

**GaussDB
24.4.0**

Getting Started

Issue 01
Date 2024-04-30



Copyright © Huawei Cloud Computing Technologies Co., Ltd. 2024. All rights reserved.

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

Trademarks and Permissions



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

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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

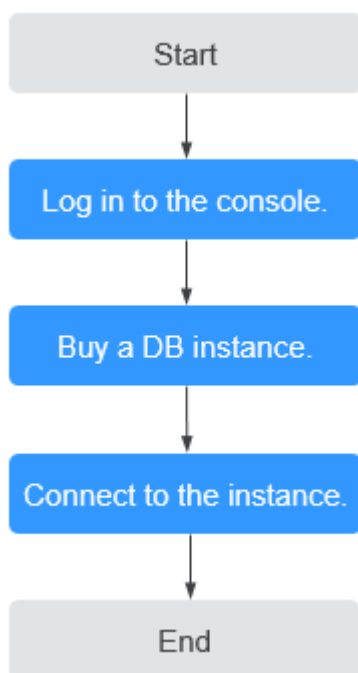
Contents

1 Operation Guide.....	1
2 Buying a DB Instance.....	3
3 Connecting to a DB Instance Using a Database Client.....	14
3.1 Connecting to a DB Instance.....	14
3.2 Connecting to an Instance Through DAS (Recommended).....	16
3.3 Using gsql to Connect to an Instance from a Linux Server.....	18
3.4 Using DBeaver to Connect to an Instance from a Windows Server.....	24
3.5 Using Navicat to Connect to an Instance from a Windows Server.....	28
4 Connecting to a DB Instance Using a Driver.....	32
4.1 Distributed Instances.....	32
4.1.1 Development Specifications.....	32
4.1.2 Using JDBC to Connect to a Database.....	33
4.1.3 Using ODBC to Connect to a Database.....	41
4.1.4 Using libpq to Connect to a Database.....	51
4.1.5 Using Psycpg to Connect to a Database.....	58
4.1.6 Using Hibernate to Connect to a Database.....	61
4.1.7 Using MyBatis to Connect to a Database.....	68
4.1.8 Using JayDeBeApi to Connect to a Database.....	70
4.2 Primary/Standby Instances.....	72
4.2.1 Development Specifications.....	72
4.2.2 Using JDBC to Connect to a Database.....	73
4.2.3 Using ODBC to Connect to a Database.....	82
4.2.4 Using libpq to Connect to a Database.....	91
4.2.5 Using Psycpg to Connect to a Database.....	99
4.2.6 Using Hibernate to Connect to a Database.....	102
4.2.7 Using MyBatis to Connect to a Database.....	109
4.2.8 Using JayDeBeApi to Connect to a Database.....	111
5 Example: Using DAS to Connect to an Instance and Execute SQL Statements....	114

1 Operation Guide

Flowchart

Figure 1-1 Flowchart



Procedure

Table 1-1 Related operations and reference

Related Operation	Reference
Creating a DB instance	Buying a DB Instance

Related Operation	Reference
Connecting to a DB instance	Select a connection method as needed: Connecting to a DB Instance Using a Database Client Connecting to a DB Instance Using a Driver

2 Buying a DB Instance

Scenarios

You can buy a DB instance on the management console.


GaussDB supports pay-per-use and yearly/monthly billing. GaussDB allows you to tailor your computing resources and storage space to your business needs.

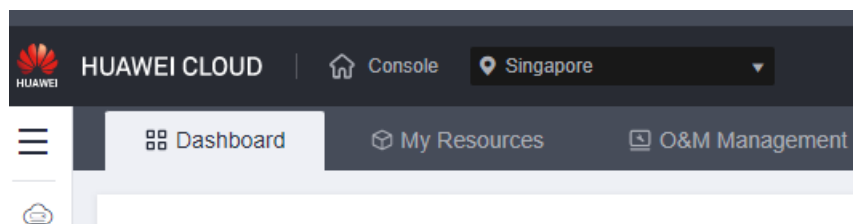
Prerequisites


You have registered a HUAWEI ID and enabled Huawei Cloud services.

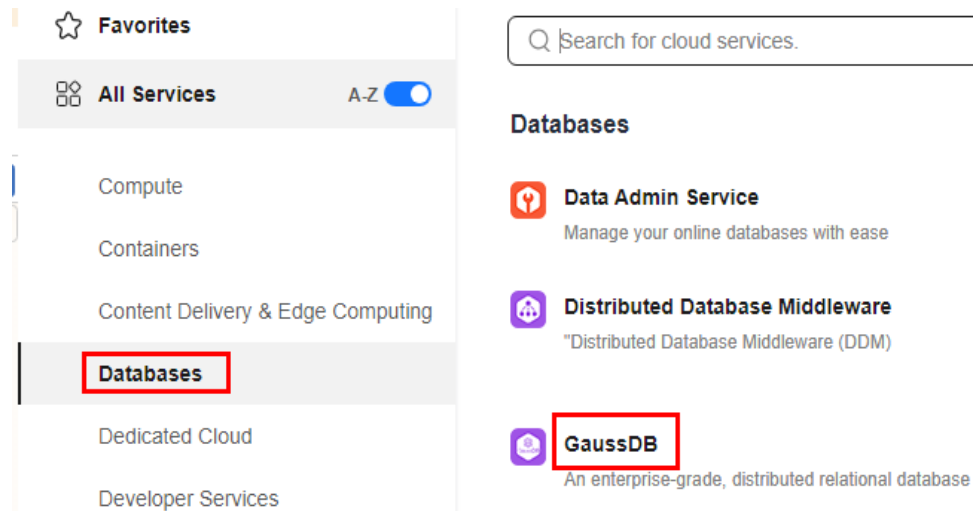
Procedure

Step 1 [Log in to the management console.](#)

Step 2 Click  in the upper left corner and select a region and project.



Step 3 Click  in the upper left corner of the page and choose **Databases > GaussDB**.



Step 4 On the **Instances** page, click **Buy DB Instance**.

Step 5 On the displayed page, select a billing mode, configure parameters about the instance, and click **Next**.

Figure 2-1 Billing mode and basic information

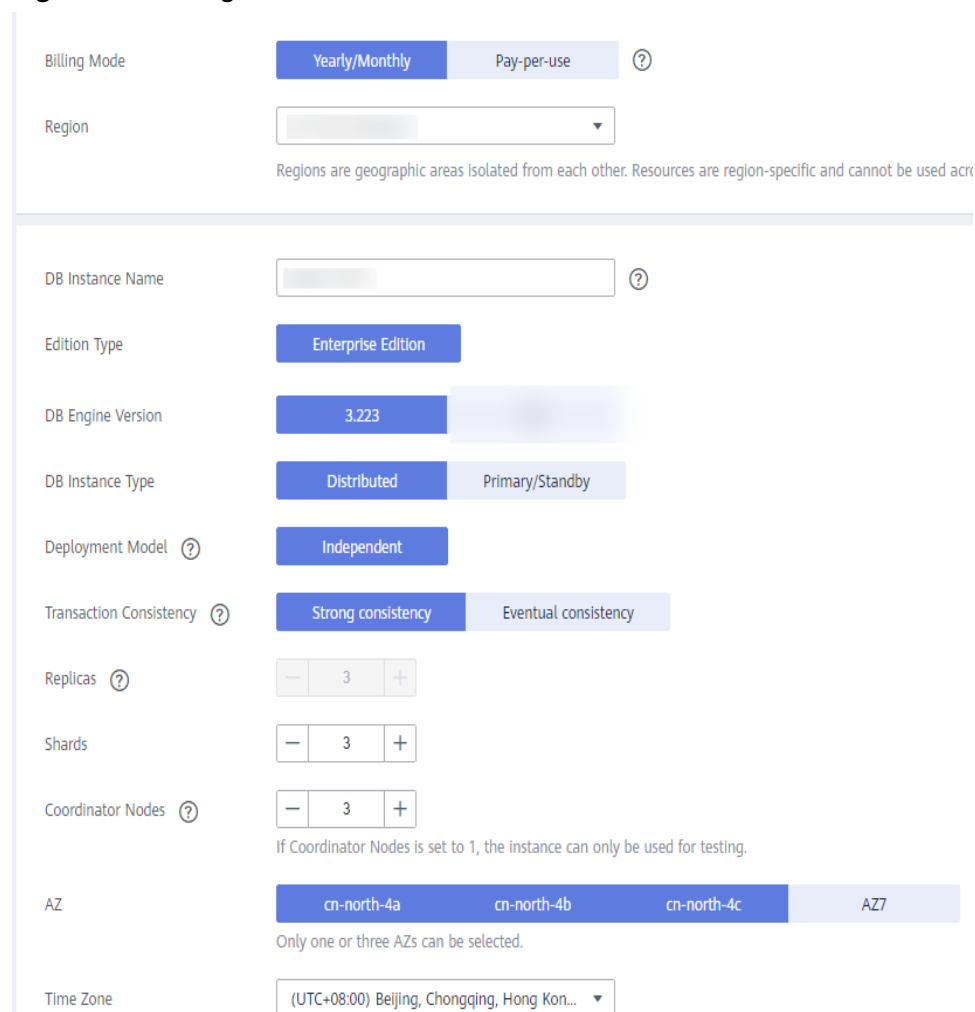


Table 2-1 Basic information

Parameter	Description
Billing Mode	<p>GaussDB provides two billing modes, yearly/monthly billing and pay-per-use billing.</p> <ul style="list-style-type: none">• Yearly/Monthly: You pay upfront for the amount of time you expect to use the DB instance for. You will need to make sure you have a top-up account with a sufficient balance or have a valid payment method configured first.• Pay-per-use: You can start using the DB instance first and then pay as you go. Pricing is listed on a per-hour basis, but bills are calculated based on the actual usage duration.
Region	<p>A region where the tenant is located. You can change the region on the instance creation page, or go back to the Instances page and change it in the upper left corner.</p> <p>NOTE Products in different regions cannot communicate with each other over a private network. After the DB instance is created, you cannot change its region.</p>
DB Instance Name	<p>The instance name must start with a letter and can contain 4 to 64 characters. Only letters (case-sensitive), digits, hyphens (-), and underscores (_) are allowed.</p>
Edition Type	<p>GaussDB provides Basic edition and Enterprise edition.</p>
DB Engine Version	<p>GaussDB 3.223 and 8.102 are supported.</p>
DB Instance Type	<ul style="list-style-type: none">• Distributed: You can add nodes for distributed instances as needed to handle large volumes of concurrent requests.• Primary/Standby: Primary/Standby instances are suitable for scenarios with small and stable volumes of data, where data reliability and service availability are extremely important.

Parameter	Description
Deployment Model	<ul style="list-style-type: none"> ● Distributed instances <ul style="list-style-type: none"> - Independent: Database components are deployed on different nodes. This model is suitable for where high reliability and stability are required and the instance scale is large. This option is available only when Edition Type is Enterprise edition. - Combined: 3-node deployment where there are three shards and each shard contains one primary DN and two standby DNs. This option is available only when Edition Type is Basic edition. ● Primary/Standby instances <ul style="list-style-type: none"> - HA (1 primary + 2 standby): 3-node deployment where there is a shard. The shard contains one primary DN and two standby DNs. - Single replica: single-node deployment where there is only one CMS component and one DN. To create a single-replica instance, ensure that the instance version is 2.2 or later. - 1 primary + 1 standby + 1 log: 3-node deployment where there is one shard with three replicas. The shard contains one primary DN, one standby DN, and one log-dedicated DN. <p>CAUTION Single replica: The availability (or SLA) cannot be guaranteed because the instance is deployed on a single server.</p> <p>NOTE</p> <ul style="list-style-type: none"> ● The combined deployment model has the following restrictions: <ul style="list-style-type: none"> - This model is available only for instances of version 3.223 or later. - Instance specifications cannot be changed. - Storage autoscaling is not supported. - Yearly/Monthly billing is not supported.
Log Nodes Supported	<p>This parameter is available only for distributed instances. If this option is selected, the distributed instance supports the 1 primary + 1 standby + 1 log deployment model.</p>

Parameter	Description
Transaction Consistency	<p>This parameter is available only to distributed instances.</p> <ul style="list-style-type: none"> • Strong consistency: When an application updates data, every user can query all data that has been successfully committed, but performance is affected. • Eventual consistency: When an application updates data, the data users queried may be different, and some users may not obtain the most current value. The most current data may take a bit of time to become available for query by all users. However, DB instances with eventual consistency generally have higher performance. Eventual consistency cannot ensure strong read consistency of distributed transactions and consistency of transactions that depend on query results, such as INSERT INTO and SELECT * FROM. Write operations that are split into multiple statements or involve in multiple nodes are not supported.
Failover Priority	<p>This function is available only to distributed instances.</p> <p>To use this parameter, contact customer service to apply for the required permissions. The default value is Reliability. For details about how to change the failover priority for an existing instance, see Changing Failover Priority.</p> <ul style="list-style-type: none"> • Reliability: Data consistency is given priority during a failover. This is recommended for applications with highest priority for data consistency. • Availability: Database availability is given priority during a failover. This is recommended for applications that require their databases to provide uninterrupted online services. <p>NOTE</p> <p>If Availability is selected, exercise caution when modifying the following database parameters. For details about how to modify parameters, see Modifying Instance Parameters.</p> <ul style="list-style-type: none"> - recovery_time_target: If this parameter is changed, the DB instance will undergo frequent forced failovers. To change this parameter, contact technical support first. - audit_system_object: If this parameter is changed, DDL audit logs will be lost. To change this parameter, contact technical support first.
Replicas	<p>This parameter is available only to distributed instances.</p> <p>Total number of DNs each shard, primary and standby DNs combined. There are three replicas in a shard, indicating that there are one primary and two standby DNs in a shard.</p>

Parameter	Description
Shards	This parameter is available only to distributed instances. It indicates the number of shards in an instance. A shard contains multiple DNs. The number of DNs in a shard depends on the value of Replicas , for example, if Replicas is set to 3 , there are three DNs (one primary and two standby DNs) in a shard.
Coordinator Nodes	This parameter is available only to distributed instances. It indicates the number of CNs in an instance. A CN provides the following functions: <ul style="list-style-type: none"> It receives access requests from applications and returns execution results to clients. It breaks down tasks and distributes task fragments to different DNs for parallel processing. <p>NOTICE It is recommended that the number of CNs be less than or equal to twice the number of shards.</p>
AZ	An AZ is a physical region where resources have their own independent power supply and networks. AZs are physically isolated but interconnected through an internal network. A DB instance can be deployed in one AZ or three AZs.
Time Zone	Select a time zone according to the region hosting your DB instance when you buy the instance.

Figure 2-2 Specifications and storage

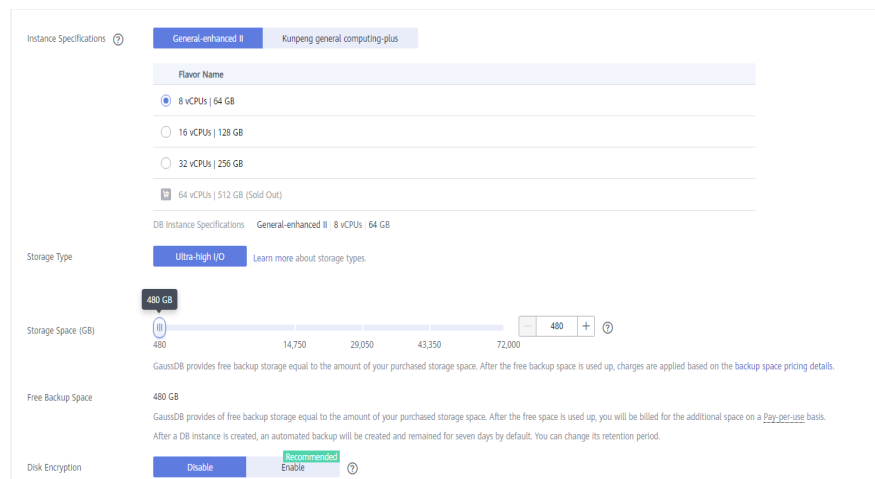


Table 2-2 Specifications and storage

Parameter	Description
Instance Specifications	Indicates the CPU and memory specifications of a DB instance. Different instance specifications have different numbers of database connections. For details, see Instance Specifications .
Dedicated Cloud	M6. NOTE This option is available only when you have purchased Dedicated Computing Cluster (DCC).
Resource Type	EVS. NOTE This option is available only when you have purchased Dedicated Computing Cluster (DCC).
Storage Type	Determines the instance read/write speed. The higher the maximum throughput is, the higher the instance read/write speed can be. GaussDB supports ultra-high I/O storage with a maximum throughput of 350 MB/s.
Storage Space (GB)	The storage space contains the system overhead required for inodes, reserved blocks, and database operation. After buying an instance, you can scale up its storage space. For details, see Scaling Up Storage Space . NOTE When you create a DB instance, the storage space for a single shard starts from 40 GB and can be increased at a step of 4 GB.
Free Backup Space	GaussDB provides free backup storage equal to the amount of your purchased storage space. After the free backup space is used up, you will be billed for the additional space used.
Disk Encryption	<ul style="list-style-type: none">● Disable: Encryption is disabled.● Enable: Encryption is enabled, which improves data security but affects system performance. Key Name: If disk encryption is enabled, you need to select or create a key, which is used by tenants.

Figure 2-3 Network and database configuration

Relationship among VPCs, subnets, security groups, and DB instances: ⓘ

VPC ⓘ ⓘ

If you want to create a VPC, go to the VPC console.

Make sure there are enough private IP addresses available for future scale-ups. After the DB instance is created, the subnet cannot be changed. Available private IP addresses in the selected subnet: 63534

Security Group ⓘ ⓘ [View Security Group](#)

In a security group, rules that authorize connections to DB instances apply to all DB instances associated with the security group.
Ensure that the TCP ports in the inbound rule of the selected security group contain 8000-8100, 20050, 5000-5001, 2379-2380, 6000, 6500, 40000-60480.

Security Group Rules

Database Port

Administrator

Administrator Password

Keep your password secure. The system cannot retrieve your password.

Confirm Password

Parameter Template ⓘ [View Parameter Template](#)

Tag

It is recommended that you use TMS's predefined tag function to add the same tag to different cloud resources. ⓘ [View predefined tags](#)

You can add 20 more tags.

Table 2-3 Network

Parameter	Description
VPC	<p>A virtual network where your GaussDB instances are located. A VPC isolates networks for different workloads. You need to create or select the required VPC. For details about how to create a VPC, see Creating a VPC.</p> <p>If no VPC is available, GaussDB allocates a default VPC for you.</p> <p>NOTICE After the GaussDB instance is created, the VPC cannot be changed.</p>
Subnet	<p>A subnet provides dedicated network resources that are logically isolated from other networks for network security. Subnets take effect only within a specific AZ. Dynamic Host Configuration Protocol (DHCP) is enabled by default for subnets in which you plan to create GaussDB instances and cannot be disabled. A private IP address is automatically assigned when you create a GaussDB instance.</p> <p>NOTE</p> <ul style="list-style-type: none"> By default, a subnet supports up to 256 IP addresses. A distributed instance can require up to 1,286 IP addresses. You are advised to use a subnet that can provide 2,048 IP addresses.

Parameter	Description
Security Group	<p>A security group controls the access that traffic has in and out of a GaussDB instance. By default, the security group associated with the instance is authorized.</p> <ul style="list-style-type: none"> • If you need to change the security group when buying a distributed instance, ensure that the TCP ports in the inbound rule include the following: 40000-60480, 20050, 5000-5001, 2379-2380, 6000, 6500, and <i><database port></i>-(<i><database port></i> + 100). (For example, if the database port is 8000, the TCP ports for the security group must include 8000-8100.) • If you need to change the security group when buying a primary/standby instance, ensure that the TCP ports in the inbound rule include the following: 20050, 5000-5001, 2379-2380, 6000, 6500, and <i><database port></i>-(<i><database port></i> + 100). (For example, if the database port is 8000, the TCP ports for the security group must include 8000-8100.) <p>The security group enhances security by controlling access to GaussDB from other services. When you select a security group, you must ensure that it allows the client to access your DB instances. If you do not need to specify a security group when creating a DB instance, you can submit a service ticket to request it at Service Tickets > Create Service Ticket in the upper right corner of the management console.</p> <p>If no security group is available, GaussDB allocates a default security group for you.</p>
Database Port	<p>The port used by applications to access the database. Value range: 1024 to 39998. Default value: 8000. The following ports are used by the system and cannot be used: 2378, 2379, 2380, 4999, 5000, 5999, 6000, 6001, 8097, 8098, 12016, 12017, 20049, 20050, 21731, 21732, 32122, 32123, and 32124.</p>
Single Floating IP Address	<p>Specifies whether to enable the single floating IP address policy. If this policy is enabled, only one floating IP address is assigned to an instance and is bound to the primary node. The floating IP address does not change after a primary/standby switchover. If this policy is disabled, each node is bound to a floating IP address. The floating IP address changes after a primary/standby switchover.</p> <p>The constraints on the single floating IP address policy are as follows:</p> <ul style="list-style-type: none"> • This policy is only available to primary/standby instances of version 3.206 or later. • The single floating IP address policy is configurable only during instance creation and cannot be modified afterwards.

Table 2-4 Database configuration

Parameter	Description
Administrator	DB administrator. The default username is root .
Administrator Password	<p>Enter a strong password and periodically change it to improve security, preventing security risks such as brute force cracking.</p> <p>NOTICE The password must contain:</p> <ul style="list-style-type: none"> • 8 to 32 characters. • At least three types of the following: uppercase letters, lowercase letters, digits, and special characters (~!@#%^*_-=+?,). <p>Keep your password secure because you cannot retrieve it from the system.</p> <p>After a DB instance is created, you can reset this password. For details, see Resetting the Administrator Password.</p>
Confirm Password	Enter the administrator password again.

Table 2-5 Parameter templates

Parameter	Description
Parameter Template	<p>A template of parameters for creating an instance. The template contains engine configuration values that are applied to one or more instances. You can modify the instance parameters as required after the DB instance is created.</p> <p>For details, see Modifying Parameters in a Parameter Template.</p>
Enterprise Project	<p>If the DB instance has been associated with an enterprise project, select the target project from the Enterprise Project drop-down list.</p> <p>You can also go to the enterprise project management console to create a project. For details, see Enterprise Management User Guide.</p>

Table 2-6 Tags

Parameter	Description
Tag	<p>This parameter is optional. Adding tags helps you better identify and manage your DB instances. Each instance can have up to 20 tags.</p> <p>If your organization has configured tag policies for GaussDB, add tags to instances based on the policies. If a tag does not comply with the policies, instance creation may fail. Contact your organization administrator to learn more about tag policies.</p>

If you have any questions about the price, click **Pricing details** at the bottom of the page.

 **NOTE**

The performance of your GaussDB instance depends on its settings. Hardware items include the instance specifications, storage type, and storage space.

Step 6 Confirm the displayed details.


Confirm your specifications for pay-per-use instances.

- If you need to modify your settings, click **Previous**.
- If you do not need to modify your settings, click **Submit**.

Confirm your order for yearly/monthly DB instances.

- If you need to modify your settings, click **Previous**.
- If you do not need to modify your settings, click **Pay Now** to go to the payment page. On the displayed page, select a payment method and click **Pay**.

Step 7 To view and manage the GaussDB instance after the creation task is submitted, go to the **Instances** page.

- When a GaussDB instance is being created, the instance status is **Creating**.
- To refresh the instance list, click  in the upper right corner of the list. When the creation process is complete, the instance status will be **Available**.
- An automated full backup is immediately triggered after once your instance is created.
- The default database port is 8000. You can change it during instance creation or after an instance is created.

----End

Related Operations

- [Creating a DB Instance Using an API](#)
- [Modifying Instance Parameters](#)

3 Connecting to a DB Instance Using a Database Client

3.1 Connecting to a DB Instance

GaussDB instances can be connected using `gsq`, DBeaver, Navicat, or Data Admin Service (DAS).

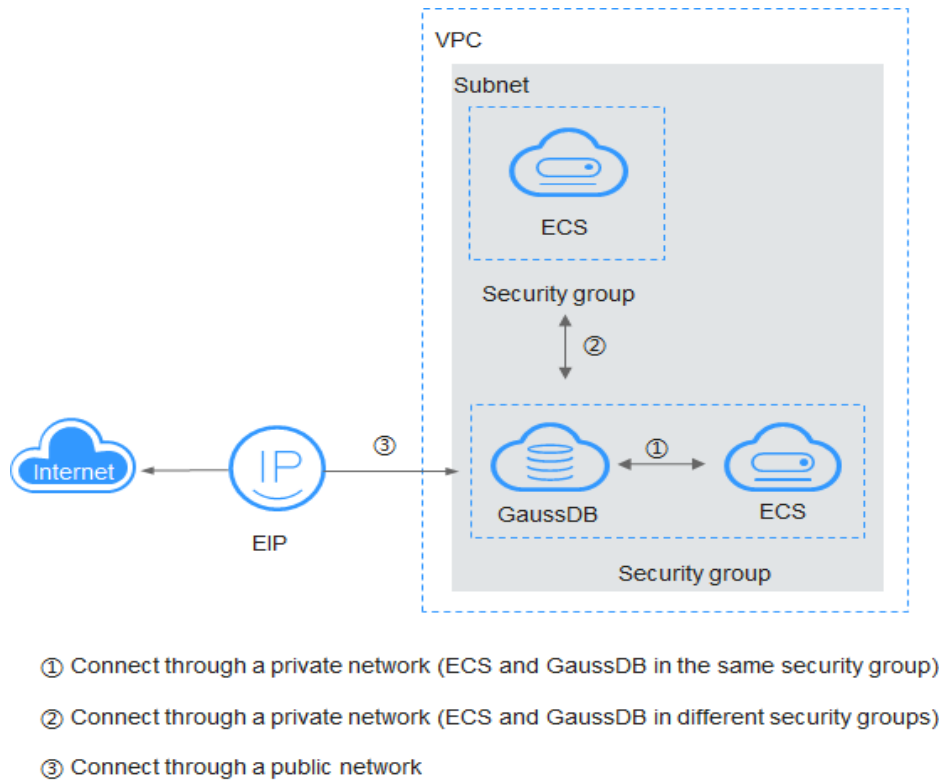
Table 3-1 GaussDB instance connection modes

Connect Through	IP Addresses	Description	Comments
DAS	Not required	DAS enables you to manage databases on a web-based console. It supports SQL execution, advanced database management, and intelligent O&M, simplifying database management and improving both efficiency and data security. The permissions required for connecting to a GaussDB instance through DAS are enabled by default.	Easy to use, secure, advanced, and intelligent

Connect Through	IP Address	Description	Comments
gsql	Private IP address/EIP	gsql is a client tool provided by GaussDB. You can use gsql to connect to the database and then enter, edit, and execute SQL statements in an interactive manner.	To achieve a higher data transmission rate and security level, migrate your applications to a server that is in the same subnet as your GaussDB instance and use a private IP address to access the instance.
DBeaver	EIP	DBeaver is a GUI-based database management tool. You can use this tool to view database schemas, execute SQL queries and scripts, browse and export data, process BLOB/CLOB data, and modify database schemas.	Open-source and easy-to-use
Navicat	EIP	Navicat is a database management tool. You can easily view and edit data on its graphical interface. For example, you can insert, delete, update, and query data, process SQL statements or scripts, use functions, and generate data.	Stable and easy to use

[Figure 3-1](#) shows how a gsql connection works.

Figure 3-1 Connecting to a DB instance using gsql



3.2 Connecting to an Instance Through DAS (Recommended)

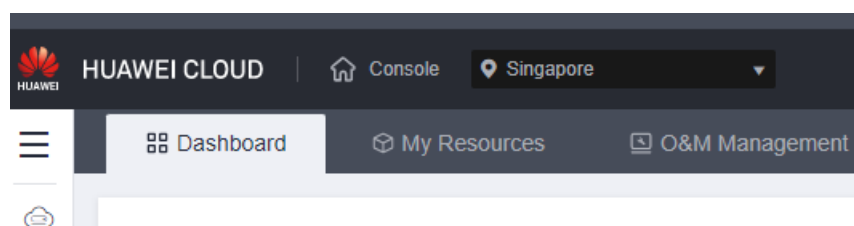
Scenarios

DAS enables you to manage your databases from a web-based console. It supports SQL execution, advanced database management, and intelligent O&M, simplifying database management and improving both efficiency and data security.

Procedure

Step 1 [Log in to the management console.](#)

Step 2 Click  in the upper left corner and select a region and project.




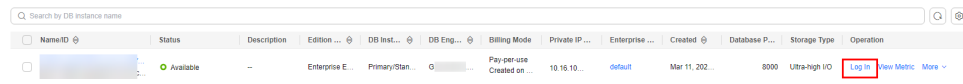
- Step 3** Click  in the upper left corner of the page and choose **Databases > GaussDB**.
- Step 4** On the **Instances** page, locate the DB instance you want to log in to and click **Log In** in the **Operation** column.

Figure 3-2 Logging in to an instance



Alternatively, click the DB instance name on the **Instances** page. On the displayed **Basic Information** page, click **Log In** in the upper right corner of the page.

Figure 3-3 Logging in to an instance



- Step 5** On the displayed login page, enter the username and password and click **Log In**.

Table 3-2 Parameter description

Parameter	Description
Login Username	Username of the GaussDB database account. The default administrator is root .
Database Name	Name of the database (postgres by default).
Password	Password of the database user.
Collect Metadata Periodically	You are advised to enable Collect Metadata Periodically . If it is disabled, DAS obtains only the structured data from databases in real time, and the performance of databases is affected. The collection time cannot be customized. Once Collect Metadata Periodically is enabled, DAS collects metadata at 20:00 every day (UTC time). If you are not using a UTC time, convert the time according to your local time zone. You can also click Collect Now to collect metadata at any time you want.
Show Executed SQL Statements	You are advised to enable Show Executed SQL Statements . With it enabled, you can view the executed SQL statements under SQL Operations > SQL History and execute them again without entering the SQL statements.

For details about how to use DAS to manage databases, see [GaussDB Management](#).

----End

Follow-up Operations

After logging in to the DB instance, you can create or migrate your databases.

- [Creating a Database](#)
- [Migrating the Database](#)

3.3 Using `gsql` to Connect to an Instance from a Linux Server

This section describes how to use the `gsql` client to connect to a GaussDB instance you have bought on the GaussDB management console.

- [Step 1: Buy an ECS](#)
- [Step 2: Query the IP Address and Port Number of the Instance to Be Connected](#)
- [Step 3: Test the Connectivity](#)
- [Step 4: Obtain the Driver Package](#)
- [Step 5: Connect to the Database](#)
 - [Non-SSL connection](#)
 - [SSL connection](#)

Buying an ECS

If you want to connect to a database using the command-line interface (CLI), like `gsql`, you need to create an ECS and install `gsql` on it.

1. [Log in to the management console](#) and check whether there is an available ECS.
 - If there is, go to [3](#).
 - If there is not, go to [2](#).

Figure 3-4 ECS instances

Name/ID	AZ	Status	Specifications/Image	IP Address	Enterprise Project	Tag	Operation
ecs-5308		Running	1 vCPU/1.2 GB CentOS7.4	192.168.0.103 (Private IP)	default	--	Remote Login More

2. Buy an ECS that runs EulerOS.
For details about how to buy a Linux ECS, see [Purchasing an ECS](#) in *Elastic Cloud Server Getting Started*.
3. On the **ECS Information** page of the target ECS, view the region and VPC of the ECS.

Figure 3-5 ECS basic information

ECS Information

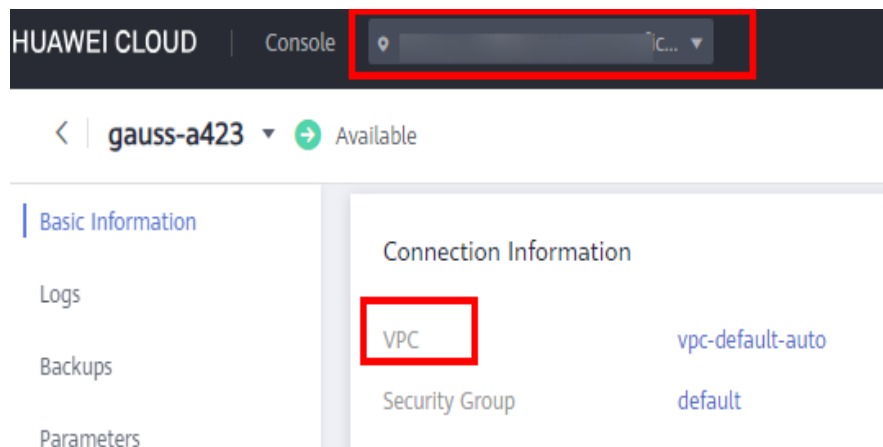
ID	604c198e-018a-4b67-91fd-80a46166ac78
Name	gau [redacted] eip
Region	[redacted]
AZ	AZ1
Specifications	General computing-plus 2 vCPUs 4 GiB c3.large.2
Image	CentOS 8.0 64bit for Tenant 20210227 Public image
VPC	vpc-default-auto
Obtained	Nov 08, 2022 09:38:48 GMT+08:00
Launched	Nov 08, 2022 09:38:55 GMT+08:00

NOTICE

The ECS must run EulerOS. gsql supports the following versions:
For x86 servers: EulerOS V2.0SP5 and Kylin V10 SP2
For Kunpeng servers: EulerOS V2.0SP8 and Kylin V10 SP1

4. On the **Basic Information** page of your GaussDB instance, view the region and VPC of the instance.



Figure 3-6 Basic information about a GaussDB instance



5. Check whether the ECS and GaussDB instance are in the same region and VPC.

- If the ECS and GaussDB instance are in the same region and VPC, the DB instance can be connected through a private network. For details about how to obtain the private IP address, see [Querying the IP Address of the Instance to Be Connected](#).
- If the ECS and DB instance are in different VPCs, the DB instance must be connected over a public network. For details about how to obtain the public IP address, see [Querying the IP Address of the Instance to Be Connected](#). Ensure that both the ECS and GaussDB instance have EIPs.
 - For details about how to bind an EIP to an ECS, see [Binding an EIP](#).
 - For details about how to bind an EIP to a GaussDB instance, see [Binding an EIP](#).

Querying the IP Address and Port Number of the Instance to Be Connected

1. [Log in to the management console](#).
2. Click  in the upper left corner and select a region and project.
3. Click  in the upper left corner of the page and choose **Databases > GaussDB**.
4. On the **Instances** page, click the name of the target instance to go to the **Basic Information** page.
5. In the **Connection Information** area, view the IP address and port number.
 - If the ECS and GaussDB instance are in the same VPC, obtain the private IP address and database port number.
 - If the ECS and GaussDB instance are in different VPCs, obtain the EIP and database port number.

Testing Connectivity

1. Log in to the ECS. For details, see [Login Using VNC](#) in *Elastic Cloud Server User Guide*.
2. On the ECS, check whether it can connect to the target GaussDB instance using the IP address and port number obtained in [Querying the IP Address and Port Number of the Instance to Be Connected](#).

telnet *IP address Port number*

Example:

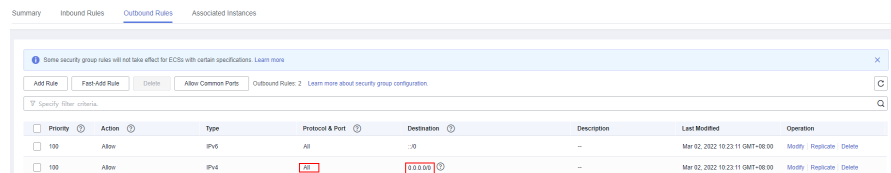
```
telnet 192.168.0.16 8000
```

NOTE

- If the message "command not found" is displayed, install a Telnet client that matches the OS of the ECS.
- If the ECS can connect to the DB instance, no further action is required.
- If the communication fails, check the security group rules.
 - If **Destination** is not **0.0.0.0/0** and **Protocol & Port** is not **All** on the **Outbound Rules** page of the ECS, add the IP address and port of the GaussDB instance to the outbound rules.

- If the ECS and GaussDB instance are in the same VPC, add the private IP address and port of the GaussDB instance to the outbound rules.
- If the ECS and GaussDB instance are in different VPCs, add the EIP address and port of the GaussDB instance to the outbound rules.

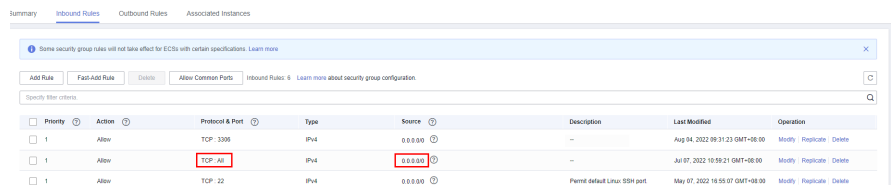
Figure 3-7 ECS security group



- If **Source** is not **0.0.0.0/0** and **Protocol & Port** is not **All** on the **Inbound Rules** page of the GaussDB instance, add the IP address and port of the ECS to the inbound rules.
 - If the ECS and GaussDB instance are in the same VPC, add the private IP address and port of the ECS to the inbound rules.
 - If the ECS and GaussDB instance are in different VPCs, add the EIP address and port of the ECS to the inbound rules.

For details, see [Configuring Security Group Rules](#).

Figure 3-8 GaussDB security group



Obtaining the Driver Package

Download particular packages listed in [Table 3-3](#) based on the version of your instance.

Table 3-3 Driver package download list

Version	Download Address
3.x	Driver package Verification package for the driver package
2.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

1. Upload the software package and verification package to the same directory on a Linux VM.
2. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```

Connecting to a Database

- **Non-SSL connection**

- a. Log in as user **root** to the ECS you have created.
- b. Upload the client tool package and configure gsql environment variables.

- i. Run the following command to create the **/tmp/tools** directory for storing the client tool package:

```
mkdir /tmp/tools
```

- ii. Download the **GaussDB_driver.zip** driver package of the required version by referring to [Obtaining the Driver Package](#), and upload it to the **/tmp/tools** directory of the created ECS.

- iii. Run the following commands to decompress the **GaussDB_driver.zip** driver package:

```
cd /tmp/tools  
unzip GaussDB_driver.zip
```

- iv. Run the following commands to copy the decompressed **GaussDB-Kernel_V***R***C**_EULER_64bit-Gsql.tar.gz** client tool package to the **/tmp/tools** directory:

NOTE

This section uses the gsql tool package suitable for the primary/standby instances running on Euler2.5_x86_64 as an example. The relative path of the tool package varies depending on where you decompressed it.

```
cd /tmp/tools/GaussDB_driver/Centralized/Euler2.5_X86_64/  
cp GaussDB-Kernel_V***R***C**_EULER_64bit-Gsql.tar.g /tmp/tools
```

- v. Run the following commands to decompress the package:

```
cd /tmp/tools  
tar -zxvf GaussDB-Kernel_V***R***C**_EULER_64bit-Gsql.tar.g
```

- vi. Configure environment variables.

Run the following command to open the **~/.bashrc** file:

```
vim ~/.bashrc
```

Press **G** to move the cursor to the last line and press **i** to enter Insert mode, and type the following information. Then, press **Esc** to exit Insert mode, and run **:wq** to save the settings and exit.

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

Run the following command to make the environment variables take effect permanently:

```
source ~/.bashrc
```


- c. Enter the password when prompted to connect to the database.

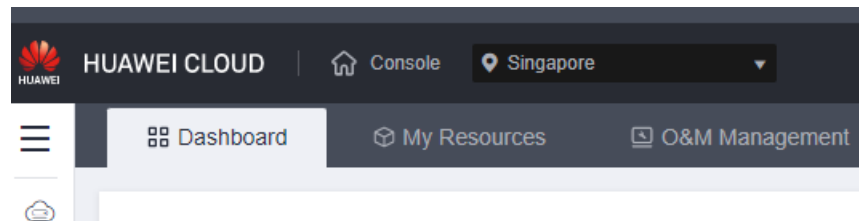
After the database is installed, a database is generated. Database **postgres** is used as an example.



```
gsqldb -d postgres -h 10.0.0.0 -U root -p 8000
Password for user root:
```

postgres is the name of the database you want to connect. **10.0.0.0** is the IP address of the instance obtained in [Querying the IP Address of the Instance to Be Connected](#). **root** is the username for logging in to the database. **8000** is the database port obtained in [Querying the Port Number of the Instance to Be Connected](#).

- **SSL connection**

- a. [Log in to the management console](#).
- b. Click  in the upper left corner and select a region and project.



- c. Click  in the upper left corner of the page and choose **Databases > GaussDB**.
- d. On the **Instances** page, click the name of the target instance. In the **DB Information** area on the **Basic Information** page, click  next to the **SSL** field to download the root certificate or certificate bundle.
- e. Upload the root certificate to the ECS or save it to the device to be connected to the GaussDB instance.

Import the root certificate to the Linux ECS. For details, see [How Can I Import the Root Certificate to a Windows or Linux OS?](#)

- f. Connect to a GaussDB instance.

A Linux ECS is used in this example. Run the following command to set environment variables on the ECS:

```
export PGSSLMODE=<sslmode>
export PGSSLROOTCERT=<ca-file-directory>
```

```
gsqldb -h <host> -p <port> -d <database> -U <user>
```

Table 3-4 Parameters

Parameter	Description
<host>	IP address of the DB instance. To obtain this parameter, go to the Basic Information page of the DB instance. If the DB instance is accessed through an ECS, the IP address can be found in the Private IP Address field of the Connection Information area.

Parameter	Description
<code><port></code>	Database port in use. The default value is 8000 . To obtain this parameter, go to the Basic Information page of the DB instance. The port number can be found in the Database Port field in the Connection Information area.
<code><database></code>	Name of the database (postgres by default).
<code><user></code>	Username of the GaussDB database account. The default administrator is root .
<code><ca-file-directory></code>	Path of the CA certificate for SSL connection.
<code><sslmode></code>	SSL connection mode. Set it to verify-ca to use a CA to check whether the service is trusted.

For example, to connect to a **postgres** database through an SSL connection as user **root**, run the following commands on the ECS:

```
export PGSSLMODE="verify-ca"
export PGSSLROOTCERT="/home/Ruby/ca.pem"
```

```
gsql -d postgres -h 10.0.0.0 -U root -p 8000
```

Password for user root:

- g. Check the command output after you log in to the database. If information similar to the following is displayed, the SSL connection has been established.

```
SSL connection (cipher: DHE-RSA-AES256-GCM-SHA384, bits: 256)
```

Helpful Links

For more information about `gsq`l commands, see [Tool Reference](#).

3.4 Using DBeaver to Connect to an Instance from a Windows Server

DBeaver is a multi-platform database client for you to connect to different databases using particular drivers. This section describes how to use DBeaver to connect to a GaussDB instance.

Step 1: Obtain the Driver Package

1. Obtain the driver package and its verification package.
Download the driver package and its verification package of the relevant version to any local directory. [Table 3-5](#) lists the download list.

Table 3-5 Driver package download list

Version	Download Address
3.x	Driver package Verification package for the driver package
2.x	Driver package Verification package for the driver package

2. Verify the driver package.

To prevent the driver package from being maliciously tampered during transfer or storage, perform the following steps to verify the driver package:

- a. Press **Win+R** to open the **Run** text box. Type **cmd** in the **Open** field and press **Enter** to open the **Command Prompt** window.
- b. Run the following command to obtain the hash value of the driver package:

```
certutil -hashfile {Local directory of the driver package}\{Driver package name} sha256
```

- Replace *{Local directory of the driver package}* with the actual download path, for example, **C:\Users**.
- Replace *{Driver package name}* with the name of the downloaded driver package, for example, **GaussDB_driver.zip**.

Example: **certutil -hashfile C:\Users\GaussDB_driver.zip sha256**

- c. Compare the hash value obtained in **2.b** with the hash value of the verification package obtained in **1**.
 - If they are consistent, the verification is successful.
 - If they are inconsistent, download the driver package again and repeat **2.a** to **2.c** to verify the driver package.

3. Decompress the driver package.

Decompress the driver package obtained in **1** to the local PC, and place the **gsjdbc4.jar** package included in it to any local directory.

Step 2: Obtain the DBeaver Client Installation Package

The DBeaver official website provides client installation packages for different OSs. [Download](#) the required DBeaver client installation package, and install it on the local PC.

Step 3: Create a Driver

1. Start the DBeaver client.
2. Choose **Database > Driver Manager**.
3. In the displayed window, click **New**.

4. On the **Settings** tab, set **Driver Name**, **Class Name**, **URL Template**, **Default Port**, **Default Database**, and **Default User**, select a driver type, and click **OK**.

Table 3-6 Parameters

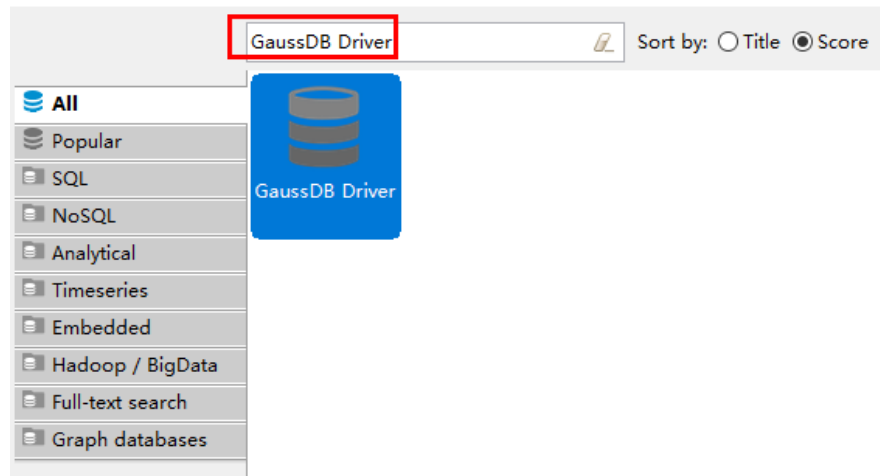
Parameter	Description
Driver Name	Use a name that is easy to identify, for example, GaussDB Driver .
Driver Type	Set it to Generic .
Class Name	Set it to org.postgresql.Driver .
URL Template	Set it to jdbc:postgresql://{host}[:{port}]/[{database}] .
Default Port	Set it to the port of your DB instance specified during instance creation. The default port of a GaussDB instance is 8000.
Default Database	Set it to the name of the database to be connected. After a DB instance is created, a database named postgres is generated by default.
Default User	Set it to the name of the user who will access the GaussDB instance. The default user is root .

5. On the **Libraries** tab, click **Add File** to add **gsjdbc4.jar** obtained in [3](#).
6. After the file is added, the driver class is empty. Click **Find Class** to set the driver class. The identified driver class must be the same as the class name specified on the **Settings** tab.
7. Click **OK** to complete the driver settings.

Step 4: Connect to the Database

1. On the DBeaver client, choose  to create a connection.
2. Search for the driver created in [Step 3](#), select the driver, and click **Next**.

Figure 3-9 Selecting a driver



3. Enter the host IP address, port number, database name, username, and password.

Table 3-7 Parameters

Parameter	Description
Host	Private IP address of the DB instance to be connected. To view the private IP address and port of the DB instance, perform the following steps: <ol style="list-style-type: none"> 1. Log in to the GaussDB management console. 2. Select the region in which the target instance is located. 3. Click the name of the target instance to enter the Basic Information page. 4. In the Connection Information area, view the EIP of the instance. If no EIP is bound to the instance, bind one to the instance first. For details, see Binding an EIP.
Port	Port of your DB instance specified during instance creation. The default port of a GaussDB instance is 8000.
Database/Schema	Name of the database to be connected. After a DB instance is created, a database named postgres is generated by default.
Username	Name of the user who will access the GaussDB instance. The default user is root .
Password	Password of the user who will access the GaussDB instance.

4. Click **Test Connection**. If **Connected** is displayed in the dialog box, the connection is successful. Click **OK**.

5. Click **Finish** to connect to the database. You can view information about the connected database in the **Database Navigator** area.

3.5 Using Navicat to Connect to an Instance from a Windows Server

Navicat Premium 16.2.8 for Windows PC now supports GaussDB management and development. This section describes how to use Navicat to connect to a GaussDB instance.

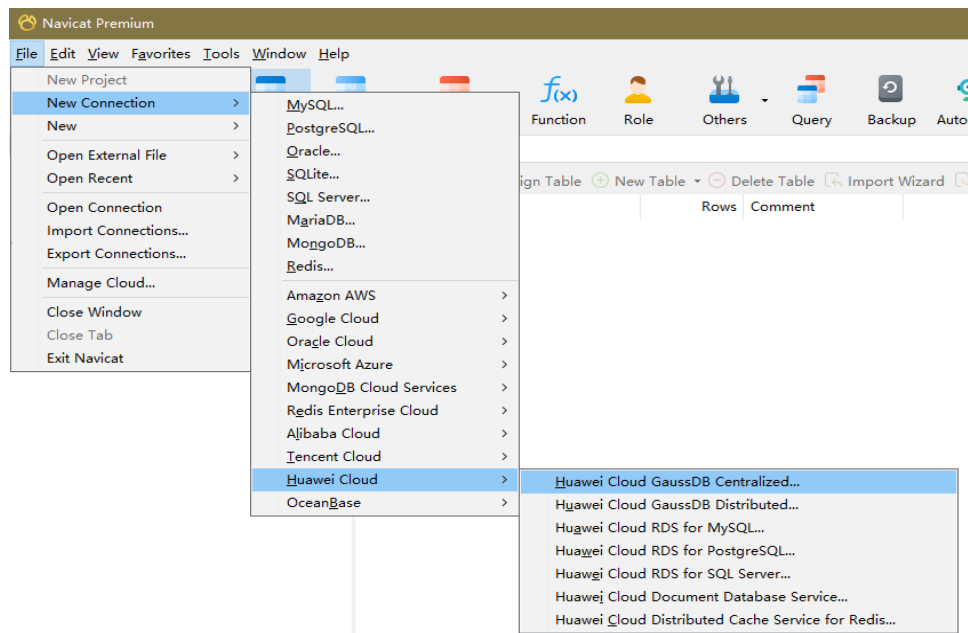
Prerequisites

You have **downloaded** or bought Navicat Premium and installed it on the local PC.

Procedure

- Step 1** Start the Navicat Premium client and choose **File > New Connection > Huawei Cloud > Huawei Cloud GaussDB Centralized** or **Huawei Cloud GaussDB Distributed**.

Figure 3-10 Creating a connection



- Step 2** In the **New Connection** window, enter the correct connection name, host, port, initial database, user name, and password.

Figure 3-11 Setting information for connecting to a primary/standby instance

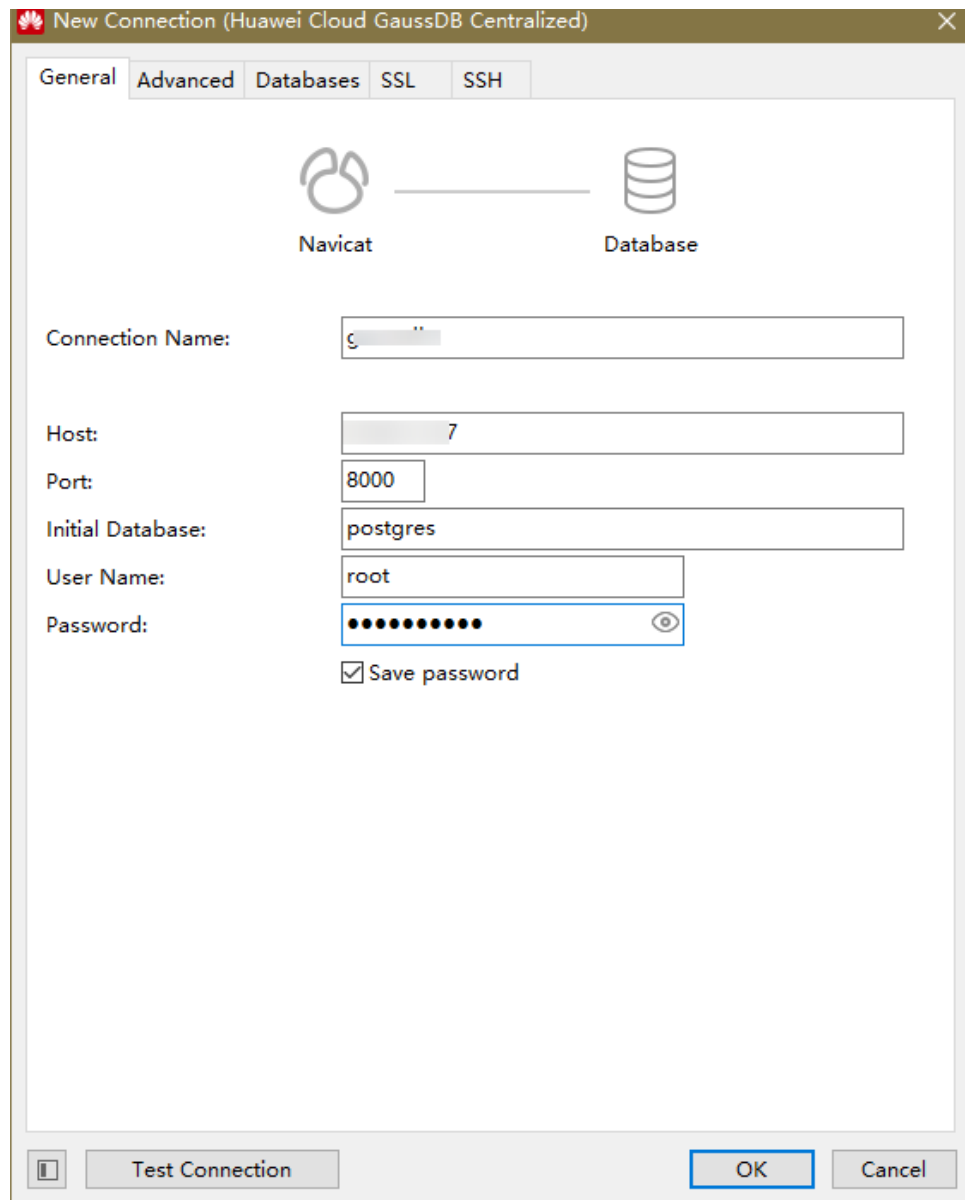


Figure 3-12 Setting information for connecting to a distributed instance

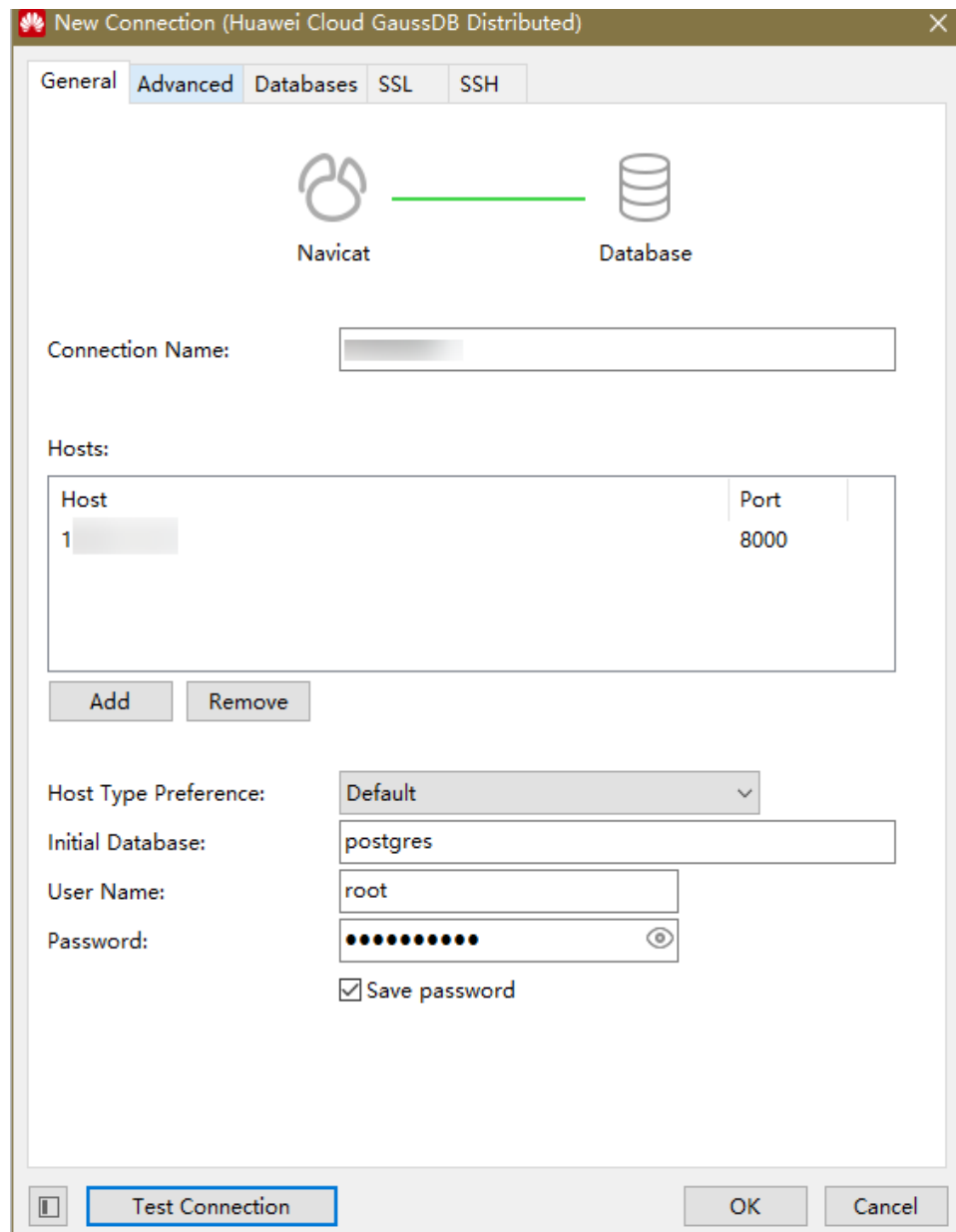


Table 3-8 Parameters

Parameter	Description
Connection Name	Use a name that is easy to identify.

Parameter	Description
Host	Private IP address of the DB instance to be connected. To view the private IP address and port of the DB instance, perform the following steps: <ol style="list-style-type: none">1. Log in to the GaussDB management console.2. Select the region in which the target instance is located.3. Click the name of the target instance to enter the Basic Information page.4. In the Connection Information area, view the EIP of the instance. If no EIP is bound to the instance, bind one to the instance first. For details, see Binding an EIP.
Port	Port of your DB instance specified during instance creation. The default port of a GaussDB instance is 8000.
Initial Database	Name of the database to be connected. After a DB instance is created, a database named postgres is generated by default.
User Name	Name of the user who will access the GaussDB instance. The default user is root .
Password	Password of the user who will access the GaussDB instance.

Step 3 Click **Test Connection**. If **Connection Successful** is displayed in the dialog box, the connection is normal. Click **OK** to close the dialog box.

Step 4 Click **OK**. The connection is disabled by default after being created.

Step 5 Right-click the connection name and choose **Open Connection** from the shortcut menu.

Step 6 Right-click the database name and choose **Open Database** from the shortcut menu.

----End

4 Connecting to a DB Instance Using a Driver

You can connect to DB instances using drivers, such as JDBC and ODBC.

4.1 Distributed Instances

4.1.1 Development Specifications

If the connection pool mechanism is used during application development, comply with the following specifications:

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

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

Table 4-1 describes the compatibility of application development drivers.

Table 4-1 Compatibility Description

Driver	Compatibility Description
JDBC, Go, ODBC, libpq, Psycopg, and ecpg	The new drivers are forward compatible with the database. To use the new features added to the driver and database, you must upgrade the database.

NOTICE

- In principle, you need to set the compatibility parameter after the database creation, instead of switching the parameters when using the database.
- The JDBC driver must be upgraded to that maps to GaussDB Kernel 503.1.0 or later if the following features are used:
 - The `s2` compatibility parameter is enabled and the validity check of `sessiontimezone` is set.

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

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

4.1.2 Using JDBC to Connect to a Database

Java Database Connectivity (JDBC) is a Java API for running SQL statements. It provides unified access APIs for different relational databases, based on which applications process data. The GaussDB library supports JDBC 4.2 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridge.

Prerequisites

Java JDK 8 has been installed on the local PC.

JDBC Package

The package name is **GaussDB-Kernel_***Database version number_OS version number_64bit_Jdbc.tar.gz*.

After the decompression, you will obtain the following JDBC packages in JAR format:

- **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package is recommended. The Java code examples in this section use the **gaussdbjdbc.jar** package by default.
- **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package contains the dependent libraries related to encryption and decryption that need to be loaded to the encrypted database. You are advised to use this driver package in encrypted scenarios. Currently, only EulerOS is supported.
- **gaussdbjdbc-JRE7.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. The **gaussdbjdbc-JRE7.jar** package is used in the JDK 1.7 environment.

 **CAUTION**

- Before using the **gscejdbc.jar** driver package, you need to set the environment variable `LD_LIBRARY_PATH`. For details, see section "Setting Encrypted Equality Queries > Using JDBC to Operate an Encrypted Database" in *Feature Guide*.
- In JDK 1.8, you are advised to use **gaussdbjdbc.jar** instead of **gaussdbjdbc-JRE7.jar**.
- For details about other JDBC JAR packages, see "Application Development Guide" > "JDBC Compatibility Package" in *Developer Guide*.

Driver Class

Before establishing a database connection, load the **com.huawei.gaussdb.jdbc.Driver** database driver class.

 **NOTE**

1. GaussDB is compatible with PostgreSQL in the use of JDBC. Therefore, when two JDBC drivers are used in the same process, class names may conflict.
2. JDBC of this version does not support identity & access management suite (IAM) for authentication.
3. The GaussDB JDBC driver has the following enhanced features:
 1. The SHA256 encryption mode is supported for login.
 2. The third-party log framework that implements the sf4j API can be connected.
 3. Distributed load balancing at the connection level is supported.
 4. DR failover is supported.

Environment Class

The JDK1.8 must be configured on the client. JDK supports multiple platforms such as Windows and Linux. The following uses Windows as an example to describe how to configure JDK:

- Step 1** Enter **java -version** in the MS-DOS window (command prompt in Windows) to check the JDK version. Ensure that the JDK version is JDK1.8. If the JDK is not installed, download the installation package and install it.
- Step 2** On the Windows desktop, right-click **This PC** and choose **Properties** from the shortcut menu.
- Step 3** In the displayed **System** window, click **Advanced system settings** in the navigation tree on the left.
- Step 4** In the **System Properties** dialog box, click **Environment Variables** in the lower right corner.
- Step 5** In the **System variables** area of the **Environment Variables** dialog box, click **New** or **Edit** to configure system variables. For details about the variables, see [Table 4-2](#).

Table 4-2 Variables

Variable	Operation	Variable Value
JAVA_HOME	<ul style="list-style-type: none"> If the variable exists, click Edit. If the variable does not exist, click New. 	Specifies the Java installation directory. Example: C:\Program Files\Java\jdk1.8.0_131 .
Path	Click Edit .	<ul style="list-style-type: none"> If <i>JAVA_HOME</i> is configured, add <i>%JAVA_HOME%\bin</i> before the variable value. If <i>JAVA_HOME</i> is not configured, add the full Java installation path before the variable value: C:\Program Files\Java\jdk1.8.0_131\bin
CLASSPATH	Click New .	%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar

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

----End

Loading the Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Implicit loading at any position before a connection is created in the code:
`Class.forName("com.huawei.gaussdb.jdbc.Driver")`
- Parameter transfer during JVM startup: `java -Djdbc.drivers=com.huawei.gaussdb.jdbc.Driver jdbctest`

 **NOTE**

`jdbctest` is the name of a test application.

Function Prototype

JDBC provides three database connection methods.

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

Parameters

Table 4-3 Database connection parameters

Parameter	Description
url	<p>gaussdbjdbc.jar database connection descriptor.</p> <p>If host is set to a server name or an IPv4 address, formats are as follows:</p> <ul style="list-style-type: none">• jdbc:gaussdb: (If the database name is left empty, the username is used.)• jdbc:gaussdb:database• jdbc:gaussdb://host/database• jdbc:gaussdb://host:port/database• jdbc:gaussdb://host:port/database?param1=value1&param2=value2• jdbc:gaussdb://host1:port1,host2:port2/database?param1=value1&param2=value2 <p>If host is set to an IPv6 address, formats are as follows:</p> <ul style="list-style-type: none">• jdbc:gaussdb: (If the database name is left empty, the username is used.)• jdbc:gaussdb:database• jdbc:gaussdb://host/database or jdbc:gaussdb://[host]/database• jdbc:gaussdb://[host]:port/database• jdbc:gaussdb://[host]:port/database?param1=value1&param2=value2• jdbc:gaussdb://[host1]:port1,[host2]:port2/database?param1=value1&param2=value2

Parameter	Description
	<p>NOTE</p> <ul style="list-style-type: none"> • database indicates the name of the database to connect. • host indicates the name or IP address of the database server. Both IPv4 and IPv6 addresses are supported. For security purposes, the database CN forbids access from other nodes in the cluster without authentication. To access the CN from inside the cluster, deploy the JDBC program on the host where the CN is located and set host to 127.0.0.1. Otherwise, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the cluster. If it is deployed inside, database performance may be affected. By default, the local host is used to connect to the server. • port indicates the port number of the database server. By default, the database on port 5432 of the local host is connected. • If host is set to an IPv6 address and the port number is specified in the URL, use square brackets ([]) to enclose the IP address. The format is [IP address]:Port number. • param indicates a database connection attribute. The parameter can be configured in the URL. The URL starts with a question mark (?), uses an equal sign (=) to assign a value to the parameter, and uses an ampersand (&) to separate parameters. You can also use the attributes of the info object for configuration. For details, see Examples. • value indicates the database connection attribute values. • The connectTimeout and socketTimeout parameters must be set for connection. If they are not set, the default value 0 is used, indicating that the connection will not time out. When the network between the DN and client is faulty, the client does not receive the ACK packet from the DN. In this case, the client starts the timeout retransmission mechanism to continuously retransmit packets. A timeout error is reported only when the timeout interval reaches the default value 600s. As a result, the RTO is high. • You are advised to ensure the validity of the URL when using the standard JDBC API to establish a connection. An invalid URL may cause an exception, and the exception contains the original URL character string, which may cause sensitive information leakage.
info	For details about common attributes of info , see "Application Development Guide" > "Development Based on JDBC" > "Connecting to a Database" in <i>Developer Guide</i> .
user	Database user.
password	Password of the database user.

 **NOTE**

After the **uppercaseAttributeName** parameter is enabled, if the database contains metadata with a mixture of uppercase and lowercase letters, only the metadata in lowercase letters can be queried and output in uppercase letters. Before using the metadata, ensure that the metadata is stored in lowercase letters to prevent data errors.

Examples

Example 1: Connect to a database.

```
// gaussdbjdbc.jar is used as an example.
// The following code encapsulates database connection operations into an API. The database can then be
// connected using an authorized username and a password.
public static Connection getConnect(String username, String passwd)
{
    // Driver class.
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    // Database connection descriptor.
    String sourceURL = "jdbc:gaussdb://$ip:$port/database";
    Connection conn = null;

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

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

    return conn;
}
```

Example 2: Use the Properties object as a parameter to create a connection.

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

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

    try
    {
        info.setProperty("user", username);
        info.setProperty("password", passwd);
        // Create a connection.
    }
}
```

```
        conn = DriverManager.getConnection(sourceURL, info);
        System.out.println("Connection succeed!");
    }
    catch(Exception e)
    {
        e.printStackTrace();
        return null;
    }

    return conn;
}
```

For details about common parameters, see "Application Development Guide" > "Development Based on JDBC" > "Common JDBC Parameters" in *Developer Guide*.

Example 3: Use the streaming read function.

// There will be security risks if the username and password used for authentication are directly written into code. It is recommended that the username and password be stored in the configuration file or environment variables (the password must be stored in ciphertext and decrypted when being used) to ensure security.

// In this example, the username and password are stored in environment variables. Before running this example, set environment variables *EXAMPLE_USERNAME_ENV* and *EXAMPLE_PASSWORD_ENV* in the local environment (set the environment variable names based on the actual situation).

// Establish a connection.

```
public static Connection getConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableStreamingQuery=true";
    Connection conn = null;
    try {
        // Load the driver.
        Class.forName(driver);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    try {
        // Create a connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
        System.out.println("Connection succeed!");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    return conn;
}
```

// Execute common SQL statements to create table **t_user**.

```
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();

        // Execute common SQL statements.
        stmt.executeUpdate("DROP TABLE IF EXISTS t_user");
        stmt.executeUpdate("CREATE TABLE t_user(id int, name VARCHAR(20));");
        stmt.close();
    } catch (SQLException e) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}
```

// Execute a prepared statement to insert data in batches.

```
public static void BatchInsertData(Connection conn) {
    PreparedStatement pst = null;

    try {
        // Generate a prepared statement.
        pst = conn.prepareStatement("INSERT INTO t_user VALUES (?,?)");
        for (int i = 0; i < 20; i++) {
            // Add parameters.
            pst.setInt(1, i + 1);
            pst.setString(2, "name " + (i + 1));
            pst.addBatch();
        }
        // Perform batch processing.
        pst.executeBatch();
        pst.close();
    } catch (SQLException e) {
        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// Enable streaming read and query the content in the t_user table.
public static void StreamingQuery(Connection conn) {
    PreparedStatement pst = null;
    ResultSet resultSet = null;

    try {
        // Query all values in the t_user table.
        pst = conn.prepareStatement("SELECT * FROM t_user");
        pst.setFetchSize(Integer.MIN_VALUE); // Functions the same as
        ((PgStatement)statement).enableStreamingResults();
        resultSet = pst.executeQuery();
        while (resultSet.next()) {
            System.out.println(resultSet.getInt(1));
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }

        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");
    Connection conn = getConnection(userName, password);

    CreateTable(conn);
}
```

```
BatchInsertData(conn);
StreamingQuery(conn);

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

NOTICE

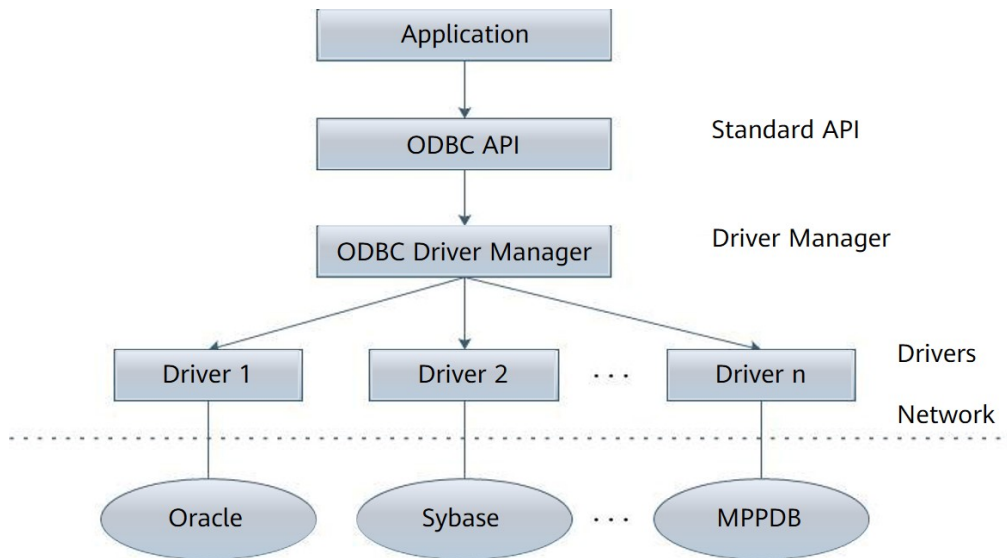
When the streaming read function is used, you need to perform the `resultSet.close()` or `statement.close()` operation after the result set is used. Otherwise, the current connection is unavailable.

4.1.3 Using ODBC to Connect to a Database

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

Figure 4-1 shows the system structure of ODBC.

Figure 4-1 ODBC system structure



GaussDB supports ODBC 3.5 in the following environments.

Table 4-4 OSs Supported by ODBC

OS	Platform
EulerOS V2.0SP5	x86_64

OS	Platform
EulerOS V2.0SP9	Arm64
EulerOS V2.0SP10	x86_64
EulerOS V2.0SP10	Arm64
Windows 7	x86_32
Windows 7	x86_64
Windows Server 2008	x86_32
Windows Server 2008	x86_64
Kylin V10	x86_64
Kylin V10	Arm64
UnionTech V20	x86_64
UnionTech V20	Arm64
Huawei Cloud EulerOS 2.0	x86_64
Huawei Cloud EulerOS 2.0	Arm64

The ODBC Driver Manager running on Unix or Linux can be unixODBC or iODBC. unixODBC-2.3.7 is used as the component for connecting to the database.

Windows has a native ODBC Driver Manager. You can locate **Data Sources (ODBC)** by choosing **Control Panel > Administrative Tools**.

NOTE

The current database ODBC driver is based on an open-source version and may be incompatible with Huawei-developed data types such as tinyint, smalldatetime, and nvarchar2.

ODBC Constraints

- ODBC does not support DR switchover.
- When the **proc_outparam_override** parameter is enabled for the database, ODBC cannot properly call the stored procedure that contains the **out** parameter.

Prerequisites

You have downloaded the ODBC driver packages for Linux and Windows.

- The package name for Linux is **GaussDB-Kernel_Database version number_OS version number_64bit_Odbc.tar.gz**. In the Linux OS, header files (including **sql.h** and **sqltext.h**) and the library (**libodbc.so**) are required in application development. These header files and library can be obtained from the unixODBC-2.3.7 installation package.

- The package name for Windows is **GaussDB-Kernel_Database version number_Windows_Odbc_X64.tar.gz** (64-bit). In the Windows OS, the required header files and library files are system-resident.

Procedure in a Linux Server

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

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

After the download, verify the integrity based on the integrity verification algorithm provided by the community. Download <https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5>, view the MD5 value, and check whether the MD5 value is the same as that in the source code package.

Step 2 Install unixODBC. It does not matter if unixODBC of another version has been installed.

For example, install unixODBC-2.3.7.

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7
```

```
./configure --enable-gui=no # To perform compilation on an Arm server, add the configure parameter --  
build=aarch64-unknown-linux-gnu.  
make  
# The installation may require root permissions.  
make install
```

NOTE

- Currently, unixODBC-2.2.1 is not supported.
- It is installed in the **/usr/local** directory by default. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.
- You can compile unixODBC with the **--enable-fastvalidate=yes** option to achieve higher performance. However, this option may cause an application that passes an invalid handle to the ODBC API to fail instead of returning an SQL_INVALID_HANDLE error.

Step 3 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel_Database version number_OS version number_64bit_Odbc.tar.gz**. After the decompression, the **lib** and **odbc** folders are generated. The **odbc** folder contains another **lib** folder. Copy all dynamic libraries in the **/lib** and **/odbc/lib** folders to the **/usr/local/lib** directory.

Step 4 Configure the data source.

1. Configure the ODBC driver file.

Add the following content to the **/usr/local/etc/odbcinst.ini** file:

```
[GaussMPP]
Driver64=/usr/local/lib/gsqlodbcw.so
setup=/usr/local/lib/gsqlodbcw.so
```

For descriptions of the parameters in the **odbcinst.ini** file, see [Table 4-5](#).

Table 4-5 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to Driver in DSN.	[DRIVER_N]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/gsqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/gsqlodbcw.so

2. Configure the data source file.

Add the following content to the `/usr/local/etc/odbc.ini` file:

```
[gaussdb]
Driver=GaussMPP
Servername=127.0.0.1 # Database server IP address
Database=db1 # Database name
Username=omm # Database username
Password= # Database user password
Port=8000 # Database listening port
Sslmode = allow
```

Table 4-6 describes the parameters in the `odbc.ini` file.

Table 4-6 odbc.ini configuration parameters

Parameter	Description	Example Value
[DSN]	Data source name.	[gaussdb]
Driver	Driver name, corresponding to DriverName in <code>odbcinst.ini</code> .	Driver = DRIVER_N
Servername	Server IP address. Multiple IP addresses can be configured. Both IPv4 and IPv6 are supported.	Servername=127.0.0.1
Database	Name of the database to connect.	Database=db1
Username	Database username.	Username=omm

Parameter	Description	Example Value
Password	<p>Database user password.</p> <p>NOTE After a user establishes a connection, the ODBC driver automatically clears their password stored in memory.</p> <p>However, if this parameter is configured, unixODBC will cache data source files, which may cause the password to be stored in the memory for a long time.</p> <p>When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored.</p> <p>CAUTION The password in the configuration file must comply with the following HTTP rules:</p> <ol style="list-style-type: none"> 1. Characters must comply with the URL encoding specifications. For example, the exclamation mark (!) must be written as %21, and the percent sign (%) must be written as %25. Therefore, pay attention to the characters. 2. A plus sign (+) will be replaced by a space. 	Password=*****

Parameter	Description	Example Value
Port	Port number of the server. When load balancing is enabled, multiple port numbers can be configured and must correspond to multiple IP addresses. If multiple IP addresses are configured and only one port number is configured when load balancing is enabled, all IP addresses share the same port number by default, that is, the configured port number.	Port=8000
Sslmode	Specifies whether to enable SSL.	Sslmode = allow

For values of the **Sslmode** parameter, see [Table 4-7](#).

Table 4-7 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified.

sslmode	Whether SSL Encryption Is Enabled	Description
verify-full	Yes	<p>SSL connection is required. In addition to the check scope specified by verify-ca, the system checks whether the name of the host where the database resides is the same as that in the certificate. If they are different, modify the /etc/hosts file as user root and add the IP address and host name of the connected database node to the file.</p> <p>NOTE This mode does not support the default certificate of the product. Contact the administrator to generate a certificate.</p>

Step 5 Enable the SSL mode. For details, contact the database administrator.

Step 6 Configure the database server. For details, contact the database administrator.

Step 7 Configure the environment variables on the client.

```
vim ~/.bashrc
```

Add the following information to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
export ODBC_SYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 8 Run the following command to validate the addition:

```
source ~/.bashrc
```

Step 9 Connect to the database.

```
isql -v GaussODBC
```

GaussODBC: data source name.

- If the following information is displayed, the configuration is correct and the connection succeeds:

```
+-----+
| Connected!                |
|                            |
| sql-statement             |
| help [tablename]         |
| quit                      |
|                            |
+-----+
```

- If error information is displayed, the configuration is incorrect. Check the configuration.
- In a cluster environment, you need to copy and configure the unixODBC file on all nodes.

----End

Procedure in a Windows Server

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

Step 1 Replace the GaussDB client driver.

Decompress the **GaussDB-Kernel_Database version number_Windows_X64_Odbc.tar.gz** (64-bit) driver package or **GaussDB-Kernel_Database version number_Windows_X86_Odbc.tar.gz** (32-bit) driver package, and click **gsqlodbc.exe** to install the driver.

Step 2 Open the driver manager.

When configuring the data source, use the ODBC driver manager corresponding to the ODBC version. If the 64-bit ODBC driver is used, the 64-bit ODBC driver manager must be used. Assume that the OS is installed on drive C (if the OS is installed on another drive, change the path accordingly):

- If you want to use 32-bit ODBC driver manager in a 64-bit OS, open **C:\Windows\SysWOW64\odbcad32.exe**. Do not choose **Control Panel > Administrative Tools > Data Sources (ODBC)**.

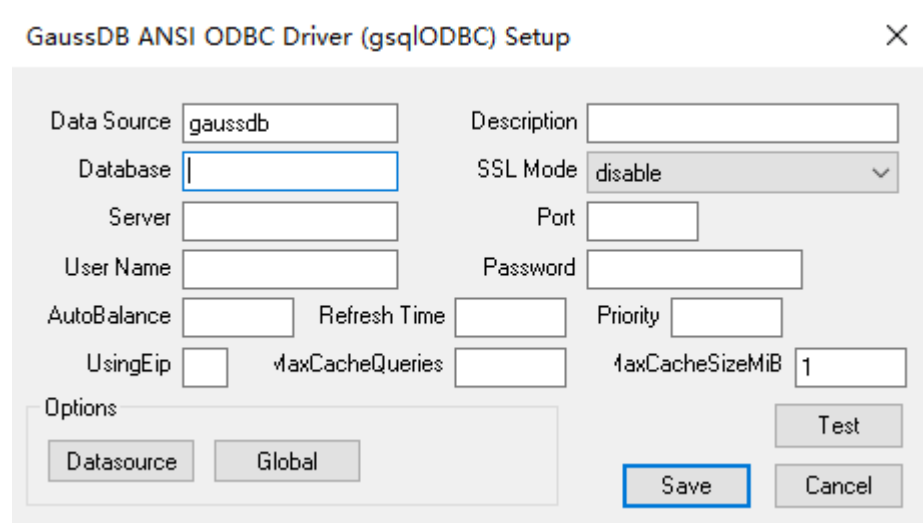
NOTE

WoW64 is short for Windows 32-bit on Windows 64-bit. **C:\Windows\SysWOW64** stores the 32-bit environment on a 64-bit system. **C:\Windows\System32** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- For a 32-bit OS, open **C:\Windows\System32\odbcad32.exe** or choose **Computer > Control Panel > Administrative Tools > Data Sources (ODBC)** to open Driver Manager.
- For a 64-bit OS, choose **Control Panel > Administrative Tools > Data Sources (ODBC)** to enable driver management.

Step 3 Configure the data source.

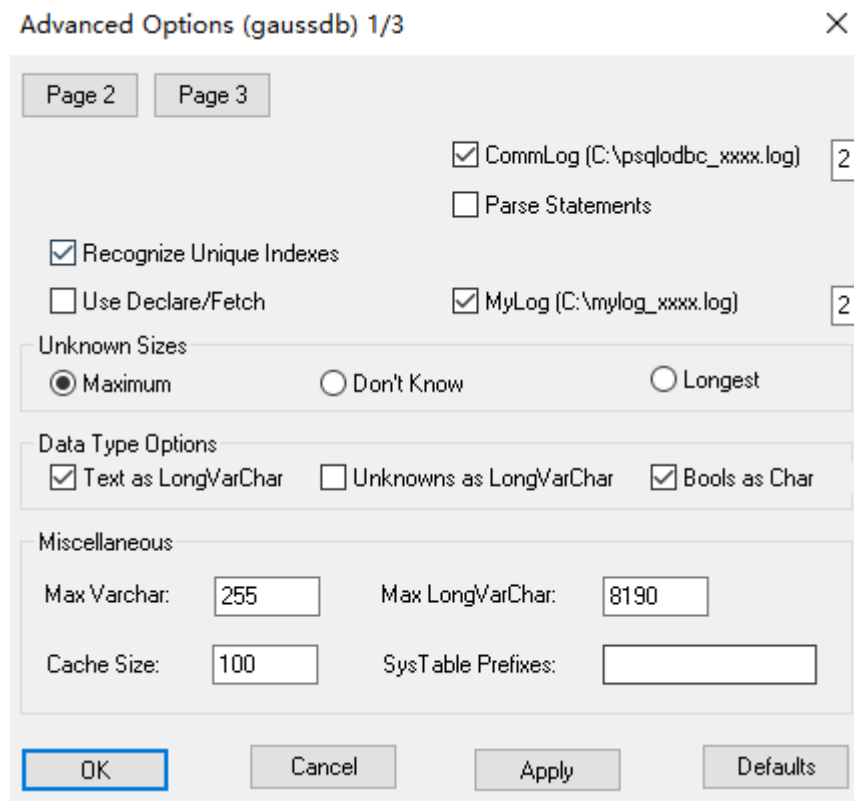
On the **User DSN** tab, click **Add** and choose **GaussDB Unicode** for setup.



The screenshot shows the 'GaussDB ANSI ODBC Driver (gsqloDBC) Setup' dialog box. The 'Data Source' field is filled with 'gaussdb'. The 'Database' field is empty. The 'Server' field is empty. The 'User Name' field is empty. The 'Password' field is empty. The 'AutoBalance' checkbox is unchecked. The 'Refresh Time' field is empty. The 'Priority' field is empty. The 'UsingEip' checkbox is unchecked. The 'MaxCacheQueries' field is empty. The 'MaxCacheSizeMiB' field is filled with '1'. The 'SSL Mode' dropdown menu is set to 'disabled'. There are 'Test', 'Save', and 'Cancel' buttons. At the bottom, there are 'Datasource' and 'Global' radio buttons.

For details about the parameters, see [Procedure in a Linux Server](#).

You can click **Datasource** to configure whether to print logs.



NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Configure SSL mode.

Change the value of **SSL Mode** in **Step 3** to **require**.

Table 4-8 sslmode options

sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.

sslmode	Whether SSL Encryption Is Enabled	Description
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified. Currently, Windows ODBC does not support the certificate-based authentication.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by verify-ca , the system checks whether the name of the host where the database resides is the same as that on the certificate. Currently, Windows ODBC does not support the certificate-based authentication.

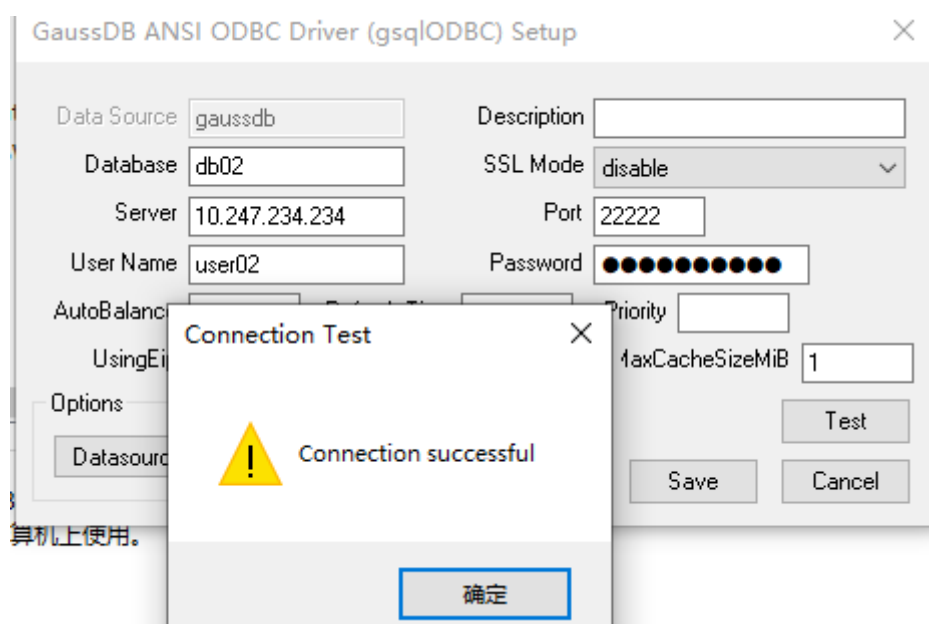
Step 5 Configure a GaussDB server. For details, contact the administrator.

Step 6 Restart the cluster.

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

Step 7 Click **Test** to test the connection.

- If the following information is displayed, the configuration is correct and the connection succeeds.



- If error information is displayed, the configuration is incorrect. Check the configuration.

----End

4.1.4 Using libpq to Connect to a Database

libpq is a C application programming interface to GaussDB. libpq contains a set of library functions that allow client programs to send query requests to GaussDB servers and obtain query results. It is also the underlying engine of other GaussDB application interfaces, such as ODBC. This chapter provides examples to show how to write code using libpq.

Prerequisites

A C development environment has been installed on the local PC.

To compile and develop source programs based on libpq, perform the following steps:

- Decompress the **GaussDB-Kernel_Database version number_OS version number_64bit_Libpq.tar.gz** file. The required header file is stored in the **include** folder, and the **lib** folder contains the required libpq library file.

NOTE

In addition to **libpq-fe.h**, the **include** folder contains the header files **postgres_ext.h**, **gs_thread.h**, and **gs_threadlocal.h** by default. These three header files are the dependency files of **libpq-fe.h**.

- Develop the source program **testlibpq.c**. The source code file needs to reference the header file provided by libpq.

Example: #include <libpq-fe.h>

- To compile the libpq source program by running **gcc**, use the **-I directory** option to provide the installation location of the header file. (Sometimes the compiler looks for the default directory, so this option can be ignored.)

Example:

```
gcc -I (Directory where the header file is located) -L (Directory where the libpq library is located)
testlibpq.c -lpq
```

Example: **gcc -I \$(GAUSSHOME)/include/libpq -L \$(GAUSSHOME)/lib -lpq testlibpq.c -o testlibpq**

- If the makefile is used, add the following option to variables **CPPFLAGS**, **LDFLAGS**, and **LIBS**:

```
CPPFLAGS += -I (Directory of the header file)
LDFLAGS += -L (Directory of the libpq library)
LIBS += -lpq
```

Example:

```
CPPFLAGS += -I$(GAUSSHOME)/include/libpq
LDFLAGS += -L$(GAUSSHOME)/lib
```

Code for Common Functions

Example 1:

```
/*
 * testlibpq.c
 * Note: testlibpq.c source program provides basic and common application scenarios of libpq.
 * The PQconnectdb, PQexec, PQntuples, and PQfinish APIs provided by libpq are used to establish database
 * connections, execute SQL statements, obtain returned results, and clear resources.
 */
```

```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or
    configuration files. Environment variables need to be configured as required. If no environment variable is
    used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn *conn;
    PGresult *res;
    int nFields;
    int i,j;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *host = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
    * This value is used when the user provides the value of the conninfo character string in the command
    line.
    * Otherwise, the environment variables or the default values
    * are used for all other connection parameters.
    */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

    /* Check whether the backend connection has been successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
        exit_nicely(conn);
    }

    /*
    * Since a cursor is used in the test case, a transaction block is required.
    * Put all data in one "select * from pg_database"
    * PQexec() is too simple and is not recommended.
    */

    /* Start a transaction block. */
    res = PQexec(conn, "BEGIN");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
}
```

```
/*
 * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
 */
PQclear(res);

/*
 * Fetch data from the pg_database system catalog.
 */
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/* First, print out the attribute name. */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* Print lines. */
for (i = 0; i < PQntuples(res); i++)
{
    for (j = 0; j < nFields; j++)
        printf("%-15s", PQgetvalue(res, i, j));
    printf("\n");
}

PQclear(res);

/* Close the portal. We do not need to check for errors. */
res = PQexec(conn, "CLOSE myportal");
PQclear(res);

/* End the transaction. */
res = PQexec(conn, "END");
PQclear(res);

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

return 0;
}
```

Example 2:

```
/*
 * testlibpq2.c Test PQprepare
 * PQprepare creates a prepared statement with specified parameters for PQexecPrepared to execute the
 * prepared statement.
 * Before running this example, create a table and insert data.
 * create table t01(a int, b int);
 * insert into t01 values(1, 23);
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
```



```
/* The values of variables such as user and passwd must be read from environment variables or
configuration files. Environment variables need to be configured as required. If no environment variable is
used, a character string can be directly assigned. */
PGconn *conn;
PGresult * res;
ConnStatusType pgstatus;
char connstr[1024];
char cmd_sql[2048];
int nParams = 0;
int paramLengths[5];
int paramFormats[5];
Oid paramTypes[5];
char * paramValues[5];
int i, cnt;
char cid[32];
int k;
char *passwd = getenv("EXAMPLE_PASSWD_ENV");
char *port = getenv("EXAMPLE_PORT_ENV");
char *hostaddr = getenv("EXAMPLE_HOST_ENV");
char *username = getenv("EXAMPLE_USERNAME_ENV");
char *dbname = getenv("EXAMPLE_DBNAME_ENV");

/* Use PQconnectdb to connect to the database. The detailed connection information is as follows:
connstr */
sprintf(connstr,
        "hostaddr=%s dbname=%s port=%s user=%s password=%s",
        hostaddr, dbname, port, username, paswswd);
conn = PQconnectdb(connstr);
pgstatus = PQstatus(conn);
if (pgstatus == CONNECTION_OK)
{
    printf("Connect database success!\n");
}
else
{
    printf("Connect database fail:%s\n", PQerrorMessage(conn));
    return -1;
}

/* Create table t01. */
res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
values(1, 23);");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    printf("Command failed: %s.\n", PQerrorMessage(conn));
    PQfinish(conn);
    return -1;
}

/* cmd_s
sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
/*Parameter corresponding to $1 in cmd_sql*/
paramTypes[0] = 23;
/* PQprepare creates a prepared statement with given parameters. */
res = PQprepare(conn,
                "pre_name",
                cmd_sql,
                1,
                paramTypes);
if (PQresultStatus(res) != PGRES_COMMAND_OK )
{
    printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
    PQfinish(conn);
    return -1;
}
PQclear(res);
paramValues[0] = cid;
for (k=0; k<2; k++)
{
```

```
    sprintf(cid, "%d", 1);
    paramLengths[0] = 6;
    paramFormats[0] = 0;
    /*Execute the prepared statement.*/
    res = PQexecPrepared(conn,
        "pre_name",
        1,
        paramValues,
        paramLengths,
        paramFormats,
        0);

    if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
    {
        printf("%s\n",PQerrorMessage(conn));
        PQclear(res);
        PQfinish(conn);
        return -1;
    }
    cnt = PQntuples(res);
    printf("return %d rows\n", cnt);
    for (i=0; i<cnt; i++)
    {
        printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
    }
    PQclear(res);
}
/* The execution is complete. Close the connection. */
PQfinish(conn);
return 0;
}
```

Example 3:

```
/*
 * testlibpq3.c
 * Test PQexecParams.
 * PQexecParams runs a command to bind parameters and requests the query result in binary format.
 * Before running this example, populate a database.
 *
 *
 * CREATE TABLE test1 (i int4, t text);
 *
 * INSERT INTO test1 values (2, 'ho there');
 *
 * Expected output:
 *
 *
 * tuple 0: got
 * i = (4 bytes) 2
 * t = (8 bytes) 'ho there'
 */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohl/htonl */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
```

```

* This function is used to print out the query results. The results are in binary format
* and fetched from the table created in the comment above.
*/
static void
show_binary_results(PGresult *res)
{
    int    i;
    int    i_fnum,
          t_fnum;

    /* Use PQfnumber to avoid assumptions about field order in the result. */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char    *iptr;
        char    *tptr;
        int     ival;

        /* Obtain the field value. (Ignore the possibility that they may be null.) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * The binary representation of INT4 is the network byte order,
         * which is better to be replaced with the local byte order.
         */
        ival = ntohl(*(uint32_t *) iptr);

        /*
         * The binary representation of TEXT is text. Since libpq can append a zero byte to it,
         * and think of it as a C string.
         */

        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
              PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
              PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or
    configuration files. Environment variables need to be configured as required. If no environment variable is
    used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn    *conn;
    PGresult  *res;
    const char *paramValues[1];
    int       paramLengths[1];
    int       paramFormats[1];
    uint32_t  binaryIntVal;
    char      *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char      *port = getenv("EXAMPLE_PORT_ENV");
    char      *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char      *username = getenv("EXAMPLE_USERNAME_ENV");
    char      *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * If the user provides a parameter on the command line,
     * The value of this parameter is a conninfo character string. Otherwise,
     * Use environment variables or default values.
     */
}
```

```
if (argc > 1)
    conninfo = argv[1];
else
    sprintf(conninfo,
        "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
        dbname, port, hostaddr, username, passwd);

/* Connect to the database. */
conn = PQconnectdb(conninfo);

/* Check whether the connection to the server was successfully established. */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
        PQerrorMessage(conn));
    exit_nicely(conn);
}

res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

PQclear(res);

res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

PQclear(res);

/* Convert the integer value "2" to the network byte order. */
binaryIntVal = htonl((uint32_t) 2);

/* Set the parameter array for PQexecParams. */
paramValues[0] = (char *) &binaryIntVal;
paramLengths[0] = sizeof(binaryIntVal);
paramFormats[0] = 1; /* Binary */
/* PQexecParams runs a command to bind parameters. */
res = PQexecParams(conn,
    "SELECT * FROM test1 WHERE i = $1::int4",
    1, /* One parameter */
    NULL, /* Enable the backend to deduce the parameter type. */
    paramValues,
    paramLengths,
    paramFormats,
    1); /* Binary result is required. */

if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
/* Output the binary result.*/
show_binary_results(res);

PQclear(res);

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

```
    return 0;  
}
```

4.1.5 Using Psycopg to Connect to a Database

Psycopg is a Python API used to execute SQL statements and provides a unified access API for GaussDB. Applications can perform data operations based on psycopg. Psycopg2 is the encapsulation of libpq and is implemented using the C language, which is efficient and secure. It provides cursors on both clients and servers, asynchronous communication and notification, and the COPY TO and COPY FROM functions. It supports multiple types of Python out-of-the-box and adapts to GaussDB data types. Through the flexible object adaptation system, you can extend and customize the adaptation. Psycopg2 is compatible with Unicode.

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

Table 4-9 Platforms supported by psycopg

OS	Platform	Python Version
EulerOS V2.0SP5	<ul style="list-style-type: none">• Arm64• x86_64	3.8.5
EulerOS V2.0SP9	<ul style="list-style-type: none">• Arm64• x86_64	3.7.4
EulerOS V2.0SP10, Kylin V10, and UnionTech20	<ul style="list-style-type: none">• Arm64• x86_64	3.7.9
EulerOS V2.0SP11 and SUSE 12.5	<ul style="list-style-type: none">• Arm64• x86_64	3.9.11
Huawei Cloud EulerOS 2.0	<ul style="list-style-type: none">• Arm64• x86_64	3.9.9

NOTICE

During `psycopg2` compilation, OpenSSL of GaussDB is linked. OpenSSL of GaussDB may be incompatible with OpenSSL of the OS. If incompatibility occurs, for example, "version 'OPENSSL_1_1_1f' not found" is displayed, use the environment variable `LD_LIBRARY_PATH` to isolate the OpenSSL provided by the OS and the OpenSSL on which GaussDB depends.

For example, when the application software **client.py** that invokes `psycopg2` is executed, the environment variable is explicitly assigned to the application software.

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

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

Prerequisites

A Python development environment has been installed on the local PC.

Connecting to a Database

Step 1 Prepare related drivers and dependent libraries. Obtain the package **GaussDB-Kernel_Database version number_OS version number_64bit_Python.tar.gz** from the release package.

After the decompression, the following folders are generated:

- **psycopg2**: `psycopg2` library file
- **lib**: `lib` library file

Step 2 Load the driver.

- Before using the driver, perform the following operations:
 - a. Decompress the driver package of the corresponding version.

```
tar zxvf xxxx-Python.tar.gz
```
 - b. Copy **psycopg2** to the **site-packages** folder in the Python installation directory as the **root** user.

```
su root
cp psycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
```
 - c. Change the **psycopg2** directory permission to **755**.

```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```
 - d. Add the **psycopg2** directory to the environment variable `$PYTHONPATH` and validate it.

```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```
 - e. For non-database users, configure the **lib** directory in `LD_LIBRARY_PATH` after decompression.

```
export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
```
- Load a database driver before creating a database connection.

```
import psycopg2
```

Step 3 Connect to a database.

Connect to the database in non-SSL mode.

1. Use the `psycopg2.connect` function to obtain the connection object.
2. Use the connection object to create a cursor object.

Connect to the database in SSL mode.

When you use `psycopy2` to connect to the GaussDB server, you can enable SSL to encrypt the communication between the client and server. To enable SSL, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

1. Use the `.ini` file (the `configparser` package of Python can parse this type of configuration file) to save the configuration information about the database connection.
2. Add SSL connection parameters `sslmode`, `sslcert`, `sslkey`, and `sslrootcert` to the connection options.
 - a. **sslmode**: For details about the options, see [Table 4-10](#).
 - b. **sslcert**: client certificate path.
 - c. **sslkey**: client key path.
 - d. **sslrootcert**: root certificate path.
3. Use the `psycopg2.connect` function to obtain the connection object.
4. Use the connection object to create a cursor object.

 **CAUTION**

To use SSL to connect to the database, ensure that the Python interpreter is compiled in the mode of generating a dynamic link library (.so) file. You can perform the following steps to check the connection mode of the Python interpreter:

1. Run the `import ssl` command in the Python interpreter to import SSL.
 2. Run the `ps ux` command to query the PID of the Python interpreter. Assume that the PID is `*****`.
 3. In the shell CLI, run the `pmap -p ***** | grep ssl` command and check whether the command output contains the path related to `libssl.so`. If the command output contains the path, the Python interpreter is compiled in dynamic link mode.
-

Table 4-10 sslmode options

sslmode	Enable SSL Encryption	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.

sslmode	Enable SSL Encryption	Description
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required, but only data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required, and the validity of the server CA must be verified.
verify-full	Yes	The SSL connection must be enabled, which is not supported by GaussDB currently.

----End

4.1.6 Using Hibernate to Connect to a Database

Hibernate is an object-relational mapping (ORM) tool for the Java programming language. It provides an easy-to-use framework for automatically mapping Java objects to database tables, so that developers can operate databases in object-oriented mode. Hibernate frees developers from manually writing a large amount of SQL and JDBC code, which significantly reduces the development workload of the data access layer.

This section describes how to use Hibernate to connect to a GaussDB database and perform operations in it, for example, creating a table, modifying a table, and inserting, deleting, updating, or querying data in a table.

Configuring the POM Dependency

```
<dependency>
  <groupId>com.huaweicloud.gaussdb</groupId>
  <artifactId>opengaussjdbc</artifactId>
  <version>503.2.T35</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.3.7.Final</version>
</dependency>
```

CAUTION

The Maven environment must have been configured before you configure the POM dependency.

Configuring the hibernate.cfg.xml Resource File

```
<?xml version="1.0" encoding="utf-8"?>
  <!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
  <hibernate-configuration>
    <session-factory>
      <!--GaussDB connection information-->
      <property name="connection.driver_class">com.huawei.opengauss.jdbc.Driver</property>
      <property name="connection.url">jdbc:opengauss://**.*.*.*.* (Replace *.*.*.*.* with the database
        IP address.)20000 (Replace 20000 with the database port.)/test? currentSchema=test (Replace test with the
        target database and schema.)</property>
      <property name="connection.username">*** (Replace *** with the correct username.)</property>
      <property name="connection.password">***** (Replace ***** with the correct password.)</property>

      <!-- The following configurations are optional. -->

      <!-- Specify whether to support dialects. -->
      <!-- In PostgreSQL compatibility mode, the following configuration must be set. -->
      <property name="dialect">org.hibernate.dialect.PostgreSQL82Dialect</property>

      <!-- Specify whether to print SQL statements when CRUD commands are executed. -->
      <property name="show_sql">>true</property>

      <!-- Enable automatic table creation (and update).-->
      <!-- <property name="hbm2ddl.auto">update</property>-->
      <!-- <property name="hbm2ddl.auto">create</property>-->

      <!-- Register resources (entity class mapping file).-->
      <mapping resource="./student.xml"/>
    </session-factory>
  </hibernate-configuration>
```

Preparing an Entity Class and Entity Class Mapping File

In this example, the **Student** class and **student.xml** file are used. The file path is **com.howtodoinjava.hibernate.test.dto**.

1. Create an entity class.

```
public class Student implements Serializable {
    int id;
    String name;
    // String age;
    // The constructor, get, and set methods are omitted here.
}
```

2. Prepare the **student.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <!-- The mapping between classes and tables is made on the class element. -->
  <!-- name: fully-qualified class name -->
  <!-- table: table name -->
  <class name="com.howtodoinjava.hibernate.test.dto.Student" table="student">
    <!-- The mapping of the primary key is made on the id element. -->
    <!-- name: name of the attribute used as the primary key in the object -->
    <!-- column: name of the primary key field in the table -->
    <!-- If the value of name is the same as that of column, column can be omitted. -->
    <id name="id" column="id">
      <!-- Set the class attribute of the generator element to "assigned". The ID must be provided. -->
    >
    <generator class="assigned" />

    <!--** Pay attention to the Hibernate primary key generation policy. **-->
```

```
</id>
<!-- The mapping between attributes and fields is made on the property element. -->
<!-- name: attribute name in the class -->
<!-- column: field name in the table -->
<!-- If the value of name is the same as that of column, column can be omitted. -->
<property name="name" />
<!-- Same as the entity class. -->
<!--<property name="age"/>-->
</class>
</hibernate-mapping>
```

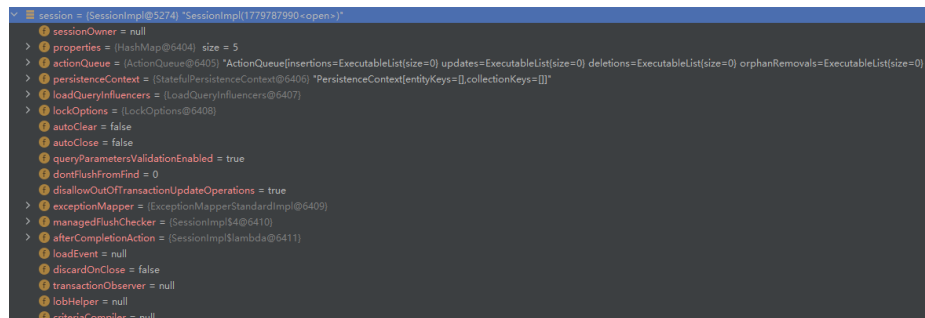
Function Test Example

1. Test the connection function.

a. Test method:

```
@Test
public void testConnection() {
    // Load the configuration information.
    Configuration conf = new Configuration().configure();
    // Create the SessionFactory object based on the configuration information.
    SessionFactory sessionFactory = conf.buildSessionFactory();
    // Open a session object related to the database.
    Session session = sessionFactory.openSession();
    System.out.println(session);
}
```

b. The breakpoint shows that the connection is successfully established.



2. Enable automatic table creation.

a. Comment out the following line in the configuration file:

```
<property name="hbm2ddl.auto">create</property>
```

b. After deleting the **student** table from the database, perform the following test method:

```
@Test
public void testCreateTableAndInsertData() {
    // Create the object to be tested.
    Student student = new Student();
    student.setId(16);
    student.setName("xiaoming");
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // Use session to save data.
    session.save(student);
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

c. View the executed SQL statements printed on the console and the result in the GaussDB database.

```

[10.58.238.107:57638/100.45.152.58:20000] connection is established. ID: b42d6e8-1cab-4d0b-half-b20d8e53a80
21, 2023 11:01:08 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: Connect complete. ID: 942d6e8-1cab-4d0b-half-b20d8e53a80
21, 2023 11:01:08 org.hibernate.dialect.Dialect <init>
INFO: HH4000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:01:45 org.hibernate.type.BasicTypeRegistry register
INFO: HH4000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
Hibernate: drop table if exists student cascade
21, 2023 11:01:45 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HH410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@6b2d69] for (non-JTA)
Hibernate: create table student (id int4 not null, name varchar(255), primary key (id))
21, 2023 11:01:45 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HH410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@3d272b4d] for (non-JTA)
21, 2023 11:01:45 org.hibernate.engine.jdbc.spl.SqlExceptionHelper$StandardWarningHandler logWarning
WARN: SQL Warning Code: 0, SQLState: 00000
21, 2023 11:01:45 org.hibernate.engine.jdbc.spl.SqlExceptionHelper$StandardWarningHandler logWarning
WARN: CREATE TABLE / PRIMARY KEY will create implicit index "student_pkey" for table "student"
INFO: HH4000370: Executing 'import script' org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@71d8cfe7
Hibernate: insert into student (name, id) values (?, ?)

```

```

test=> select * from test.student;
id | name
---+---
16 | xiaoming
(1 row)

```

The **student** table is successfully created, and the **id = 16, name = "xiaoming"** record is inserted into the table.

3. Modify the table (by adding data records).

- a. Modify the configuration file and configure the path of the schema where the table to be modified is located in the URL.

In this test example, the **student** table is in schema **test** under database **test**. That is, the table path is **test.test.student**.

```

<property name="connection.url">jdbc:opengauss://xxx.xxx.xxx.xxx (Replace xxx.xxx.xxx.xxx with the database IP address.):20000 (Replace 20000 with the database port.)/test?currentSchema=test (Replace test with the target database and schema.)</property>
<!-- Uncomment update. -->
<property name="hbm2ddl.auto">update</property>

```

- b. Uncomment the **age** attribute in the entity class and XML file and execute the following method:

```

@Test
public void testAlterTable() {
    // Create the object to be tested.
    Student student = new Student();
    student.setId(15);
    student.setName("xiaohong");
    student.setAge("20");
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // Use session to save data.
    session.save(student);
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}

```

- c. View the executed SQL statements printed on the console and the result in the GaussDB database.

```

21, 2023 11:06:20 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HH410001001: Connection properties: (user=test, password=****)
21, 2023 11:06:20 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HH410001002: Autocommit mode: false
21, 2023 11:06:20 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HH4000115: Hibernate connection pool size: 20 (min=1)
21, 2023 11:06:20 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [ae4dbdd-045c-4dbb-9ccd-7cc6f60bc316] try to connect. IP: 100.45.152.58:20000
21, 2023 11:06:20 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [10.58.238.107:57770/100.45.152.58:20000] connection is established. ID: ae4dbdd-045c-4dbb-9ccd-7cc6f60bc316
21, 2023 11:06:20 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: Connect complete. ID: ae4dbdd-045c-4dbb-9ccd-7cc6f60bc316
21, 2023 11:06:20 org.hibernate.dialect.Dialect <init>
INFO: HH4000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:07:01 org.hibernate.type.BasicTypeRegistry register
INFO: HH4000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@93cf3f0f
21, 2023 11:07:01 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HH410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@5c68bb48] for (non-JTA)
Hibernate: alter table test.student add column age int4
Hibernate: insert into student (name, age, id) values (?, ?, ?)

```

```
test=> select * from test.student;
id | name | age
---+---+---
16 | xiaoming | 20
15 | xiaohong | 20
(2 rows)
```

The framework automatically adds data records to the table based on the entity class and XML file.

4. Save data into the database.

a. Test method:

```
@Test
public void testInsert() {
    Student s1 = new Student(1,"q");
    Student s2 = new Student(2,"w");
    Student s3 = new Student(3,"e");
    ArrayList<Student> students = new ArrayList<>();
    students.add(s1);
    students.add(s2);
    students.add(s3);
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // Use session to save data.
    for (Student student : students) {
        session.save(student);
    }
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

b. The result is as follows.

```
INFO: H4410001001: Connection properties: [user=test, password=****]
21, 2023 11:10:53 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: H4410001003: Autocommit mode: false
21, 2023 11:10:53 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: H44000115: Hibernate connection pool size: 20 (min=1)
21, 2023 11:10:53 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
[0897050-0a76-4e3e-990b-02f092e04f41] try to connect. IP: 100.105.152.85:20000
21, 2023 11:10:53 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
[10.50.238.107:57975/100.85.152.58:20000] Connection is established. ID: d897050-0a76-4e3e-990b-02f092e04f41
21, 2023 11:10:53 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
Connect complete. ID: d897050-0a76-4e3e-990b-02f092e04f41
21, 2023 11:10:54 org.hibernate.dialect.Dialect <init>
INFO: H44000005: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:11:13 org.hibernate.type.BasicTypeRegistry register
INFO: H44000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@3e1162e7
21, 2023 11:11:34 org.hibernate.resource.transaction.backend.jdbc.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bc0fac1 getIsolatedConnection
INFO: H4410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bc0fac1 for (non-JTA)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
```

```
test=> select * from test.student;
id | name | age
---+---+---
16 | xiaoming | 20
15 | xiaohong | 20
1 | q | 0
2 | w | 0
3 | e | 0
(5 rows)
```

Three rows of data are successfully inserted.

5. Query data in HQL mode.

a. Test method:

```
@Test
public void testHQL() {
```

```
// HQL mode
Configuration conf = new Configuration().configure();
SessionFactory sessionFactory = conf.buildSessionFactory();
// Create a session.
Session session = sessionFactory.openSession();
// Start a transaction.
Transaction tx = session.beginTransaction();

// Create an HQL query.
String hql = "FROM Student S WHERE S.id = 15";
Query query = session.createQuery(hql);

// Execute the query and obtain the result.
List results = query.list();

// Commit the transaction.
tx.commit();

// End the session.
session.close();
}
```

b. The result is as follows.

```
21, 2023 11:15:51 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [79760384-6903-4760-8afd-ba92fb927eed] try to connect. IP: 100.85.152.58:28000
21, 2023 11:15:51 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [10_58_238_107:58070/100.85.152.58:28000] connection is established. ID: 79760384-6903-4760-8afd-ba92fb927eed
21, 2023 11:15:51 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: connect complete. ID: 79760384-6903-4760-8afd-ba92fb927eed
21, 2023 11:15:52 org.hibernate.dialect.Dialect cinit()
INFO: ##### Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:16:29 org.hibernate.type.BasicTypeRegistry register
INFO: ##### Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@405325cf
21, 2023 11:16:30 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: ##### Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bca6fac] for (non-JTA)
21, 2023 11:16:30 org.hibernate.internal.QueryTranslatorFactoryInitiator initiateService
INFO: ##### Using ASTQueryTranslatorFactory
Hibernate: select student0_id as id1_0, student0_name as name2_0, student0_age as age3_0 from student student0 where student0_id=15
```

6. Query data in SQL mode.

a. Test method:

```
@Test
public void testQuery() {
// Start a transaction based on session.
Configuration conf = new Configuration().configure();
SessionFactory sessionFactory = conf.buildSessionFactory();
Session session = sessionFactory.openSession();
Transaction transaction = session.beginTransaction();
// SQL mode
List<Student> students = session.createSQLQuery("select * from test.student where id = 1").addEntity(Student.class).list();
for (int i = 0; i < students.size(); i++) {
System.out.println(students.get(i));
}
students.get(0).setAge("20");
// Commit the transaction.
transaction.commit();
// After the operation is complete, close the session connection object.
session.close();
}
```

b. The result is as follows.

```
21, 2023 11:15:13 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [afdd16eb-c28b-487c-8768-1a65ee115399] try to connect. IP: 100.85.152.58:28000
21, 2023 11:15:13 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [10_58_238_107:58070/100.85.152.58:28000] connection is established. ID: afdd16eb-c28b-487c-8768-1a65ee115399
21, 2023 11:15:13 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: connect complete. ID: afdd16eb-c28b-487c-8768-1a65ee115399
21, 2023 11:15:13 org.hibernate.dialect.Dialect cinit()
21, 2023 11:15:52 org.hibernate.type.BasicTypeRegistry register
INFO: ##### Using dialect: org.hibernate.dialect.PostgreSQLDialect
INFO: ##### Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@3e162e7
21, 2023 11:15:53 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: ##### Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@5c60bba0] for (non-JTA)
Hibernate: select * from test.student where id = 1
Student[id=1, name=, age=0]
Hibernate: update student set name=?, age=? where id=?
```

```
test=> select * from test.student;
id | name | age
---+---+---
16 | xiaoming |
15 | xiaohong | 20
1 | q | 20
2 | w | 0
3 | e | 0
(5 rows)
```

The data record of the student whose ID is 1 is found and the value of **age** is changed to 20.

7. Modify data.

- Test method:

```
@Test
public void testUpdate() {
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // SQL mode
    session.createQuery("update test.student set age = 19 where id =
16").executeUpdate();
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

- The result is as follows.

```
21, 2023 11:21:18 org.hibernate.engine.jdbc.connections.internal.UrlManager$ConnectionProviderImpl$PoolConnections$Client
INFO: H8800015: Hibernate connection pool size: 20 (min=1)
21, 2023 11:21:18 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [7b5430f9-cf89-4b06-93de-642abd92aaf] try to connect. IP: 100.85.152.58:20000
21, 2023 11:21:18 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [10_58_238_107:58129/100.85.152.58:20000] connection is established. ID: 7b5430f9-cf89-4b06-93de-642abd92aaf
21, 2023 11:21:18 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: connect complete. ID: 7b5430f9-cf89-4b06-93de-642abd92aaf
21, 2023 11:21:19 org.hibernate.dialect.Dialect$Client
INFO: H8800040: Using dialect: org.hibernate.dialect.PostgresSQLDialect
21, 2023 11:21:50 org.hibernate.type.BasicTypeRegistry register
INFO: H8800020: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e16267
21, 2023 11:21:50 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorImpl getIsolatedConnection
INFO: H8810001501: connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@5c60bbad] for (non-JTA)
Hibernate: update test.student set age = 19 where id = 16
```

```
test=> select * from test.student;
id | name | age
---+---+---
15 | xiaohong | 20
1 | q | 20
2 | w | 0
3 | e | 0
16 | xiaoming | 19
(5 rows)
```

The **age** field of the student whose ID is 16 is successfully changed to 19.

8. Delete data.

- a. Test method:

```
@Test
public void testDelete() {
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // SQL mode
    List<Student> students = session.createQuery("select * from
```

```
test.student").addEntity(Student.class).list();
    System.out.println(students);
    session.createSQLQuery("delete from test.student where id = " +
students.get(0).getId()).executeUpdate();
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

b. The result is as follows.

```
21, 2023-11-23:15 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [a525c88e-c87c-4cfd-8884-99ba1a0635ab] try to connect. IP: 100.85.152.58:28800
21, 2023-11-23:15 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [10c58-238-107:58168/100.85.152.58:28800] connection is established. ID: a525c88e-c87c-4cfd-8884-99ba1a0635ab
21, 2023-11-23:15 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: Connect complete. ID: a525c88e-c87c-4cfd-8884-99ba1a0635ab
21, 2023-11-23:15 org.hibernate.dialect.Dialect cinit>
INFO: HHH000000: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023-11-23:15 org.hibernate.type.BasicTypeRegistry register
INFO: HHH000020: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
21, 2023-11-23:15 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH10001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@9c66bbab] for (non-JTA)
Hibernate: select * from test.student
[Student(id=15, name=xiaohong, age=20), Student(id=1, name=q, age=20), Student(id=2, name=w, age=0), Student(id=3, name=e, age=0), Student(id=16, name=xiaoming, age=19)]
Hibernate: delete from test.student where id = 15
```

```
test=> select * from test.student;
id | name | age
---+---+---
 1 | q    | 20
 2 | w    |  0
 3 | e    |  0
16 | xiaoming | 19
(4 rows)
```

The record whose ID is 15 has been deleted from the **student** table.

4.1.7 Using MyBatis to Connect to a Database

MyBatis is a first class persistence framework with support for custom SQL, stored procedures, and advanced mappings. MyBatis eliminates almost all of the JDBC code and manual setting of parameters and retrieval of results. MyBatis can use simple XML or annotations for configuration and map primitive. It can map interfaces and Java Plain Old Java Objects (POJOs) to database records.

This section describes how to use MyBatis to connect to a GaussDB database.

Configuring the POM Dependency

```
<dependency>
  <groupId>com.huaweicloud.gaussdb</groupId>
  <artifactId>opengaussjdbc</artifactId>
  <version>503.2.T35</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
  <version>3.5.6</version>
</dependency>
```

CAUTION

The Maven environment must have been configured before you configure the POM dependency.

Configuring the Required File

Configuring the **mybatis-config.xml** file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<!-- Root element of the configuration file -->
<configuration>
  <!--Configure the global attributes.-->
  <settings>
    <!--Use getGeneratedKeys of JDBC to obtain the auto-increment primary key value of the database.-->
    <setting name="useGeneratedKeys" value="true"/>
    <!--Replace the column alias with the column label. The default value is true.-->
    <setting name="useColumnLabel" value="true" />
    <!--Enable camel-case naming conversion: Table{create_time} -> Entity{createTime}-->
    <setting name="mapUnderscoreToCamelCase" value="true" />
    <setting name="logImpl" value="STDOUT_LOGGING" />
  </settings>

  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="com.huawei.opengauss.jdbc.Driver"/>
        <property name="url" value="jdbc:opengauss://***.***.***.*** (Replace ***.***.***.*** with the
database IP address):20000 (Replace 20000 with the database port)/test? (Replace test with the
corresponding database name.)connectionTimeout=10"/>
        <property name="username" value="*** (Replace *** with the correct username.)"/>
        <property name="password" value="***** (Replace ***** with the correct password.)"/>
      </dataSource>
    </environment>
  </environments>
  <!--Register mapper (address of mapper.xml).-->
  <mapper>
    <mapper resource="mapper/StudentDaoMapper.xml"/></mapper>
  </mapper>
</configuration>
```

Examples

1. Test the entity class **StudentEntity.java** (in **com.huawei.entity**).

```
public class StudentEntity {
    Integer id;
    String name;
}
```

2. Configure the **StudentDaoMapper.xml** file corresponding to the entity class (in **resources.mapper**).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-
mapper.dtd" >
<mapper namespace="StudentMapper">
  <!-- Query by primary key -->
  <select id="getList" resultType="com.huawei.entity.StudentEntity" >
    select * from student;
  </select>
</mapper>
```

3. Test table query.

```
@Test
public void mainTest() throws IOException {
    // 1. Read the core configuration file of MyBatis (mybatis-config.xml).
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
    // 2. Obtain a SqlSessionFactory factory object based on the configuration information.
    SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
    // 3. Obtain a SqlSession object through the factory.
    SqlSession session = fac.openSession();
}
```



```
// 4. Find the SQL statement to be executed based on the namespace and ID and execute the SQL
statement.
List<StudentEntity> list = session.selectList("StudentMapper.getList");
// 5. Output the result.
list.forEach(i -> {
    System.out.println(i.toString());
});
}
```

4. Query result logs.

```
Setting autocommit to false on JDBC Connection [com.huawei.opengauss.jdbc.jdbc.PgConnection@47caedad]
==> Preparing: select * from student;
==> Parameters:
<==      Columns: id, name
<==      Row: 1, test
<==      Total: 1
StudentEntity(id=1, name=test)
```

CAUTION

Currently, **PaginationInnerInterceptor** in the MybatisPlus plug-in does not adapt to the GaussDB driver. To resolve this problem, set **DbType** to **POSTGRE_SQL** when creating the **PaginationInnerInterceptor** object. Example:
**PaginationInnerInterceptor innerInterceptor = new
PaginationInnerInterceptor(DbType.POSTGRE_SQL)**

4.1.8 Using JayDeBeApi to Connect to a Database

JayDeBeApi is a Python module that provides a convenient, efficient way for Python developers to use the Java JDBC driver to connect to and perform operations on databases.

This section describes how to use JayDeBeApi to connect to the GaussDB database.

Environment Configuration

1. Configure the GaussDB development environment.

Prepare the basic development environment of GaussDB and obtain the database connection parameters. For example:

```
gsqll -h ***.***.***.*** -p 20000 -U *** -W ***** -d test
```

Parameter description:

-h: IP address of the server hosting the GaussDB instance

-p: connection port of the GaussDB instance

-U: username for the connection

-W: user password

-d: name of the database you want to connect.

2. Install the JayDeBeApi driver.

a. Install Java JDK 8 and Python 3 on the local PC. To check the software versions, run the following commands:

```
java -version
python --version
pip --version
```

- b. If the server can connect to the Python Package Index (PyPI), run the pip command to install JayDeBeApi:


```
pip install jaydebeapi
```

 If the server cannot connect to PyPI, download an offline installation package of JayDeBeApi and install it on the local PC.
3. Obtain the GaussDB driver package.
Download particular packages listed in [Table 4-11](#) based on the version of your instance.

Table 4-11 Driver package download list

Version	Download Address
3.x	Driver package Verification package for the driver package
2.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- a. Upload the software package and verification package to the same directory on a Linux VM.
- b. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```

Example

1. Create a script file.
 - Create a test_jaydebeapi.py file and write the following code into the file:

```
#!/usr/bin/env python3.x
# -*- coding: UTF-8 -*-
encoding = "utf8"
import jaydebeapi

def test_jaydebeapi():
    #Set required parameters.
    url = 'jdbc:opengauss://***.***.***.***.20000/test'
    user = '****'
    password = '*****'
    driver = 'com.huawei.opengauss.jdbc.Driver'
    jarFile = './opengaussjdbc.jar'

    conn = jaydebeapi.connect(driver, url, [user, password], jarFile)
    cur = conn.cursor()

    #Create a table named students.
    sql = 'create table students (id int, name varchar(20))'
    cur.execute(sql)
```

```
#Insert three groups of data into the students table.
sql = "insert into students values(1,'xiaoming'),(2,'xiaohong'),(3,'xiaolan')"
cur.execute(sql)

#Query all data in the students table.
sql = 'select * from students'
cur.execute(sql)
ans = cur.fetchall()
print(ans)

#Update data in the students table.
sql = 'update students set name = \'xiaolv\' where id = 1'
cur.execute(sql)

#Query all data in the students table again.
sql = 'select * from students'
cur.execute(sql)
ans = cur.fetchall()
print(ans)

#Delete the students table.
sql = 'drop table students'
cur.execute(sql)

cur.close()
conn.close()

test_jaydebeapi()
```

- Configure required parameters in the code.

```
#Set the connection URL, which requires the IP address, port number, and database name of the
database server you want to connect.
url = 'jdbc:opengauss://***.***.***.***:20000/test'
#Enter the username.
user = '***'
#Enter the password.
password = '*****'
#Specify the path to the JDBC driver class.
driver = 'com.huawei.opengauss.jdbc.Driver'
#Specify the path to the JAR package of the JDBC driver. By default, it is stored in the same
directory as the test_jaydebeapi.py file.
jarFile = './opengaussjdbc.jar'
```

2. Execute the program.

Run the following command to execute the **test_jaydebeapi.py** file:

```
python ./test_jaydebeapi.py
```

3. Check the result.

The GaussDB database is successfully connected, and two query results are returned, as shown in the following figure.

```
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: [f42f7374-3bfd-4f91-b370-4bbd007aea16] Try to connect. IP: 127.0.0.1:20000
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: [127.0.0.1:48426/127.0.0.1:20000] Connection is established. ID: f42f7374-3bfd-4f91-b370-4bbd007aea16
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: Connect complete. ID: f42f7374-3bfd-4f91-b370-4bbd007aea16
[[('1', 'xiaoming'), (2, 'xiaohong'), (3, 'xiaolan')]]
[[('1', 'xiaolv'), (2, 'xiaohong'), (3, 'xiaolan')]]
```

4.2 Primary/Standby Instances

4.2.1 Development Specifications

If the connection pool mechanism is used during application development, comply with the following specifications:

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

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

Table 4-12 describes the compatibility of application development drivers.

Table 4-12 Compatibility Description

Driver	Compatibility Description
JDBC, Go, ODBC, libpq, Psycopg, and ecpg	The new drivers are forward compatible with the database. To use the new features added to the driver and database, you must upgrade the database.

NOTICE

- Setting **behavior_compat_options** to '**proc_outparam_override**' is applicable only in A-compatible mode.
- In principle, you need to set the compatibility parameter after the database creation, instead of switching the parameters when using the database.
- The JDBC driver must be upgraded to that maps to GaussDB Kernel 503.1.0 or later if the following features are used:
 1. The fully-encrypted memory decryption emergency channel is required.
 2. JDBC is required to use user-defined nested types such as record, array, and tableof.
 3. The **s2** compatibility parameter is enabled and the validity check of **sessiontimezone** is set.

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

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

4.2.2 Using JDBC to Connect to a Database

Java Database Connectivity (JDBC) is a Java API for running SQL statements. It provides unified access APIs for different relational databases, based on which applications process data. The GaussDB library supports JDBC 4.2 and requires JDK 1.8 for code compiling. It does not support JDBC-ODBC bridge.

Prerequisites

Java JDK 8 has been installed on the local PC.

JDBC Package

The package name is **GaussDB-Kernel_Database version number_OS version number_64bit_Jdbc.tar.gz**. After the decompression, you will obtain the following JDBC packages in JAR format:

- **gaussdbjdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package is recommended. The Java code examples in this section use the **gaussdbjdbc.jar** package by default.
- **gscejdbc.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. This driver package contains the dependent libraries related to encryption and decryption that need to be loaded to the encrypted database. You are advised to use this driver package in encrypted scenarios. Currently, only EulerOS is supported.
- **gaussdbjdbc-JRE7.jar**: The main class name is **com.huawei.gaussdb.jdbc.Driver**. The URL prefix of the database connection is **jdbc:gaussdb**. The **gaussdbjdbc-JRE7.jar** package is used in the JDK 1.7 environment.

CAUTION

- Before using the **gscejdbc.jar** driver package, you need to set the environment variable **LD_LIBRARY_PATH**. For details, see section "Setting Encrypted Equality Queries > Using JDBC to Operate an Encrypted Database" in *Feature Guide*.
 - In JDK 1.8, you are advised to use **gaussdbjdbc.jar** instead of **gaussdbjdbc-JRE7.jar**.
 - For details about other JDBC JAR packages, see "Application Development Guide" > "JDBC Compatibility Package" in *Developer Guide*.
-

Driver Class

Before establishing a database connection, load the **com.huawei.gaussdb.jdbc.Driver** database driver class.

NOTE

1. GaussDB is compatible with PostgreSQL in the use of JDBC. Therefore, when two JDBC drivers are used in the same process, class names may conflict.
2. GaussDB JDBC driver has the following enhanced features:
 1. The SHA256 encryption mode is supported for login.
 2. The third-party log framework that implements the sf4j API can be connected.
 3. DR failover is supported.

Environment Class

JDK1.8 must be configured on the client. JDK supports multiple platforms such as Windows and Linux. The following uses Windows as an example to describe how to configure JDK 1.8:

Step 1 Enter **java -version** in the MS-DOS window (command prompt in Windows) to check the JDK version. Ensure that the JDK version is JDK1.8. If JDK is not installed, download the installation package from the official website and install it.

Step 2 Configure system environment variables.

1. Right-click **My computer** and choose **Properties**.
2. In the navigation pane, choose **Advanced system settings**.
3. In the **System Properties** dialog box, click **Environment Variables** on the **Advanced** tab page.
4. In the **System variables** area of the **Environment Variables** dialog box, click **New** or **Edit** to configure system variables. For details about the variables, see [Table 4-13](#).

Table 4-13 Variables

Variable	Operation	Variable Value
JAVA_HOME	<ul style="list-style-type: none">- If the variable exists, click Edit.- If the variable does not exist, click New.	Specifies the Java installation directory. Example: C:\Program Files\Java\jdk1.8.0_131 .
Path	Click Edit .	<ul style="list-style-type: none">- If <i>JAVA_HOME</i> is configured, add <i>%JAVA_HOME%\bin</i> before the variable value.- If <i>JAVA_HOME</i> is not configured, add the following full Java installation path before the variable value: C:\Program Files\Java\jdk1.8.0_131\bin
CLASSPATH	Click New .	%JAVA_HOME%\lib;%JAVA_HOME%\lib\tools.jar

----End

Loading the Driver

Load the database driver before creating a database connection.

You can load the driver in the following ways:

- Implicit loading at any position before a connection is created in the code:
Class.forName("com.huawei.gaussdb.jdbc.Driver");
- Parameter transfer during JVM startup: **java -Djdbc.drivers=com.huawei.gaussdb.jdbc.Driver jdbctest**

NOTE

jdbctest is the name of a test application.

Function Prototype

JDBC provides the following three database connection methods:

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

Parameters

Table 4-14 Database connection parameters

Parameter	Description
url	<p>gaussdbjdbc.jar database connection descriptor.</p> <p>If host is set to a server name or an IPv4 address, formats are as follows:</p> <ul style="list-style-type: none">• jdbc:gaussdb: (If the database name is left empty, the username is used.)• jdbc:gaussdb:database• jdbc:gaussdb://host/database• jdbc:gaussdb://host:port/database• jdbc:gaussdb://host:port/database?param1=value1&param2=value2• jdbc:gaussdb://host1:port1,host2:port2/database?param1=value1&param2=value2 <p>If host is set to an IPv6 address, formats are as follows:</p> <ul style="list-style-type: none">• jdbc:gaussdb: (If the database name is left empty, the username is used.)• jdbc:gaussdb:database• jdbc:gaussdb://host/database or jdbc:gaussdb://[host]/database• jdbc:gaussdb://[host]:port/database• jdbc:gaussdb://[host]:port/database?param1=value1&param2=value2• jdbc:gaussdb://[host1]:port1,[host2]:port2/database?param1=value1&param2=value2

Parameter	Description
	<p>NOTE</p> <ul style="list-style-type: none"> • database indicates the name of the database to connect. • host indicates the name or IP address of the database server. Both IPv4 and IPv6 addresses are supported. For security purposes, the primary database node forbids access from other nodes in the database without authentication. To access the primary database node from inside the database, deploy the JDBC program on the host where the primary database node is located and set host to 127.0.0.1. Otherwise, the error message "FATAL: Forbid remote connection with trust method!" may be displayed. It is recommended that the service system be deployed outside the database. Otherwise, the database performance may be affected. By default, the local host is used to connect to the server. • port indicates the port number of the database server. By default, the database on port 5432 of the local host is connected. • If host is set to an IPv6 address and the port number is specified in the URL, use square brackets ([]) to enclose the IP address. The format is [IP address]:Port number. • param indicates a database connection attribute. The parameter can be configured in the URL. The URL starts with a question mark (?), uses an equal sign (=) to assign a value to the parameter, and uses an ampersand (&) to separate parameters. You can also use the attributes of the info object for configuration. For details, see Examples. • value indicates the database connection attribute values. • The connectTimeout and socketTimeout parameters must be set for connection. If they are not set, the default value 0 is used, indicating that the connection will not time out. When the network between the DN and client is faulty, the client does not receive the ACK packet from the DN. In this case, the client starts the timeout retransmission mechanism to continuously retransmit packets. A timeout error is reported only when the timeout interval reaches the default value 600s. As a result, the RTO is high. • You are advised to ensure the validity of the URL when using the standard JDBC interface to establish a connection. An invalid URL may cause an exception, and the exception contains the original URL character string, which may cause sensitive information leakage.
info	For details about common attributes of info , see "Application Development Guide" > "Development Based on JDBC" > "Connecting to a Database" in <i>Developer Guide</i> .
user	Database user.
password	Password of the database user.

 **NOTE**

After the **uppercaseAttributeName** parameter is enabled, if the database contains metadata with a mixture of uppercase and lowercase letters, only the metadata in lowercase letters can be queried and output in uppercase letters. Before using the metadata, ensure that the metadata is stored in lowercase letters to prevent data errors.

Examples

Example 1: Connect to a database.

```
// The following code encapsulates database connection operations into an interface. The database can
// then be connected using an authorized username and a password.
public static Connection getConnect(String username, String passwd)
{
    // Driver class.
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    // Database connection descriptor.
    String sourceURL = "jdbc:gaussdb://$ip:$port/database";
    Connection conn = null;

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

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

    return conn;
}
```

Example 2: Use the Properties object as a parameter to create a connection.

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

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

    try
    {
        info.setProperty("user", username);
        info.setProperty("password", passwd);
        // Create a connection.
        conn = DriverManager.getConnection(sourceURL, info);
    }
}
```

```
        System.out.println("Connection succeed!");
    }
    catch(Exception e)
    {
        e.printStackTrace();
        return null;
    }

    return conn;
}
```

For details about common parameters, see "Application Development Guide" > "Development Based on JDBC" > "Common JDBC Parameters" in *Developer Guide*.

Example 3: Use the streaming read function.

```
// There will be security risks if the username and password used for authentication are directly written into
// code. It is recommended that the username and password be stored in the configuration file or
// environment variables (the password must be stored in ciphertext and decrypted when being used) to
// ensure security.
// In this example, the username and password are stored in environment variables. Before running this
// example, set environment variables EXAMPLE_USERNAME_ENV and EXAMPLE_PASSWORD_ENV in the local
// environment (set the environment variable names based on the actual situation).
// Establish a connection.
public static Connection getConnection(String username, String passwd) {
    String driver = "com.huawei.gaussdb.jdbc.Driver";
    String sourceURL = "jdbc:gaussdb://$ip:$port/database?enableStreamingQuery=true";
    Connection conn = null;
    try {
        // Load the driver.
        Class.forName(driver);
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    try {
        // Create a connection.
        conn = DriverManager.getConnection(sourceURL, username, passwd);
        System.out.println("Connection succeed!");
    } catch (Exception e) {
        e.printStackTrace();
        return null;
    }
    return conn;
}

// Execute common SQL statements to create table t_user.
public static void CreateTable(Connection conn) {
    Statement stmt = null;
    try {
        stmt = conn.createStatement();

        // Execute common SQL statements.
        stmt.executeUpdate("DROP TABLE IF EXISTS t_user");
        stmt.executeUpdate("CREATE TABLE t_user(id int, name VARCHAR(20));");
        stmt.close();
    } catch (SQLException e) {
        if (stmt != null) {
            try {
                stmt.close();
            } catch (SQLException e1) {
                e1.printStackTrace();
            }
        }
        e.printStackTrace();
    }
}

// Execute a prepared statement to insert data in batches.
public static void BatchInsertData(Connection conn) {
```

```
PreparedStatement pst = null;

try {
    // Generate a prepared statement.
    pst = conn.prepareStatement("INSERT INTO t_user VALUES (?,?)");
    for (int i = 0; i < 20; i++) {
        // Add parameters.
        pst.setInt(1, i + 1);
        pst.setString(2, "name " + (i + 1));
        pst.addBatch();
    }
    // Perform batch processing.
    pst.executeBatch();
    pst.close();
} catch (SQLException e) {
    if (pst != null) {
        try {
            pst.close();
        } catch (SQLException e1) {
            e1.printStackTrace();
        }
    }
    e.printStackTrace();
}

// Enable streaming read and query the content in the t_user table.
public static void StreamingQuery(Connection conn) {
    PreparedStatement pst = null;
    ResultSet resultSet = null;

    try {
        // Query all values in the t_user table.
        pst = conn.prepareStatement("SELECT * FROM t_user");
        pst.setFetchSize(Integer.MIN_VALUE); // Functions the same as
        ((PgStatement)statement).enableStreamingResults();
        resultSet = pst.executeQuery();
        while (resultSet.next()) {
            System.out.println(resultSet.getInt(1));
        }
    } catch (SQLException e) {
        throw new RuntimeException(e);
    } finally {
        if (resultSet != null) {
            try {
                resultSet.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }

        if (pst != null) {
            try {
                pst.close();
            } catch (SQLException e) {
                throw new RuntimeException(e);
            }
        }
    }
}

public static void main(String[] args) throws Exception {
    String userName = System.getenv("EXAMPLE_USERNAME_ENV");
    String password = System.getenv("EXAMPLE_PASSWORD_ENV");

    Connection conn = getConnection(userName, password);

    CreateTable(conn);
}
```

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

NOTICE

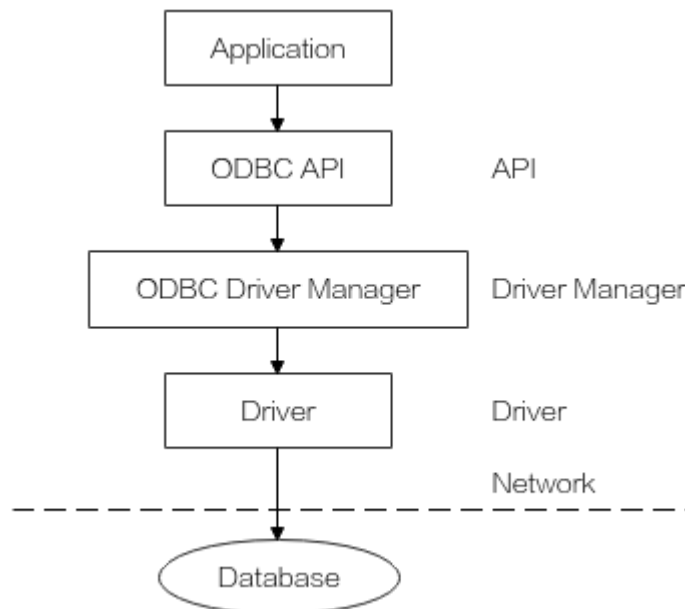
When the streaming read function is used, you need to perform the `resultSet.close()` or `statement.close()` operation after the result set is used. Otherwise, the current connection is unavailable.

4.2.3 Using ODBC to Connect to a Database

Open Database Connectivity (ODBC) is a Microsoft API for accessing databases based on the X/OPEN CLI. Applications interact with the database through the APIs provided by ODBC, which enhances their portability, scalability, and maintainability.

Figure 4-2 shows the system structure of ODBC.

Figure 4-2 ODBC system structure



GaussDB supports ODBC in the following environments.

Table 4-15 OSs Supported by ODBC

OS	Platform
EulerOS V2.0SP5	x86_64
EulerOS V2.0SP9	Arm64
EulerOS V2.0SP10	x86_64
EulerOS V2.0SP10	Arm64
Windows 7	x86_32
Windows 7	x86_64
Windows Server 2008	x86_32
Windows Server 2008	x86_64
Kylin V10	x86_64
Kylin V10	Arm64
UnionTech V20	x86_64
UnionTech V20	Arm64
Huawei Cloud EulerOS 2.0	x86_64
Huawei Cloud EulerOS 2.0	Arm64

The ODBC Driver Manager running on Unix or Linux can be unixODBC or iODBC. unixODBC-2.3.7 is used as the component for connecting to the database.

Windows has a native ODBC Driver Manager. You can locate **Data Sources (ODBC)** by choosing **Control Panel > Administrative Tools**.

NOTE

The current database ODBC driver is based on an open-source version and may be incompatible with data types tinyint, smalldatetime, nvarchar, and nvarchar2.

ODBC Constraints

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

Prerequisites

You have downloaded the ODBC driver packages for Linux and Windows.

- The package name for Linux is **GaussDB-Kernel_Database version number_OS version number_64bit_Odbc.tar.gz**. In the Linux OS, header files (including **sql.h** and **sqlext.h**) and the library (**libodbc.so**) are required in application development. These header files and library can be obtained from the unixODBC-2.3.7 installation package.
- The package name for Windows is **GaussDB-Kernel_Database version number_Windows_Odbc_X64.tar.gz** (64-bit). In the Windows OS, the required header files and library files are system-resident.

Procedure in a Linux Server

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

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

After the download, verify the integrity based on the integrity verification algorithm provided by the community. Download <https://www.unixodbc.org/unixODBC-2.3.7.tar.gz.md5>, view the MD5 value, and check whether the MD5 value is the same as that in the source code package.

Step 2 Install unixODBC. It does not matter if unixODBC of another version has been installed.

For example, to install unixODBC-2.3.7, run the commands below.

```
tar zxvf unixODBC-2.3.7.tar.gz
cd unixODBC-2.3.7

./configure --enable-gui=no # To perform compilation on an Arm server, add the configure parameter --build=aarch64-unknown-linux-gnu.
make
# The installation may require root permissions.
make install
```

NOTE

- Currently, unixODBC-2.2.1 is not supported.
- By default, it is installed in the **/usr/local** directory. The data source file is generated in the **/usr/local/etc** directory, and the library file is generated in the **/usr/local/lib** directory.
- You can compile unixODBC with the **--enable-fastvalidate=yes** option to achieve higher performance. However, this option may cause an application that passes an invalid handle to the ODBC API to fail instead of returning an SQL_INVALID_HANDLE error.

Step 3 Replace the GaussDB client driver.

Decompress **GaussDB-Kernel_Database version number_OS version number_64bit_Odbc.tar.gz**. After the decompression, the **lib** and **odbc** folders are generated. The **odbc** folder contains another **lib** folder. Copy all dynamic libraries in the **/lib** and **/odbc/lib** folders to the **/usr/local/lib** directory.

Step 4 Configure the data source.

1. Configure the ODBC driver file.

Add the following content to the **/usr/local/etc/odbcinst.ini** file:

```
[GaussMPP]
Driver64=/usr/local/lib/gsqlodbcw.so
setup=/usr/local/lib/gsqlodbcw.so
```

For descriptions of the parameters in the **odbcinst.ini** file, see [Table 4-16](#).

Table 4-16 odbcinst.ini configuration parameters

Parameter	Description	Example
[DriverName]	Driver name, corresponding to Driver in DSN.	[DRIVER_N]
Driver64	Path of the dynamic driver library.	Driver64=/usr/local/lib/gsqlodbcw.so
setup	Driver installation path, which is the same as the dynamic library path in Driver64.	setup=/usr/local/lib/gsqlodbcw.so

2. Configure the data source file.

Add the following content to the `/usr/local/etc/odbc.ini` file:

```
[MPPODBC]
Driver=GaussMPP
Servername=127.0.0.1 # Database server IP address
Database=db1 # Database name
Username=omm # Database username
Password= # Database user password
Port=8000 # Database listening port
Sslmode = allow
```

Table 4-17 describes the parameters in the `odbc.ini` file.

Table 4-17 odbc.ini configuration parameters

Parameter	Description	Example Value
[DSN]	Data source name.	[MPPODBC]
Driver	Driver name, corresponding to DriverName in <code>odbcinst.ini</code> .	Driver = DRIVER_N
Servername	Server IP address. Multiple IP addresses can be configured. Both IPv4 and IPv6 are supported.	Servername=127.0.0.1
Database	Name of the database to connect.	Database=db1
Username	Database username.	Username = omm

Parameter	Description	Example Value
Password	<p>Database user password.</p> <p>NOTE After a user establishes a connection, the ODBC driver automatically clears their password stored in memory.</p> <p>However, if this parameter is configured, unixODBC will cache data source files, which may cause the password to be stored in the memory for a long time.</p> <p>When you connect to an application, you are advised to send your password through an API instead of writing it in a data source configuration file. After the connection has been established, immediately clear the memory segment where your password is stored.</p> <p>CAUTION The password in the configuration file must comply with the following HTTP rules:</p> <ol style="list-style-type: none"> 1. Characters must comply with the URL encoding specifications. For example, the exclamation mark (!) must be written as %21, and the percent sign (%) must be written as %25. Therefore, pay attention to the characters. 2. A plus sign (+) will be replaced by a space. 	Password=*****
Port	Port number of the server.	Port = 8000
Sslmode	Specifies whether to enable SSL.	Sslmode = allow

For values of the **Sslmode** parameter, see [Table 4-18](#).

Table 4-18 Sslmode options

Sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by verify-ca , the system checks whether the name of the host where the database resides is the same as that in the certificate. GaussDB does not support this mode.

Step 5 Enable the SSL mode. For details, contact the database administrator.

Step 6 Configure the database server. For details, contact the database administrator.

Step 7 Configure the environment variables on the client.

```
vim ~/.bashrc
```

Add the following information to the configuration file:

```
export LD_LIBRARY_PATH=/usr/local/lib/:$LD_LIBRARY_PATH
export ODBCYSINI=/usr/local/etc
export ODBCINI=/usr/local/etc/odbc.ini
```

Step 8 Run the following command to validate the addition:

```
source ~/.bashrc
```

Step 9 Connect to the database.

```
isql -v GaussODBC
```

GaussODBC: data source name.

- If the following information is displayed, the configuration is correct and the connection succeeds:

```
+-----+
| Connected!          |
|                    |
| sql-statement      |
| help [tablename]   |
```

```
| quit |
|-----|
```

- If error information is displayed, the configuration is incorrect. Check the configuration.

----End

Procedure in a Windows Server

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

Step 1 Replace the GaussDB client driver.

Decompress the **GaussDB-Kernel_Database version number_Windows_X64_Odbc.tar.gz** (64-bit) driver package or **GaussDB-Kernel_Database version number_Windows_X86_Odbc.tar.gz** (32-bit) driver package, and click **gsqlodbc.exe** to install the driver.

Step 2 Open the driver manager.

When configuring the data source, use the ODBC driver manager corresponding to the ODBC version. If the 64-bit ODBC driver is used, the 64-bit ODBC driver manager must be used. Assume that the OS is installed on drive C (if the OS is installed on another drive, change the path accordingly):

- If you want to use 32-bit ODBC driver manager in a 64-bit OS, open **C:\Windows\SysWOW64\odbcad32.exe**. Do not choose **Control Panel > Administrative Tools > Data Sources (ODBC)**.

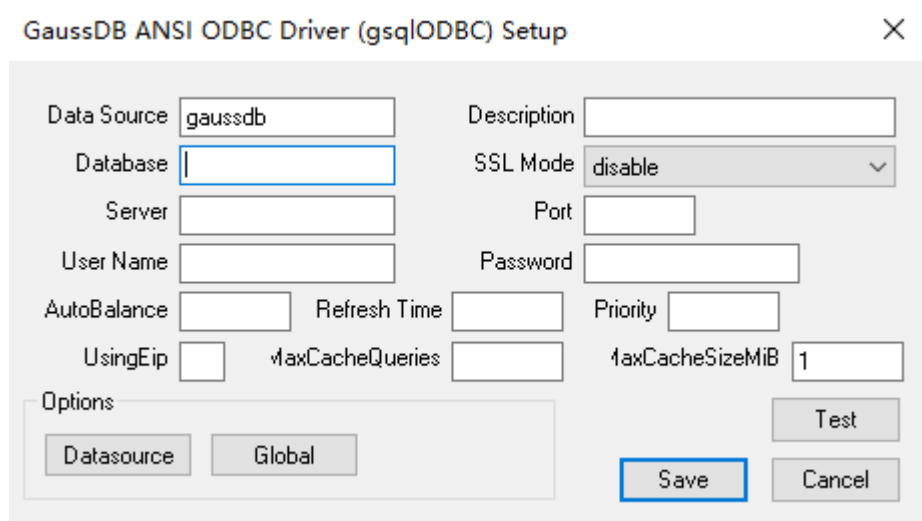
NOTE

WoW64 is short for Windows 32-bit on Windows 64-bit. **C:\Windows\SysWOW64** stores the 32-bit environment on a 64-bit system. **C:\Windows\System32** stores the environment consistent with the current OS. For technical details, see Windows technical documents.

- For a 32-bit OS, open **C:\Windows\System32\odbcad32.exe** or choose **Computer > Control Panel > Administrative Tools > Data Sources (ODBC)** to open Driver Manager.
- For a 64-bit OS, choose **Control Panel > Administrative Tools > Data Sources (ODBC)** to enable driver management.

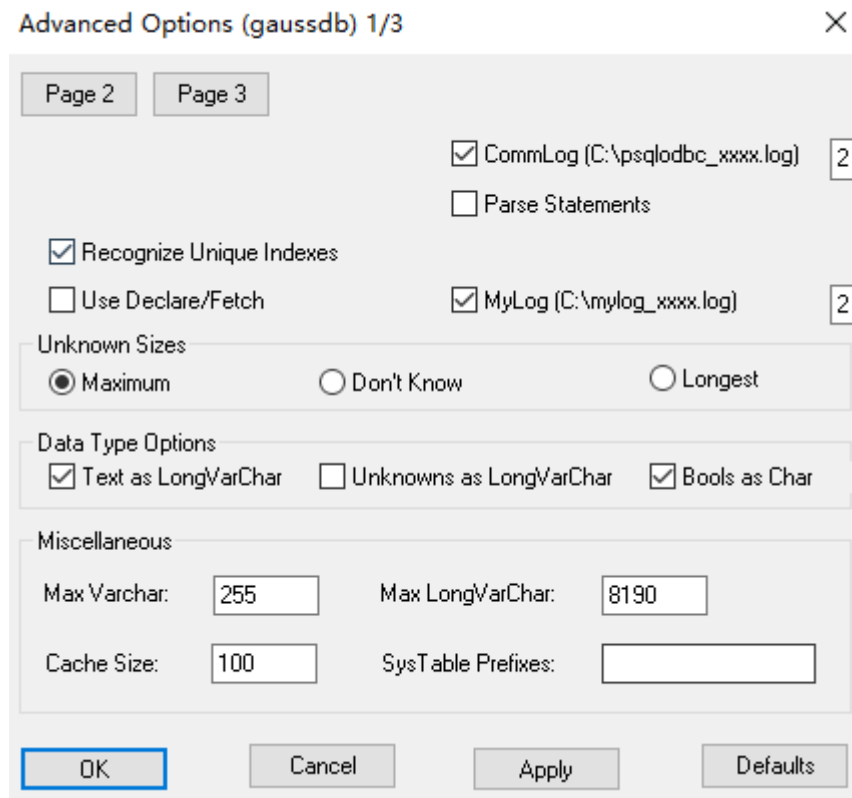
Step 3 Configure the data source.

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



For details about the parameters, see [Procedure in a Linux Server](#).

You can click **Datasource** to configure whether to print logs.



NOTICE

The entered username and password will be recorded in the Windows registry and you do not need to enter them again when connecting to the database next time. For security purposes, you are advised to delete sensitive information before clicking **Save** and enter the required username and password again when using ODBC APIs to connect to the database.

Step 4 Configure SSL mode.

Change the value of **SSL Mode** in **Step 3** to **require**.

Table 4-19 sslmode options

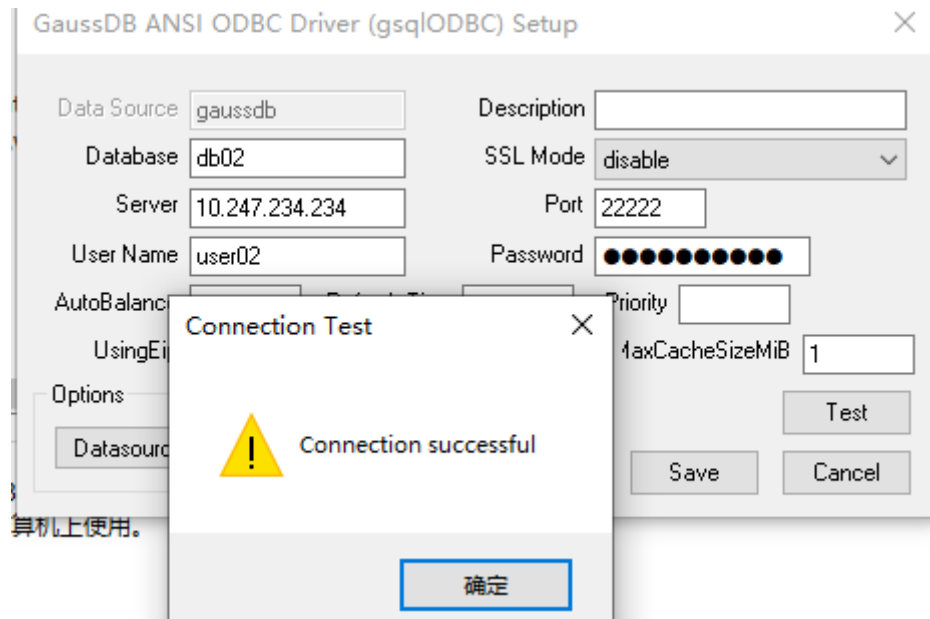
sslmode	Whether SSL Encryption Is Enabled	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required and data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required and whether the database has a trusted certificate will be verified. Currently, Windows ODBC does not support the cert authentication.
verify-full	Yes	SSL connection is required. In addition to the check scope specified by verify-ca , the system checks whether the name of the host where the database resides is the same as that in the certificate. Currently, Windows ODBC does not support the cert authentication.

Step 5 Configure a GaussDB server. For details, contact the administrator.**Step 6** Restart the database instance.

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

Step 7 Click **Test** to test the connection.

- If the following information is displayed, the configuration is correct and the connection succeeds.



- If error information is displayed, the configuration is incorrect. Check the configuration.

----End

4.2.4 Using libpq to Connect to a Database

libpq is a C application programming interface to GaussDB. libpq contains a set of library functions that allow client programs to send query requests to the GaussDB servers and obtain query results. It is also the underlying engine of other GaussDB APIs, such as ODBC. This chapter provides examples to show how to write code using libpq.

Prerequisites

A C development environment has been installed on the local PC.

To compile and develop source programs based on libpq, perform the following steps:

1. Decompress the **GaussDB-Kernel_Database version number_OS version number_64bit_Libpq.tar.gz** file. The required header file is stored in the **include** folder, and the **lib** folder contains the required libpq library file.

NOTE

In addition to **libpq-fe.h**, the **include** folder contains the header files **postgres_ext.h**, **gs_thread.h**, and **gs_threadlocal.h** by default. These three header files are the dependency files of **libpq-fe.h**.

2. Develop the source program **testlibpq.c**. The source code file needs to reference the header file provided by libpq.
Example: `#include <libpq-fe.h>`
3. To compile the libpq source program by running **gcc**, use the **-I directory** option to provide the installation location of the header file. (Sometimes the compiler looks for the default directory, so this option can be ignored.)
Example:

```
gcc -I (Directory where the header file is located) -L (Directory where the libpq library is located)
testlibpq.c -lpq
```

4. If the makefile is used, add the following option to variables *CPPFLAGS*, *LDLFLAGS*, and *LIBS*:

```
CPPFLAGS += -I (Directory of the header file)
LDLFLAGS += -L (Directory of the libpq library)
LIBS += -lpq
For example:
CPPFLAGS += -I$(GAUSSHOME)/include/libpq
LDLFLAGS += -L$(GAUSSHOME)/lib
```

Code for Common Functions

Example 1:

```
/*
 * testlibpq.c
 * Note: testlibpq.c source program provides basic and common application scenarios of libpq.
 * The PQconnectdb, PQexec, PQntuples, and PQfinish interfaces provided by libpq are used to establish
 database connections, execute SQL statements, obtain returned results, and clear resources.
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int
main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or
 configuration files. Environment variables need to be configured as required. If no environment variable is
 used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn *conn;
    PGresult *res;
    int nFields;
    int i,j;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *host = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
     * This value is used when the user provides the value of the conninfo character string in the command
 line.
     * Otherwise, the environment variables or the default values
     * are used for all other connection parameters.
     */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, host, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);
```

```
/* Check whether the backend connection has been successfully established. */
if (PQstatus(conn) != CONNECTION_OK)
{
    fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
    exit_nicely(conn);
}

/*
 * Since a cursor is used in the test case, a transaction block is required.
 * Put all data in one "select * from pg_database"
 * PQexec() is too simple and is not recommended.
 */

/* Start a transaction block. */
res = PQexec(conn, "BEGIN");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "BEGIN command failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/*
 * PQclear PGresult should be executed when it is no longer needed, to avoid memory leakage.
 */
PQclear(res);

/*
 * Fetch data from the pg_database system catalog.
 */
res = PQexec(conn, "DECLARE myportal CURSOR FOR select * from pg_database");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    fprintf(stderr, "DECLARE CURSOR failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
PQclear(res);

res = PQexec(conn, "FETCH ALL in myportal");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "FETCH ALL failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}

/* First, print out the attribute name. */
nFields = PQnfields(res);
for (i = 0; i < nFields; i++)
    printf("%-15s", PQfname(res, i));
printf("\n\n");

/* Print lines. */
for (i = 0; i < PQntuples(res); i++)
{
    for (j = 0; j < nFields; j++)
        printf("%-15s", PQgetvalue(res, i, j));
    printf("\n");
}

PQclear(res);

/* Close the portal. We do not need to check for errors. */
res = PQexec(conn, "CLOSE myportal");
PQclear(res);

/* End the transaction. */
```



```
res = PQexec(conn, "END");
PQclear(res);

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

return 0;
}
```

Example 2:

```
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
/* Test the automatic primary node selection of the PQconnectStart + loop PQconnectStart APIs when
multiple IP addresses are configured. */

static void exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

int main()
{
    char *conninfo = "postgresql://10.442.13.173:53600,10.442.13.177:54600/postgres?
user=bot&password=Gaussdba@Mpp&target_session_attrs=read-write";
    //PGconn *conn = PQconnectdb(conninfo);
    PGconn *conn = PQconnectStart(conninfo);
    if (conn == NULL) {
        fprintf(stderr, "Connection initialization failed\n");
        return 1;
    }

    ConnStatusType status;
    int sock;

    while (1) {
        PQconnectPoll(conn);
        status = PQstatus(conn);
        fprintf(stderr, "\n-----conn->status: %d\n", PQstatus(conn));
        if (status == CONNECTION_BAD) {
            fprintf(stderr, "Connection failed\n");
            PQfinish(conn);
            return 1;
        }

        if ((int)status == 0) {
            fprintf(stderr, "Connection established\n");
            break;
        }

        /* Obtain the bottom-layer socket descriptor to perform non-blocking I/O operations. */
        sock = PQsocket(conn);

        /* Other non-blocking operations can be performed here, for example, waiting for an event. */

        /* Simulate other operations in the non-blocking process by using sleep. */
        sleep(1);
    }
    fprintf(stderr, "\n-----conn->status: %d\n", PQstatus(conn));

    PGresult *res;
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s", PQerrorMessage(conn));
        exit_nicely(conn);
    }
}
```

```
res = PQexec(conn, "show transaction_read_only;");
printf("\n-----transaction_read_only is %-15s\n", PQgetvalue(res, 0, 0));
PQclear(res);

res = PQexec(conn, "select 1;");
if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "select failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
printf("%-15s\n", PQgetvalue(res, 0, 0));
PQclear(res);

PQfinish(conn);
return 0;
}
```

Example 3:

```
/*
 * testlibpq3.c Test PQprepare
 * PQprepare creates a prepared statement with specified parameters for PQexecPrepared to execute the
 * prepared statement.
 * Before running this example, run the following command to create a table:
 * create table t01(a int, b int);
 * insert into t01 values(1, 23);
 */
#include <stdio.h>
#include <stdlib.h>
#include <libpq-fe.h>
#include <string.h>
int main(int argc, char * argv[])
{
    /* The values of variables such as user and passwd must be read from environment variables or
    configuration files. Environment variables need to be configured as required. If no environment variable is
    used, a character string can be directly assigned. */
    PGconn *conn;
    PGresult * res;
    ConnStatusType pgstatus;
    char connstr[1024];
    char cmd_sql[2048];
    int nParams = 0;
    int paramLengths[5];
    int paramFormats[5];
    Oid paramTypes[5];
    char * paramValues[5];
    int i, cnt;
    char cid[32];
    int k;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /* Use PQconnectdb to connect to the database. The detailed connection information is as follows:
    connstr */
    sprintf(connstr,
            "hostaddr=%s dbname=%s port=%s user=%s password=%s",
            hostaddr, dbname, port, username, passwd);
    conn = PQconnectdb(connstr);
    pgstatus = PQstatus(conn);
    if (pgstatus == CONNECTION_OK)
    {
        printf("Connect database success!\n");
    }
    else
    {
        printf("Connect database fail:%s\n",PQerrorMessage(conn));
    }
}
```

```
    return -1;
}

/* Create table t01. */
res = PQexec(conn, "DROP TABLE IF EXISTS t01;CREATE TABLE t01(a int, b int);INSERT INTO t01
values(1, 23);");
if (PQresultStatus(res) != PGRES_COMMAND_OK)
{
    printf("Command failed: %s.\n", PQerrorMessage(conn));
    PQfinish(conn);
    return -1;
}

/* cmd_sql query */
sprintf(cmd_sql, "SELECT b FROM t01 WHERE a = $1");
/*Parameter corresponding to $1 in cmd_sql*/
paramTypes[0] = 23;
/* PQprepare creates a prepared statement with given parameters. */
res = PQprepare(conn,
                "pre_name",
                cmd_sql,
                1,
                paramTypes);
if (PQresultStatus(res) != PGRES_COMMAND_OK )
{
    printf("Failed to prepare SQL : %s\n: %s\n",cmd_sql, PQerrorMessage(conn));
    PQfinish(conn);
    return -1;
}
PQclear(res);
paramValues[0] = cid;
for (k=0; k<2; k++)
{
    sprintf(cid, "%d", 1);
    paramLengths[0] = 6;
    paramFormats[0] = 0;
    /*Execute the prepared statement.*/
    res = PQexecPrepared(conn,
                        "pre_name",
                        1,
                        paramValues,
                        paramLengths,
                        paramFormats,
                        0);
    if( (PQresultStatus(res) != PGRES_COMMAND_OK ) && (PQresultStatus(res) != PGRES_TUPLES_OK))
    {
        printf("%s\n",PQerrorMessage(conn));
        PQclear(res);
        PQfinish(conn);
        return -1;
    }
    cnt = PQntuples(res);
    printf("return %d rows\n", cnt);
    for (i=0; i<cnt; i++)
    {
        printf("row %d: %s\n", i, PQgetvalue(res, i, 0));
    }
    PQclear(res);
}
/* The execution is complete. Close the connection. */
PQfinish(conn);
return 0;
}
```

Example 4:

```
/*
 * testlibpq4.c
 * Test PQexecParams.
 * PQexecParams executes a command with parameters and requests the query result in binary format.
```

```
* Before running this example, run the following command to populate a database:
*
*
* CREATE TABLE test1 (i int4, t text);
*
* INSERT INTO test1 values (2, 'ho there');
*
* The expected output is as follows:
*
*
* tuple 0: got
* i = (4 bytes) 2
* t = (8 bytes) 'ho there'
*
*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <libpq-fe.h>

/* for ntohs/htons */
#include <netinet/in.h>
#include <arpa/inet.h>

static void
exit_nicely(PGconn *conn)
{
    PQfinish(conn);
    exit(1);
}

/*
 * This function is used to print out the query results. The results are in binary format
 * and fetched from the table created in the comment above.
 */
static void
show_binary_results(PGresult *res)
{
    int    i;
    int    i_fnum,
          t_fnum;

    /* Use PQfnumber to avoid assumptions about field order in the result. */
    i_fnum = PQfnumber(res, "i");
    t_fnum = PQfnumber(res, "t");

    for (i = 0; i < PQntuples(res); i++)
    {
        char    *iptr;
        char    *tptr;
        int     ival;

        /* Obtain the field value. (Ignore the possibility that they may be null.) */
        iptr = PQgetvalue(res, i, i_fnum);
        tptr = PQgetvalue(res, i, t_fnum);

        /*
         * The binary representation of INT4 is the network byte order,
         * which is better to be replaced with the local byte order.
         */
        ival = ntohs(*(uint32_t *) iptr);

        /*
         * The binary representation of TEXT is text. Since libpq can append a zero byte to it,
         * and think of it as a C string.
         */
    }
}
```

```
        printf("tuple %d: got\n", i);
        printf(" i = (%d bytes) %d\n",
            PQgetlength(res, i, i_fnum), ival);
        printf(" t = (%d bytes) '%s'\n",
            PQgetlength(res, i, t_fnum), tptr);
        printf("\n\n");
    }
}

int
main(int argc, char **argv)
{
    /* The values of variables such as user and passwd must be read from environment variables or
    configuration files. Environment variables need to be configured as required. If no environment variable is
    used, a character string can be directly assigned. */
    const char conninfo[1024];
    PGconn *conn;
    PGresult *res;
    const char *paramValues[1];
    int paramLengths[1];
    int paramFormats[1];
    uint32_t binaryIntVal;
    char *passwd = getenv("EXAMPLE_PASSWD_ENV");
    char *port = getenv("EXAMPLE_PORT_ENV");
    char *hostaddr = getenv("EXAMPLE_HOST_ENV");
    char *username = getenv("EXAMPLE_USERNAME_ENV");
    char *dbname = getenv("EXAMPLE_DBNAME_ENV");

    /*
    * If the user provides a parameter on the command line,
    * The value of this parameter is a conninfo character string. Otherwise,
    * Use environment variables or default values.
    */
    if (argc > 1)
        conninfo = argv[1];
    else
        sprintf(conninfo,
            "dbname=%s port=%s host=%s application_name=test connect_timeout=5 sslmode=allow user=%s
password=%s",
            dbname, port, hostaddr, username, passwd);

    /* Connect to the database. */
    conn = PQconnectdb(conninfo);

    /* Check whether the connection to the server was successfully established. */
    if (PQstatus(conn) != CONNECTION_OK)
    {
        fprintf(stderr, "Connection to database failed: %s",
            PQerrorMessage(conn));
        exit_nicely(conn);
    }

    res = PQexec(conn, "drop table if exists test1;CREATE TABLE test1 (i int4, t text);");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }

    PQclear(res);

    res = PQexec(conn, "INSERT INTO test1 values (2, 'ho there');");
    if (PQresultStatus(res) != PGRES_COMMAND_OK)
    {
        fprintf(stderr, "command failed: %s", PQerrorMessage(conn));
        PQclear(res);
        exit_nicely(conn);
    }
}
```

```
PQclear(res);

/* Convert the integer value "2" to the network byte order. */
binaryIntVal = htonl((uint32_t) 2);

/* Set the parameter array for PQexecParams. */
paramValues[0] = (char *) &binaryIntVal;
paramLengths[0] = sizeof(binaryIntVal);
paramFormats[0] = 1; /* Binary */

/* PQexecParams executes a command with parameters. */
res = PQexecParams(conn,
    "SELECT * FROM test1 WHERE i = $1::int4",
    1, /* One parameter */
    NULL, /* Enable the backend to deduce the parameter type. */
    paramValues,
    paramLengths,
    paramFormats,
    1); /* Binary result is required. */

if (PQresultStatus(res) != PGRES_TUPLES_OK)
{
    fprintf(stderr, "SELECT failed: %s", PQerrorMessage(conn));
    PQclear(res);
    exit_nicely(conn);
}
/* Output the binary result.*/
show_binary_results(res);

PQclear(res);

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

return 0;
}
```

4.2.5 Using Psycopg to Connect to a Database

Psycopg is a Python API used to execute SQL statements and provides a unified access API for GaussDB. Applications can perform data operations based on psycopg. Psycopg2 is the encapsulation of libpq and is implemented using the C language, which is efficient and secure. It provides cursors on both clients and servers, asynchronous communication and notification, and the COPY TO and COPY FROM functions. It supports multiple types of Python out-of-the-box and adapts to GaussDB data types. Through the flexible object adaptation system, you can extend and customize the adaptation. Psycopg2 is compatible with Unicode.

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

Table 4-20 Platforms supported by psycopg

OS	Platform	Python Version
EulerOS V2.0SP5	<ul style="list-style-type: none">● Arm64● x86_64	3.8.5
EulerOS V2.0SP9	<ul style="list-style-type: none">● Arm64● x86_64	3.7.4

OS	Platform	Python Version
EulerOS V2.0SP10, Kylin V10, and UnionTech20	<ul style="list-style-type: none">• Arm64• x86_64	3.7.9
EulerOS V2.0SP11 and SUSE 12.5	<ul style="list-style-type: none">• Arm64• x86_64	3.9.11
Huawei Cloud EulerOS 2.0	<ul style="list-style-type: none">• Arm64• x86_64	3.9.9

NOTICE

During pycopg2 compilation, OpenSSL of GaussDB is linked. OpenSSL of GaussDB may be incompatible with OpenSSL of the OS. If incompatibility occurs, for example, "version 'OPENSSL_1_1_1f' not found" is displayed, use the environment variable `LD_LIBRARY_PATH` to isolate the OpenSSL provided by the OS and the OpenSSL on which GaussDB depends.

For example, when the application software **client.py** that invokes pycopg2 is executed, the environment variable is explicitly assigned to the application software.

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

In the preceding command, `/path/to/pycopg2/lib` indicates the directory where the OpenSSL library on which the GaussDB depends is located. Change it as required.

Prerequisites

A Python development environment has been installed on the local PC.

Connecting to a Database

Step 1 Prepare related drivers and dependent libraries. Obtain the package **GaussDB-Kernel_Database version number_OS version number_64bit_Python.tar.gz** from the release package.

After the decompression, the following folders are generated:

- **pycopg2**: **pycopg2** library file
- **lib**: **lib** library file

Step 2 Load the driver.

- Before using the driver, perform the following operations:
 - a. Decompress the driver package of the corresponding version.

```
tar zxvf xxxx-Python.tar.gz
```
 - b. Copy **pycopg2** to the **site-packages** folder in the Python installation directory as the **root** user.

```
su root  
cp pycopg2 $(python3 -c 'import site; print(site.getsitepackages()[0])') -r
```

- c. Change the **psycopg2** directory permission to **755**.

```
chmod 755 $(python3 -c 'import site; print(site.getsitepackages()[0])')/psycopg2 -R
```
 - d. Add the **psycopg2** directory to the environment variable **\$PYTHONPATH** and validate it.

```
export PYTHONPATH=$(python3 -c 'import site; print(site.getsitepackages()[0])'):$PYTHONPATH
```
 - e. For non-database users, configure the **lib** directory in **LD_LIBRARY_PATH** after decompression.

```
export LD_LIBRARY_PATH=path/to/lib:$LD_LIBRARY_PATH
```
- Load a database driver before creating a database connection.

```
import psycopg2
```

Step 3 Connect to a database.

Connect to the database in non-SSL mode.

1. Use the `psycopg2.connect` function to obtain the connection object.
2. Use the connection object to create a cursor object.

Connect to the database in SSL mode.

When you use `psycopy2` to connect to the GaussDB server, you can enable SSL to encrypt the communication between the client and server. To enable SSL, you must have the server certificate, client certificate, and private key files. For details on how to obtain these files, see related documents and commands of OpenSSL.

1. Use the `.ini` file (the **configparser** package of Python can parse this type of configuration file) to save the configuration information about the database connection.
2. Add SSL connection parameters **sslmode**, **sslcert**, **sslkey**, and **sslrootcert** to the connection options.
 - a. **sslmode**: For details about the options, see [Table 4-21](#).
 - b. **sslcert**: client certificate path.
 - c. **sslkey**: client key path.
 - d. **sslrootcert**: root certificate path.
3. Use the `psycopg2.connect` function to obtain the connection object.
4. Use the connection object to create a cursor object.

CAUTION

To use SSL to connect to the database, ensure that the Python interpreter is compiled in the mode of generating a dynamic link library (.so) file. You can perform the following steps to check the connection mode of the Python interpreter:

1. Run the **import ssl** command in the Python interpreter to import SSL.
 2. Run the **ps ux** command to query the PID of the Python interpreter. Assume that the PID is *********.
 3. In the shell CLI, run the **pmap -p ***** | grep ssl** command and check whether the command output contains the path related to **libssl.so**. If yes, the Python interpreter is compiled in dynamic link mode.
-

Table 4-21 sslmode options

sslmode	Enable SSL Encryption	Description
disable	No	SSL connection is not enabled.
allow	Possible	If the database server requires SSL connection, SSL connection can be enabled. However, authenticity of the database server will not be verified.
prefer	Possible	If the database supports SSL connection, SSL connection is preferred. However, authenticity of the database server will not be verified.
require	Yes	SSL connection is required, but only data is encrypted. However, authenticity of the database server will not be verified.
verify-ca	Yes	SSL connection is required, and the validity of the server CA must be verified.
verify-full	Yes	The SSL connection must be enabled, which is not supported by GaussDB currently.

----End

4.2.6 Using Hibernate to Connect to a Database

Hibernate is an object-relational mapping (ORM) tool for the Java programming language. It provides an easy-to-use framework for automatically mapping Java objects to database tables, so that developers can operate databases in object-oriented mode. Hibernate frees developers from manually writing a large amount of SQL and JDBC code, which significantly reduces the development workload of the data access layer.

This section describes how to use Hibernate to connect to a GaussDB database and perform operations in it, for example, creating a table, modifying a table, and inserting, deleting, updating, or querying data in a table.

Configuring the POM Dependency

```
<dependency>
  <groupId>com.huaweicloud.gaussdb</groupId>
  <artifactId>opengaussjdbc</artifactId>
  <version>503.2.T35</version>
</dependency>
<dependency>
  <groupId>org.hibernate</groupId>
  <artifactId>hibernate-core</artifactId>
  <version>5.3.7.Final</version>
</dependency>
```

CAUTION

The Maven environment must have been configured before you configure the POM dependency.

Configuring the hibernate.cfg.xml Resource File

```
<?xml version="1.0" encoding="utf-8"?>
  <!DOCTYPE hibernate-configuration PUBLIC
    "-//Hibernate/Hibernate Configuration DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
  <session-factory>
    <!--GaussDB connection information-->
    <property name="connection.driver_class">com.huawei.opengauss.jdbc.Driver</property>
    <property name="connection.url">jdbc:opengauss://*****(Replace **** with the database
    IP address.);20000 (Replace 20000 with the database port.)/test? currentSchema=test (Replace test with the
    target database and schema.)</property>
    <property name="connection.username">*** (Replace *** with the correct username.)</property>
    <property name="connection.password">***** (Replace ***** with the correct password.)</property>

    <!-- The following configurations are optional. -->

    <!-- Specify whether to support dialects. -->
    <!-- In PostgreSQL compatibility mode, the following configuration must be set. -->
    <property name="dialect">org.hibernate.dialect.PostgreSQL82Dialect</property>

    <!-- Specify whether to print SQL statements when CURD commands are executed. -->
    <property name="show_sql">>true</property>

    <!-- Enable automatic table creation (and update).-->
    <!-- <property name="hbm2ddl.auto">update</property>-->
    <!-- <property name="hbm2ddl.auto">create</property>-->

    <!-- Register resources (entity class mapping file).-->
    <mapping resource="./student.xml"/>
  </session-factory>
</hibernate-configuration>
```

Preparing an Entity Class and Entity Class Mapping File

In this example, the **Student** class and **student.xml** file are used. The file path is **com.howtodoinjava.hibernate.test.dto**.

1. Create an entity class.

```
public class Student implements Serializable {
    int id;
    String name;
    // String age;
    // The constructor, get, and set methods are omitted here.
}
```

2. Prepare the **student.xml** file.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
  "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
  "http://www.hibernate.org/dtd/hibernate-mapping-3.0.dtd">
<hibernate-mapping>
  <!-- The mapping between classes and tables is made on the class element. -->
  <!-- name: fully-qualified class name -->
  <!-- table: table name -->
  <class name="com.howtodoinjava.hibernate.test.dto.Student" table="student">
    <!-- The mapping of the primary key is made on the id element. -->
    <!-- name: name of the attribute used as the primary key in the object -->
    <!-- column: name of the primary key field in the table -->
```

```
<!-- If the value of name is the same as that of column, column can be omitted. -->
<id name="id" column="id">
  <!-- Set the class attribute of the generator element to "assigned". The ID must be provided. --
>
  <generator class="assigned" />

  <!--** Pay attention to the Hibernate primary key generation policy. **-->

</id>
<!-- The mapping between attributes and fields is made on the property element. -->
<!-- name: attribute name in the class -->
<!-- column: field name in the table -->
<!-- If the value of name is the same as that of column, column can be omitted. -->
<property name="name" />
<!-- Same as the entity class. -->
<!--<property name="age"/-->
</class>
</hibernate-mapping>
```

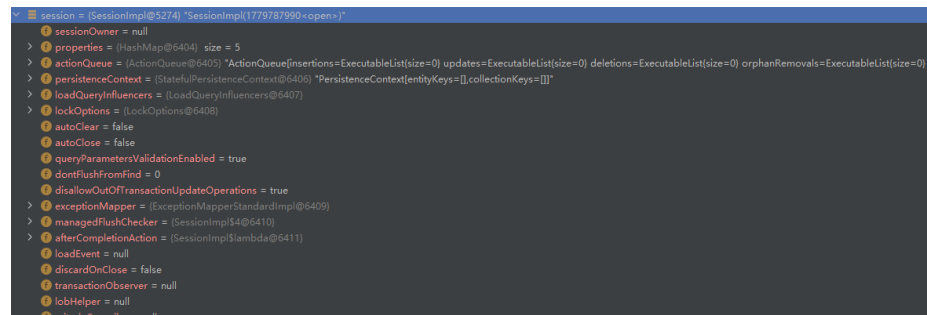
Function Test Example

1. Test the connection function.

a. Test method:

```
@Test
public void testConnection() {
    // Load the configuration information.
    Configuration conf = new Configuration().configure();
    // Create the SessionFactory object based on the configuration information.
    SessionFactory sessionFactory = conf.buildSessionFactory();
    // Open a session object related to the database.
    Session session = sessionFactory.openSession();
    System.out.println(session);
}
```

b. The breakpoint shows that the connection is successfully established.



2. Enable automatic table creation.

a. Comment out the following line in the configuration file:

```
<property name="hbm2ddl.auto">create</property>
```

b. After deleting the **student** table from the database, perform the following test method:

```
@Test
public void testCreateTableAndInsertData() {
    // Create the object to be tested.
    Student student = new Student();
    student.setId(16);
    student.setName("xiaoming");
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // Use session to save data.
    session.save(student);
}
```

```
// Commit the transaction.
transaction.commit();
// After the operation is complete, close the session connection object.
session.close();
}
```

- c. View the executed SQL statements printed on the console and the result in the GaussDB database.

```
[10.58.238.187:57638/100.85.152.50:20000] Connection is established. ID: b420808-1cab-408b-b1af-b20d8e53a80
21, 2023 11:01:08 com.hazelcast.opengauss.jdbc.core.v3.ConnectionFactoryImpl.openConnectionImpl
: Connect complete. ID: b420808-1cab-408b-b1af-b20d8e53a80
21, 2023 11:01:08 org.hibernate.dialect.Dialect <init>
INFO: HHH000040: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:01:45 org.hibernate.type.BasicTypeRegistry register
INFO: HHH000070: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
Hibernate: drop table if exists student cascade
21, 2023 11:01:45 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorJtaImpl.getIsolatedConnection
INFO: HHH000150: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@6b2d89] for (non-JTA)
Hibernate: create table student (id int4 not null, name varchar(255), primary key (id))
21, 2023 11:01:45 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorJtaImpl.getIsolatedConnection
INFO: HHH000150: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@34278b4d] for (non-JTA)
21, 2023 11:01:45 org.hibernate.engine.jdbc.spi.SqlExceptionHelper$StandardWarningHandler logWarning
WARN: SQL Warning Code: 0, SQLState: 08000
21, 2023 11:01:45 org.hibernate.engine.jdbc.spi.SqlExceptionHelper$StandardWarningHandler logWarning
WARN: CREATE TABLE / PRIMARY KEY will create implicit index "student_pkey" for table "student"
21, 2023 11:01:45 org.hibernate.tool.schema.internal.SchemaCreatorImpl.applyImportSources
INFO: HHH000470: Executing import script 'org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@1d8cfe7'
Hibernate: insert into student (name, id) values (?, ?)
```

```
test=> select * from test.student;
id | name
----+-----
 16 | xiaoming
(1 row)
```

The **student** table is successfully created, and the **id = 16, name = "xiaoming"** record is inserted into the table.

3. Modify the table (by adding data records).
 - a. Modify the configuration file and configure the path of the schema where the table to be modified is located in the URL.

In this test example, the **student** table is in schema **test** under database **test**. That is, the table path is **test.test.student**.

```
<property name="connection.url">jdbc:opengauss://xxx.xxx.xxx.xxx (Replace xxx.xxx.xxx.xxx with the database IP address.):20000 (Replace 20000 with the database port.)/test?currentSchema=test (Replace test with the target database and schema.)</property>
<!-- Uncomment update. -->
<property name="hbm2ddl.auto">update</property>
```

- b. Uncomment the **age** attribute in the entity class and XML file and execute the following method:

```
@Test
public void testAlterTable() {
    // Create the object to be tested.
    Student student = new Student();
    student.setId(15);
    student.setName("xiaohong");
    student.setAge("20");
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // Use session to save data.
    session.save(student);
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

- c. View the executed SQL statements printed on the console and the result in the GaussDB database.

```

[10.58.238.107:57638/100.85.152.58:20000] connection is established. ID: b42d6e8-1cab-4d0b-half-b2dd8e53a80
21, 2023 11:01:08 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: Connect complete. ID: b42d6e8-1cab-4d0b-half-b2dd8e53a80
21, 2023 11:01:08 org.hibernate.dialect.Dialect <init>
INFO: HHH000040: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:01:45 org.hibernate.type.BasicTypeRegistry register
INFO: HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
Hibernate: drop table if exists student cascade
21, 2023 11:01:45 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH000150: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@6b2d69] for (non-JTA)
Hibernate: create table student (id int4 not null, name varchar(255), primary key (id))
21, 2023 11:01:45 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH000150: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@3d278d4] for (non-JTA)
21, 2023 11:01:45 org.hibernate.engine.jdbc.spi.SqlExceptionHelperStandardWarningHandler logWarning
WARN: SQL Warning Code: 0, SQLState: 00000
21, 2023 11:01:45 org.hibernate.engine.jdbc.spi.SqlExceptionHelperStandardWarningHandler logWarning
WARN: CREATE TABLE / PRIMARY KEY will create implicit index "student_pkey" for table "student"
21, 2023 11:01:45 org.hibernate.tool.schema.internal.SchemaCreatorImpl applyImportSources
INFO: HHH000470: Executing 'import script' org.hibernate.tool.schema.internal.exec.ScriptSourceInputNonExistentImpl@71d8cfe7
Hibernate: insert into student (name, id) values (?, ?)

```

```

test=> select * from test.student;
id | name | age
---+---+---
16 | xiaoming |
15 | xiaohong | 20
(2 rows)

```

The framework automatically adds data records to the table based on the entity class and XML file.

4. Save data into the database.

a. Test method:

```

@Test
public void testInsert() {
    Student s1 = new Student(1,"q");
    Student s2 = new Student(2,"w");
    Student s3 = new Student(3,"e");
    ArrayList<Student> students = new ArrayList<>();
    students.add(s1);
    students.add(s2);
    students.add(s3);
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // Use session to save data.
    for (Student student : students) {
        session.save(student);
    }
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}

```

b. The result is as follows.

```

INFO: HHH000100: Connection properties: {user=test, password=****}
21, 2023 11:10:53 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl buildCreator
INFO: HHH000100: Autocommit mode: false
21, 2023 11:10:53 org.hibernate.engine.jdbc.connections.internal.DriverManagerConnectionProviderImpl$PooledConnections <init>
INFO: HHH000100: Hibernate connection pool size: 20 (min=1)
21, 2023 11:10:53 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
[08970650-ea76-4e3e-999b-02f4926e4f41] try to connect. IP: 100.85.152.58:20000
21, 2023 11:10:53 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
[10.58.238.107:57875/100.85.152.58:20000] connection is established. ID: d8970650-ea76-4e3e-999b-02f4926e4f41
21, 2023 11:10:53 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
Connect complete. ID: d8970650-ea76-4e3e-999b-02f4926e4f41
21, 2023 11:10:54 org.hibernate.dialect.Dialect <init>
INFO: HHH000040: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:11:33 org.hibernate.type.BasicTypeRegistry register
INFO: HHH000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
21, 2023 11:11:34 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: HHH000150: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bc5fac] for (non-JTA)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
Hibernate: insert into student (name, age, id) values (?, ?, ?)
Hibernate: insert into student (name, age, id) values (?, ?, ?)

```

```
test=> select * from test.student;
id | name | age
---+---+---
16 | xiaoming |
15 | xiaohong | 20
1 | q | 0
2 | w | 0
3 | e | 0
(5 rows)
```

Three rows of data are successfully inserted.

5. Query data in HQL mode.

a. Test method:

```
@Test
public void testHQL() {
    // HQL mode
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    // Create a session.
    Session session = sessionFactory.openSession();
    // Start a transaction.
    Transaction tx = session.beginTransaction();

    // Create an HQL query.
    String hql = "FROM Student S WHERE S.id = 15";
    Query query = session.createQuery(hql);

    // Execute the query and obtain the result.
    List results = query.list();

    // Commit the transaction.
    tx.commit();

    // End the session.
    session.close();
}
```

b. The result is as follows.

```
21, 2023-11-15:51 com.huawei.opengauss.jdbc.core.v1.ConnectionFactoryImpl.openConnectionImpl
INFO: [79760384-6903-4769-8afd-ba92fb927eed] try to connect, ip: 100.85.152.58:28800
21, 2023-11-15:51 com.huawei.opengauss.jdbc.core.v1.ConnectionFactoryImpl.openConnectionImpl
INFO: [10-58-238-107-57909/100.85.152.58:28800] connection is established, ID: 79760384-6903-4769-8afd-ba92fb927eed
21, 2023-11-15:51 com.huawei.opengauss.jdbc.core.v1.ConnectionFactoryImpl.openConnectionImpl
INFO: connect complete, ID: 79760384-6903-4769-8afd-ba92fb927eed
21, 2023-11-15:52 org.hibernate.dialect.Dialect.<init>
INFO: HH4000002: using dialect: org.hibernate.dialect.nosql.NoSQLDialect
21, 2023-11-16:29 org.hibernate.type.BasicTypeRegistry.register
INFO: HH4000270: type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@405325cf
21, 2023-11-16:30 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl.getIsolatedConnection
INFO: HH410001581: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@7bca6fac] for (non-JTA)
21, 2023-11-16:30 org.hibernate.internal.QueryTranslatorFactoryInitiator.initiateService
INFO: HH4000397: using ASTQueryTranslatorFactory
Hibernate: select student0_id as id1_0, student0_name as name2_0, student0_age as age3_0 from student student0 where student0_id=15
```

6. Query data in SQL mode.

a. Test method:

```
@Test
public void testQuery() {
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // SQL mode
    List<Student> students = session.createSQLQuery("select * from test.student where id = 1").addEntity(Student.class).list();
    for (int i = 0; i < students.size(); i++) {
        System.out.println(students.get(i));
    }
    students.get(0).setAge("20");
    // Commit the transaction.
    transaction.commit();
}
```

```
// After the operation is complete, close the session connection object.
session.close();
}
```

b. The result is as follows.

```
21, 2023 11:19:13 com.huawei.opengauss.jdbc.core.v3.connection.factory.impl.openConnectionImpl
: [afdd16eb-c28b-487c-8768-1a56ee115399] try to connect, IP: 100.85.152.58:20000
21, 2023 11:19:13 com.huawei.opengauss.jdbc.core.v3.connection.factory.impl.openConnectionImpl
: [16,58,238,187:58078/100.85.152.58:20000] connection is established, ID: afdd16eb-c28b-487c-8768-1a56ee115399
21, 2023 11:19:13 com.huawei.opengauss.jdbc.core.v3.connection.factory.impl.openConnectionImpl
: Connect complete, ID: afdd16eb-c28b-487c-8768-1a56ee115399
21, 2023 11:19:13 org.hibernate.dialect.Dialect <init>
INFO: H484000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:19:52 org.hibernate.type.BasicTypeRegistry register
INFO: H484000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
21, 2023 11:19:53 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: H48410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@5c60bb0a] for (non-JTA)
Hibernate: select * from test.student where id = 1
Student(id=1, name=q, age=0)
Hibernate: update student set name=?, age=? where id=?
```

```
test=> select * from test.student;
id | name | age
---+---+---
16 | xiaoming |
15 | xiaohong | 20
1 | q | 20
2 | w | 0
3 | e | 0
(5 rows)
```

The data record of the student whose ID is 1 is found and the value of **age** is changed to **20**.

7. Modify data.

- Test method:

```
@Test
public void testUpdate() {
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // SQL mode
    session.createQuery("update test.student set age = 19 where id = 16").executeUpdate();
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

- The result is as follows.

```
21, 2023 11:21:18 org.hibernate.engine.jdbc.connections.internal.JdbcConnectionProviderImpl$JdbcConnectionPool <init>
INFO: H484000115: Hibernate connection pool size: 20 (min=1)
21, 2023 11:21:18 com.huawei.opengauss.jdbc.core.v3.connection.factory.impl.openConnectionImpl
: [7b5438f9-cf89-4b06-93de-6428abd92aaf] try to connect, IP: 100.85.152.58:20000
21, 2023 11:21:18 com.huawei.opengauss.jdbc.core.v3.connection.factory.impl.openConnectionImpl
: [16,58,238,187:58129/100.85.152.58:20000] connection is established, ID: 7b5438f9-cf89-4b06-93de-6428abd92aaf
21, 2023 11:21:18 com.huawei.opengauss.jdbc.core.v3.connection.factory.impl.openConnectionImpl
: Connect complete, ID: 7b5438f9-cf89-4b06-93de-6428abd92aaf
21, 2023 11:21:19 org.hibernate.dialect.Dialect <init>
INFO: H484000400: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023 11:21:59 org.hibernate.type.BasicTypeRegistry register
INFO: H484000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
21, 2023 11:21:59 org.hibernate.resource.transaction.backend.jdbc.internal.DdlTransactionIsolatorNonJtaImpl getIsolatedConnection
INFO: H48410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@5c60bb0a] for (non-JTA)
Hibernate: update test.student set age = 19 where id = 16
```

```
test=> select * from test.student;
id | name | age
---+---+---
15 | xiaohong | 20
1 | q | 20
2 | w | 0
3 | e | 0
16 | xiaoming | 19
(5 rows)
```

The **age** field of the student whose ID is **16** is successfully changed to **19**.

8. Delete data.

a. Test method:

```
@Test
public void testDelete() {
    // Start a transaction based on session.
    Configuration conf = new Configuration().configure();
    SessionFactory sessionFactory = conf.buildSessionFactory();
    Session session = sessionFactory.openSession();
    Transaction transaction = session.beginTransaction();
    // SQL mode
    List<Student> students = session.createQuery("select * from
test.student").addEntity(Student.class).list();
    System.out.println(students);
    session.createQuery("delete from test.student where id = " +
students.get(0).getId()).executeUpdate();
    // Commit the transaction.
    transaction.commit();
    // After the operation is complete, close the session connection object.
    session.close();
}
```

b. The result is as follows.

```
21, 2023-11-22 15:15:15 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [4525c68e-c87c-4cfd-8884-99ba1a0635ab] Try to connect. IP: 100.85.152.58:20000
21, 2023-11-22 15:15:15 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: [10c58c238-1075816d/100.85.152.58:20000] connection is established. ID: 4525c68e-c87c-4cfd-8884-99ba1a0635ab
21, 2023-11-22 15:15:15 com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
: connect complete. ID: 4525c68e-c87c-4cfd-8884-99ba1a0635ab
21, 2023-11-22 15:15:15 org.hibernate.dialect.Dialect <init>
INFO: 48400000: Using dialect: org.hibernate.dialect.PostgreSQLDialect
21, 2023-11-22 15:15:15 org.hibernate.type.BasicTypeRegistry register
INFO: 484000270: Type registration [java.util.UUID] overrides previous : org.hibernate.type.UUIDBinaryType@e1162e7
21, 2023-11-22 15:15:15 org.hibernate.resource.transaction.backend.jdbc.internal.jdt.TransactionIsolationImpl getIsolatedConnection
INFO: 48410001501: Connection obtained from JdbcConnectionAccess [org.hibernate.engine.jdbc.env.internal.JdbcEnvironmentInitiator$ConnectionProviderJdbcConnectionAccess@5c66bbab] for (non-JTA)
Hibernate: select * from test.student
[Student(id=1, name=xiaohong, age=20), Student(id=2, name=q, age=0), Student(id=3, name=e, age=0), Student(id=16, name=xiaoming, age=19)]
Hibernate: delete from test.student where id = 15
```

```
test=> select * from test.student;
id | name | age
---+---+---
1 | q | 20
2 | w | 0
3 | e | 0
16 | xiaoming | 19
(4 rows)
```

The record whose ID is 15 has been deleted from the **student** table.

4.2.7 Using MyBatis to Connect to a Database

MyBatis is a first class persistence framework with support for custom SQL, stored procedures, and advanced mappings. MyBatis eliminates almost all of the JDBC code and manual setting of parameters and retrieval of results. MyBatis can use simple XML or annotations for configuration and map primitive. It can map interfaces and Java Plain Old Java Objects (POJOs) to database records.

This section describes how to use MyBatis to connect to a GaussDB database.

Configuring the POM Dependency

```
<dependency>
  <groupId>com.huaweicloud.gaussdb</groupId>
  <artifactId>opengaussjdbc</artifactId>
  <version>503.2.T35</version>
</dependency>
<dependency>
  <groupId>org.mybatis</groupId>
  <artifactId>mybatis</artifactId>
```



```
<version>3.5.6</version>
</dependency>
```

⚠ CAUTION

The Maven environment must have been configured before you configure the POM dependency.

Configuring the Required File

Configuring the **mybatis-config.xml** file.

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE configuration
  PUBLIC "-//mybatis.org//DTD Config 3.0//EN"
  "http://mybatis.org/dtd/mybatis-3-config.dtd">
<!-- Root element of the configuration file -->
<configuration>
  <!--Configure the global attributes.-->
  <settings>
    <!--Use getGeneratedKeys of JDBC to obtain the auto-increment primary key value of the database.-->
    <setting name="useGeneratedKeys" value="true"/>
    <!--Replace the column alias with the column label. The default value is true.-->
    <setting name="useColumnLabel" value="true" />
    <!--Enable camel-case naming conversion: Table{create_time} -> Entity{createTime}-->
    <setting name="mapUnderscoreToCamelCase" value="true" />
    <setting name="logImpl" value="STDOUT_LOGGING" />
  </settings>

  <environments default="development">
    <environment id="development">
      <transactionManager type="JDBC"/>
      <dataSource type="POOLED">
        <property name="driver" value="com.huawei.opengauss.jdbc.Driver"/>
        <property name="url" value="jdbc:opengauss://***.***.***.*** (Replace ***.***.***.*** with the
database IP address):20000 (Replace 20000 with the database port)/test? (Replace test with the
corresponding database name.)connectionTimeout=10"/>
        <property name="username" value="*** (Replace *** with the correct username.)"/>
        <property name="password" value="***** (Replace ***** with the correct password.)"/>
      </dataSource>
    </environment>
  </environments>
  <!--Register mapper (address of mapper.xml).-->
  <mappers>
    <mapper resource="mapper/StudentDaoMapper.xml"/></mapper>
  </mappers>
</configuration>
```

Examples

1. Test the entity class **StudentEntity.java** (in **com.huawei.entity**).

```
public class StudentEntity {
    Integer id;
    String name;
}
```

2. Configure the **StudentDaoMapper.xml** file corresponding to the entity class (in **resources.mapper**).

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE mapper PUBLIC "-//mybatis.org//DTD Mapper 3.0//EN" "http://mybatis.org/dtd/mybatis-3-
mapper.dtd" >
<mapper namespace="StudentMapper">
  <!-- Query by primary key -->
  <select id="getList" resultType="com.huawei.entity.StudentEntity" >
    select * from student;
```

```
</select>
</mapper>
```

3. Test table query.

```
@Test
public void mainTest() throws IOException {
    // 1. Read the core configuration file of MyBatis (mybatis-config.xml).
    InputStream in = Resources.getResourceAsStream("mybatis-config.xml");
    // 2. Obtain a SqlSessionFactory factory object based on the configuration information.
    SqlSessionFactory fac = new SqlSessionFactoryBuilder().build(in);
    // 3. Obtain a SqlSession object through the factory.
    SqlSession session = fac.openSession();
    // 4. Find the SQL statement to be executed based on the namespace and ID and execute the SQL
    statement.
    List<StudentEntity> list = session.selectList("StudentMapper.getList");
    // 5. Output the result.
    list.forEach(i -> {
        System.out.println(i.toString());
    });
}
```

4. Query result logs.

```
Setting autocommit to false on JDBC Connection [com.huawei.opengauss.jdbc.jdbc.PgConnection@47caedad]
==> Preparing: select * from student;
==> Parameters:
<== Columns: id, name
<== Row: 1, test
<== Total: 1
StudentEntity(id=1, name=test)
```

CAUTION

Currently, **PaginationInnerInterceptor** in the MybatisPlus plug-in does not adapt to the GaussDB driver. To resolve this problem, set **DbType** to **POSTGRE_SQL** when creating the **PaginationInnerInterceptor** object. Example:

```
PaginationInnerInterceptor innerInterceptor = new
PaginationInnerInterceptor(DbType.POSTGRE_SQL)
```

4.2.8 Using JayDeBeApi to Connect to a Database

JayDeBeApi is a Python module that provides a convenient, efficient way for Python developers to use the Java JDBC driver to connect to and perform operations on databases.

This section describes how to use JayDeBeApi to connect to the GaussDB database.

Environment Configuration

1. Configure the GaussDB development environment.

Prepare the basic development environment of GaussDB and obtain the database connection parameters. For example:

```
gsqll -h ***.***.***.*** -p 20000 -U *** -W ***** -d test
```

Parameter description:

-h: IP address of the server hosting the GaussDB instance

-p: connection port of the GaussDB instance

-U: username for the connection

- W**: user password
 - d**: name of the database you want to connect.
2. Install the JayDeBeApi driver.
 - a. Install Java JDK 8 and Python 3 on the local PC. To check the software versions, run the following commands:


```
java -version
python --version
pip --version
```
 - b. If the server can connect to the Python Package Index (PyPI), run the pip command to install JayDeBeApi:


```
pip install jaydebeapi
```

If the server cannot connect to PyPI, download an offline installation package of JayDeBeApi and install it on the local PC.
 3. Obtain the GaussDB driver package.

Download particular packages listed in [Table 4-22](#) based on the version of your instance.

Table 4-22 Driver package download list

Version	Download Address
3.x	Driver package Verification package for the driver package
2.x	Driver package Verification package for the driver package

To prevent a software package from being tampered with during transmission or storage, download the corresponding verification package and perform the following steps to verify the software package:

- a. Upload the software package and verification package to the same directory on a Linux VM.
- b. Run the following command to verify the integrity of the software package:

```
cat GaussDB_driver.zip.sha256 | sha256sum --check
```

If **OK** is displayed in the command output, the verification is successful.

```
GaussDB_driver.zip: OK
```

Example

1. Create a script file.
 - Create a test_jaydebeapi.py file and write the following code into the file:


```
#!/usr/bin/env python3.x
# -*- coding: UTF-8 -*-
encoding = "utf8"
import jaydebeapi

def test_jaydebeapi():
    #Set required parameters.
    url = 'jdbc:opengauss://***.***.***.***:20000/test'
```

```
user = '****'
password = '*****'
driver = 'com.huawei.opengauss.jdbc.Driver'
jarFile = './opengaussjdbc.jar'

conn = jaydebeapi.connect(driver, url, [user, password], jarFile)
cur = conn.cursor()

#Create a table named students.
sql = 'create table students (id int, name varchar(20))'
cur.execute(sql)

#Insert three groups of data into the students table.
sql = "insert into students values(1,'xiaoming'),(2,'xiaohong'),(3,'xiaolan')"
cur.execute(sql)

#Query all data in the students table.
sql = 'select * from students'
cur.execute(sql)
ans = cur.fetchall()
print(ans)

#Update data in the students table.
sql = 'update students set name = \'xiaolv\' where id = 1'
cur.execute(sql)

#Query all data in the students table again.
sql = 'select * from students'
cur.execute(sql)
ans = cur.fetchall()
print(ans)

#Delete the students table.
sql = 'drop table students'
cur.execute(sql)

cur.close()
conn.close()

test_jaydebeapi()
```

– Configure required parameters in the code.

```
#Set the connection URL, which requires the IP address, port number, and database name of the
database server you want to connect.
url = 'jdbc:opengauss://***.***.***.***:20000/test'
#Enter the username.
user = '****'
#Enter the password.
password = '*****'
#Specify the path to the JDBC driver class.
driver = 'com.huawei.opengauss.jdbc.Driver'
#Specify the path to the JAR package of the JDBC driver. By default, it is stored in the same
directory as the test_jaydebeapi.py file.
jarFile = './opengaussjdbc.jar'
```

2. Execute the program.

Run the following command to execute the **test_jaydebeapi.py** file:

```
python ./test_jaydebeapi.py
```

3. Check the result.

The GaussDB database is successfully connected, and two query results are returned, as shown in the following figure.

```
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: [f42f7374-3bfd-4f91-b370-4bbd007aea16] Try to connect. IP: 127.0.0.1:20000
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: [127.0.0.1:48426/127.0.0.1:20000] Connection is established. ID: f42f7374-3bfd-4f91-b370-4bbd007aea16
Oct 11, 2023 12:50:41 PM com.huawei.opengauss.jdbc.core.v3.ConnectionFactoryImpl openConnectionImpl
INFO: Connect complete. ID: f42f7374-3bfd-4f91-b370-4bbd007aea16
[[('1', 'xiaoming'), (2, 'xiaohong'), (3, 'xiaolan')]]
[[('1', 'xiaolv'), (2, 'xiaohong'), (3, 'xiaolan')]]
```


5 Example: Using DAS to Connect to an Instance and Execute SQL Statements


This section describes how to create a pay-per-use GaussDB instance with the minimum specifications and execute basic SQL syntax.

- [Buying an Instance](#)
- [Connecting to an Instance Through DAS](#)
- [Getting Started with SQL](#)

Buying an Instance

Step 1 Log in to the [management console](#).

Step 2 Click  in the upper left corner and select a region.

Step 3 Click  on the left and choose **Databases > GaussDB**.

Step 4 In the navigation pane on the left, choose **GaussDB > Instances**.

Step 5 Click **Buy DB Instance**.

Step 6 Configure the basic information, such as the billing mode and instance name.

Figure 5-1 Billing mode and basic information

Billing Mode: Yearly/Monthly Pay-per-use ?

Region:

Regions are geographic areas isolated from each other. Resources are region-specific and cannot be used across regions through internal network connections. For low network latency and quick resource access, select the nearest region.

DB Instance Name: ?

DB Engine:

DB Engine Version: 1.4 Enterprise Edition 2.7 Enterprise Edition

DB Instance Type: Distributed Primary/Standby

Deployment Model ?:

Transaction Consistency ?: Strong consistency Eventual consistency

Replicas ?: 3 ?

Shards: 3 ?

Coordinator Nodes ?: 3 ?

If Coordinator Nodes is set to 1, the instance can only be used for testing.

AZ: AZ3 AZ1 AZ2

Only one or three AZs can be selected.

Time Zone: ?

Step 7 Configure instance specifications.

Instance Specifications ?: General-enhanced II

Flavor Name:

8 vCPUs | 64 GB Unavailable for production environment

16 vCPUs | 128 GB

32 vCPUs | 256 GB

64 vCPUs | 512 GB

DB Instance Specifications: General-enhanced II | 2 vCPUs | 16 GB

Storage Type: Ultra-high I/O [Learn more about storage types.](#)

Storage Space (GB): ?

Disk Encryption: Disable Recommended Enable ?

Step 8 Select a VPC and security group for the instance and configure the database port.
The VPC you selected must contain sufficient subnets.

The screenshot shows a configuration page for a database instance. It includes sections for VPC (set to vpc-default), Security Group (set to default), Database Port (set to default), Administrator (set to root), Administrator Password (with a confirmation field), Parameter Template (set to Default-Enterprise-Edition-GaussDB-3.103-IN...), Enterprise Project (set to --Select--), and Tag (with fields for Tag key and Tag value).

Step 9 Configure the administrator password, enterprise project, and parameter template.

This screenshot focuses on the password and template configuration. It shows the Administrator Password field with a confirmation field, the Parameter Template dropdown set to 'Default-Enterprise-Edition-GaussDB-3.103-IN...', the Enterprise Project dropdown set to '--Select--', and the Tag section with 'Tag key' and 'Tag value' input fields.

Step 10 Click **Next**, confirm the instance information, and click **Submit**.


Step 11 Go to the instance list.


If status of the instance becomes available, the instance has been created.

----End

Connecting to an Instance Through DAS

Step 1 Log in to the [management console](#).

Step 2 Click  in the upper left corner and select a region.

Step 3 Click  on the left and choose **Databases > Data Admin Service**.

Step 4 In the navigation pane on the left, choose **Development Tool** to go to the login list page.

Step 5 Click **Add Login**.

NOTICE

After the database is created, the **root** user is added by default. You do not need to create a **root** user.

Step 6 Set **DB Engine** to GaussDB, retain the default value of **Source Database**, and configure required parameters.

You are advised to enable **Collect Metadata Periodically** and **Show Executed SQL Statements**.

If a message is displayed indicating that a connection has been established, go to **Step 9**.

Step 7 Click **Test Connection**.

If a message is displayed indicating connection successful, continue with the operation. If a message is displayed indicating connection failed and the failure cause is provided, make modifications according to the error message.

Step 8 Click **OK**.

Step 9 Locate the added instance, click **Log In** in the **Operation** column.

Step 10 Go to the **SQL Query** page.

----End

Getting Started with SQL

Step 1 Create a database user.

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

CREATE USER joe WITH PASSWORD "xxxxxxxx";

If information similar to the following is displayed, the creation is successful.

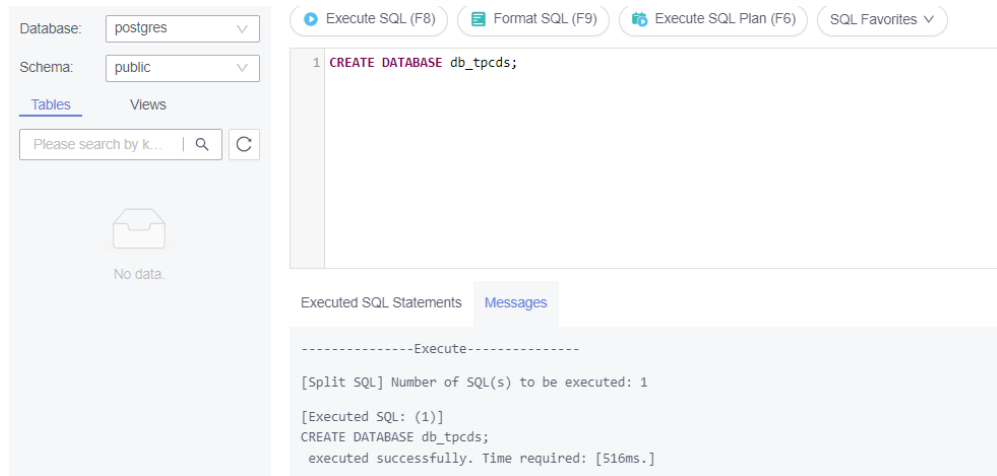
The screenshot shows the SQL Query page in a web interface. On the left, there are dropdown menus for 'Database: postgres' and 'Schema: public'. Below these are tabs for 'Tables' and 'Views', and a search bar with the text 'Please search by k...'. The main area contains a single SQL statement: '1 CREATE USER joe WITH PASSWORD '...' ;'. Below the statement, there are tabs for 'Executed SQL Statements' and 'Messages'. The 'Messages' tab is active, displaying the following output: '-----Execute-----', '[Split SQL] Number of SQL(s) to be executed: 1', '[Executed SQL: (1)]', 'CREATE USER joe WITH PASSWORD '...' ;', and 'executed successfully. Time required: [52ms.]'.

In this case, you have created a user named **joe**, and the user password is **xxxxxxx**.
For more information about database users, see [Users and Permissions](#).

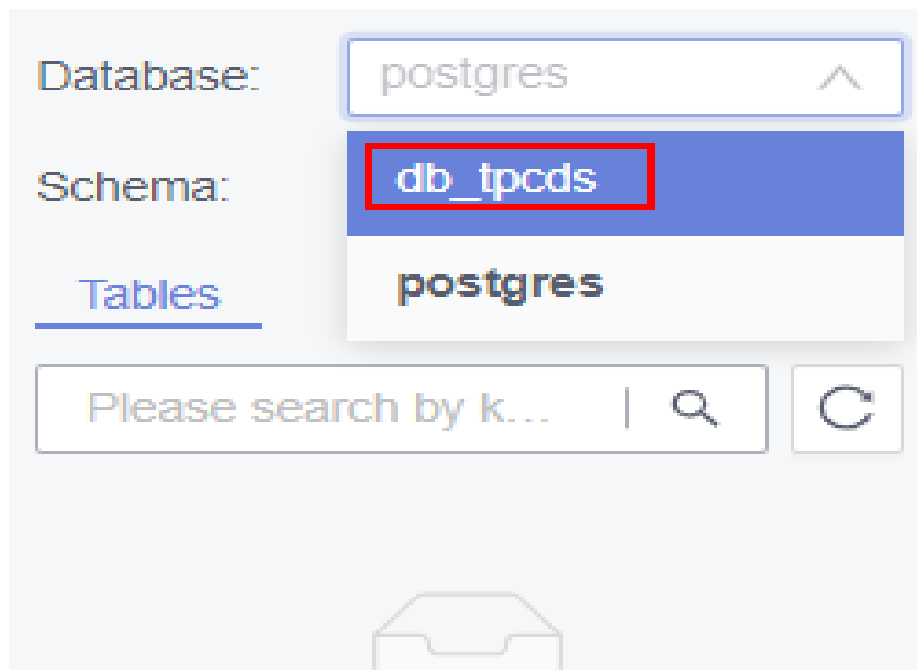
Step 2 Create a database.

CREATE DATABASE *db_tpcds*;

If information similar to the following is displayed, the creation is successful.



Switch to the newly created database in the upper left corner.



Step 3 Create a table.

- Run the following command to create a schema:

CREATE SCHEMA *myschema*;

- Create a table named **mytable** that has only one column. The column name is **firstcol** and the column type is integer.

CREATE TABLE *myschema.mytable (firstcol int);*

- Insert data to the table.
INSERT INTO myschema.mytable values (100);
- View data in the table.
SELECT * FROM myschema.mytable;

Note:

- By default, new database objects, such as the **mytable** table, are created in the *\$user* schema. For more information about schemas, see [Creating and Managing Schemas](#).
- For details about how to create a table, see [Creating and Managing Tables](#).
- In addition to the created tables, a database contains many system catalogs. These system catalogs contain information about instance installation as well as GaussDB queries and processes. You can collect information about the database by querying the system catalogs. For details about querying system catalogs, see [Querying a System Catalog](#).

Step 4 In the **db_tpcds** database, run the following statement as user **root** to grant all permissions of the **db_tpcds** database to user **joe**:

```
GRANT ALL ON DATABASE db_tpcds TO joe;
```

```
GRANT USAGE ON schema myschema TO joe;
```

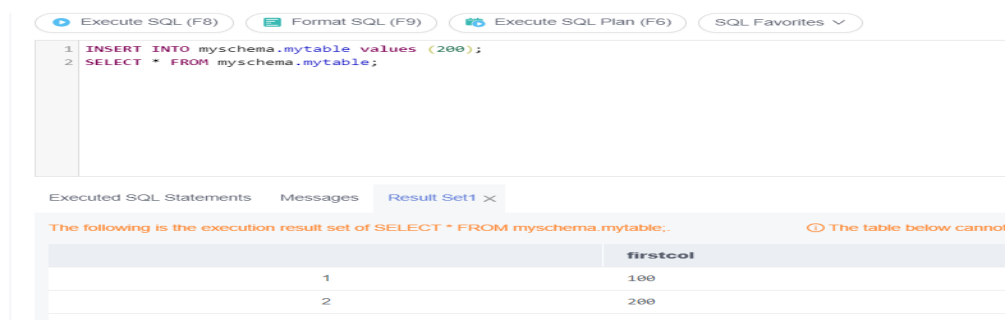
```
GRANT ALL ON TABLE myschema.mytable TO joe;
```

Step 5 Log in to the **db_tpcds** database as user **joe**.

Step 6 After login, insert data into the table and verify the data.

```
INSERT INTO myschema.mytable values (200);
```

```
SELECT * FROM myschema.mytable;
```



The screenshot shows a SQL execution interface with the following elements:

- Buttons: Execute SQL (F8), Format SQL (F9), Execute SQL Plan (F6), SQL Favorites.
- SQL Statements:

```
1 INSERT INTO myschema.mytable values (200);
2 SELECT * FROM myschema.mytable;
```
- Executed SQL Statements, Messages, Result Set1 x
- Message: The following is the execution result set of SELECT * FROM myschema.mytable.
- Table with 2 columns: firstcol, and 2 rows of data.

	firstcol
1	100
2	200

----End