TaurusDB for PostgreSQL

User Guide

Issue 01

Date 2025-11-14

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Huawei Cloud Computing Technologies Co., Ltd.

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

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

Contents

1 Using IAM to Grant Access to TaurusDB for PostgreSQL

1.1 Creating a User and Granting TaurusDB for PostgreSQL Permissions

This chapter describes how to use **Identity and Access Management (IAM)** for fine-grained permissions management for your TaurusDB for PostgreSQL resources. With IAM, you can:

- Create IAM users for employees based on your enterprise's organizational structure. Each IAM user has their own identity credentials for accessing TaurusDB for PostgreSQL resources.
- Grant only the permissions required for users to perform a specific task.
- Entrust a Huawei Cloud account or cloud service to perform efficient O&M on your TaurusDB for PostgreSQL resources.

If your Huawei Cloud account does not require individual IAM users, skip this chapter.

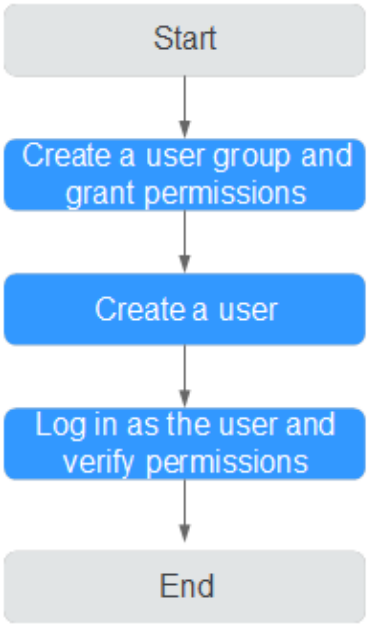
This section describes the procedure for granting permissions (see **Figure 1-1**).

Prerequisites

Before assigning permissions to user groups, you should learn about TaurusDB for PostgreSQL system-defined policies and select the policies as required. For the system-defined policies of other services, see **System-defined Policies/Roles**.

Process Flow

Figure 1-1 Process for granting TaurusDB for PostgreSQL permissions



1. **Create a user group and assign permissions** to it.

   Create a user group on the IAM console and attach the **TaurusDB FullAccess** policy to the group.

   NOTE

   To use some interconnected services, you also need to configure permissions of such services.

   For example, to connect to your DB instance through the console, configure the **DAS FullAccess** permission of Data Admin Service (DAS) besides **TaurusDB FullAccess**.

2. **Create an IAM user and add it to the user group**.

   Create a user on the IAM console and add the user to the group created in **1**.

3. **Log in** and verify permissions.

   Log in to the TaurusDB console by using the created user, and verify that the user only has read permissions for TaurusDB for PostgreSQL.

   Choose **Service List** > **TaurusDB** and click **Buy DB Instance**. If you can buy a TaurusDB for PostgreSQL instance, the required permission policy has already been applied.

1.2 Creating a TaurusDB for PostgreSQL Custom Policy

Custom policies can be created to supplement the system-defined policies of TaurusDB for PostgreSQL.

You can create custom policies in either of the following two ways:

- Visual editor: Select cloud services, actions, resources, and request conditions without the need to know policy syntax.

- JSON: Edit JSON policies from scratch or based on an existing policy.

For details, see **Creating a Custom Policy**. This section provides examples of common TaurusDB for PostgreSQL custom policies.

Example Custom Policies

- Example 1: Allowing users to create TaurusDB for PostgreSQL instances

```
{
    "Version": "1.1",
    "Statement": [{
        "Effect": "Allow",
        "Action": ["gaussdb:instance:create"]
    }]
}
```

- Example 2: Denying TaurusDB for PostgreSQL instance deletion

  A policy with only "Deny" permissions must be used in conjunction with other policies to take effect. If the permissions assigned to a user include both "Allow" and "Deny", the "Deny" permissions take precedence over the "Allow" permissions.

  The following method can be used if you need to assign permissions of the **TaurusDB FullAccess** policy to a user but you want to prevent the user from deleting TaurusDB for PostgreSQL instances. Create a custom policy for denying TaurusDB for PostgreSQL instance deletion, and attach both policies to the group the user belongs to. Then, the user can perform all operations on TaurusDB for PostgreSQL instances except deleting them. The following is an example deny policy:

```
{
    "Version": "1.1",
    "Statement": [{
        "Action": ["gaussdb:instance:delete"],
        "Effect": "Deny"
    }]
}
```


2 Buying a TaurusDB for PostgreSQL Instance

Scenarios

You can on the TaurusDB for PostgreSQL console.

TaurusDB for PostgreSQL only supports the pay-per-use billing mode. You can tailor your compute resources and storage space to your business needs.

Prerequisites

- You have **created a Huawei ID and enabled Huawei Cloud services**.
- You can create an IAM user or user group on the IAM console and grant it specific operation permissions, to perform refined management on Huawei Cloud. For details, see **Creating a User and Granting TaurusDB for PostgreSQL Permissions**.

Procedure

Step 1 Go to the **Buy DB Instance** page.

Step 2 On that page, click the **Custom Config** tab, select a billing mode, configure parameters about your instance, and click **Next**.

- Basic configuration

  Only pay-per-use billing is supported.
- Resource selection

Figure 2-1 Resource selection

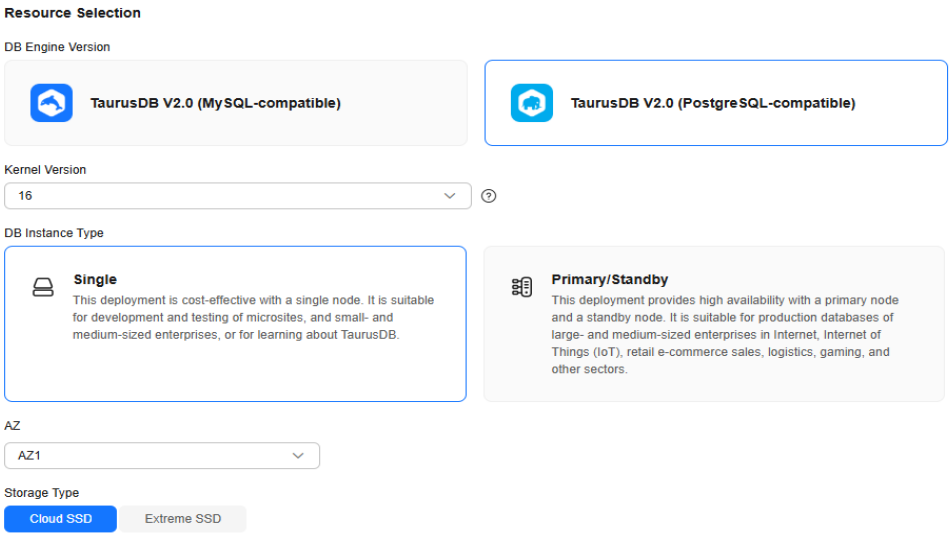


Table 2-1 Basic information

Parameter	Description
Region	Region where your resources are located. **NOTE** Products in different regions cannot communicate with each other through a private network. After a DB instance is created, the region cannot be changed. Therefore, exercise caution when selecting a region.
Compatible DB Engine	TaurusDB for PostgreSQL
Kernel Version	TaurusDB for PostgreSQL 16

Parameter	Description
DB Instance Type	– **Primary/Standby**: uses an HA architecture with a primary DB instance and a synchronous standby DB instance. It is suitable for production databases of large and medium enterprises in Internet, Internet of Things (IoT), retail e-commerce sales, logistics, gaming, and other sectors. The standby DB instance improves instance reliability and is invisible to you after being created. An AZ is a physical region where resources use independent power supply and networks. AZs are physically isolated but interconnected through an internal network. Some regions support both single AZs and multiple AZs and some only support single AZs. To achieve high reliability, TaurusDB for PostgreSQL will automatically deploy your primary and standby DB instances in different physical servers even if you deploy them in the same AZ. You can deploy primary and standby DB instances in a single AZ or across AZs to achieve failover and high availability. – **Single**: uses a single-node architecture, which is less expensive than primary/standby DB instances. It is suitable for development and testing of microsites, and small- and medium-sized enterprises, or for learning about TaurusDB for PostgreSQL.
Storage Type	Determines the DB instance read/write speed. The higher the maximum throughput is, the higher the DB instance read/write speed can be. – **Cloud SSD**: cloud disks used to decouple storage from compute. – **Extreme SSD**: uses 25GE network and RDMA technologies to provide you with up to 1,000 MB/s throughput per disk and sub-millisecond latency. **NOTE** – The cloud SSD and extreme SSD storage types are supported with general-purpose, dedicated, and Kunpeng general-enhanced DB instances. – The IOPS supported by cloud SSDs depends on the I/O performance of Elastic Volume Service (EVS) disks. For details, see the description about ultra-high I/O in **Disk Types and Performance** of *Elastic Volume Service Service Overview*. – The IOPS supported by extreme SSDs depends on the I/O performance of Elastic Volume Service (EVS) disks. For details, see the description about extreme SSDs in **Disk Types and Performance** of *Elastic Volume Service Service Overview*.

- Instance options

Figure 2-2 Specifications and storage

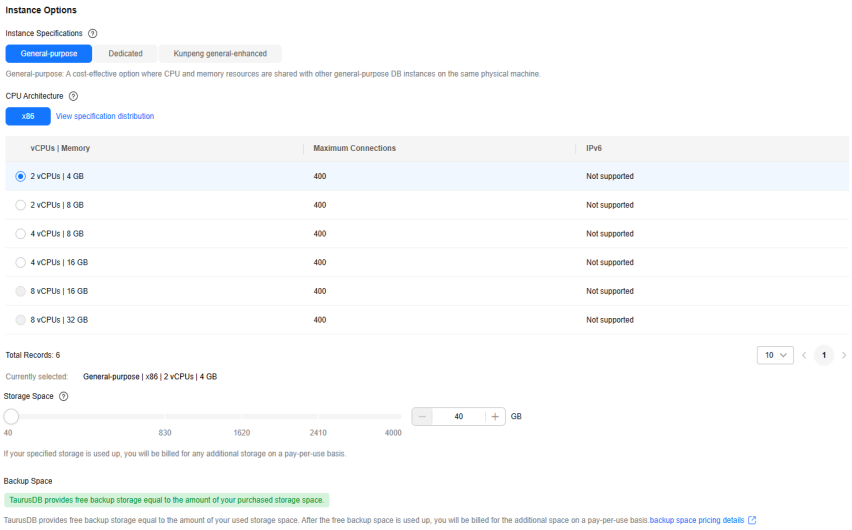


Table 2-2 Specifications and storage

Parameter	Description
Instance Specifications	Refers to the vCPU and memory of a DB instance. Different instance specifications support different numbers of database connections and maximum IOPS.
Storage Space (GB)	Contains the system overhead required for inodes, reserved blocks, and database operation. Storage space can range in size from 40 GB to 4,000 GB and can be scaled up only by a multiple of 10 GB. After a DB instance is created, you can scale up its storage space. For details, see **Scaling up Storage Space**.

- Basic settings and connectivity

Figure 2-3 Basic settings and connectivity

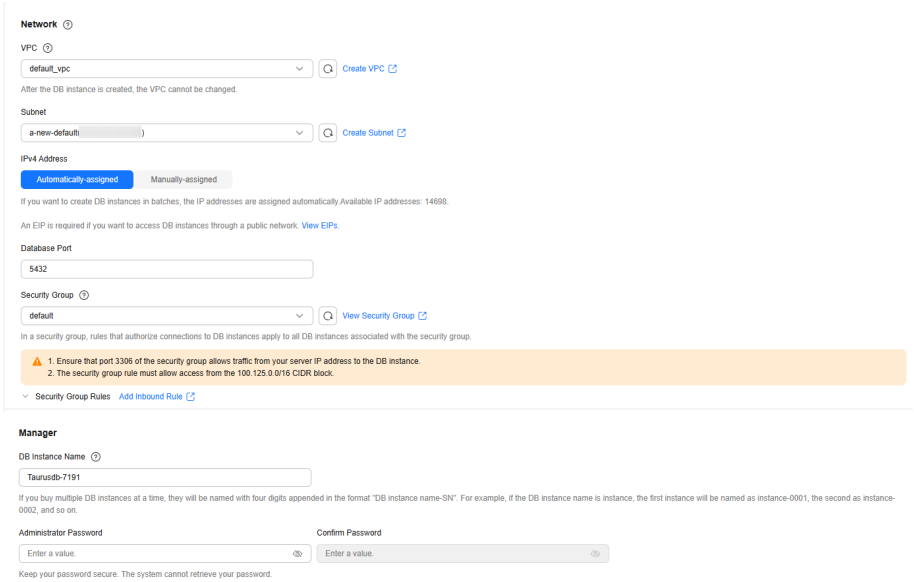


Table 2-3 Network

Parameter	Description
DB Instance Name	The instance name must start with a letter and consist of 4 to 64 characters. Only letters (case-sensitive), digits, hyphens (-), and underscores (_) are allowed.
Password	– **Configure** (default setting): Configure a password for your DB instance during the creation process. – **Skip**: Configure a password later after the DB instance is created. **NOTICE** If you select **Skip** for **Password**, you need to reset the password before you can log in to the instance. After a DB instance is created, you can reset the password. For details, see section **Resetting the Administrator Password to Restore root Access**.
Administrator	The default login name for the database is **root**.

Parameter	Description
Administrator Password	Must consist of 8 to 32 characters and contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters (~!@#$ %^*-_=+?,). Enter a strong password and periodically change it for security reasons. If the password you provide is regarded as a weak password by the system, you will be prompted to enter a stronger password. Keep this password secure. The system cannot retrieve it. After a DB instance is created, you can reset this password. For details, see **Resetting the Administrator Password to Restore root Access**.
Confirm Password	Must be the same as **Administrator Password**.
VPC	The virtual network in which your TaurusDB for PostgreSQL instance is located. A VPC can isolate networks for different workloads. You can select an existing VPC or create a VPC. For details on how to create a VPC, see the "Creating a VPC" section in the *Virtual Private Cloud User Guide*. If no VPC is available, TaurusDB for PostgreSQL allocates a VPC to you by default. To use a shared VPC, select a VPC that another account shares with the current account from the drop-down list. VPC owners can share the subnets in a VPC with one or multiple accounts through **Resource Access Manager (RAM)**. Through VPC sharing, you can easily configure, operate, and manage multiple accounts' resources at low costs. For more information about VPC and subnet sharing, see **VPC Sharing**. **NOTICE** After a TaurusDB for PostgreSQL instance is created, the VPC cannot be changed.
Subnet	Improves network security by providing dedicated network resources that are logically isolated from other networks. Subnets are only valid within a specific AZ. Dynamic Host Configuration Protocol (DHCP) is enabled by default for subnets where you plan to create DB instances and cannot be disabled. A private IPv4 address is automatically assigned when you create a DB instance. You can also enter an idle private IPv4 address within the subnet CIDR block. After the DB instance is created, you can change the private IP address.

Parameter	Description
Security Group	Controls the access that traffic has in and out of a DB instance. By default, the security group associated with the DB instance is authorized. In addition, a network **access control list (ACL)** can help control inbound and outbound traffic of subnets in your VPC. A security group enhances security by controlling access to TaurusDB for PostgreSQL from other services. You need to add inbound rules to a security group so that you can connect to your DB instance. If no security group is available, TaurusDB for PostgreSQL allocates a security group to you by default.

- Advanced settings

Figure 2-4 Advanced settings

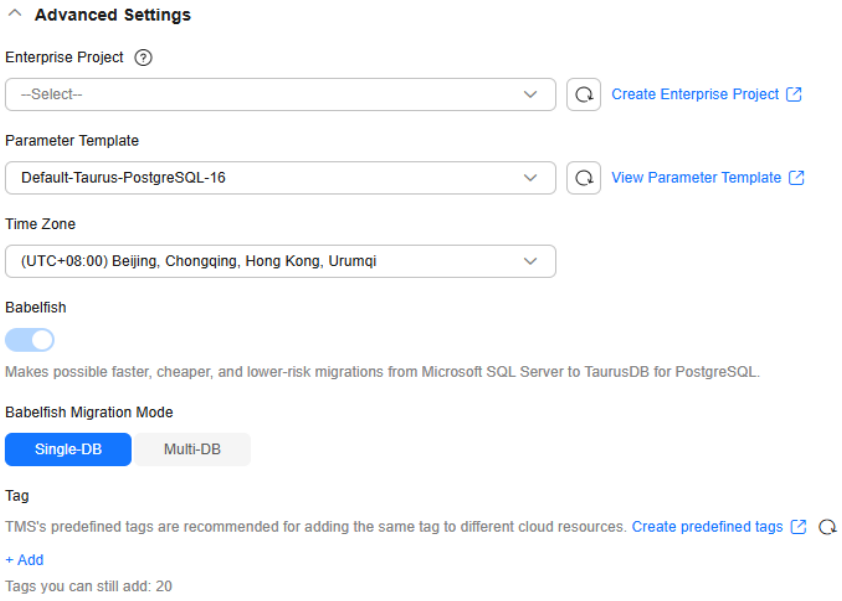


Table 2-4 Advanced settings

Parameter	Description
Enterprise Project	If your account has been associated with an enterprise project, select the target project from the **Enterprise Project** drop-down list. For more information about enterprise projects, see *Enterprise Management User Guide*.

Parameter	Description
Parameter Template	Contains engine configuration values that can be applied to one or more DB instances. If you intend to create primary/standby DB instances, they use the same parameter template. You can modify the instance parameters as required after the DB instance is created. **NOTICE** If you use a custom parameter template when creating a DB instance, the following specification-related parameters in the custom template are not delivered. Instead, the default values are used. – **maintenance_work_mem** – **shared_buffers** – **max_connections** – **effective_cache_size** You can modify the instance parameters as required after the DB instance is created. For details, see **Modifying Parameters of a TaurusDB for PostgreSQL Instance**.
Time Zone	You need to select a time zone for your instance based on the region hosting your instance. You can select a time zone during instance creation and change it later as needed.
Babelfish	Provides a SQL Server syntax compatibility layer, enabling Transact-SQL (T-SQL) applications to run seamlessly in a PostgreSQL environment. This option is enabled by default. Currently, it cannot be disabled.
Babelfish Migration Mode	Babelfish supports **Single-DB** and **Multi-DB** migration modes. – **Single-DB**: Only one T-SQL database can be created in Babelfish, and the T-SQL schema is created as a regular PostgreSQL schema in the Babelfish database. – **Multi-DB**: Multiple T-SQL databases can be created in Babelfish (each database has its own schema). The T-SQL schema is created as a PostgreSQL schema (*<database_name>_<schema_name>*) to avoid name conflicts. **CAUTION** After an instance with Babelfish enabled is created, to ensure access to all previously created SQL objects, you are advised not to change the migration mode. If you need to change the migration mode, create a new instance.

Parameter	Description
Tag	Tags a DB instance. This parameter is optional. Adding tags helps you better identify and manage your DB instances. Up to 20 tags can be added for each DB instance. If your organization has configured tag policies for TaurusDB for PostgreSQL, add tags to DB instances based on the policies. If a tag does not comply with the policies, DB instance creation may fail. Contact your organization administrator to learn more about tag policies.

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

Step 3 Confirm the specifications for pay-per-use DB instances.

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

Step 4 To view and manage DB instances, go to the **Instances** page.

- When your DB instance is being created, the status is **Creating**. The status changes to **Available** after the instance is created.
- The automated backup policy is enabled by default. You can change it after the DB instance is created. An automated full backup is immediately triggered once your DB instance is created.
- After a DB instance is created, you can enter a description for it.
- The default database port is **5432**. You can change it after the DB instance is created. For details, see **Changing a Database Port**.

----End

3 Instance Connection

3.1 Overview

Before connecting to a DB instance, you must create one first. For details about how to create a DB instance, see **Buying a TaurusDB for PostgreSQL Instance**. You can connect to a TaurusDB for PostgreSQL instance through a command-line interface (CLI) or graphical user interface (GUI).

Connecting to a DB Instance over a Private Network Using CLI

Table 3-1 lists how to use CLI to connect to a TaurusDB for PostgreSQL instance over a private network.

Table 3-1 Connecting to a DB instance over a private network

Connection Method	IP Address	Security Group Rule	Description
Private network	Private IP address	• If the ECS and TaurusDB for PostgreSQL instance are in the same security group, they can communicate with each other over a private network by default. No security group rules need to be configured. • If the ECS and TaurusDB for PostgreSQL instance are in different security groups, configure security group rules for the ECS and TaurusDB for PostgreSQL instance, respectively. – TaurusDB for PostgreSQL instance: Configure an **inbound rule** for the security group associated with the TaurusDB for PostgreSQL instance. For details, see **Configuring Security Group Rules**. – ECS: The default security group rule allows all outbound data packets. In this case, you do not need to configure a security group rule for the ECS. If **not all outbound traffic** is allowed in the security group, you need to configure an **outbound rule** for the ECS to allow all outbound packets.	• Secure and high-performance • Recommended

Connection Methods

Table 3-2 Connection methods

Connection Method	Description
Connecting to a TaurusDB for PostgreSQL Instance Through the psql CLI Client	In Linux, you need to install a PostgreSQL client on the ECS and connect to the instance through the psql CLI. A private IP address is provided by default. When your applications are deployed on an ECS that is in the same region and VPC as the TaurusDB for PostgreSQL instance, you are advised to use a private IP address to connect to the instance through the ECS.
• **Connecting to a TaurusDB for PostgreSQL Instance Through pgAdmin** • **Connecting to a TaurusDB for PostgreSQL Instance Through sqlcmd** • **Connecting to a TaurusDB for PostgreSQL Instance Through SSMS**	In Windows, you can use the pgAdmin, sqlcmd, or SSMS client to connect to a TaurusDB for PostgreSQL instance.

3.2 Connecting to a TaurusDB for PostgreSQL Instance Through the psql CLI Client

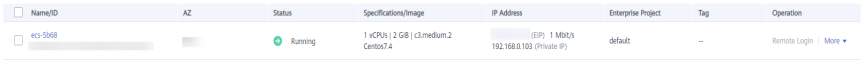
3.2.1 Connecting to a DB Instance from a Linux ECS over a Private Network

You can connect to your DB instance using a Linux ECS installed with a PostgreSQL client over a private network.

Step 1: Buy an ECS

1. **Log in to the management console** and check whether there is an ECS available.
   - If there is a Linux ECS, go to **3**.
   - If no Linux ECS is available, go to **2**.

   Figure 3-1 ECS

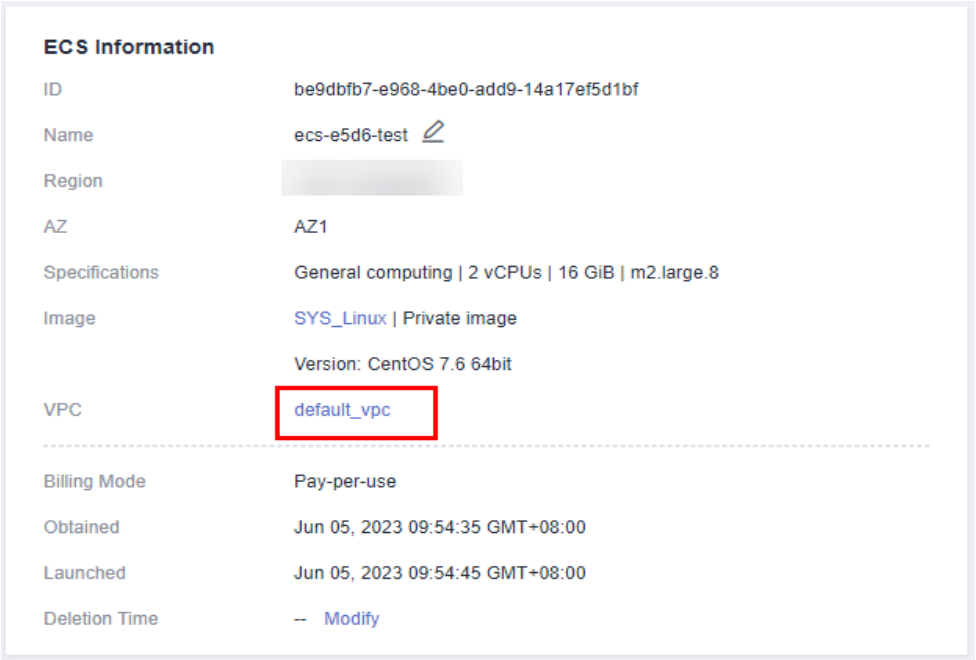
2. Buy an ECS and select Linux (for example, CentOS) as its OS.

   To download a PostgreSQL client to the ECS, bind an EIP to the ECS. The ECS must be in the same region, VPC, and security group as the TaurusDB for PostgreSQL instance for mutual communications.

   For details about how to purchase a Linux ECS, see **Purchasing an ECS in Custom Config Mode** in *Elastic Cloud Server User Guide*.

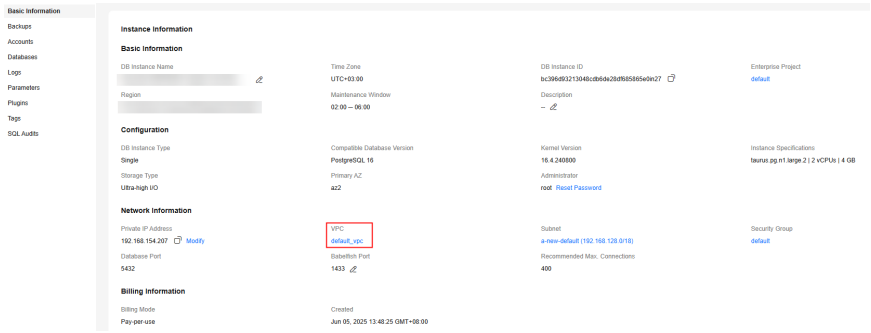
3. On the **ECS Information** page, check the region and VPC of the ECS.

   Figure 3-2 ECS information

4. On the **Basic Information** page of the TaurusDB for PostgreSQL instance, check the region and VPC of the instance.

Figure 3-3 Checking the region and VPC



5. Check whether the ECS and the TaurusDB for PostgreSQL instance are in the same region and VPC.
   - If they are in the same region and VPC, go to **Step 2: Test Connectivity and Install the PostgreSQL Client**.
   - If they are not in the same region, purchase another ECS or DB instance. The ECS and DB instance in different regions cannot communicate with each other. To reduce network latency, deploy your DB instance in the region nearest to your workloads.
   - If the ECS and DB instance are in different VPCs, change the VPC of the ECS to that of the DB instance. For details, see **Changing a VPC**.

Step 2: Test Connectivity and Install the PostgreSQL Client

1. Log in to the ECS. For details, see **Logging In to a Linux ECS Using VNC** in *Elastic Cloud Server User Guide*.
2. On the **Instances** page of the TaurusDB console, click the instance name to go to the **Basic Information** page.
3. In the **Network Information** area, obtain the private IP address and database port of the DB instance.

   Figure 3-4 Obtaining the private IP address and database port

4. On the ECS, check whether the private IP address and database port of the DB instance can be connected.

   ```
   curl -kv 192.168.0.7:5432
   ```

   - If yes, network connectivity is normal.
   - If no, check the security group rules.
     - If in the security group of the ECS, there is no outbound rule with **Destination** set to **0.0.0.0/0** and **Protocol & Port** set to **All**, add an outbound rule for the private IP address and port of the DB instance.
     - If in the security group of the DB instance, there is no inbound rule allowing the access from the private IP address and port of the ECS,

add an inbound rule for the private IP address and port of the ECS. For details, see **Configuring Security Group Rules**.

5. Install the PostgreSQL client.

   Installation from source code: This installation method has no restrictions on TaurusDB for PostgreSQL instance versions and ECS OS types.

   The following uses an ECS using the Huawei Cloud EulerOS 2.0 image as an example to describe how to install a PostgreSQL 16.4 client.

   Figure 3-5 Checking the ECS image

   a. To use SSL, download OpenSSL to the ECS in advance.
   ```
   sudo yum install -y openssl-devel
   ```

   b. Obtain the **code download link**, run **wget** to download the installation package to the ECS, or **download the installation package** to the local PC and then upload it to the ECS.
   ```
   wget https://ftp.postgresql.org/pub/source/v16.4/postgresql-16.4.tar.gz
   ```

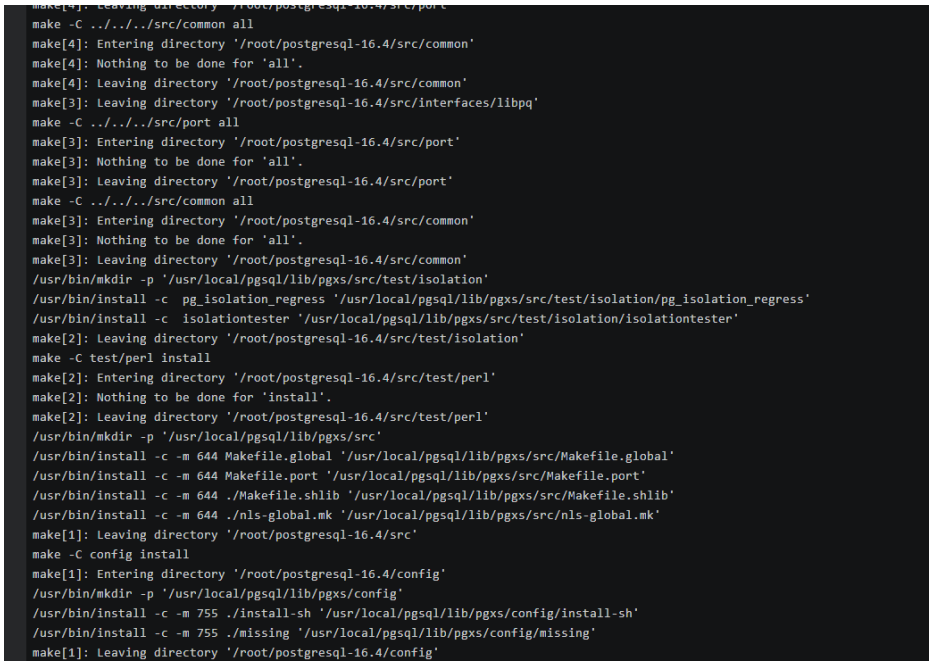
   c. Decompress the installation package.
   ```
   tar xf postgresql-16.4.tar.gz
   ```

   d. Compile the source code and then install the client.
   ```
   cd postgresql-16.4
   ./configure --without-icu --without-readline --without-zlib --with-openssl
   make -j 8 && make install
   ```

   NOTE

   If **--prefix** is not specified, the default path is **/usr/local/pgsql**. The client can be installed in the simplest way.

Figure 3-6 Compilation and installation



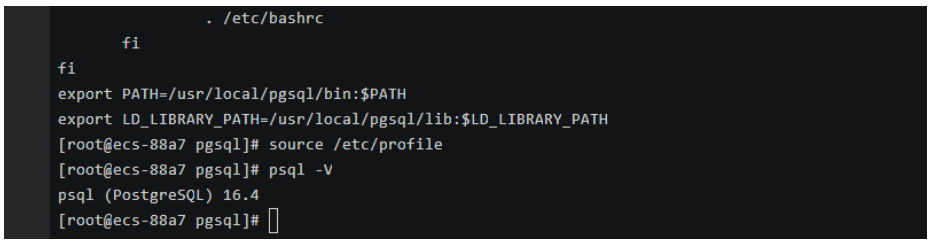
e. Add the following code to the **/etc/profile** file to configure environment variables:

```
export PATH=/usr/local/pgsql/bin:$PATH
export LD_LIBRARY_PATH=/usr/local/pgsql/lib:$LD_LIBRARY_PATH
source /etc/profile
```

f. Test whether the psql is available.

```
psql -V
```

Figure 3-7 Testing psql



Step 3: Connect to the DB Instance Using a CLI (Non-SSL Connection)

1. Run the following command on the ECS to connect to the DB instance:

   psql --no-readline -h *<host>* **-p** *<port>* **"dbname=***<database>* **user=***<user>***"**

   Example:

   psql --no-readline -h 192.168.0.7 -p 5432 "dbname=postgres user=root"

Table 3-3 Parameter description

Parameter	Description
<host>	Private IP address obtained in **3**.

Parameter	Description
<port>	Database port obtained in 3. The default value is **5432**.
<database>	Name of the database to be connected. The default database name is **postgres**.
<user>	Administrator account **root**.

2. Enter the password of the database account as prompted.
   ```
   Password:
   ```
   If no error message is displayed, the connection is successful.

3.3 Connecting to a TaurusDB for PostgreSQL Instance Through pgAdmin

pgAdmin is an administration and development tool for PostgreSQL. Using pgAdmin, you can connect to specific databases from your clients, create tables, and run simple and complex SQL statements. pgAdmin can be used on Windows, Linux, macOS, and other operating systems. The latest version of pgAdmin is based on the browser/server (B/S) architecture. For more information, see the **pgAdmin documentation**.

This section uses pgAdmin 4-4.17 as an example to describe how to use pgAdmin to connect to a TaurusDB for PostgreSQL instance and create databases and tables. All the following database operations are performed by connecting to the TaurusDB PostgreSQL instance through the PostgreSQL port.

NOTICE

The pgAdmin version must be 4 or later.

Procedure

Step 1 Obtain the pgAdmin installation package.

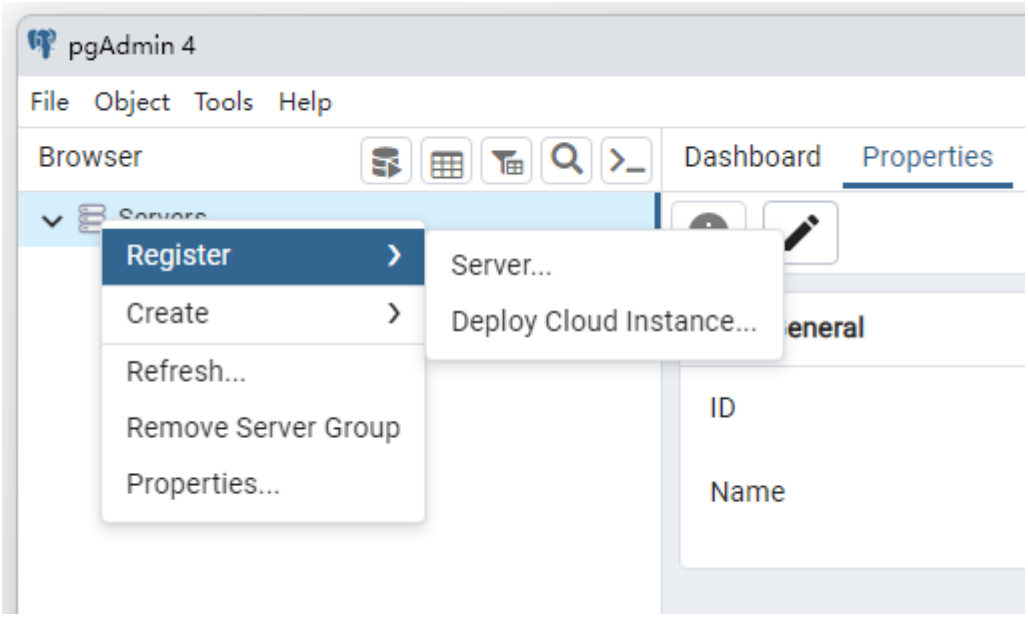
Download the pgAdmin installation package for Windows from the **pgAdmin official website**.

Step 2 Double-click the installation package and complete the installation as instructed.

Step 3 Start the pgAdmin client after the installation.

Step 4 In the displayed login window, right-click **Servers** and choose **Register** > **Server** from the shortcut menu.

Figure 3-8 Login information



Step 5 On the **General** page, specify **Name**. On the **Connection** page, specify information about the TaurusDB for PostgreSQL instance to be connected. Then, click **Save**.

Figure 3-9 Entering basic information

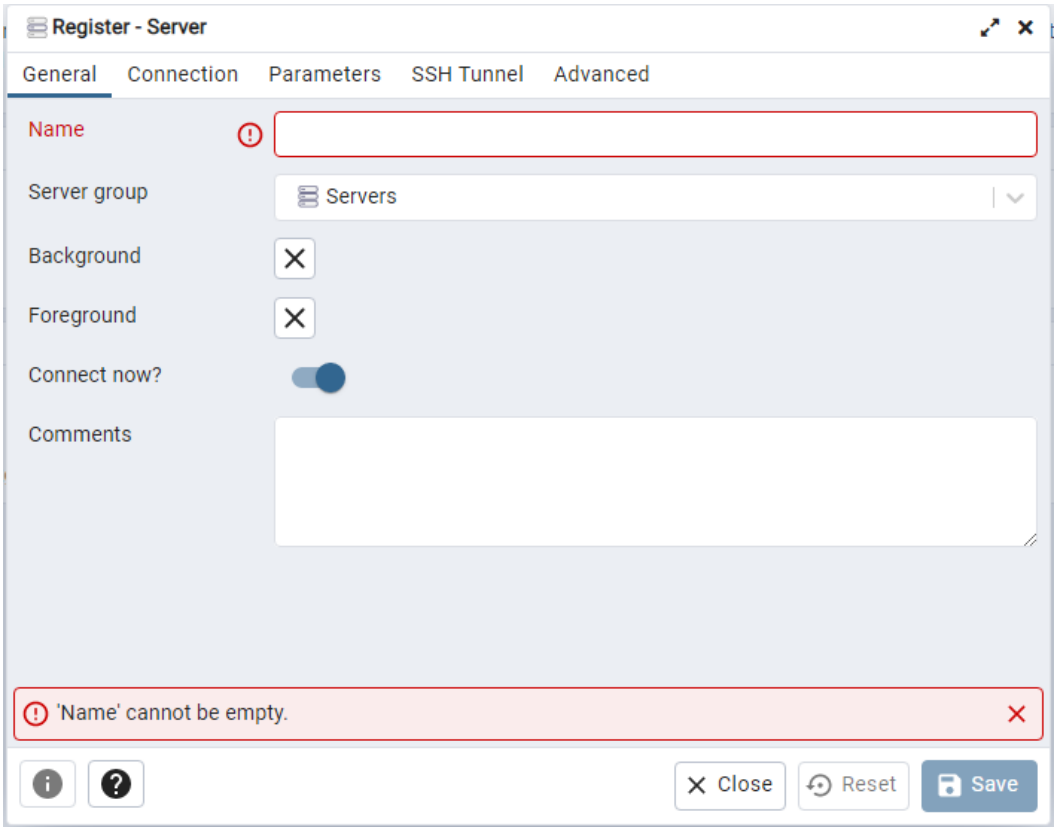
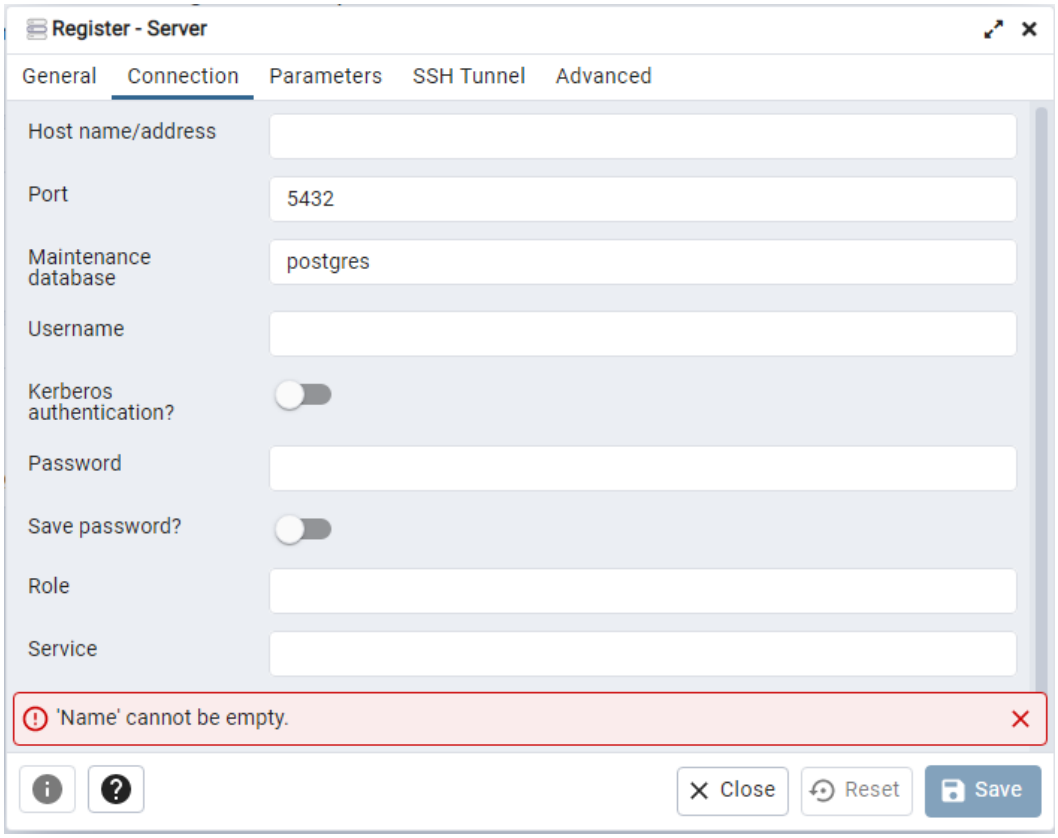


Figure 3-10 Entering connection information



Parameter description:

- **Host name/address**: indicates the private IP address of the TaurusDB for PostgreSQL instance.
- **Port**: indicates the database port. By default, the value is **5432**.
- **Username**: indicates the username. By default, the value is **root**.
- **Password**: indicates the password for accessing the TaurusDB for PostgreSQL instance.

Step 6 In the login window, check that the connection information is correct. The target DB instance is successfully connected.

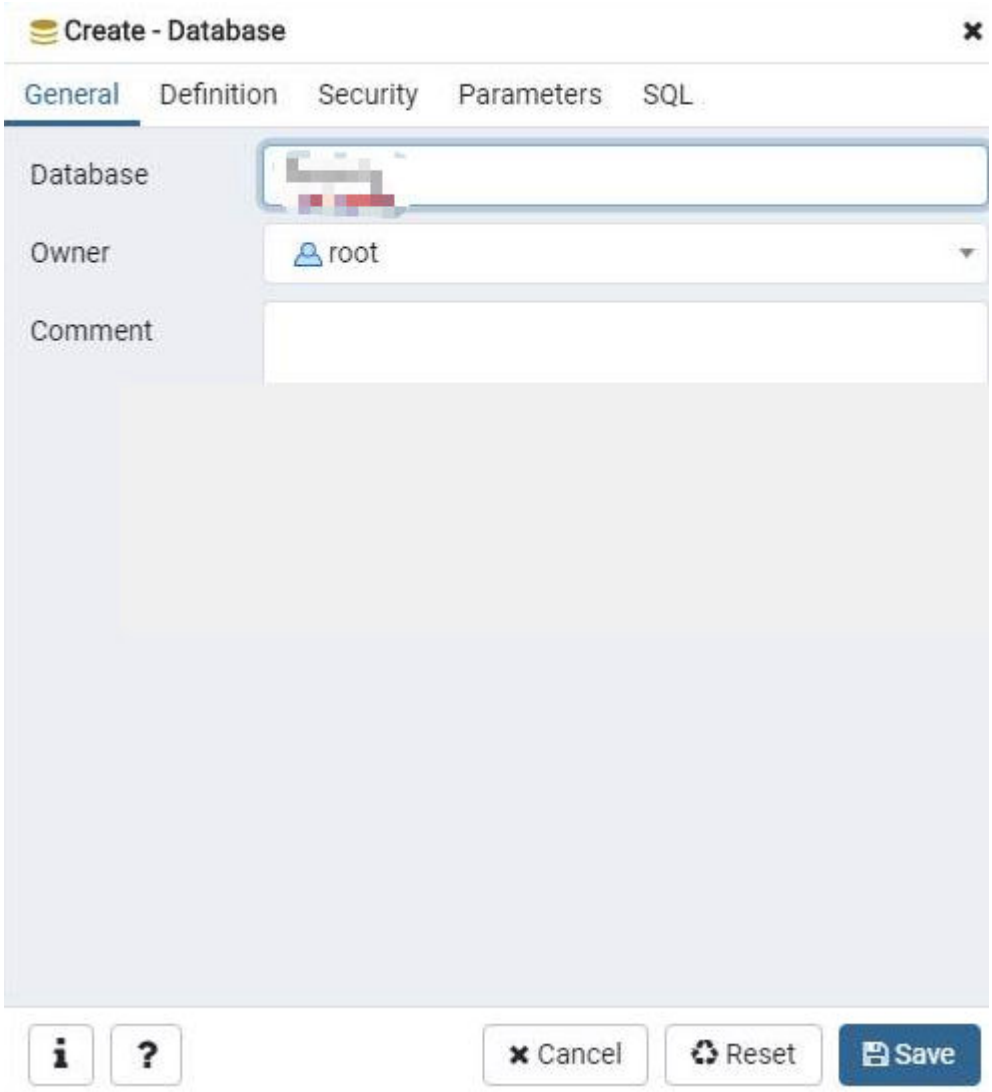
----End

Creating a Database

Step 1 In the navigation pane on the left of pgAdmin, right-click the target instance node and choose **Create** > **Database** from the shortcut menu.

Step 2 On the **General** tab, specify **Database** and click **Save**.

Figure 3-11 Creating a database



----End

Creating a Table

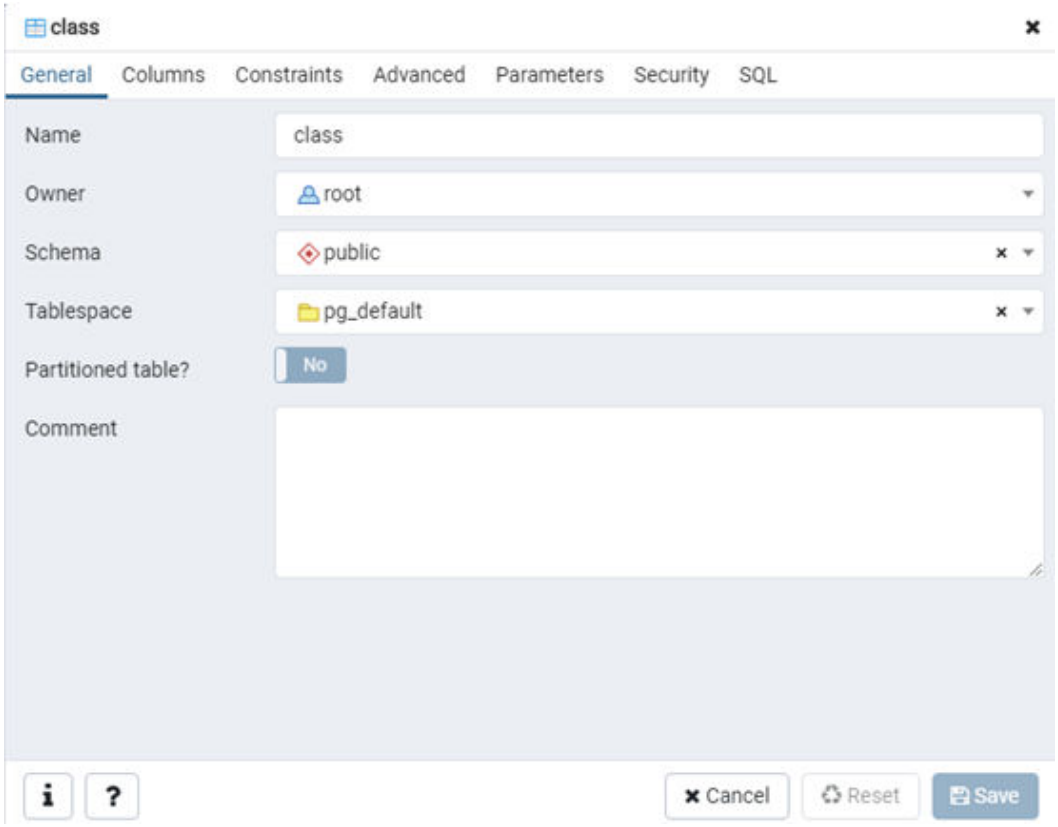
Step 1 Access the created database. In the navigation pane on the left, right-click **Tables** and choose **Create** > **Table** from the shortcut menu.

NOTE

Create tables in the schema of the database created by the current user.

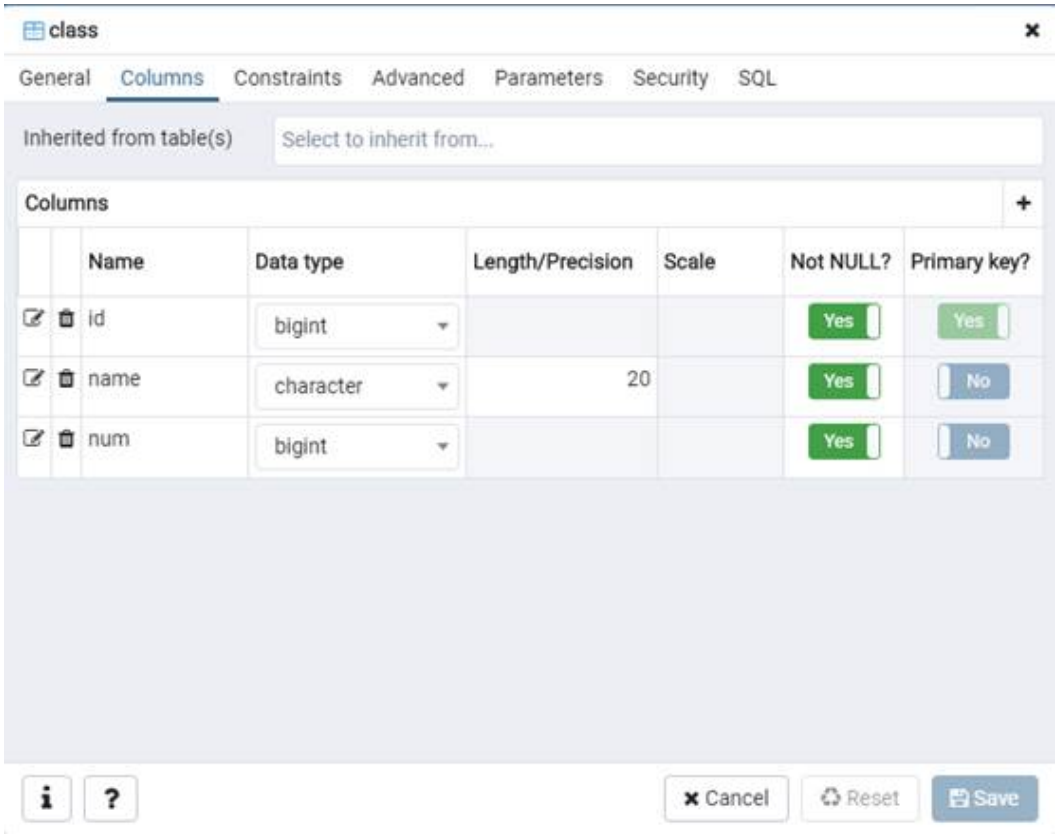
Step 2 On the **General** tab, enter required information and click **Save**.

Figure 3-12 Basic information



Step 3 On the **Columns** tab, add table columns and click **Save**.

Figure 3-13 Adding table columns



----End

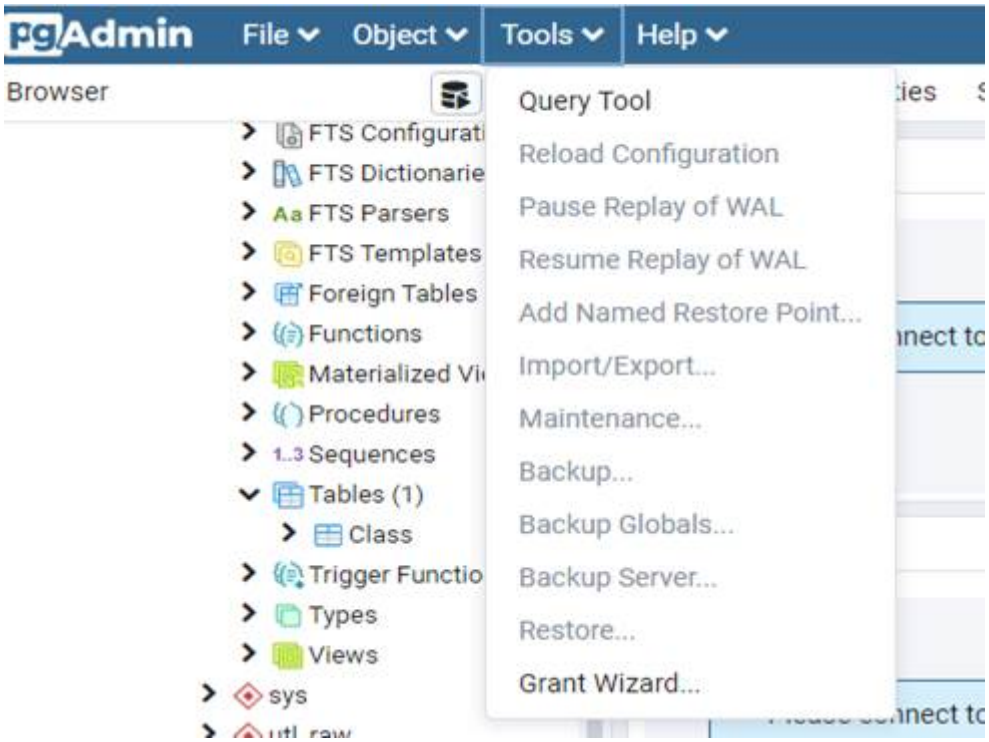
Executing SQL Statements

On the top menu bar, choose **Tools** > **Query Tool**. The SQL CLI is displayed.

NOTE

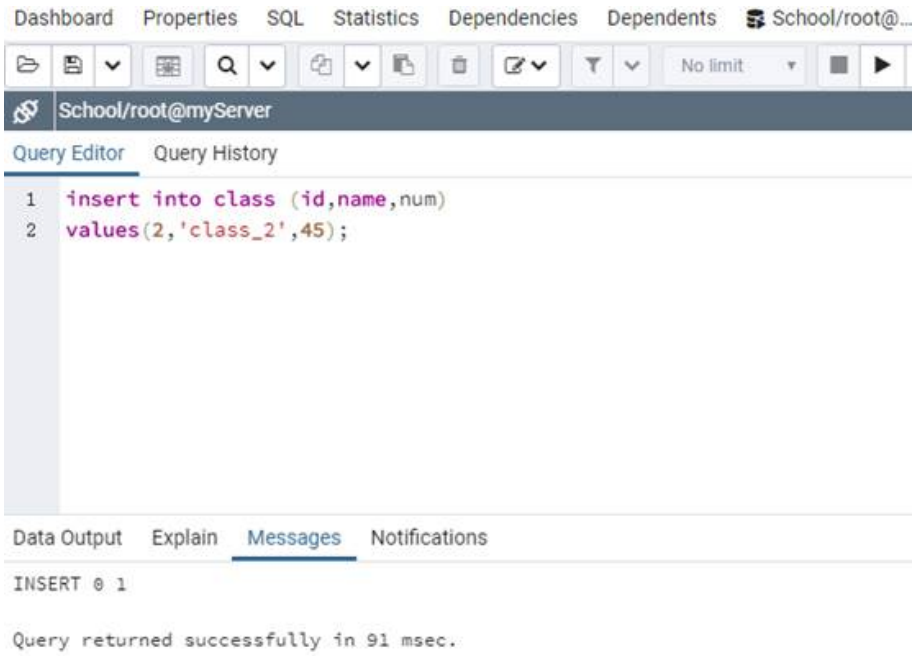
Take care when adding, deleting, or modifying data in the instance. Improper operations can cause instance or workload exceptions.

Figure 3-14 SQL execution



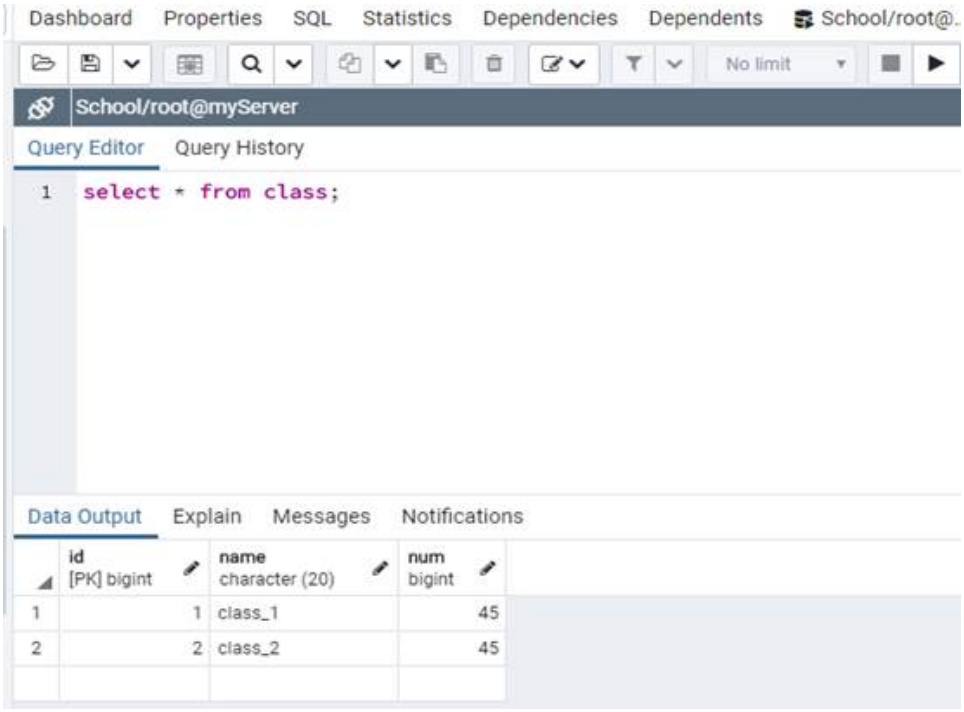
- Enter an **INSERT** command and click **Execute** to insert data into the table.

Figure 3-15 Inserting data



- Enter an **SELECT** command and click **Execute** to query data in the table.

Figure 3-16 Querying data



Viewing Monitoring Data

In the navigation pane on the left, select a database and click the **Dashboard** tab on the right pane to view database metrics, including **Database sessions**, **Transactions per second**, **Tuples in**, **Tuples out**, and **Block I/O**.

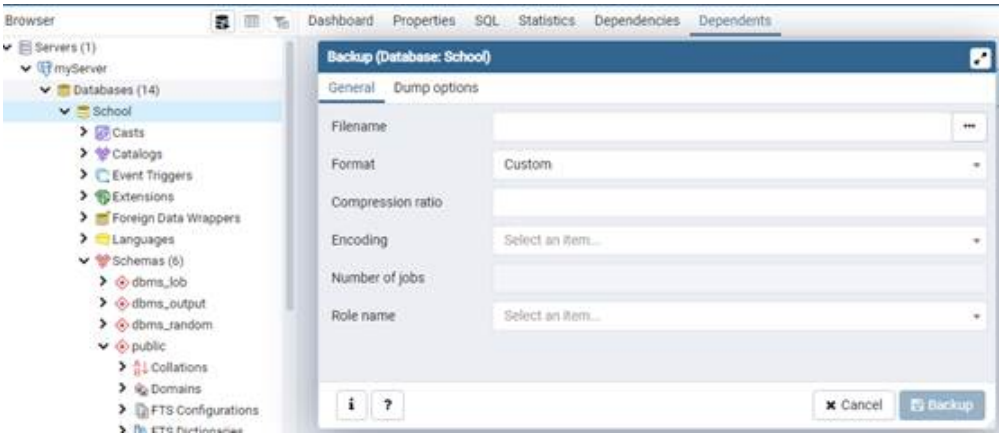
NOTE

Cloud Eye monitors operating statuses of DB instances. You can view the DB instance metrics on the management console.

Backing Up Data

1. In the navigation pane on the left, right-click the database to be backed up and choose **Backup** from the shortcut menu.
2. On the **General** tab, enter required information and click **Backup**.

Figure 3-17 Backing up data



Restoring Data

1. In the navigation pane on the left, right-click the database to be restored and choose **Restore** from the shortcut menu.
2. On the displayed tab in the right pane, select a file name and click **Restore**.

Figure 3-18 Restoring data

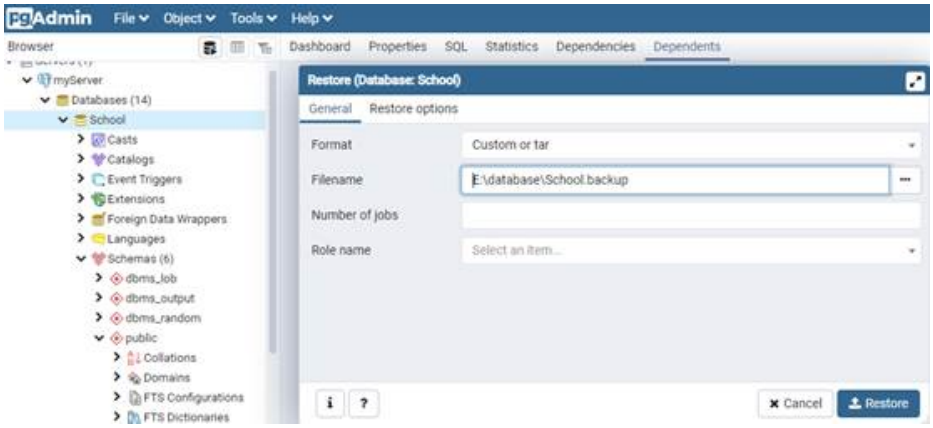
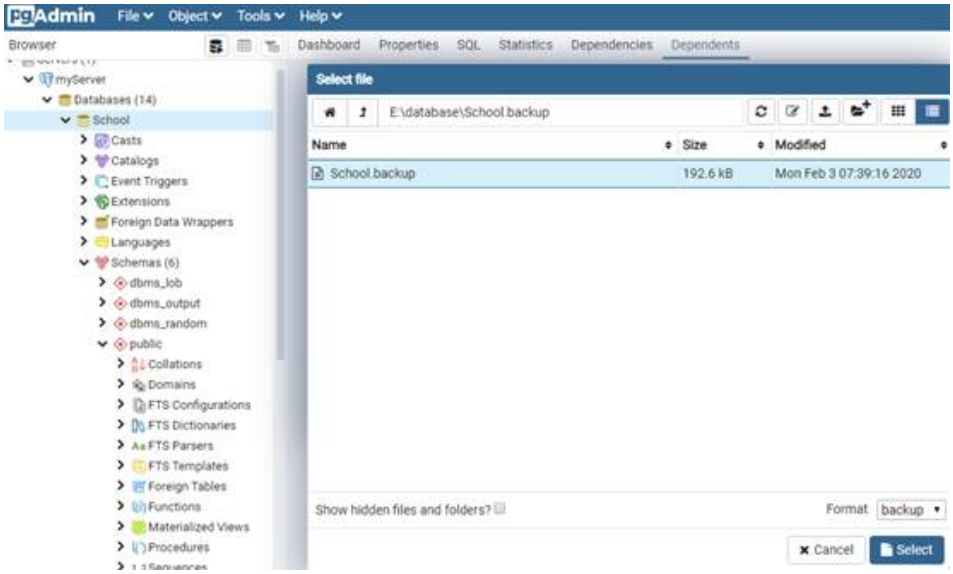


Figure 3-19 Restoration completed



3.4 Connecting to a TaurusDB for PostgreSQL Instance Through sqlcmd

sqlcmd is a command-line tool that allows you to run T-SQL statements, system procedures, and scripts in SQL Server databases.

Prerequisites

- You have purchased a TaurusDB for PostgreSQL instance.
- You have enabled Babelfish.

Procedure

Step 1 Install the sqlcmd CLI client.

For details, see the SQL Server official documentation (**Download and install sqlcmd**).

Step 2 Use sqlcmd to run the following command to connect to a database:

```
sqlcmd -S <host>,<port> -U <login> -P <password> -d <database>
```

Table 3-4 Parameter description

Parameter	Description
<host>	Private IP address of an instance.
<port>	Database port. The default value is **1433**.
<database>	Name of the Babelfish database to be connected.
<login>	Login name of the Babelfish user.
<password>	Password associated with the user.

----End

3.5 Connecting to a TaurusDB for PostgreSQL Instance Through SSMS

SQL Server Management Studio (SSMS) is an integrated environment for managing SQL Server databases. It provides a range of tools for configuring, monitoring, and managing SQL Server instances. It can be used to compile queries, deploy components at the data layer, and manage databases.

This section uses SSMS 19.0.2 as an example to describe how to use SSMS to connect to a TaurusDB for PostgreSQL instance.

Prerequisites

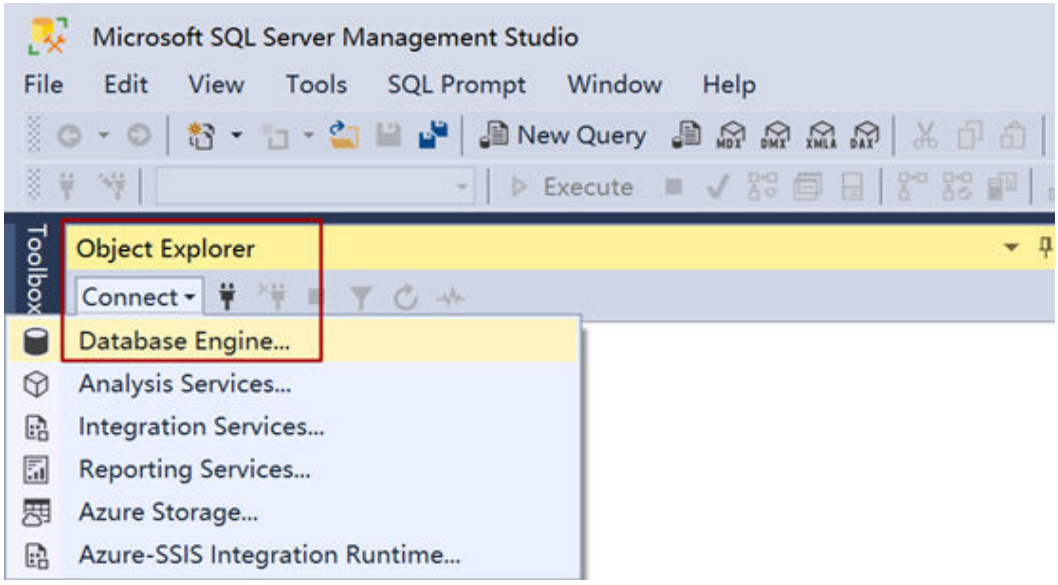
- You have purchased a TaurusDB for PostgreSQL instance.
- You have enabled Babelfish.

Procedure

Step 1 Start SSMS.

Step 2 When you start SSMS for the first time, the **Connect to Server** window is displayed. Skip this step.

If the window is not displayed, select **Connect** in **Object Explorer** and click **Database Engine** to open it.



Step 3 In the login window, set parameters and click **Connect**.

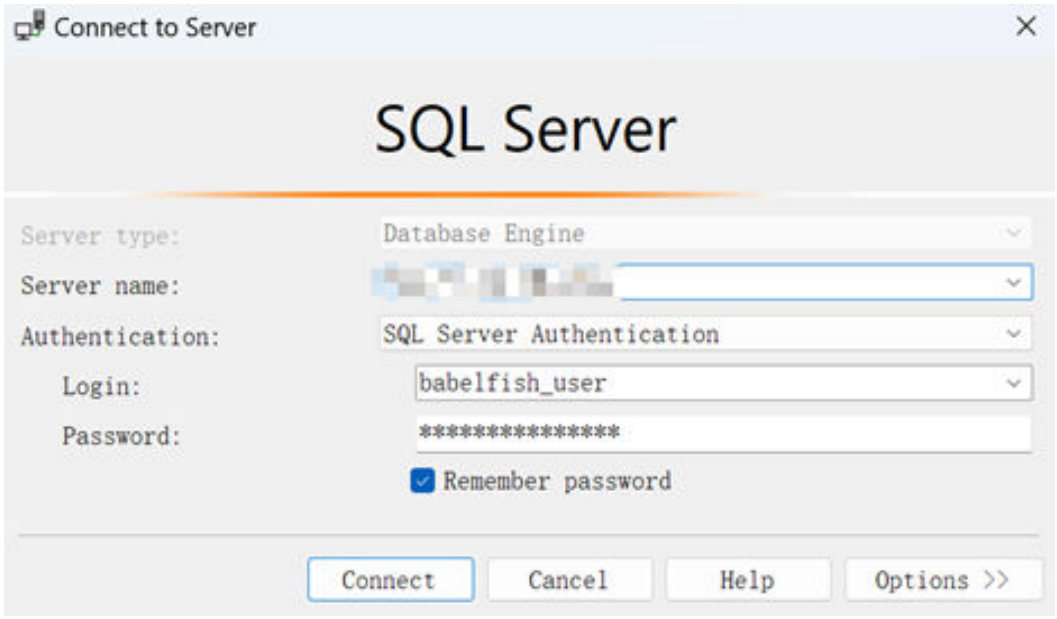


Table 3-5 Login parameters

Parameter	Description
Server type	The value defaults to **Database Engine**.
Server name	Private IP address of an instance.
Authentication	The value defaults to **SQL Server Authentication**.
Login	Login name of the Babelfish user.
Password	Password associated with the Babelfish user.

Step 4 Check whether the instance has been connected. If the connection information is correct, the instance will be connected.

----End

3.6 Connection Information Management

3.6.1 Changing a Database Port

Scenarios

After Babelfish is enabled, TaurusDB for PostgreSQL listens on two ports. The default port of a PostgreSQL database is **5432**, and that of a Babelfish database is **1433**.

This section describes how to change the database port of a primary DB instance. For primary/standby DB instances, changing the database port of the primary DB instance will cause the database port of the standby DB instance to also be changed.

If specific security group rules have been configured for a DB instance, you need to change the inbound rules of the security group to which the DB instance belongs after changing the database port.

Procedure

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the primary instance name.

Step 5 On the **Basic Information** page, find **Database Port** and click **Configure** under it.

TaurusDB for PostgreSQL instances can use database ports 1433 and 2100 to 9500.

Step 6 In the displayed dialog box, enter a new port and click **OK**.

- Changing the primary DB instance's port updates the standby DB instance's port as well and causes both of them to reboot.
- This process takes 1 to 5 minutes.

Step 7 If you have enabled operation protection, click **Send Code** in the displayed **Identity Verification** dialog box and enter the obtained verification code. Then, click **OK**.

Two-factor authentication improves the security of your account and cloud product. For details about how to enable operation protection, see the *Identity and Access Management User Guide*.

Step 8 View the results on the **Basic Information** page.

----End

3.6.2 Configuring Security Group Rules

Scenarios

A security group is a collection of access control rules for ECSs and TaurusDB for PostgreSQL instances that have the same security requirements and are mutually trusted within a VPC.

To ensure database security and reliability, you need to configure security group rules to allow only specific IP addresses and ports to access TaurusDB for PostgreSQL instances.

- When you attempt to connect to a TaurusDB for PostgreSQL instance through a private network, check whether the ECS and the TaurusDB for PostgreSQL instance are in the same security group.
  - If yes, they can communicate with each other by default. No security group rules need to be configured.
  - If no, you need to configure security group rules for them, separately.
    - TaurusDB for PostgreSQL instance: Configure an **inbound rule** for the security group associated with the TaurusDB for PostgreSQL instance.
    - ECS: The default security group rule allows all outbound data packets. In this case, you do not need to configure a security group rule for the ECS. If **not all outbound traffic** is allowed in the security group, you need to configure an **outbound rule** for the ECS to allow all outbound packets.

This section describes how to configure an inbound rule for a TaurusDB for PostgreSQL instance.

For details about the requirements of security group rules, see **Adding a Security Group Rule** in *Virtual Private Cloud User Guide*.

Precautions

The default security group rule allows all outbound data packets. ECSs and TaurusDB for PostgreSQL instances can access each other if they are deployed in the same security group. After a security group is created, you can configure security group rules to control access to and from TaurusDB for PostgreSQL instances associated with that security group.

- By default, you can create up to 100 security groups in your cloud account.
- By default, you can add up to 50 security group rules to a security group.
- A security group can be associated with multiple TaurusDB for PostgreSQL instances.
- Too many security group rules will increase the first packet latency. You are advised to create up to 50 rules for each security group.

- To enable access to a TaurusDB for PostgreSQL instance from resources outside the security group, you need to configure an **inbound rule** for the security group associated with the TaurusDB for PostgreSQL instance.

NOTE

To ensure data and instance security, use permissions properly. You are advised to use the minimum access permission, change the default database port **5432**, and set the accessible IP address to the remote server's address or the remote server's minimum subnet address to control the access scope of the remote server.

The default value of **Source** is **0.0.0.0/0**, indicating that all IP addresses can access the TaurusDB for PostgreSQL instance as long as they are associated with the same security group as the instance.

For details about the requirements of security group rules, see **Adding a Security Group Rule** in *Virtual Private Cloud User Guide*.

Procedure

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name to go to the **Basic Information** page.

Step 5 In the **Network Information** area, click the security group name under **Security Group**.

Figure 3-20 Security Group

Step 6 On the **Inbound Rules** tab, click **Add Rule**. In the displayed dialog box, set required parameters and click **OK**.

To add more inbound rules, click .

----End

4 Database Usage

4.1 Suggestions on Using TaurusDB for PostgreSQL

4.1.1 Instance Usage Suggestions

Database Connection

TaurusDB for PostgreSQL uses a process architecture, providing a backend service process for each client connection.

- Set **max_connections** depending on the type of your application. Use the parameter settings provided on pgtune as examples:
  - Set **max_connections** to **200** for web applications.
  - Set **max_connections** to **300** for OLTP applications.
  - Set **max_connections** to **40** for data warehouses.
  - Set **max_connections** to **20** for desktop applications.
  - Set **max_connections** to **100** for hybrid applications.
- Limit the maximum number of connections allowed for a single user based on workload requirements.
  ```
  ALTER ROLE xxx CONNECTION LIMIT xxx;
  ```
- Set the number of active connections to two to three times the number of vCPUs.
- Avoid long transactions, which may block autovacuum and affect database performance.
- Periodically release persistent connections because maintaining them may generate a large cache and use up memory. You can configure parameters such as **idle_session_timeout** and **idle_in_transaction_session_timeout** to release idle connections.
- Check the application framework to prevent the application from automatically starting transactions without performing any operations.

Reliability and Availability

- Select primary/standby DB instances for production databases.
- Keep vCPU, memory, and storage usage less than 85% for production databases to prevent problems such as out of memory (OOM) and full storage.
- Deploy primary and standby instances in different AZs to improve availability.
- Set the time window for automated backup to off-peak hours. Do not disable full backup.
- Configure asynchronous replication between primary and standby DB instances to prevent workloads on the primary instance from being blocked due to a fault on the standby instance.
- Pay attention to the size of temporary files and the generation rate. Too many temporary files affect database performance, slow down database startup, and even cause service unavailability.
- Do not create too many objects in one instance. Generally, the number of tables in a single instance does not exceed 20,000, and that in a single database does not exceed 4,000. This prevents service unavailability caused by long-time table file scanning during database startup.

Logical Replication

- Keep the name of a logical replication slot less than 40 bytes to prevent full backup failures.
- Delete replication slots that are no longer used for logical replication to prevent database bloat.
- Replication slots will be lost after a primary/standby switchover is performed. When this occurs, you need to create replication slots again.
- Use failover slots for TaurusDB for PostgreSQL to prevent replication slot loss after a primary/standby switchover or instance reboot.
- When using logical replication, avoid long transactions and commit discarded two-phase transactions in a timely manner to prevent stacked WAL logs from occupying too much storage space.
- When using logical replication, avoid using a large number of sub-transactions (such as savepoints and exception clauses in a transaction) to prevent high memory usage.
- When using Data Replication Service (DRS) to synchronize or migrate data, delete the logical replication slots contained in the databases that are rarely accessed or add heartbeat tables to periodically push the replication slots to prevent stacked WAL logs.

Database Age

- Definition of database age:
  - Database age refers to the latest transaction ID minus the oldest transaction ID in the database.
  - The maximum age allowed for a database is 2 billion transactions old. When a database reaches the maximum age, it will be forcibly shut down. In this case, contact technical support to vacuum the database.

  - To view the age of a database, run the following SQL statement:

    select datname, age(datfrozenxid) from pg_database;

- You are advised to use the **db_max_age** metric to monitor the database age and set the alarm threshold to 1 billion.

Stability

- Commit or roll back two-phase transactions in a timely manner to prevent database bloat.
- Change the table structure, for example, adding fields or indexes, during off-peak hours.
- To create indexes during peak hours, use the CONCURRENTLY syntax to avoid blocking the DML of the table.
- Before modifying the structure of a table during peak hours, perform a verification test to prevent the table from being rewritten.
- Configure a lock wait timeout duration for DDL operations to avoid blocking operations on related tables.
- Partition your database if its capacity exceeds 2 TB.
- If a frequently accessed table contains more than 20 million records or its size exceeds 10 GB, split the table or create partitions.
- To prevent replication exceptions on the standby instance, control the data write speed of the primary instance under 50 MB/s. That's because the standby instance replays WAL logs in a single process at a maximum speed of 50 MB/s to 70 MB/s.

Routine O&M

- Periodically download and view slow query logs on the **Logs** page to identify and resolve performance issues in a timely manner.
- Periodically check the resource usage of your instance. If the service pressure fluctuates greatly, you are advised to configure resource alarms. High write pressure will slow down database reboots and affect service availability.
- Run the **SELECT** statement before deleting or modifying a record.
- After a large amount of data is deleted or updated in a table, run VACUUM on the table.
- Note the number of available replication slots and ensure that at least one replication slot is available for database backup.
- Remove any replication slots that are no longer used to prevent the replication slots from blocking log reclaiming.
- Do not use unlogged tables because data in these tables will be lost after a database exception (such as OOM or underlying faults) or primary/standby switchover.
- Do not run VACUUM FULL on system catalogs. If necessary, run VACUUM. Running VACUUM FULL on system catalogs causes the instance to reboot and the instance cannot be connected for a long time.

Security

- Avoid enabling access to your database from the Internet. If you do need to enable Internet access, bind an EIP to your DB instance and configure a whitelist.
- Use SSL to connect to your DB instance.

4.1.2 Database Usage Suggestions

Naming

- The names of objects (such as databases, tables, and indexes) must be no more than 63 bytes. Note that some characters may occupy multiple bytes.
- Do not use reserved database keywords in object names or start an object name with pg or a digit.
- A database name can contain 1 to 63 characters. Only letters, digits, and underscores (_) are allowed. It cannot start with **pg** or a digit and cannot be the same as TaurusDB for PostgreSQL system or template database names, which cannot be changed. The system database of TaurusDB for PostgreSQL is **postgres**, and the template databases are **template0** and **template1**.

Table Design

- The table structure must be designed in advance to avoid frequent structure changes, such as adding fields or changing data types.
- There cannot be more than 64 fields in a single table.
- Create partitioned tables for the tables whose data needs to be deleted periodically. For example, you can create partitions by time and delete data from the partitions using DROP or TRUNCATE.
- Use appropriate data types for table fields. For example, do not use the character type for numeric or date data.
- When using the numeric data type, ensure that the values are within allowed ranges and meet precision requirements.

Index Design

- Design primary keys or unique keys for tables that require logical replication.
- When creating a foreign key, specify the action for deleting or updating the foreign key, for example, ON DELETE CASCADE.
- Create indexes for fields that are frequently used (such as fields for data query and sorting).
- Create partial indexes for queries using fixed conditions.
- Create expression indexes for queries using conditional expressions.
- A single table cannot contain too many indexes because indexes also occupy storage. For example, there should be fewer than 5 single-column indexes and fewer than 3 composite indexes.

SQL Design

- Specify the required fields to be returned in a query.

- Only use IS NULL or IS NOT NULL to determine whether a field is NULL.
- Use NOT EXISTS instead of NOT IN in a query.
- Use UNION ALL instead of UNION to concatenate result sets.
- Use TRUNCATE instead of DELETE to delete an entire table.
- Submit data changes in large transactions in batches to prevent high pressure during transaction commit or rollback.
- When creating a function, define the volatility of the function as the strictest category, instead of the default VOLATILE. Too many concurrent calls of VOLATILE functions may result in failures to establish new connections.

Security

- Do not assign the public role to the owner of an application database object. Assign a specific role to the owner.
- A database password must meet complexity requirements.
- Allocate a unique database account for each service.
- When accessing an object, explicitly specify the schema of the object to avoid accessing objects with the same name in other schemas.

4.2 Databases

4.2.1 Creating a Database

Scenarios

After your TaurusDB for PostgreSQL instance is created, you can create databases on it.

Constraints

- Databases cannot be created for DB instances that are being restored.
- Database names must be unique.
- After a database is created, its name cannot be changed.

Procedure

Step 1 **Log in to the management console**.

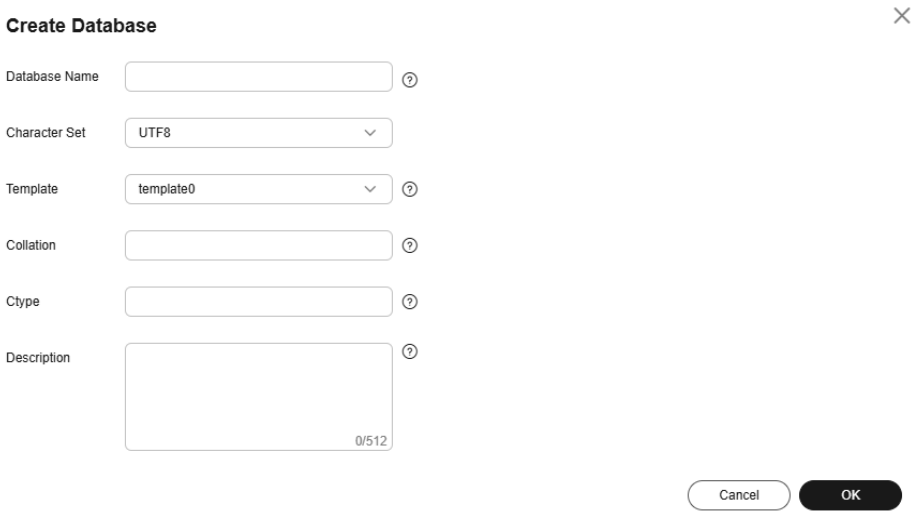
Step 2 Click  in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 On the **Databases** page, click **Create Database**. In the displayed dialog box, configure required parameters and click **OK**.

Figure 4-1 Creating a database



- The database name can contain 1 to 63 characters. Only letters, digits, and underscores (_) are allowed. It cannot start with **pg** or a digit and cannot be the same as TaurusDB for PostgreSQL system or template database names. The system database of TaurusDB for PostgreSQL is **postgres**, and the template databases are **template0** and **template1**.
- The default character set is **utf-8**. You can change it as required.
- You can specify a template database. A new database will be created using this template. Option **template1** (default option) is adapted to TaurusDB for PostgreSQL and template0 complies with the PostgreSQL community settings.
- A collation defines how characters are ordered. en_US.UTF-8 is used by default. Different collations may result in varied ordering results. For example, select 'a'>'A'; is false under en_US.UTF-8, but true under 'C', and also, 'C' must be used to obtain the migration results as expected when data is migrated from Oracle to TaurusDB for PostgreSQL. Collations supported by a database can be queried from system catalog **pg_collation**.
- Ctype refers to character classification to be used in the new database (LC_CTYPE). The use of this parameter affects the classification of characters, such as lowercase letters, uppercase letters, and digits. By default, the character classification of the template database is used.
- The description can contain 0 to 512 characters.

Step 6 After the database is created, manage it on the **Databases** page.

----End

4.2.2 Deleting a Database

Scenarios

You can delete databases that you have created.

NOTICE

Deleted databases cannot be recovered. Exercise caution when performing this operation.

Constraints

Databases cannot be deleted from DB instances that are being restored.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, click **Databases**.

Step 6 Locate the target database and click **Delete** in the **Operation** column.

Step 7 In the displayed dialog box, click **OK**.

----End

4.3 Accounts (Non-Administrator)

4.3.1 Creating a Database Account

Scenarios

When you create a DB instance, account **root** is created at the same time by default. You can create other database accounts as needed.

Constraints

- The instance must be in the running state.
- This operation is not allowed for DB instances that are being restored.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 On the **Accounts** page, click **Create Account**.

Step 6 In the displayed dialog box, set required parameters and click **OK**.

Figure 4-2 Creating a database account

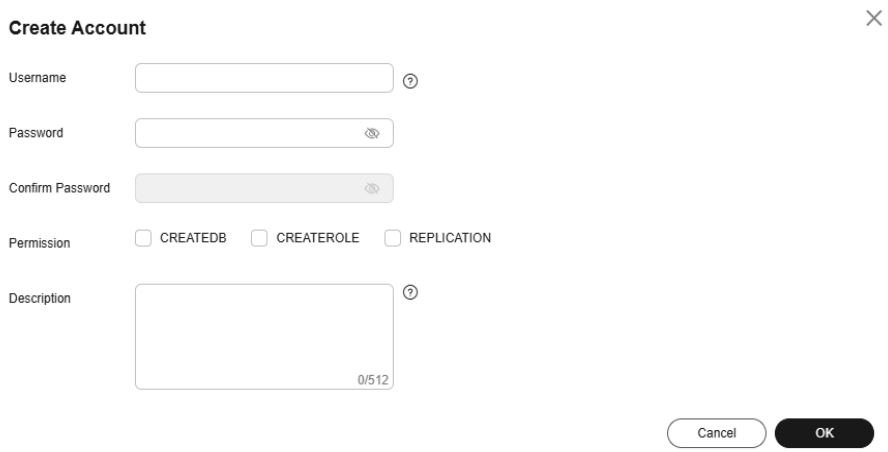


Table 4-1 Parameter description

Parameter	Description
Username	The username can contain 1 to 128 characters. It can include letters, digits, hyphens (-), and underscores (_), and it must be different from system accounts. System accounts include **rdsadmin**, **rdsuser**, **rdsbackup**, and **rdsmirror**.
Password	• The password must consist of 8 to 32 characters and contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters (**~!@#$%^*-_=+?,**). • The password cannot contain the username or the username spelled backwards. • The password must be strong to avoid being easily cracked, as weak passwords will block the creation of your account. You are advised to enter a strong password to improve security and prevent security risks such as brute force cracking.

Parameter	Description
Permission	You can assign permissions, including **CREATEDB**, **CREATEROLE**, and **REPLICATION**, to the user. • **CREATEDB**: indicates that the user has the permission to create a database. If this attribute is not specified, the user cannot create databases by default. • **CREATEROLE**: indicates that the user has the permission to create other users. If this attribute is not specified, the user cannot be used to create new users by default. • **REPLICATION**: indicates that the user can use streaming replication or logical replication. If this attribute is not specified, the user cannot be used to set up streaming replication or logical replication by default.
Description	The description can contain 0 to 512 characters.

Step 7 After the account is created, manage it on the **Accounts** page.

----End

Privileges of the Root User

TaurusDB for PostgreSQL provides privileges for the **root** user. To create objects on a TaurusDB for PostgreSQL database without operation risks, escalate your account to root privileges when necessary.

The following table describes root privilege escalation in different versions.

Table 4-2 Privileges of the **root** user

Version	Whether to Escalate Privileges	Initial Version for Privilege Escalation
pgcore16	Yes	16.2

Escalate to root privileges when you need to:

- Create an event trigger.
- Create a wrapper.
- Create a logical replication publication.
- Create a logical replication subscription.
- Query and maintain replication sources.
- Create a replication user.
- Create a full-text index template and parser.

- Run the **vacuum** command on a system catalog.
- Run the **analyze** command on a system catalog.
- Create an extension.
- Grant an object permission to a user.

Creating a Common Babelfish User

After connecting to the TDS port of a TaurusDB for PostgreSQL instance with Babelfish enabled, you can create a common Babelfish user.

1. Connect to an instance through the TDS port.
   ```
   sqlcmd -S <host>,1433 -U babelfish_user
   ```
2. Create a common user. For more information, see **CREATE USER (Transact-SQL)**.
   ```
   -- Create a login.
   CREATE LOGIN test_babelfish_login WITH PASSWORD = '***';
   GO
   -- Creates a user for the login.
   CREATE USER test_babelfish_user FOR LOGIN test_babelfish_login;
   GO
   ```

4.3.2 Resetting a Password for a Database Account

Scenarios

You can reset passwords for the accounts you have created. To protect your instance against brute force cracking, change your password periodically, such as every three or six months.

Procedure

Step 1 **Log in to the management console**.

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

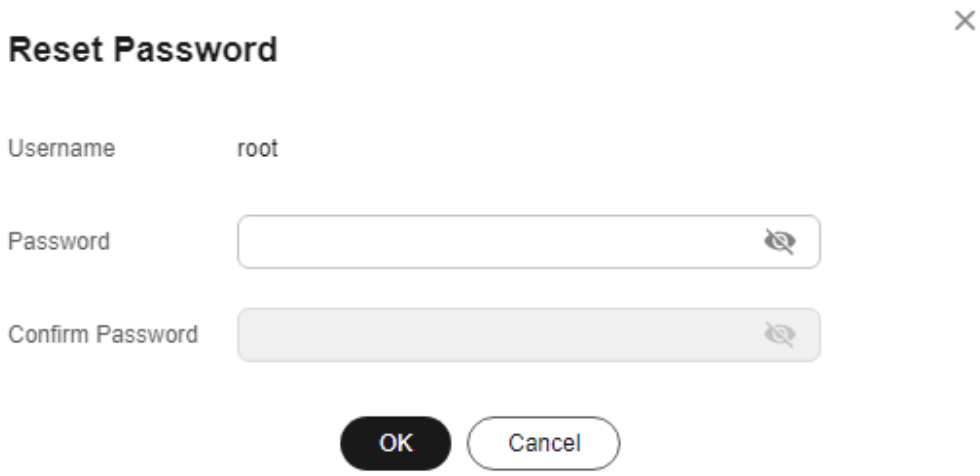
Step 3 Click  in the upper left corner of the page and choose **Databases** > **TaurusDB**.

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Accounts**. On the displayed page, locate the target username and click **Reset Password** in the **Operation** column.

Step 6 In the displayed dialog box, enter a new password, confirm the password, and click **OK**.

Figure 4-3 Resetting a password



- The password must consist of 8 to 32 characters and contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters (**~!@#$%^*-_=+?,**).
- The password cannot contain the username or the username spelled backwards.
- The password must be strong to avoid being easily cracked, as weak passwords will block your operation. You are advised to enter a strong password to improve security and prevent security risks such as brute force cracking.
- You can use Cloud Trace Service (CTS) to query the password reset records.

----End

4.3.3 Deleting a Database Account

Scenarios

You can delete database accounts you have created.

NOTICE

Deleted database accounts cannot be restored. Exercise caution when deleting an account.

Constraints

Accounts cannot be deleted from DB instances that are being restored.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Accounts**. On the displayed page, locate the target username and click **Delete** in the **Operation** column.

Step 6 In the displayed dialog box, click **OK**.

----End

4.4 Tablespace Management

Scenarios

TaurusDB for PostgreSQL provides the PostgreSQL tablespace management solution based on user **root**.

Tablespace Management

You can create and delete tablespaces, and grant tablespace permissions.

Creating a Tablespace

Step 1 Connect to the database as user **root** and create a tablespace.

 # psql --host=*<TaurusDB_ADDRESS>* **--port=***<DB_PORT>* **--dbname=***<DB_NAME>* **--username=root -c "select control_tablespace (**'**create**', '*<TABLESPACE_NAME>*');"

Table 4-3 Parameter description

Parameter	Description
TaurusDB_ADDRESS	IP address of the TaurusDB for PostgreSQL instance
DB_PORT	Port of the TaurusDB for PostgreSQL instance
DB_NAME	Database name
TABLESPACE_NAME	Tablespace name

Step 2 Enter the password of user **root** when prompted.

Log in to the **my_db** database and create the **tbspc1** tablespace. Example:

 # psql --host=192.168.6.141 --port=5432 --dbname=my_db --username=root -c "select control_tablespace('create', 'tbspc1');"

```
Password for user root:
        control_tablespace
```



```
------------------------------
create tablespace tbspc1 successfully.
(1 row)
```

If the creation fails, view error logs of the DB instance.

NOTE

To ensure performance, a maximum of 100 tablespaces can be created.

----End

Granting Tablespace Permissions

Step 1 Connect to a database as user **root** and grant the tablespace usage permissions to specified users.

psql --host=*<TaurusDB_ADDRESS>* **--port=***<DB_PORT>* **--dbname=***<DB_NAME>* **--username=root -c** "**select control_tablespace ('alter',** '*<TABLESPACE_NAME>*', '*<USER_NAME>*');"

Table 4-4 Parameter description

Parameter	Description
TaurusDB_ADDRESS	IP address of the TaurusDB for PostgreSQL instance
DB_PORT	Port of the TaurusDB for PostgreSQL instance
DB_NAME	Database name
TABLESPACE_NAME	Tablespace name
USER_NAME	Tablespace username

Step 2 Enter the password of user **root** when prompted.

Log in to the **my_db** database and grant permissions to tablespace **tbspc1**, for example:

 # psql --host=192.168.6.141 --port=5432 --dbname=my_db --username=root -c "select control_tablespace('alter', 'tbspc1', 'user1');"

```
Password for user root:
        control_tablespace
----------------------------
alter tablespace tbspc1 successfully.
(1 row)
```

If the permissions fail to be granted, view error logs of the DB instance.

----End

Deleting a Tablespace

Step 1 Connect to a database as user **root** and delete a tablespace.

 # psql --host=<*TaurusDB_ADDRESS*> **--port=**<*DB_PORT*> **--username=root --dbname=**<*DB_NAME*> **-c** "**select control_tablespace**('**drop',** '<*TABLESPACE_NAME*>');"

Table 4-5 Parameter description

Parameter	Description
TaurusDB_ADDRESS	IP address of the TaurusDB for PostgreSQL instance
DB_PORT	Port of the TaurusDB for PostgreSQL instance
DB_NAME	Database name
TABLESPACE_NAME	Tablespace name

Step 2 Enter the password of user **root** when prompted.

Example:

 # psql --host=192.168.6.141 --port=8635 --dbname=my_db --username=root -c "select control_tablespace('drop', 'tbspc1');"

```
Password for user root:
        control_tablespace
----------------------------
drop tablespace tbspc1 successfully.
(1 row)
```

Before deleting the tablespace, ensure that it is empty. If the deletion fails, view error logs of the DB instance.

----End

5 Database Migration

5.1 Migration Solution Overview

You can migrate data from RDS for MySQL, self-managed MySQL databases, MySQL databases built on other clouds, self-managed Oracle databases, RDS for PostgreSQL, self-managed PostgreSQL databases, or PostgreSQL databases built on other clouds to TaurusDB for PostgreSQL, or from one TaurusDB for PostgreSQL instance to another.

Data migration tools include Data Replication Service (DRS) and pg_dump. You are advised to use DRS because it is easy to use and can complete a migration task in minutes. DRS facilitates data transfer between databases, helping you reduce DBA labor costs and hardware costs.

DRS provides real-time synchronization. Real-time synchronization refers to the real-time flow of workload data from sources to destinations through a synchronization instance while consistency of data is ensured. It is different from migration. Migration means moving entire data of a database to another. Synchronization refers to the continuous flow of data between different applications.

For more information, see **What Is DRS?**

Migration Solutions

Table 5-1 TaurusDB for PostgreSQL migration solutions

Source Database	Data Size	One-Time or Continuous Migration	Application Downtime	Solution
TaurusDB for PostgreSQL	Small	One-time	Some time	Use pg_dump to copy data from the source to the destination TaurusDB for PostgreSQL instance.
	Any	One-time or continuous	Minimal	Use DRS to synchronize data from the source to the destination TaurusDB for PostgreSQL instance.
RDS for PostgreSQL	Small	One-time	Some time	Use pg_dump to copy data from the source to the destination TaurusDB for PostgreSQL instance.
	Any	One-time or continuous	Minimal	Use DRS to synchronize data from the source to the destination TaurusDB for PostgreSQL instance.
• On-premises PostgreSQL databases • ECS-hosted PostgreSQL databases	Any	One-time or continuous	Minimal	Use DRS to synchronize data from self-managed PostgreSQL databases to TaurusDB for PostgreSQL.
PostgreSQL databases on other clouds	Any	One-time or continuous	Minimal	Use DRS to synchronize data from PostgreSQL databases on other clouds to TaurusDB for PostgreSQL.
• On-premises Oracle databases • ECS-hosted Oracle databases	Any	One-time or continuous	Minimal	Use DRS to synchronize data from self-managed Oracle databases to TaurusDB for PostgreSQL.

Source Database	Data Size	One-Time or Continuous Migration	Application Downtime	Solution
RDS for MySQL	Any	One-time or continuous	Minimal	Use DRS to synchronize data from RDS for MySQL to TaurusDB for PostgreSQL.
• On-premises MySQL databases • ECS-hosted MySQL databases	Any	One-time or continuous	Minimal	Use DRS to synchronize data from self-managed MySQL databases to TaurusDB for PostgreSQL.
MySQL databases on other clouds	Any	One-time or continuous	Minimal	Use DRS to synchronize data from MySQL databases on other clouds to TaurusDB for PostgreSQL.

5.2 Migrating Data to TaurusDB for PostgreSQL Using psql

Preparing for Data Migration

PostgreSQL supports logical backups. You can use the pg_dump logical backup function to export backup files and then import them to TaurusDB for PostgreSQL using psql.

Preparations

1. Prepare an ECS for accessing TaurusDB for PostgreSQL instances.

   To connect to a TaurusDB for PostgreSQL instance through an ECS, you need to create an ECS first.

2. Install the PostgreSQL client on the ECS prepared in **1** or a device that can access TaurusDB for PostgreSQL instances.

   NOTE

   The PostgreSQL client version must be the same as the DB engine version of your TaurusDB for PostgreSQL instance. A PostgreSQL database or client will provide pg_dump and psql.

Exporting Data

Before migrating an existing PostgreSQL database to TaurusDB for PostgreSQL, you need to export data first.

NOTICE

- The export tool must match the DB engine version.
- Database migration is performed offline. Before the migration, you must stop any applications using the source database.
- Take care when exporting or importing data. Improper operations can cause instance or workload exceptions.

Step 1 Log in to the ECS or the device that can access the TaurusDB for PostgreSQL instance.

Step 2 Use the pg_dump tool to export the source database into a SQL file.

```
pg_dump--username=<DB_USER> --host=<DB_ADDRESS> --port=<DB_PORT> --format=plain --file=<BACKUP_FILE><DB_NAME>
```

- *DB_USER* indicates the database username.
- *DB_ADDRESS* indicates the database address.
- *DB_PORT* indicates the database port.
- *BACKUP_FILE* indicates the name of the file to which the data will be exported.
- *DB_NAME* indicates the name of the database to be migrated.

Enter the database password as prompted.

NOTE

If the exported SQL file uses INSERT statements, you can easily edit and modify the file. However, the speed of importing data may be slower than that of using COPY statements. You are advised to select a right statement format as needed.

- If both the source and destination databases are PostgreSQL databases, you are advised to export COPY statements (default). For details, see **Example 1: Exporting the source database to a SQL file (COPY)**.
- If either of the source and destination databases is a non-PostgreSQL database, you are advised to export INSERT statements. For details, see **Example 2: Exporting the source database to a SQL file (INSERT)**.

For more information, see **pg_dump options**.

Examples:

- Example 1: Exporting the source database to a SQL file (COPY)

  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --file=backup.sql my_db
  ```

  Password for user root:

- Example 2: Exporting the source database to a SQL file (INSERT)

  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --inserts --file=backup.sql my_db
  ```

  Password for user root:

- Example 3: Exporting all table structures from the source database to a SQL file
  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --schema-only --file=backup.sql my_db
  ```
  Password for user root:
- Example 4: Exporting all table data from the source database to a SQL file
  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --data-only --file=backup.sql my_db
  ```
  Password for user root:

After the commands in any of the above examples are executed, a **backup.sql** file will be generated as follows:

```
$ ll backup.sql
-rw-r-----. 1 rds rds 2714 Sep 21 08:23 backup.sql
```

Step 3 Use pg_dump to export tables from the source database to a SQL file.

```
pg_dump --username=<DB_USER> --host=<DB_ADDRESS> --port=<DB_PORT> --format=plain --file=<BACKUP_FILE> <DB_NAME> --table=<TABLE_NAME>
```

- *DB_USER* indicates the database username.
- *DB_ADDRESS* indicates the database address.
- *DB_PORT* indicates the database port.
- *BACKUP_FILE* indicates the name of the file to which the data will be exported.
- *DB_NAME* indicates the name of the database to be migrated.
- *TABLE_NAME* indicates the name of the specified table in the database to be migrated.

Enter the database password as prompted.

Examples:

- Example 1: Exporting one table from the source database to a SQL file
  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --file=backup.sql my_db --table=test
  ```
  Password for user root:
- Example 2: Exporting multiple tables from the source database to a SQL file
  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --file=backup.sql my_db --table=test1 --table=test2
  ```
  Password for user root:
- Example 3: Exporting all tables starting with **ts_** from the source database to a SQL file
  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --file=backup.sql my_db --table=ts_*
  ```
  Password for user root:
- Example 4: Exporting all tables except those starting with **ts_** from the source database to a SQL file
  ```
  $ pg_dump --username=root --host=192.168.151.18 --port=5432 --format=plain --file=backup.sql my_db -T=ts_*
  ```
  Password for user root:

After the commands in any of the above examples are executed, a **backup.sql** file will be generated as follows:


```
$ ll backup.sql
-rw-r-----. 1 rds rds 2714 Sep 21 08:23 backup.sql
```

----End

Importing Data

Step 1 Log in to the ECS or the device that can access the TaurusDB for PostgreSQL instance.

Step 2 Ensure that the destination database to which data is to be imported exists.

If the destination database does not exist, run the following command to create a database:

```
# psql --host=<TaurusDB_ADDRESS>--port=<DB_PORT>--username=root--dbname=postgres-c "create database<DB_NAME>;"
```

- *TaurusDB_ADDRESS* indicates the IP address of the TaurusDB for PostgreSQL instance.
- *DB_PORT* indicates the database port of the instance.
- *DB_NAME* indicates the name of the database to which data is to be imported.

Step 3 Import the exported file to the TaurusDB for PostgreSQL database.

```
# psql --host=<TaurusDB_ADDRESS> --port=<DB_PORT>--username=root--dbname=<DB_NAME>--file=<BACKUP_DIR>/backup.sql
```

- *TaurusDB_ADDRESS* indicates the IP address of the TaurusDB for PostgreSQL instance.
- *DB_PORT* indicates the database port of the instance.
- *DB_NAME* indicates the name of the database to which data is to be imported. Ensure that the database exists.
- *BACKUP_DIR* indicates the directory where the **backup.sql** file is stored.

Enter the password for the TaurusDB for PostgreSQL instance when prompted.

Example:

```
# psql --host=172.16.66.198 --port=5432 --username=root --dbname=my_db --file=backup.sql
```

Password for user root:

Step 4 View the import result.

my_db=> \l my_db

In this example, the database named **my_db** has been imported.

```
my_db=> \l my_db
List of databases
Name  | Owner | Encoding | Collate     | Ctype       | Access privileges
------+-------+----------+-------------+-------------+-----------
my_db | root  | UTF8     | en_US.UTF-8 | en_US.UTF-8 |
(1 row)
```

----End

5.3 Migrating a Microsoft SQL Server Database to TaurusDB for PostgreSQL Using Babelfish

Scenarios

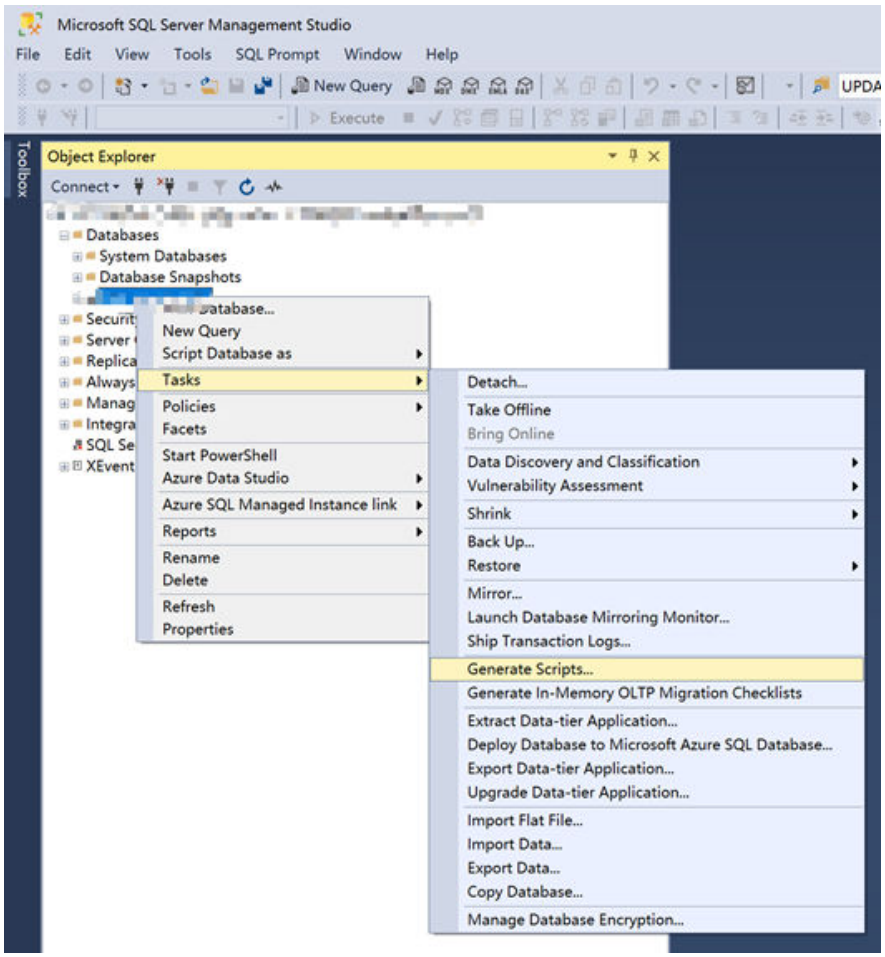
This section describes how to migrate a Microsoft SQL Server database to TaurusDB for PostgreSQL. Babelfish's compatibility with Microsoft SQL Server helps minimize the need for code adaptation, streamline the migration process, improve efficiency, and save resources.

Prerequisites

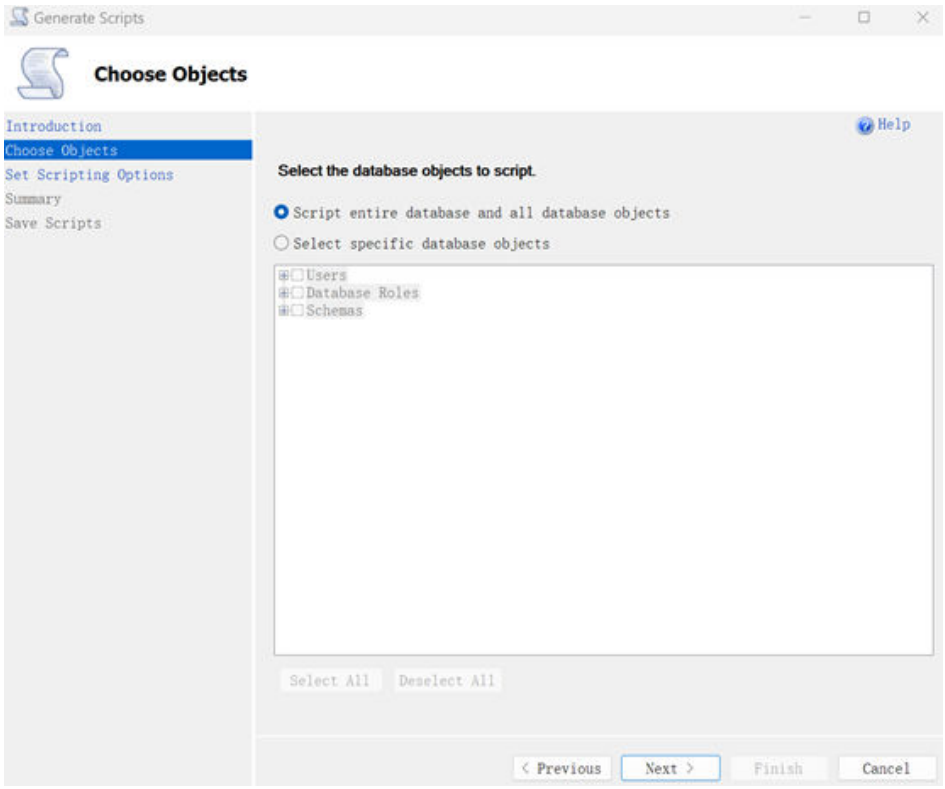
- You have deployed a TaurusDB for PostgreSQL instance with Babelfish enabled.
- You have created a Babelfish account.
- You have configured an IP address whitelist to allow the server where the client is located to access the target instance.
- You have installed a Microsoft SQL Server client tool (for example, sqlcmd).
- You have enabled Huawei Cloud Data Replication Service (DRS) if you plan to migrate data using it.

Step 1: Export Objects and Data of a Microsoft SQL Server Database

1. Connect to the Microsoft SQL Server database.
   a. Use SSMS to connect to the destination database.
   b. In the **Object Explorer** window, right-click the destination database and choose **Tasks** > **Generate Scripts...**.

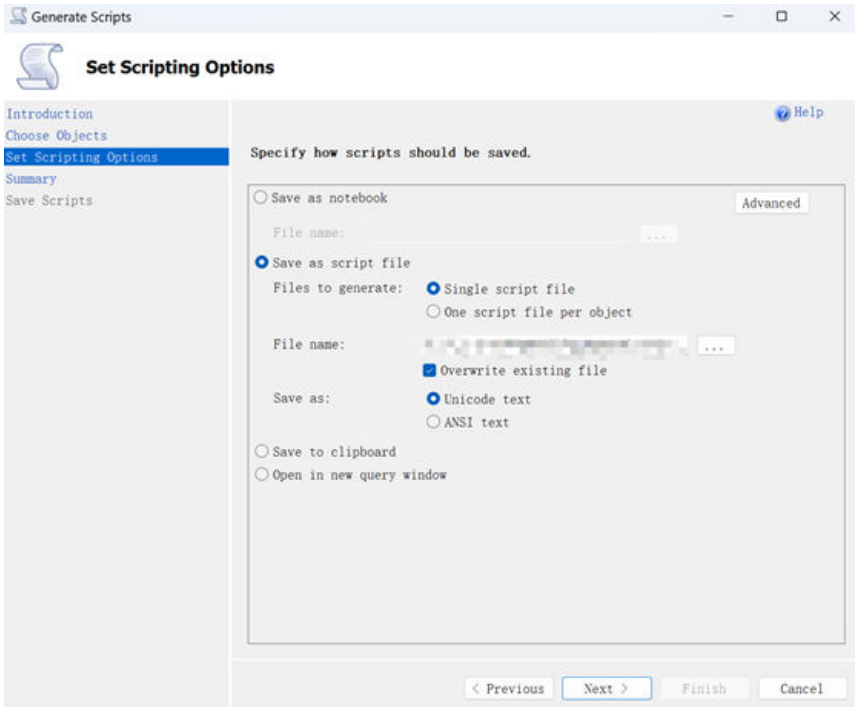


2. Generate a script.
   a. **Introduction**: Click **Next**.
   b. **Choose Objects**: Select **Script entire database and all database objects** and click **Next**.

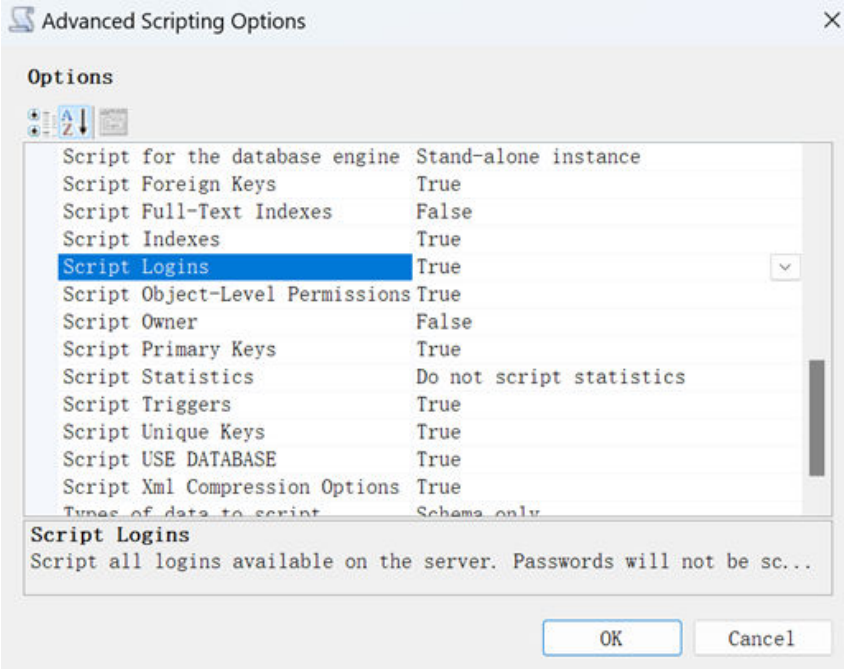


c. **Set Scripting Options**:

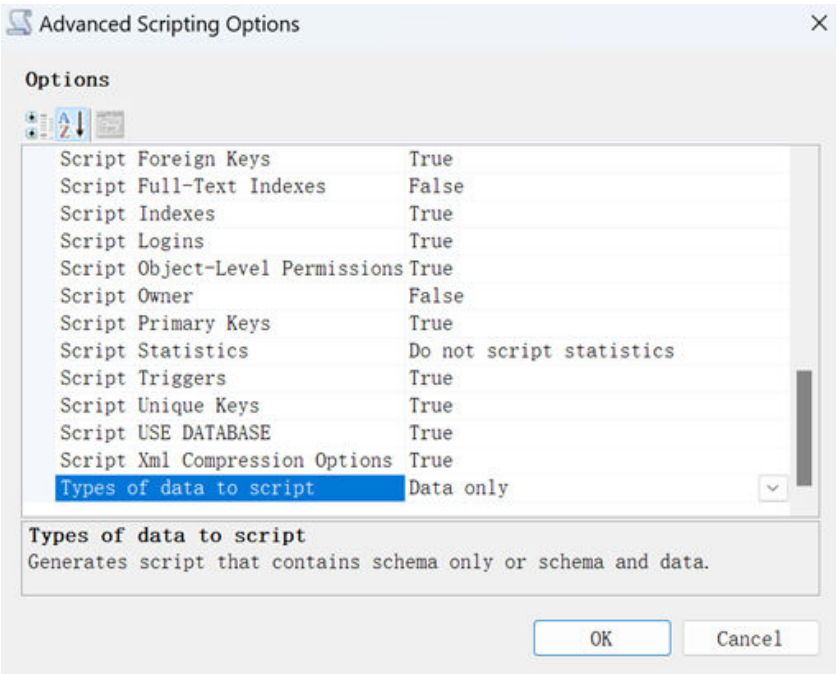
i. Select **Save as script file** and specify the output file name.



ii. To export the data definition language (DDL) script, click **Advanced** and enable the corresponding configuration.



iii. To export the data manipulation language (DML) script, click **Advanced** and set **Types of data to script** to **Data only**.



3. Obtain the exported SQL file.

   After the script is generated, obtain the exported **.sql** file from the preset path.

Step 2: Use Babelfish Compass to Evaluate Compatibility

Run Babelfish Compass to analyze the support for T-SQL statements in Babelfish and optimize the statements.

Note: For details about how to obtain and use the tool, see the **Babelfish Compass official documentation**.

Step 3: Import the DDL Script to the Babelfish Database

1. Connect to the TDS port of the Babelfish instance.

   For details, see **Connecting to a TaurusDB for PostgreSQL Instance Through sqlcmd**.

2. Execute the DDL script.

```
:r /path/to/your_ddl_script.sql
GO
```

Step 4: Migrate Data to Babelfish

Use Huawei Cloud DRS (recommended for a large amount of data in a production environment) to migrate data.

6 Instance Management

6.1 Viewing Instance Overview Data

You can view instance overview data to learn about the statuses and alarms of DB instances in real time.

Learning About TaurusDB for PostgreSQL

If you are new to TaurusDB for PostgreSQL, you can quickly learn about its concepts, common functions, and how to purchase and get started with a DB instance by referring to *Progressive Knowledge*.

1. To purchase a TaurusDB for PostgreSQL instance, click **Buy DB Instance** on the **Overview** page and select your desired version and specifications. For details, see **Buying a TaurusDB for PostgreSQL Instance**.
2. After the purchase is complete, view the instance status and active alarms on the **Overview** page.

Viewing Instances by Status

The statuses of all TaurusDB for PostgreSQL instances under your account are displayed in the upper part of the **Overview** page.

Table 6-1 Status description

Item	Description
Total instances	Total number of TaurusDB for PostgreSQL instances in all states
Abnormal	Total number of TaurusDB for PostgreSQL instances in the **Abnormal** state
Frozen	Total number of TaurusDB for PostgreSQL instances in the **Frozen** state

Item	Description
Pending reboot	Total number of TaurusDB for PostgreSQL instances in the **Pending reboot** state **NOTE** Modifications to some parameters require an instance reboot before they can be applied.
Available	Total number of TaurusDB for PostgreSQL instances in the **Available** state

6.2 Instance Lifecycle

6.2.1 Stopping a DB Instance

Scenarios

If you use DB instances only for routine development, you can temporarily stop pay-per-use instances to save money. You can stop an instance for up to 15 days.

Billing

After a DB instance is stopped, the VM where the DB instance is located is no longer billed. Other resources, including storage resources and backups, are still billed.

Constraints

- Only cloud SSD and extreme SSD pay-per-use instances can be stopped.
- A stopped instance cannot be deleted through the console.
- Stopping a DB instance will also stop its automated backups. After the DB instance is started, a full backup is automatically triggered.
- If you do not manually start your stopped DB instance after 15 days, your DB instance is automatically started during the next maintenance window. To start a DB instance, see **Starting a DB Instance**.
- A stopped pay-per-use instance may fail to start due to insufficient ECS resources. If this happens, try again later. If you need assistance, **submit a service ticket**.

Procedure

Step 1 **Log in to the management console**.

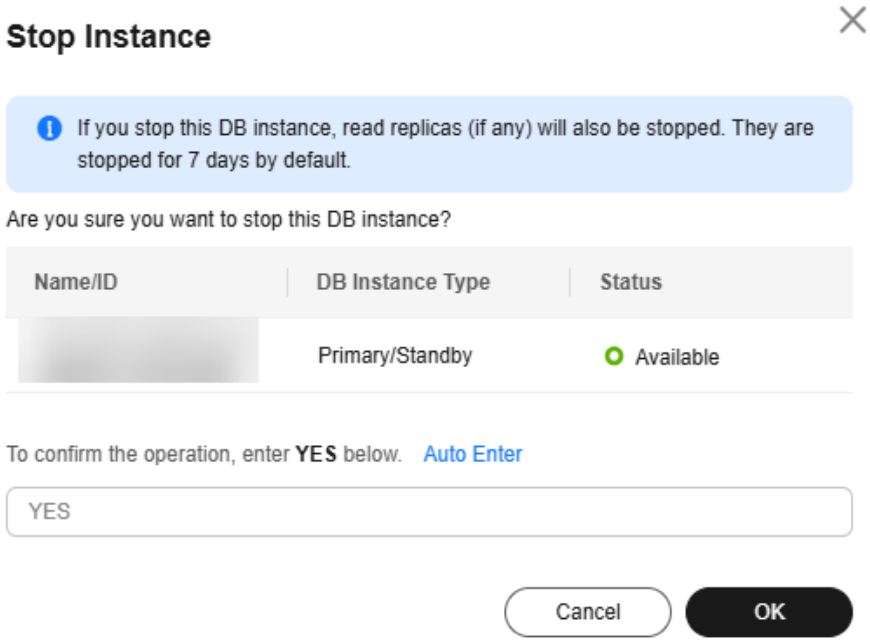
Step 2 Click  in the upper left corner and select a region.

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

Step 4 On the **Instances** page, locate the primary instance that you want to stop and choose **More** > **Stop** in the **Operation** column.

Step 5 In the displayed dialog box, click **OK**.

Figure 6-1 Stopping a DB instance



Step 6 Refresh the instance list and view the status of the instance. If the status is **Stopped**, the instance is stopped successfully.

----End

6.2.2 Starting a DB Instance

Scenarios

You can stop your instance temporarily to save money. After stopping your instance, you can restart it to begin using it again.

Billing

After a DB instance is started, the VM where the DB instance is located is billed again.

Constraints

- When a stopped DB instance is started, a full backup is automatically triggered.
- Only instances in **Stopped** state can be started.
- A stopped pay-per-use instance may fail to start due to insufficient ECS resources. If this happens, try again later. If you need assistance, **submit a service ticket**.

Procedure

Step 1 Log in to the management console.

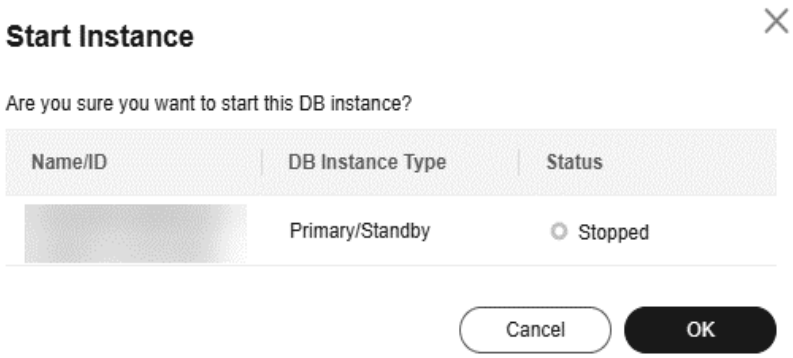
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Instances** page, locate the primary instance that you want to start and choose **More** > **Start** in the **Operation** column.

Step 5 In the displayed dialog box, click **OK**.

Figure 6-2 Starting a DB instance



Step 6 Refresh the instance list and view the status of the instance. If the status is **Available**, the instance is started successfully.

----End

6.2.3 Rebooting a DB Instance

Scenarios

You may need to reboot a DB instance for maintenance reasons. For example, after you modify some parameters, a reboot is required for the modifications to take effect. You can reboot a DB instance on the console.

Constraints

- If the DB instance status is **Abnormal**, the reboot may fail.
- An instance cannot be rebooted if its storage is full.
- Rebooting a DB instance will cause service interruptions. During this period, the DB instance status is **Rebooting**.
- Rebooting DB instances will cause instance unavailability and clear cached memory. To prevent traffic congestion during peak hours, you are advised to reboot DB instances during off-peak hours.

Rebooting a DB Instance

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, locate the instance you want to reboot and choose **More** > **Reboot** in the **Operation** column.

Alternatively, click the target DB instance on the **Instances** page to go to the **Basic Information** page. In the upper right corner, click **Reboot**.

For primary/standby DB instances, if you reboot the primary DB instance, the standby DB instance is also rebooted automatically.

Step 5 In the displayed dialog box, select the check box and click **OK**.

Step 6 Refresh the DB instance list and view the status of the DB instance. If its status is **Available**, it has rebooted successfully.

----End

6.2.4 Selecting Displayed Items

Scenarios

You can customize which instance items are displayed on the **Instances** page.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click to edit columns displayed in the DB instance list.

- **Table Text Wrapping**: If you enable this function, excess text will move down to the next line.
- **Operation Column**: If you enable this function, the **Operation** column is always fixed at the rightmost position of the table.
- The following items can be displayed: **Name/ID**, **Description**, **DB Instance Type**, **DB Engine Version**, **Status**, **Billing Mode**, **Private IP Address**, **Private IP Address for Read**, **Proxy Address**, **Private Domain Name**, **Enterprise Project**, **Created**, **Database Port**, **Storage Type**, and **Operation**.

----End

6.2.5 Exporting DB Instance Information

Scenarios

You can export information about all or selected DB instances to view and analyze DB instance information.

Constraints

A tenant can export a maximum of 3,000 instances at a time. The time required for the export depends on the number of instances.

Exporting Information About All DB Instances

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click **Export Instance Info** above the DB instance list. By default, information about all DB instances is exported. In the displayed dialog box, you can select the items to be exported and click **OK**.

Step 5 Find a .csv file locally after the export task is completed.

----End

Exporting Information About Selected DB Instances

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, filter DB instances by DB engine, DB instance name, DB instance ID, private IP address, or enterprise project, or select the DB instances to be exported, and click **Export Instance Info** above the DB instance list. In the displayed dialog box, select the items to be exported and click **OK**.

Step 5 Find a .csv file locally after the export task is completed.

----End

6.2.6 Deleting a Pay-per-Use DB Instance

Scenarios

You can manually delete a pay-per-use DB instance on the **Instances** page.

Billing

- You will not be billed for the instances that were not successfully created.
- If you delete a pay-per-use DB instance, its automated backups will also be deleted and you will no longer be billed for them. Manual backups, however, will be retained and generate additional costs.

Constraints

- DB instances cannot be deleted when operations are being performed on them. They can be deleted only after the operations are complete.
- If a backup of a DB instance is being restored, the instance cannot be deleted.
- A stopped instance cannot be deleted through the console.
- If you delete a primary DB instance, its standby DB instance is also deleted automatically. Exercise caution when performing this operation.
- Deleted DB instances cannot be recovered and resources are released. Exercise caution when performing this operation. If you want to retain data, **create a manual backup** first before deleting the DB instance.
- You can **rebuild a DB instance** that was deleted up to 7 days ago from the recycle bin.
- You can use a manual backup to restore a DB instance. For details, see **Restoring a DB Instance from Backups**.

Deleting a Pay-per-Use DB Instance

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, locate the primary DB instance to be deleted and click **More** > **Delete** in the **Operation** column.

Step 5 In the displayed dialog box, click **OK**.

Step 6 If you have enabled operation protection, click **Send Code** in the displayed **Identity Verification** dialog box and enter the obtained verification code. Then, click **OK**.

Two-factor authentication improves the security of your account and cloud product. For details about how to enable operation protection, see ***Identity and Access Management User Guide***.

Step 7 Refresh the DB instance list later to confirm that the deletion was successful.

----End

6.2.7 Recycling a DB Instance

Scenarios

TaurusDB for PostgreSQL allows you to move deleted pay-per-use DB instances to the recycle bin. You can rebuild a DB instance that was deleted up to 7 days ago from the recycle bin.

If resources are not renewed after expiration, you can rebuild DB instances from the recycle bin to restore data.

Constraints

- The recycle bin is free for use.
- The recycle bin is enabled by default and cannot be disabled.
- After you submit a deletion request for your DB instance, a full backup will be performed. You can rebuild the DB instance only after the full backup is complete.

Modifying Recycling Policy

NOTICE

Instances in the recycle bin are retained for 7 days by default. A new recycling policy only applies to DB instances that were put in the recycle bin after the new policy was put into effect. For DB instances that were in the recycle bin before the modification, the original recycling policy takes effect.

Step 1 **Log in to the management console**.

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

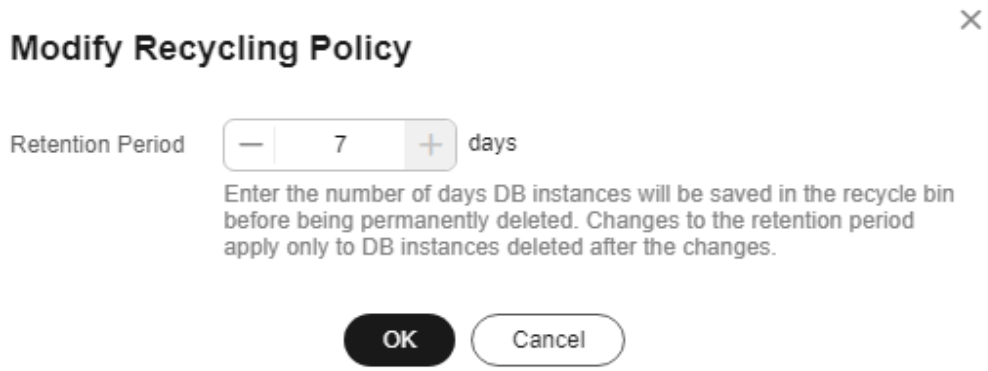
Step 3 Click  in the upper left corner of the page and choose **Databases** > **TaurusDB**.

Step 4 In the navigation pane, choose **Recycle Bin**.

Step 5 On the **Recycle Bin** page, click **Modify Recycling Policy**. In the displayed dialog box, set the retention period for the deleted DB instances to 1 to 7 days.

Step 6 Then, click **OK**.

Figure 6-3 Modifying the recycling policy



----End

Rebuilding a DB Instance

You can rebuild the primary DB instances in the recycle bin within the retention period.

Step 1 **Log in to the management console**.

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

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

Step 4 In the navigation pane, choose **Recycle Bin**.

Step 5 On the **Recycle Bin** page, locate the target DB instance to be rebuilt and click **Rebuild** in the **Operation** column.

Step 6 On the **Rebuild DB Instance** page, configure required information and submit the rebuild task. For details, see **Restoring a DB Instance from Backups**.

----End

7 Instance Modifications

7.1 Changing a DB Instance Name

Scenarios

You can change the name of a primary DB instance.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, locate the target DB instance and click next to it to edit the DB instance name. Then, click **OK**.

Alternatively, click the instance name to go to the **Basic Information** page. Under **DB Instance Name**, click to edit the instance name.

The instance name must start with a letter and consist of 4 to 64 characters. Only letters (case-sensitive), digits, hyphens (-), and underscores (_) are allowed.

- To submit the change, click .
- To cancel the change, click .

Step 5 View the results on the **Basic Information** page.

----End

7.2 Changing a DB Instance Description

Scenarios

You can add and change descriptions for your DB instances.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, locate the DB instance you wish to edit the description for and click in the **Description** column to make your modification. When you are finished, click **OK**.

Alternatively, click the instance name to go to the **Basic Information** page. Under **Description**, click to edit the instance description.

NOTE

The DB instance description can include up to 64 characters, and can include letters, digits, hyphens (-), underscores (_), and periods (.).

- To submit the change, click .
- To cancel the change, click .

Step 5 View the results on the **Basic Information** page.

----End

7.3 Changing the Replication Mode

Scenarios

TaurusDB for PostgreSQL allows you to change the replication mode between primary and standby instances. Data can be asynchronously or synchronously replicated from the primary instance to the standby instance.

- **Asynchronous** (default): When an application writes data to the primary instance, the primary instance returns a response to the application immediately without waiting for the standby instance to receive logs.
  - Advantages: Asynchronous replication involves low overhead and ensures that write operations are not blocked during a failover of your primary/standby instances.
  - Disadvantages: In rare cases, replication is delayed between the primary and standby instances, and data may be lost after the failover.
- **Synchronous**: When an application writes data to the primary instance, the primary instance returns a response to the application only after the standby instance receives logs (which are flushed to the disk).
  - Advantages: Data remains strongly consistent between the primary and standby instances, and no data loss occurs after a failover.
  - Disadvantages: Synchronous replication involves high overhead and causes write operations to be blocked when the primary or standby instance is faulty.

NOTE

- Asynchronous replication is recommended for applications requiring a guarantee of high availability.
- Synchronous replication is recommended for applications that require strong data consistency and can tolerate a short-time blocking of write operations.
- Write operations refer to non-SELECT operations, such as DDL and DML.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, locate a primary/standby instance and click the instance name.

Step 5 On the displayed **Basic Information** page, find **Replication Mode** and click **Change** under it. In the displayed dialog box, select a mode and click **OK**.

Step 6 View the results on the **Basic Information** page.

----End

7.4 Changing the Failover Priority

Scenarios

TaurusDB for PostgreSQL allows you to change the failover priority for primary and standby instances. You can select **reliability** or **availability**.

- **Reliability** (default setting): Data consistency is preferentially ensured during a primary/standby failover. This is recommended for applications whose highest priority is data consistency. In extreme scenarios, there may be a small amount of data lost if your instance uses asynchronous replication.
- **Availability**: Database availability is preferentially ensured during a primary/standby failover. This is recommended for applications that require databases to provide uninterrupted online services.

Constraints

The failover priority cannot be changed when the DB instance is stopped.

Procedure

Step 1 Log in to the management console.

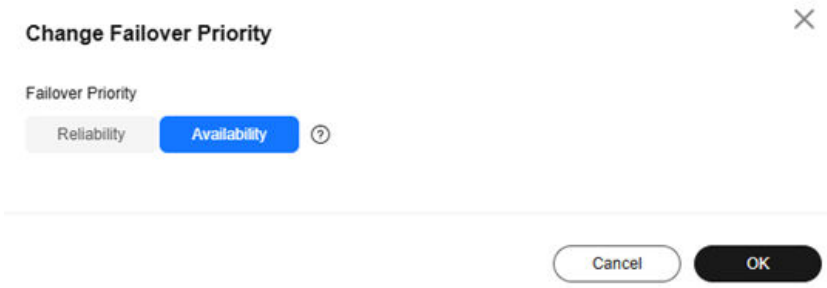
Step 2 Click  in the upper left corner and select a region.

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

Step 4 On the **Instances** page, locate a primary/standby instance and click the instance name.

Step 5 On the displayed **Basic Information** page, find **Failover Priority** and click **Change** under it. In the displayed dialog box, select a priority and click **OK**.

Figure 7-1 Changing the failover priority



Step 6 View the results on the **Basic Information** page.

----End

7.5 Scaling up Storage Space

Scenarios

If the original storage space is insufficient as your workloads grow, scale up storage space of your DB instance.

When the storage usage is greater than or equal to 97%, the DB instance status changes to **Storage full**. Your connection to the instance is intermittently interrupted and the instance becomes read-only. When this happens, data cannot be written to the instance.

A DB instance needs to preserve at least 15% of its capacity to work properly. The new minimum storage space required to make this instance available has been automatically calculated for you. You are advised to set alarm rules for the **Storage Space Usage** metric to learn about the storage usage in a timely manner.

Workloads are not interrupted during storage scale-up.

Constraints on Scale-up

- You can scale up storage space only when your account balance is greater than or equal to $0 USD.
- The maximum allowed storage is 4,000 GB. There is no limit on the number of scale-ups.
- For primary/standby DB instances, scaling up the primary DB instance will cause the standby DB instance to also be scaled up.

- A DB instance cannot be deleted during scale-up.

Billing

Table 7-1 Billing

Billing Mode	Operation	Impact on Fees
Pay-per-use	Storage scaling	The new storage space is billed by hour.

Scaling Storage Space

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, locate the target DB instance and choose **More** > **Scale Storage Space** in the **Operation** column.

Or use either of the following methods:

- Click the instance name to go to the **Basic Information** page. In the **Storage & Backup** area, click **Scale Storage Space**.
- If the instance storage is full, locate the instance on the **Instances** page and click **Scale** in the **Status** column.

Step 5 On the displayed page, specify the new storage space and click **Next**.

You can increase or decrease the storage by at least 10 GB. Enter a value that is a multiple of 10. The instance supports a storage space range from 40 GB to 4,000 GB.

The final storage space range on the console may differ slightly. It varies with the storage usage of the instance.

Step 6 Confirm the information.

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

Step 7 Check the results.

Scaling up storage space takes 3 to 5 minutes. During the time period, the status of the instance on the **Instances** page will be **Scaling up**. After a while, click the instance name and view the new storage space on the displayed **Basic Information** page to verify that the scale-up is successful.

----End

8 Data Backups

8.1 Introduction to Backups

What Are Database Backups?

TaurusDB for PostgreSQL automatically creates backups for DB instances during the backup time window. The backups are stored based on a preset retention period (1 to 732 days).

A backup file is generated each time a backup is complete. If the instance fails or data is damaged, you can use the backup file to restore the instance, ensuring data reliability.

Backup Types

TaurusDB for PostgreSQL supports multiple backup types. For details, see **Backup Types**.

- Full backup: A full backup is to back up all data, even if no data has changed since the last backup.

  Full backup can be initiated manually or automatically.
- Incremental backup: Incremental backups refer to Write-Ahead Logging (WAL) backups. TaurusDB for PostgreSQL performs an incremental backup every 5 minutes.

Where Data Is Backed Up

- Single-node instance

  A single-node architecture, which is more cost-effective than mainstream primary/standby DB instances. After a backup is triggered, data is backed up from the primary instance and stored as a package on OBS. The backup does not take up storage space of the instance.
- Primary/standby instance

  An HA architecture. In a primary/standby pair, each instance has the same instance specifications. After a backup is triggered, data is backed up from the

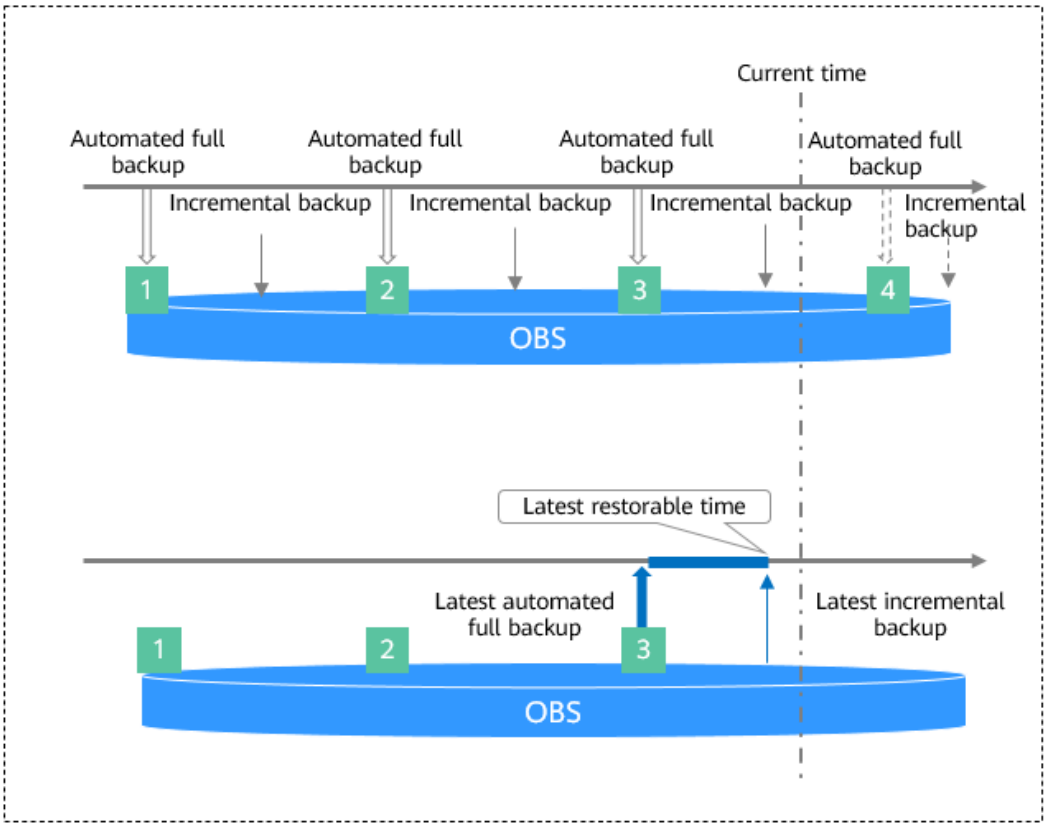
primary instance and stored as a package on OBS. The backup does not take up storage space of the instance.

If a database or table in the primary instance is maliciously or mistakenly deleted, the database or table in the standby instance will also be deleted. In this case, you can only use backups to restore the deleted data.

How Data Is Backed Up

TaurusDB for PostgreSQL automated backup is enabled by default and cannot be disabled. TaurusDB for PostgreSQL performs automated full backups based on the time window and interval specified in the backup policy. It also backs up data modifications made after the most recent automated full or incremental backup every five minutes. When you restore an instance to a point in time, the most recent full backup will be downloaded from OBS for restoration. After the restoration is complete, incremental backups will be replayed to the specified point in time.

Figure 8-1 How data is backed up



Backup Solutions

Table 8-1 describes how to back up data and download backups.

Table 8-1 Backup solutions

Task	Backup Type	Description
Backing up data in the same region	**Automated backups**	TaurusDB for PostgreSQL automatically creates full backups for your instance during a backup window you specified and saves the backups based on the configured retention period. If necessary, you can restore data to any point in time within the backup retention period. Once the automated backup policy is enabled, a full physical backup is triggered immediately. After that, full backups will be created according to the specified time window and backup cycle.
	Manual backups	Manual backups are user-initiated full backups of instances. The backup method is physical backup. Manual backups are always retained even though a TaurusDB for PostgreSQL instance is deleted. They can only be deleted manually.
	Incremental backups	WAL is enabled for TaurusDB for PostgreSQL instances by default. The system automatically backs up data modifications made after the most recent automated or incremental backup every five minutes.

Backup Storage and Billing

Backups are saved as packages in OBS buckets. Backups occupy backup space in OBS. If the free space TaurusDB for PostgreSQL provides is used up, the additional space required will be billed.

Deleting Backups

Backups can be deleted in different ways:

- Manual backups can only be manually deleted.
- Automated backups cannot be manually deleted. To delete them, set the retention period specified in your **automated backup policy**. Retained backups will be automatically deleted at the end of the retention period.

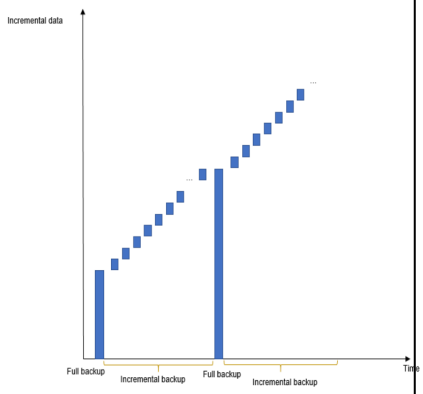
8.2 Backup Types

TaurusDB for PostgreSQL supports multiple backup types. Based on different dimensions, there are the following backup types:

Full Backups and Incremental Backups Based on Data Volume

Table 8-2 Comparison between full backups and incremental backups

Backup Type	Full backups	Incremental backups
Description	All data in an instance is backed up.	Only data changes within a certain period of time are backed up.
Enabled by Default	Yes	Yes
Retention Period	• You can specify how many days automated backups can be retained for. If you shorten the retention period, the new backup policy takes effect for existing backups. • Manual backups are always retained even though a TaurusDB for PostgreSQL instance is deleted. They can only be deleted manually.	Incremental backups will be deleted along with automated full backups.

Characteristics	• A full backup is to back up all data of your DB instance in the current point of time. • You can use a full backup to restore the complete data generated when the backup was created. • Full backups can be created automatically or manually.	• The system automatically backs up data modifications made after the most recent automated or incremental backup every five minutes. • Incremental backups can only be created automatically. • Restoring data from incremental backups relies on the most recent full backup, as shown in **Figure 8-2**, so the most recent full backup that exceeds the retention period will not be deleted to ensure successful restoration. **Figure 8-2** Incremental data restoration 
How to Check Backup Size	Click the instance name. On the **Backups** page, click the **Full Backups** tab and check the backup size.	Click the instance name. On the **Backups** page, click the **Incremental Backups** tab and check the backup size.

Automated Backups and Manual Backups Based on Backup Methods

Table 8-3 Comparison between automated backups and manual backups

Backup Type	Automated backups	Manual backups

Description	• You can set an automated backup policy on the console, and the system will back up your instance data based on the time window and backup cycle you set in the backup policy and will store the backups for the retention period you specified. • Automated backups cannot be manually deleted. To delete them, you can adjust the retention period specified in your automated backup policy. Retained backups (including full and incremental backups) will be automatically deleted at the end of the retention period. **NOTE** Once a DB instance is created, a backup is restored to a new instance, or a minor version is upgraded, a full backup is automatically triggered by default and this function cannot be disabled. Backups that are being created can be stopped. If such a backup task is stopped and the first full backup of your instance is not complete, no backup is available for restoration. To stop a backup, **submit a service ticket** to apply for required permissions.	• Manual backups are user-initiated full backups of your DB instance. They are retained until you delete them manually. • Regularly backing up your DB instance is recommended, so if your DB instance fails or data is corrupted, you can restore it using backups.
Enabled by Default	Yes	Yes
Retention Period	Automated backups are retained for the number of days you specified. The retention period ranges from 1 to 732 days.	Manual backups are always retained until you delete them manually.
How to Configure	See **Configuring a Same-Region Backup Policy**.	See **Creating a Manual Backup**.

8.3 Working with Backups

TaurusDB for PostgreSQL supports backups and restorations to ensure data reliability.

NOTE

TaurusDB for PostgreSQL uses **sysbench** to import data models and a certain amount of data. After data is backed up, the compression ratio is about 80%. The more duplicate data there is, the higher the compression ratio is.

Compression ratio = Space occupied by backup files/Space occupied by data files x 100%

Functions

If a database or table in the primary instance is maliciously or mistakenly deleted, the database or table in the standby instance will also be deleted. In this case, you can only restore the deleted data from backups.

Billing

Backups are saved as packages in OBS buckets.

Automated Backups

Automated backups are created during the backup time window for your DB instances. TaurusDB for PostgreSQL saves automated backups based on a retention period you specify. If necessary, you can restore to any point in time during your backup retention period. For details, see **Configuring a Same-Region Backup Policy**.

Manual Backups

Manual backups are user-initiated full backups of DB instances. They are retained until you delete them manually. For details, see **Creating a Manual Backup**.

Full Backups

Full backups are used to back up all data even if no data was updated since the last backup.

Incremental Backups

Incremental backups refer to Write-Ahead Logging (WAL) backups. TaurusDB for PostgreSQL automatically backs up data modifications made after the most recent automated or incremental backup every five minutes.

Exporting Backup Information

You can export backup information of DB instances to an EXCEL file for further analysis. The exported information includes the DB instance names, backup start and end times, backup statuses, and backup sizes. For details, see **Checking and Exporting Backup Information**.

8.4 Instance-Level Backups

8.4.1 Configuring a Same-Region Backup Policy

Scenarios

When you create a DB instance, an automated backup policy is enabled by default. For security purposes, the automated backup policy cannot be disabled. After the DB instance is created, you can customize the automated backup policy as required and then TaurusDB for PostgreSQL backs up data based on the automated backup policy you have set.

TaurusDB for PostgreSQL backs up data at the DB instance level, rather than the database level. If a database is faulty or data is damaged, you can restore it from backups. Backups are saved as packages in OBS buckets to ensure data confidentiality and durability. Since backing up data affects database read and write performance, the automated backup time window should be set to off-peak hours.

After an automated backup policy is configured, full backups are created based on the time window and backup cycle specified in the policy. The time required for creating a backup depends on how much data there is in the instance. Backups are stored for as long as you specified in the backup policy.

You do not need to set an interval for incremental backup because TaurusDB for PostgreSQL automatically backs up incremental data every 5 minutes. Incremental backups can be used to restore data to a specific point in time.

Billing

Backups are saved as packages in OBS buckets.

Modifying an Automated Backup Policy

Step 1 **Log in to the management console**.

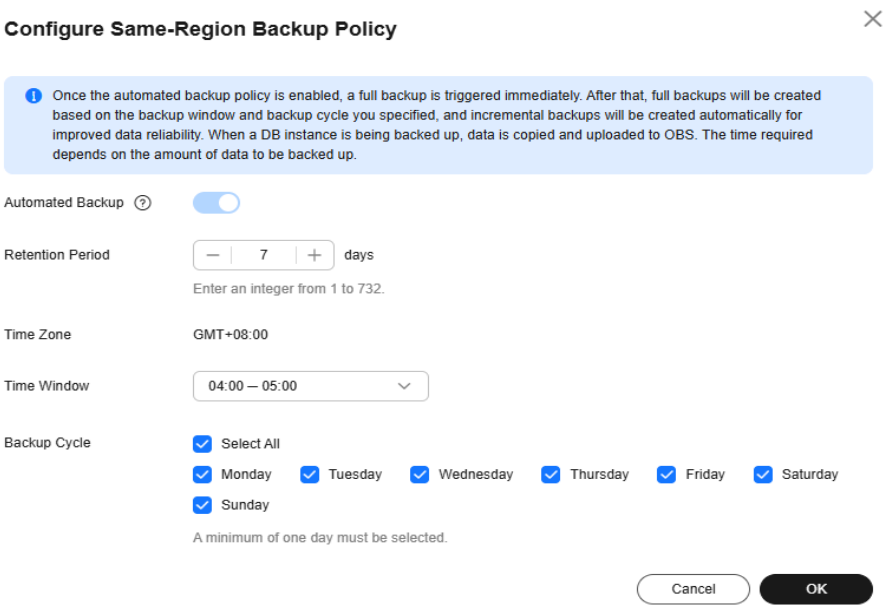
Step 2 Click  in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 On the **Backups** page, click **Configure Same-Region Backup Policy**. On the displayed page, you can view the existing backup policy. If you want to modify the policy, adjust the values of the following parameters.

Figure 8-3 Modifying an automated backup policy



- **Retention Period**: How many days your automated full backups and incremental backups can be retained. The retention period is from 1 to 732 days and the default value is **7**. To extend the retention period, **submit a service ticket**.
  - Extending the retention period improves data reliability.
  - Reducing the retention period takes effect for existing backups. Any backups (except manual backups) that have expired will be automatically deleted. Exercise caution when performing this operation.

  Policy for automatically deleting automated full backups:

  To ensure data integrity, even after the retention period expires, the most recent full backup will be retained, for example, if **Backup Cycle** was set to **Monday** and **Tuesday** and **Retention Period** was set to **2**:
  - The full backup generated on Monday will be automatically deleted on Thursday because:

    The backup generated on Monday expires on Wednesday, but it is the last backup, so it will be retained until a new backup expires. The next backup will be generated on Tuesday and will expire on Thursday. So the full backup generated on Monday will not be automatically deleted until Thursday.
  - The full backup generated on Tuesday will be automatically deleted on the following Wednesday because:

    The backup generated on Tuesday will expire on Thursday, but as it is the last backup, it will be retained until a new backup expires. The next backup will be generated on the following Monday and will expire on the following Wednesday, so the full backup generated on Tuesday will not be automatically deleted until the following Wednesday.
- **Time Window**: A one-hour period the backup will be scheduled for each day, such as 01:00-02:00 or 12:00-13:00. The backup time window indicates when

the backup starts. The backup duration depends on the data volume of your instance.

NOTE

To minimize the potential impact on services, set the time window to off-peak hours. The backup time is in UTC format. The backup time segment changes with the time zone during the switch between the DST and standard time.

- **Backup Cycle**: Daily backups are selected by default, but you can change it. At least one day must be selected.

Step 6 Click **OK**.

----End

8.4.2 Creating a Manual Backup

Scenarios

TaurusDB for PostgreSQL allows you to create manual backups for an available DB instance. You can use these backups to restore data.

Constraints

- When you delete a DB instance, its automated backups are also deleted but its manual backups are retained.
- You can create manual backups only when your account balance is no less than $0 USD.
- The backup name must be unique.

Billing

Backups are saved as packages in OBS buckets.

Method 1

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, locate the target DB instance and choose **More** > **Create Backup** in the **Operation** column.

Step 5 In the displayed dialog box, enter a backup name and description. Then, click **OK**.

- The backup name can contain 4 to 64 characters and must start with a letter. Only letters (case-sensitive), digits, hyphens (-), and underscores (_) are allowed.
- The backup description can consist of a maximum of 256 characters and cannot contain carriage return characters or the following special characters: >!<"&'=

- The time required for creating a manual backup depends on the amount of data.

To check whether the backup has been created, you can click  in the upper right corner of the page to check the DB instance status. If the DB instance status becomes **Available** from **Backing up**, the backup has been created.

Step 6 View and manage the created backup on the **Backups** page.

Alternatively, click the instance name. On the **Backups** page, you can view and manage the manual backup.

----End

Method 2

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 On the **Backups** page, click **Create Backup**. In the displayed dialog box, enter a backup name and description and click **OK**.

- The backup name can contain 4 to 64 characters and must start with a letter. Only letters (case-sensitive), digits, hyphens (-), and underscores (_) are allowed.
- The backup description can consist of a maximum of 256 characters and cannot contain carriage return characters or the following special characters: >!<"&'=
- The time required for creating a manual backup depends on the amount of data.

Step 6 View and manage the created backup on the **Backups** page.

Alternatively, click the instance name. On the **Backups** page, you can view and manage the manual backup.

----End

8.5 Managing Backups

8.5.1 Checking and Exporting Backup Information

Scenarios

You can export backup information of TaurusDB for PostgreSQL instances to an Excel file for further analysis. The exported information includes the DB instance name, backup start and end time, backup status, and backup size.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 In the navigation pane, choose **Backups**. On the displayed page, select the backups you want to export and click to export backup information.

- Only the backup information displayed on the current page can be exported. The backup information displayed on other pages cannot be exported.
- The backup information is exported to an Excel file for your further analysis.

Step 5 Check the exported backup information.

----End

8.5.2 Deleting a Manual Backup

Scenarios

You can delete manual backups to free up backup storage.

Constraints

- Deleted manual backups cannot be recovered.
- Manual backups that are being created cannot be deleted.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 In the navigation pane, choose **Backups**. On the displayed page, locate the manual backup to be deleted and choose **More** > **Delete** in the **Operation** column.

The following backups cannot be deleted:

- Automated backups
- Backups that are being restored
- Backups that are being replicated

Step 5 In the displayed dialog box, click **Yes**.

Step 6 If you have enabled operation protection, click **Send Code** in the displayed **Identity Verification** dialog box and enter the obtained verification code. Then, click **OK**.

Two-factor authentication improves the security of your account and cloud product. For details about how to enable operation protection, see ***Identity and Access Management User Guide***.

----End

9 Data Restorations

9.1 Restoration Solutions

If your database is damaged or mistakenly deleted, you can restore it from backups.

Restoring a Mistakenly Deleted Instance

You can restore a deleted instance from its retained manual backups. For details, see **Restoring a DB Instance from Backups**.

Restoring Mistakenly Deleted or Modified Data of a DB Instance

Table 9-1 Restoration solutions

Solution	Category	Storage Type		Restoration Time Point	Scope		Restore To			Time Required
		Cloud SSD	Extreme SSD	Point in Time When a Backup Was Generated	All Databases and Tables	Certain Databases and Tables	New Instance	Original Instance	Existing Instance Other than the Original Instance	

Restoring an entire instance	Restoration from backups	√	√	x	√	x	√	x	√	Depending on how much data there is in the instance
	Point-in-time recovery (PITR)	√	√	√	√	x	√	x	√	Depending on how much data there is in the instance
Restoring databases and tables	Restoration of databases and tables	√	√	√	x	√	x	√	x	Depending on how much data there is in the instance and in the databases and tables

Restoring or Migrating Data to a TaurusDB for PostgreSQL Instance

You can restore data to a TaurusDB for PostgreSQL instance from backups. For details, see **Restoring Data to TaurusDB for PostgreSQL**.

9.2 Restoring Data to TaurusDB for PostgreSQL

9.2.1 Restoring a DB Instance from Backups

Scenarios

You can use an automated or manual backup to restore data to the state at the time when the backup was created. The restoration is at the DB instance level.

Function Description

Table 9-2 Function description

Item	Description
Restoration scope	The entire instance
Instance data after restoration	The instance data after restoration is consistent with that in the full backup used for the restoration. • Restoring data to a new instance creates an instance with the same data as that in the backup. • Restoring data to an existing instance will overwrite the instance data.
Restoration type	• Restoration to a new instance • Restoration to the original instance • Restoration to an existing instance other than the original one
Configurations for restoring to a new instance	• The DB engine and engine version of the new instance are the same as those of the original instance. • The storage space of the new instance is the same as that of the original instance by default and the new instance must be at least as large as the original instance. • Other parameters need to be reconfigured.
Time required	The time required depends on how much data there is in the instance. The average restoration speed is 40 MB/s.

Constraints

- You can restore to new DB instances from backups only when your account balance is greater than or equal to $0 USD. You will pay for the new instance specifications.
- TaurusDB for PostgreSQL supports restoration to the original DB instance from backups. To use this function, **submit a service ticket**.
- Constraints on restoring data to a new DB instance:

  The new DB instance must use the same Babelfish migration mode as the original DB instance.
- Constraints on restoring data to the original DB instance:
  - If the DB instance for which the backup is created has been deleted, data cannot be restored to the original DB instance.
  - Restoring to the original DB instance will overwrite data on it and cause the DB instance to be unavailable during the restoration.
- Constraints on restoring data to an existing DB instance (other than the original instance):
  - If the target existing DB instance has been deleted, data cannot be restored to it.
  - Restoring to an existing DB instance will overwrite data on it and cause the existing DB instance to be unavailable.
  - To restore backup data to an existing DB instance, the selected DB instance must have the same DB engine, version, and Babelfish migration mode as the original DB instance.
  - Ensure that the storage space of the selected DB instance is greater than or equal to the storage space of the original DB instance. Otherwise, data will not be restored.

Procedure

Step 1 **Log in to the management console**.

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

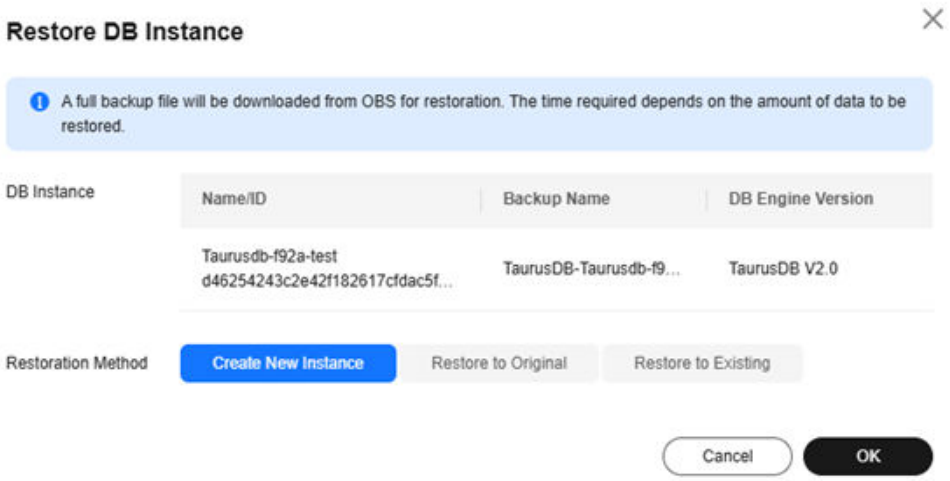
Step 3 Click  in the upper left corner of the page and choose **Databases** > **TaurusDB**.

Step 4 On the **Backups** page, select the backup to be restored and click **Restore** in the **Operation** column.

Alternatively, click the target DB instance on the **Instances** page. In the navigation pane, choose **Backups**. On the displayed page, select the backup to be restored and click **Restore** in the **Operation** column.

Step 5 Select a restoration method and click **OK**.

Figure 9-1 Restoring to a new DB instance



- Create New Instance

  The **Create New Instance** page is displayed.

  - The DB engine and engine version of the new instance are the same as those of the original instance.
  - Storage space of the new DB instance is the same as that of the original DB instance by default and the new instance must be at least as large as the original DB instance.
  - Other settings are the same as those of the original DB instance by default and can be modified. For details, see **Buying a TaurusDB for PostgreSQL Instance**.

- Restore to Original

  1. Select "I acknowledge that after I select Restore to Original, data on the original databases will be overwritten and the original DB instance will be unavailable during the restoration." and click **Next**.

  2. Confirm the information and click **OK**.

- Restore to Existing

Step 6 View the restoration result. The result depends on which restoration method was selected:

- Create New Instance

  A new instance that contains the same data as the backup is created. When the instance status changes from **Creating** to **Available**, the restoration is complete.

  The new instance is independent from the original one.

  After the new instance is created, a full backup will be automatically triggered.

- Restore to Original

  On the **Instances** page, when the instance status changes from **Restoring** to **Available**, the restoration is complete. After the restoration, a full backup will be automatically triggered.

- Restore to Existing

  On the **Instances** page, when the instance status changes from **Restoring** to **Available**, the restoration is complete. After the restoration, a full backup will be automatically triggered.

----End

9.2.2 Restoring a DB Instance to a Point in Time

Scenarios

You can use an automated backup to restore a DB instance to a specified point in time.

When you enter the time point that you want to restore the DB instance to, TaurusDB for PostgreSQL downloads the most recent full backup file from OBS to the DB instance. Then, incremental backups are also restored to the specified point in time on the DB instance. Data is restored at an average speed of 30 MB/s.

Function Description

Table 9-3 Function description

Item	Description
Restoration scope	The entire instance
Instance data after restoration	The instance data after restoration is consistent with that in the full backup plus the incremental backup used for the restoration. • Restoring data to a new instance creates an instance with the same data as that generated by that time point. • Restoring data to an existing instance will overwrite the instance data.
Restorable time point	Any time point within the retention period after the earliest full backup is generated
Scenario	• Restoration to a new instance • Restoration to an existing instance other than the original one
Configurations for restoring to a new instance	• The DB engine and engine version of the new instance are the same as those of the original instance. • Other parameters need to be reconfigured.
Time required	The time required depends on how much data there is in the instance. The average restoration speed is 30 MB/s.

Constraints

- You can restore to new DB instances from backups only when your account balance is greater than or equal to $0 USD. A new DB instance is billed based on the instance specifications.
- Constraints on restoring data to the original DB instance:

  Restoring to the original DB instance will overwrite data on it and cause the DB instance to be unavailable during the restoration.
- Constraints on restoring data to an existing DB instance (other than the original instance):
  - Restoring to an existing DB instance will overwrite data on it and cause the existing DB instance to be unavailable.
  - To restore backup data to an existing DB instance, the selected DB instance must have the same DB engine and version as the original DB instance.
  - Ensure that the storage space of the selected DB instance is no less than that of the original DB instance. Otherwise, data will not be restored.

Procedure

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Backups**. On the displayed page, click **Restore to Point in Time**.

Step 6 Select the restoration date and time range, enter a time point within the selected time range, and select a restoration method. Then, click **OK**.

- Create New Instance

  The **Create New Instance** page is displayed.
  - The DB engine and version of the new DB instance are the same as those of the original DB instance and cannot be changed.
  - Storage space of the new DB instance is the same as that of the original DB instance by default and the new instance must be at least as large as the original DB instance.
  - Other settings are the same as those of the original DB instance by default and can be modified. For details, see **Buying a TaurusDB for PostgreSQL Instance**.
- Restore to Original
  a. Select "I acknowledge that after I select Restore to Original, data on the original databases will be overwritten and the original DB instance will be unavailable during the restoration." and click **Next**.
  b. Confirm the information and click **OK**.
- Restore to Existing

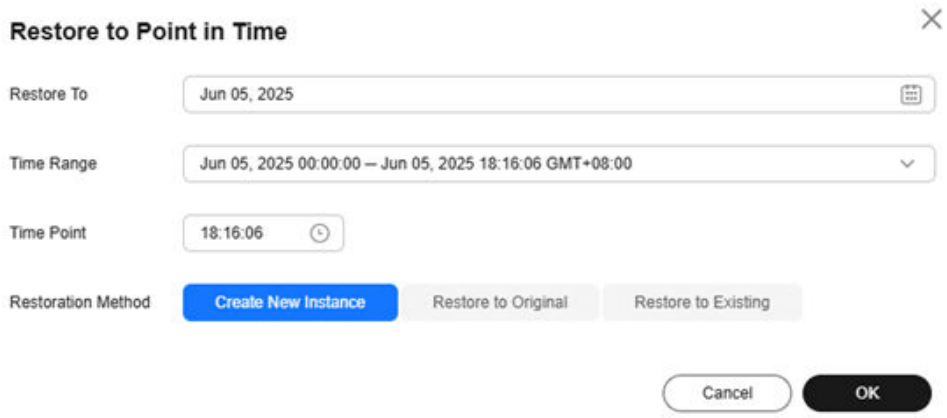
a. Select "I acknowledge that restoring to an existing DB instance will overwrite data on the instance and will cause the existing DB instance to be unavailable during the restoration. Only DB instances that can be used as target instances for the restoration are displayed here. Eligible instances must have the same DB engine type, version, and at least as much storage as the instance being restored." and click **Next**.

b. Confirm the information and click **OK**.

c. If you have enabled operation protection, click **Send Code** in the displayed **Identity Verification** dialog box and enter the obtained verification code. Then, click **OK**.

d. Two-factor authentication improves the security of your account and cloud product. For details about how to enable operation protection, see ***Identity and Access Management User Guide***.

Figure 9-2 Restoring a TaurusDB for PostgreSQL instance to a point in time



Step 7 View the restoration result. The result depends on which restoration method was selected:

- Create New Instance

  A new DB instance is created using the backup data. The status of the DB instance changes from **Creating** to **Available**.

  The new DB instance is independent from the original one.

  A full backup is triggered after the new DB instance is created.

- Restore to Original

  On the **Instances** page, the status of the DB instance changes from **Restoring** to **Available**.

  A new restoration time range is available on the **Restore to Point in Time** page. There will be a difference between the new and original time ranges. This difference reflects the duration of the restoration.

  After the restoration is complete, a full backup will be automatically triggered.

- Restore to Existing

  On the **Instances** page, the status of the DB instance changes from **Restoring** to **Available**.

After the restoration is complete, a full backup will be automatically triggered.

----End

9.2.3 Restoring Databases or Tables to a Point in Time

Scenarios

TaurusDB for PostgreSQL allows you to restore databases or tables using point-in-time recovery (PITR). This ensures your data integrity and minimizes impact on the original instance performance. You can select databases or tables and restore them to a specified point in time. During database or table PITR, TaurusDB for PostgreSQL downloads the most recent full backup from OBS and restores it to a temporary DB instance, and then replays WAL to the specified point in time on the temporary instance. After that, data on the temporary instance is written to the target databases or tables of the original instance. The time required depends on how much data needs to be restored.

The time required depends on the amount of data to be restored on the DB instance. Restoring databases or tables will not overwrite data in the DB instance. You can select the databases or tables to be restored.

TaurusDB for PostgreSQL supports restoration of databases or tables of only one DB instance at a time.

Constraints

- Take care when restoring tables. Improper operations can cause instance or workload exceptions.
- To prevent restoration failures and impact on original data, table-level restoration removes foreign key constraints, inheritance relationships, partition relationships and triggers, and renames indexes and associated sequences. Database-level restoration does not restore subscriptions.
- During table restoration, a maximum of 20,000 tables can be restored for one instance at a time. If the number of tables to be restored exceeds 20,000, you can restore the entire instance using PITR. For details, see **Restoring a DB Instance to a Point in Time**.
- During database restoration, a maximum of 2,000 databases and 20,000 tables can be restored for one instance at a time. If you want to restore more databases or tables at a time, you can restore the entire instance using PITR. For details, see **Restoring a DB Instance to a Point in Time**.
- During a database or table PITR, DB instances cannot be rebooted or deleted.
- In a database or table PITR, the database or table information to be restored is read from the latest full backup before the selected time point. You can select any time point within the restorable time period. Therefore, a database or table can be restored to the earliest full backup time point when its information exists.
- If there is no backup data about the specified tables at the point in time, the restoration will still be completed, but no data of the tables is restored.
- Data is restored at an average speed of 20 MB/s.

- Restoring databases or tables to a specified point in time does not affect new data. The restored database or table is a temporary database or table with a timestamp suffix. You can manage the data in the temporary database or table as required.

Prerequisites

After the restoration, a new database or table will be generated in the DB instance. Ensure that the DB instance has sufficient storage space for the generated database or table.

Restoring Databases or Tables of a DB Instance

Step 1 Log in to the management console.

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

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

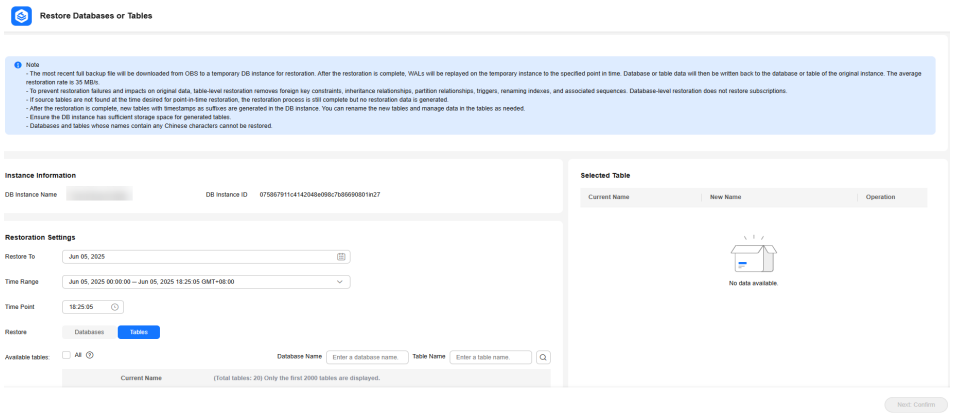
Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Backups**. On the displayed page, click **Restore Databases or Tables**.

Step 6 Specify restoration information and click **Next: Confirm**.

- To facilitate your operations, you can search for the databases or tables to be restored.
- After the restoration is complete, new databases or tables with timestamps appended as suffixes to original database or table names are generated in the DB instance. You can rename the new databases or tables.
- The new table name must be unique and consist of 1 to 64 characters. Only letters, digits, underscores (_), hyphens (-), and dollar signs ($) are allowed.
- Databases whose names contain periods (.) cannot be restored.
- To prevent data loss, new databases with unique names must be specified for database PITR.
- During database PITR, a maximum of 2,000 databases and 20,000 tables can be restored for a single instance at a time.

Figure 9-3 Restoring databases or tables to a point in time



Step 7 On the displayed page, confirm the information and click **Submit**.

Step 8 On the **Instances** page, check that the DB instance status is **Restoring**. During the restoration, services are not interrupted.

You can also view the progress and result of restoring databases or tables to a specified point in time on the **Task Center** page.

After the restoration is successful, you can manage data in the databases or tables as required.

----End

10 Extension Management

10.1 Installing and Uninstalling an Extension on the TaurusDB Console

Scenarios

TaurusDB for PostgreSQL allows you to install and uninstall extensions on the console.

TaurusDB for PostgreSQL extensions only take effect on the databases you created the extensions for. To use an extension on databases, it has to be created separately for each database.

Prerequisites

Before installing or uninstalling extensions, ensure that there are databases in your instance.

Precautions

- plpgsql is a built-in extension and cannot be uninstalled.
- Logical replication extensions, such as wal2json, can be used as-is. There is no installation required.
- Some extensions depend on the **shared_preload_libraries** parameter. They can be installed only after related libraries are loaded.
- Before using the pg_cron extension, change the value of **cron.database_name** to the name of the database this extension is used for (only one database is supported), and change the value of **cron.use_background_workers** to **on**.
- Installing or uninstalling some extensions will cause their dependent extensions and tables to be installed or uninstalled synchronously.

Modifying the shared_preload_libraries Parameter

Some extensions require corresponding parameter values to be loaded before the extensions can be installed.

You can modify the **shared_preload_libraries** parameter to load parameter values in batches or load each required parameter value independently before installing an extension.

Step 1 **Log in to the management console**.

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

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

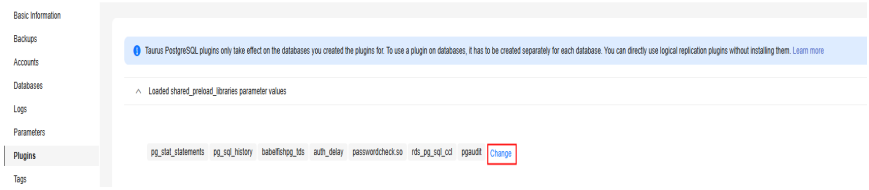
Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Plugins**.

Step 6 On the **Plugins** page, click next to **Loaded shared_preload_libraries parameter values** to view the loaded parameter values.

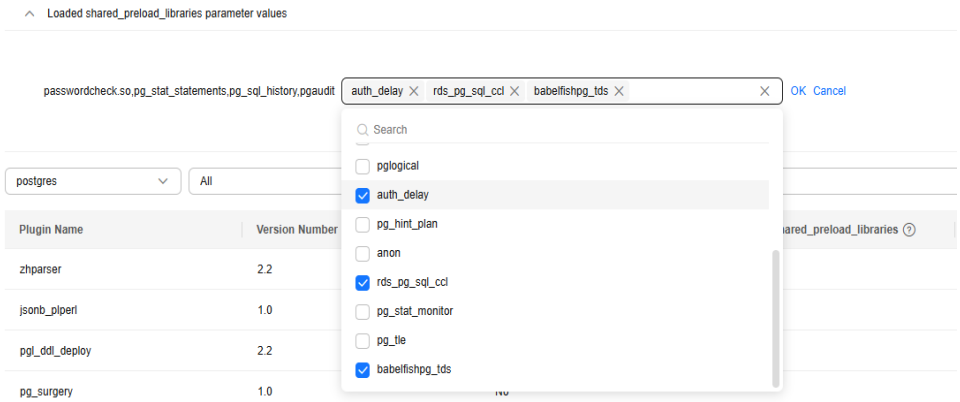
Step 7 Click **Change**.

Figure 10-1 Viewing loaded parameter values



Step 8 Select the parameter values to be loaded from the drop-down list box and click **OK**.

Figure 10-2 Selecting parameter values to be loaded



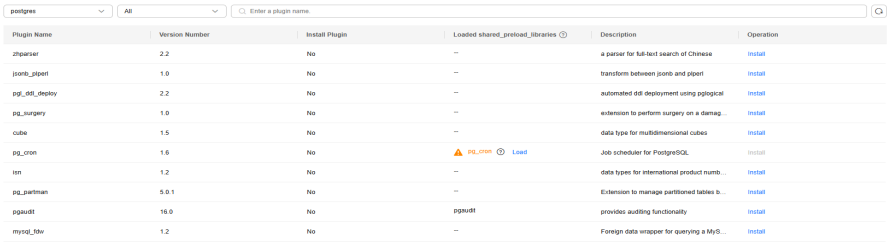
Step 9 In the displayed dialog box, click **OK**.

CAUTION

- The modified parameter values take effect only after the instance is rebooted.
- To ensure security and O&M functions of TaurusDB for PostgreSQL, the following parameter values are loaded by default and cannot be deleted:
  - **passwordcheck.so**
  - **pg_stat_statements**
  - **pg_sql_history**
  - **pgaudit**

Step 10 You can also load each parameter value independently before installing an extension.

Figure 10-3 Loading a parameter value



----End

Installing and Uninstalling an Extension

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Plugins**.

Step 6 In the **Database** drop-down list above the extension list, select the database where the extension is to be installed.

Step 7 Locate the extension to be installed and click **Install** in the **Operation** column.

Step 8 To uninstall an extension, click **Uninstall**.

----End

10.2 Installing and Uninstalling an Extension Using SQL Commands

TaurusDB for PostgreSQL provides the extension management solution for user **root**. Except the following extensions, you need to manually create other extensions by referring to this section.

- auto_explain
- passwordcheck
- pg_profile_pro
- pg_sql_history
- plpgsql
- wal2json
- test_decoding

NOTE

TaurusDB for PostgreSQL extensions only take effect on the databases you created the extensions for. To use an extension on databases, it has to be created separately for each database.

Creating an Extension

Connect to the database where an extension needs to be created as user **root** and run the following SQL statements:

select control_extension('create','*<EXTENSION_NAME>***', '***<SCHEMA>***');**

- *EXTENSION_NAME* indicates the extension name. For more information, see **Supported Extensions**.
- *SCHEMA* indicates the name of the schema where the extension is created. If this parameter is not specified, the **public** schema is used by default.

Example:

Create postgis in the public schema.

```
-- Specify the public schema for creating the extension.
select control_extension('create','postgis', 'public');
      control_extension
 ------------------------------
  create postgis successfully.
 (1 row)
-- If the schema parameter is not specified, the default schema is public.
select control_extension('create', 'postgis');
     control_extension
------------------------------
 create postgis successfully.
(1 row)
```

Deleting an Extension

Connect to the database where an extension needs to be created as user **root** and run the following SQL statements:

select control_extension('drop','*<EXTENSION_NAME>***', '***<SCHEMA>***');**

- *EXTENSION_NAME* indicates the extension name. For more information, see **Supported Extensions**.
- SCHEMA indicates the schema name. This parameter does not matter much when you delete an extension, so you do not need to specify this parameter.

Example:

```
select control_extension('drop','postgis');
     control_extension
 ---------------------------
  drop postgis successfully.
 (1 row)
```

Common Errors

- Error 1
  ```
  ERROR:  permission denied for function control_extension
  ```
  Solution: Use **root** to run the **control_extension** function.
- Error 2
  ```
  ERROR:  function control_extension(unknown, unknown) is not unique
  ```
  Solution: Add the schema parameter in the function. If the schema is not specified, there may be functions with the same name, causing execution failures.
- Error 3
  ```
  ERROR:  function control_extension(unknown, unknown) does not exist
  ```
  Solution: Do not create extensions in the **postgres** database. The **control_extension** function does not exist in the **postgres** database because this database is used as an O&M database.

10.3 Supported Extensions

NOTE

The following table lists the extensions supported by the latest minor versions of TaurusDB for PostgreSQL. You can use **SELECT name FROM pg_available_extensions;** to view the extensions supported by your DB instance.

The extensions mysql_fdw, dblink, pgsql-ogr-fdw, postgres_fdw, and tds_fdw are used to access data stored in remote database servers. Before using any of them, ensure that the server IP addresses of the two DB instances are in the same VPC and subnet.

Supported Extensions

Table 10-1 Supported extensions

Extension Name	PostgreSQL 16
address_standardizer	3.4.3
address_standardizer_data_us	3.4.3
amcheck	1.3
auth_delay	2

Extension Name	PostgreSQL 16
autoinc	1
dblink	1.2
dict_int	1
dict_xsyn	1
fuzzystrmatch	1.2
hll	2.18
insert_username	1
intagg	1.1
intarray	1.5
ip4r	2.4.2
isn	1.2
jsonb_plperl	1
lo	1.1
moddatetime	1
mysql_fdw	2.9.1
orafce	4.14
pageinspect	1.12
passwordcheck	2
pgAudit	16
pg_bigm	1.2
pg_buffercache	1.4
pg_cron	1.6.2
pg_freespacemap	1.2
pg_hint_plan	1.6.1
pg_partman	5.0.1
pg_prewarm	1.2
pg_qualstats	2.1.1
pg_repack	1.5.2
pg_roaringbitmap	0.5.4
pg_stat_statements	1.1

Extension Name	PostgreSQL 16
pg_sugry	1
pg_trgm	1.6
pg_visibility	1.2
pgl_ddl_deploy	2.2.1
pgrouting	3.6.1
pgrowlocks	1.2
pg_sql_history	1.2
pgstattuple	1.5
plpgsql	1
plperl	1
plproxy	2.11.0
postgis	3.4.3
postgis_raster	3.4.3
postgis_sfcgal	3.4.3
postgis_topology	3.4.3
postgres_fdw	1.1
rds_hwdrs_ddl	1
rds_hwdrs_privs	1
rds_pg_sql_ccl	1
rum	1.3.13
seg	1.4
sslinfo	1.2
tablefunc	1
tcn	1
tds_fdw	2.0.4
test_decoding	2
tsm_system_rows	1
tsm_system_time	1
unaccent	1.1
uuid-ossp	1.1

Extension Name	PostgreSQL 16
xml2	1.1

10.4 pg_repack

Scenarios

pg_repack can reorganize tables and indexes with minimal locks to restore the physical order. Unlike CLUSTER and VACUUM FULL it works online, without holding an exclusive lock on the processed tables during processing.

Constraints

- Only the **root** user can use pg_repack.
- The target table must have a primary key or at least a unique total index on a NOT NULL column.
- Performing a full-table repack requires free disk space about twice as large as the target table and its indexes.
- pg_repack cannot reorganize temp tables or cluster tables by GiST indexes.
- You will not be able to perform DDL commands of the target table except VACUUM or ANALYZE while pg_repack is working.
- pg_repack can be used only after a client is deployed locally. For details, see the official documentation at **https://reorg.github.io/pg_repack/**.

How to Use

- Install the extension.
  ```
  select control_extension('create', 'pg_repack');
  ```
- Delete the extension.
  ```
  select control_extension('drop', 'pg_repack');
  ```

For more information, see **Installing and Uninstalling an Extension on the TaurusDB Console** and **Installing and Uninstalling an Extension Using SQL Commands**.

Example

Use pg_repack to repack a table.

1. **Create a test table pg_repack_test.**
   ```
   create table pg_repack_test(id bigint primary key, name varchar);
   insert into pg_repack_test select i , to_char(random()*100000, 'FM000000') from generate_series(1, 1000000) i;
   delete from pg_repack_test where id in (select i  from generate_series(1, 600000, 2) i);
   select pg_size_pretty(pg_relation_size('pg_repack_test'));
   ```
2. **Repack the test table.**
   ```
   pg_repack --host=<TaurusDB_ADDRESS> --port=<DB_PORT> --dbname=<DB_NAME> --username=root --no-superuser-check --no-kill-backend -t pg_repack_test
   ```
   - *TaurusDB_ADDRESS*: IP address of the TaurusDB for PostgreSQL instance

- *DB_PORT*: port of the TaurusDB for PostgreSQL instance
- *DB_NAME*: name of the database where the **pg_repack_test** table is located

3. **Check the size of the repacked table.**

```
select pg_size_pretty(pg_relation_size('pg_repack_test'));
```

FAQs

Table 10-2 Common error information and solutions

Error Information	Solution
ERROR: pg_repack failed with error: ERROR: permission denied for schema repack	Use the **root** user.
ERROR: pg_repack failed with error: You must be a superuser to use pg_repack	Add **--no-superuser-check** to skip superuser checks.
NOTICE: Waiting for 1 transactions to finish. First PID: xxxx	Wait until the transaction is complete.
ERROR: pg_repack failed with error: program 'pg_repack 1.5.2' does not match database library 'pg_repack 1.5.0'	Ensure that the pg_repack client tool version is the same as the pg_repack extension version installed on the PostgreSQL server.

10.5 pgl_ddl_deploy

Introduction

There are many databases that require replicating data to other databases for various purposes. One of the most useful database technologies that is used to move data from point A to point B is called "logical replication". In database jargon, there are two categories of SQL statements: DML and DDL. For a number of reasons, DDL has to be handled separately. During the migration, the DBA is required to manually deploy the SQL in the correct order for all involved database clusters, manage locking contention, and add new tables to replication if necessary. Built on top of pglogical, pgl_ddl_deploy enables any DDL SQL statement to be directly propagated to subscribers. This solves the problem that pglogical cannot synchronize DDL statements.

For more information, see **official pgl_ddl_deploy documentation**.

Features

TaurusDB for PostgreSQL supports the **pgl_ddl_deploy** extension, which is used to automatically synchronize DDL statements. In many cases, most DDL statements executed in application environments can be synchronized.

- Any DDL statement can be synchronized to subscribers.
- Tables can be automatically added to replication upon creation.
- Filtering by regular expression or a specific set of tables is supported.
- There is an option to deploy in a lock-safe way on subscribers.
- There is an option to fail certain events on the subscriber to be retried later.
- In some edge cases, alerting can be built around provided logging for the DBA to then handle possible manual deployments.
- **ALTER TABLE** statements can be filtered by subcommand tags.
- Support for automatically killing blocking processes that are preventing DDL execution on the subscriber system is optional.

Extension Installation and Uninstallation

- Installing the extension
  ```
  SELECT control_extension ('create', 'pgl_ddl_deploy');
  ```
- Deleting the extension
  ```
  SELECT control_extension ('drop', 'pgl_ddl_deploy');
  ```

For more information, see **Installing and Uninstalling an Extension on the TaurusDB Console** and **Installing and Uninstalling an Extension Using SQL Commands**.

Basic Usage

This extension involves publication and subscription and depends on pglogical. You need to add and configure parameters.

```
wal_level = 'logical'
shared_preload_libraries = 'pglogical'
```

For details about how to modify the **shared_preload_libraries** parameter, see **Modifying the shared_preload_libraries Parameter**.

- Configuring parameters on the provider

```
SELECT control_extension ('create', 'pglogical');
SELECT control_extension ('create', 'pgl_ddl_deploy');
CREATE TABLE foo (id INT PRIMARY KEY);
```

- Creating a publication
  ```
  CREATE PUBLICATION testpub FOR TABLE foo;
  ```
- Configuring a replication set
  ```
  INSERT INTO pgl_ddl_deploy.set_configs (set_name, include_schema_regex, driver) VALUES ('testpub',
  '.*', 'native'::pgl_ddl_deploy.driver);
  ```
- Deploying the publication
  ```
  SELECT pgl_ddl_deploy.deploy('testpub');
  ```
- Adding roles for the user
  ```
  SELECT pgl_ddl_deploy.add_role(oid) FROM pg_roles WHERE rolname='root';
  ```
- Configuring parameters on the subscriber
  ```
  SELECT control_extension ('create', 'pglogical');
  SELECT control_extension ('create', 'pgl_ddl_deploy');
  CREATE TABLE foo (id INT PRIMARY KEY);
  ```
- Creating a subscription
  ```
  CREATE SUBSCRIPTION testsub CONNECTION conninfo PUBLICATION testpub;
  ALTER SUBSCRIPTION testsub REFRESH PUBLICATION;
  ```


After the configuration is complete, run the following DDL statements on the provider:

```
ALTET TABLE foo ADD COLUMN bla INT;
CREATE TABLE bra (id INT PRIMARY KEY);
```

You can verify the following on the subscriber:

```
\d foo
                Table "public.foo"
 Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
 bla    | integer |           |          |
Indexes:
    "foo_pkey" PRIMARY KEY, btree (id)
\dt
          List of relations
 Schema |  Name   | Type  |  Owner
--------+---------+-------+----------
 public | bar     | table | root
 public | foo     | table | root
(2 rows)
```

Restrictions

This extension has some limitations. Although most DDL statements executed in application environments can be synchronized, it does not cover 100% of edge cases.

DDL Involving Multiple Tables

A single DDL SQL statement which alters both replicated and non-replicated tables cannot be supported. For example, if you set the **include_schema_regex** parameter to **'^replicated.*'**:

```
DROP TABLE replicated.foo, notreplicated.bar;
```

The following message will be displayed on the provider:

```
WARNING: Unhandled deployment logged in pgl_ddl_deploy.unhandled
DROP TABLE
```

The **replicated.foo** table exists on the subscriber.

```
\d replicated.foo
              Table "replicated.foo"
Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
 id     | integer |           | not null |
Indexes:
    "foo_pkey" PRIMARY KEY, btree (id)
```

Similarly, if filtered replication is used, an error may occur when you run the following statement:

```
ALTER TABLE replicated.foo ADD COLUMN bar_id INT REFERENCES notreplicated.bar (id);
```

The statement is not synchronized to the subscriber.

```
\d replicated.foo
              Table "replicated.foo"
Column |  Type   | Collation | Nullable | Default
--------+---------+-----------+----------+---------
id     | integer |           | not null |
```



```
Indexes:
    "foo_pkey" PRIMARY KEY, btree (id)
```

Unsupported Commands

CREATE TABLE AS and SELECT INTO are not supported to replicate DDL due to limitations on transactional consistency. That is, if a table is created from a set of data on the provider, running the same SQL on the subscriber will in no way guarantee consistent data. For example:

```
CREATE TABLE foo AS
SELECT field_1, field_2, now() AS refreshed_at
FROM table_1;
```

Similar to CREATE TABLE AS, the following message will be displayed for SELECT INTO:

```
WARNING:  Unhandled deployment logged in pgl_ddl_deploy.unhandled
```

Multi-Statement Client SQL Limitations

The complexities and limitations come when the client sends all SQL statements as one single string to PostgreSQL. Assume the following SQL statements:

```
CREATE TABLE foo (id serial primary key, bla text);
INSERT INTO foo (bla) VALUES ('hello world');
```

If this was in a file that was called using psql, it would run as two separate SQL command strings. However, if in Python or Ruby's ActiveRecord you create a single string as above and execute it, then it would be sent to PostgreSQL as one single SQL command string. The replication depends on the value of the **allow_multi_statements** parameter:

- If the value is **false**, pgl_ddl_deploy will only auto-replicate a client SQL statement containing one command tag that matches the event trigger command tag. That is really safe, but it means you may have a lot more unhandled deployments.
- If the value is **true**, pgl_ddl_deploy will only auto-replicate DDL that contains **safe** command tags to propagate. For example, mixed DDL and DML is forbidden. If a command contains more than two DDL statements and the statements are used on both replicated and non-replicated tables, the problem described in **DDL Involving Multiple Tables** occurs.

In any case that a SQL statement cannot be automatically run on the subscriber based on these analyses, instead it will be logged as a WARNING and put into the unhandled table for manual processing. For more details and solutions to problems that occur during replication, see **official pgl_ddl_deploy documentation**.

10.6 pg_stat_statements

Introduction

The pg_stat_statements extension provides a means for tracking planning and execution statistics of all SQL statements executed by a server.

For more information, see **pg_stat_statements** in the PostgreSQL documentation.

Supported Versions

You can run the following SQL statement to check whether your DB instance supports this extension:

```
SELECT * FROM pg_available_extension_versions WHERE name  = 'pg_stat_statements';
```

To see more extensions supported by TaurusDB for PostgreSQL, go to **Supported Extensions**.

Extension Installation and Uninstallation

To check whether pg_stat_statements is installed in a database, run the following SQL statement:

```
select * from pg_extension  where extname = 'pg_stat_statements';
```

If the command output is empty, the extension is not installed. If the extension information is displayed, the extension is installed.

By default, pg_stat_statements is preloaded in the **shared_preload_libraries** parameter. Perform the following steps to install or delete the extension:

- Installing the extension
  ```
  SELECT control_extension('create', 'pg_stat_statements');
  ```
- Deleting the extension
  ```
  SELECT control_extension('drop', 'pg_stat_statements');
  ```

For more information, see **Installing and Uninstalling an Extension on the TaurusDB Console** and **Installing and Uninstalling an Extension Using SQL Commands**.

Basic Functions

1. After the pg_stat_statements extension is installed, configure the parameters below. You can adjust the values based on your workload requirements.

   Table 10-3 Parameter description

Parameter	Reboot Required	Default Value	Allowed Values	Description
pg_stat_statements.max	Yes	5000	100-5,000,000	Specifies the maximum number of statements tracked by pg_stat_statements.
pg_stat_statements.save	No	on	on, off	Specifies whether to save statement statistics across server shutdowns.
pg_stat_statements.track	No	top	top, all, none	Controls which statements are counted by pg_stat_statements.

Parameter	Reboot Required	Default Value	Allowed Values	Description
pg_stat_statements.track_planning	No	off	on, off	Controls whether planning duration is tracked by pg_stat_statements.
pg_stat_statements.track_utility	No	on	on, off	Controls whether utility commands are tracked by pg_stat_statements.

2. Query the pg_stat_statements view to obtain statistics.
```
select * from pg_stat_statements;
```
3. Query the SQL statements with high I/O consumption.

   Top 5 SQL statements

```
select userid::regrole, dbid, query
from pg_stat_statements
order by (blk_read_time+blk_write_time) desc limit 5;
```
4. Query the SQL statements with high consumption of shared memory.
```
select userid::regrole, dbid, query
from pg_stat_statements
order by (shared_blks_hit+shared_blks_dirtied) desc limit 5;
```
5. Reset the statistics.
```
select pg_stat_statements_reset();
```

Advanced Functions

You can use pg_stat_statements to troubleshoot high CPU usage. The process is as follows:

1. Reset the pg_stat_statements counter.
```
select pg_stat_statements_reset();
```
   Leave enough time for pg_stat_statements to collect information.
2. Obtain the most time-consuming SQL statements.
```
select * from pg_stat_statements order by total_exec_time desc limit 10;
```
   The obtained SQL statements have been occupying the user-mode CPU for a long time. Analyze these SQL statements.
3. Obtain the SQL statements that read the buffer for the most times.
```
select * from pg_stat_statements order by shared_blks_hit + shared_blks_read desc limit 10;
```
   The obtained SQL statements may cause too many buffer reads due to a lack of indexes, consuming a large number of CPU resources.
4. Obtain the SQL statements that have been executed for the most times.
```
select * from pg_stat_statements order by calls desc limit 10;
```
   It takes a short time to execute some simple SQL statements separately. However, in some cases (for example, cyclic executions in a transaction or concurrent executions), the CPU usage increases.

10.7 pg_cron

Introduction

The pg_cron extension is a cron-based job scheduler. It uses the same syntax as regular cron, but it allows you to schedule PostgreSQL commands directly from the database. For more information, see the **official pg_cron documentation**.

To see more extensions supported by TaurusDB for PostgreSQL, go to **Supported Extensions**.

Features

The standard cron syntax is used, in which the asterisk (*) means "run every time period", and a specific number means "but only at this time".

```
┌──────────────── min (0 - 59)
│ ┌────────────── hour (0 - 23)
│ │ ┌──────────── day of month (1 - 31)
│ │ │ ┌────────── month (1 - 12)
│ │ │ │ ┌──────── day of week (0 - 6) (0 to 6 are Sunday to
│ │ │ │ │         Saturday, or use names; 7 is also Sunday)
│ │ │ │ │
```

For example, the syntax for 9:30 a.m. (GMT) every Saturday is as follows:

```
30 9 * * 6
```

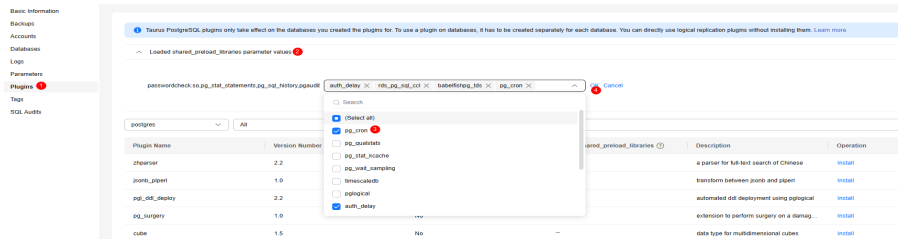
Precautions

- pg_cron requires a daemon process. Therefore, before starting a database, you need to add pg_cron to the **shared_preload_libraries** parameter value.
- Scheduled jobs do not run on the standby instance. However, if the standby instance is promoted to primary, scheduled jobs automatically start.
- Scheduled jobs are executed with the permissions of the job creator.
- Scheduled jobs are executed using the GMT time.
- An instance can run multiple jobs concurrently, but one job can only run once at a given time.
- If a job needs to wait for the completion of the previous scheduled job, it will enter the wait queue and will be started as soon as possible after the previous job completes.
- Before using this extension, you need to change the value of **cron.database_name** to the name of the database where scheduled jobs are created. This parameter can only be set to one database name instead of multiple names.

Installing the Extension

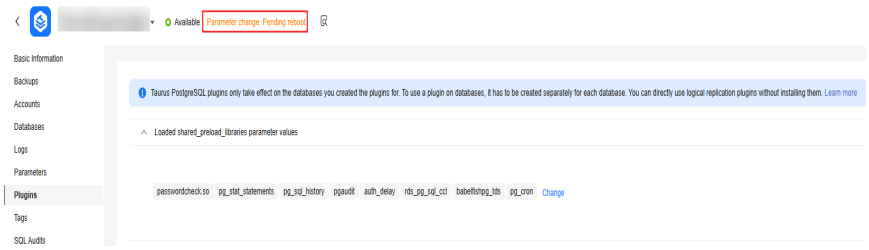
1. In the instance list, click the target instance name to go to the **Basic Information** page.
2. Choose **Plugins** and add pg_cron to the **shared_preload_libraries** parameter values.

Figure 10-4 Plugins



3. Reboot the DB instance for the modification to be applied.

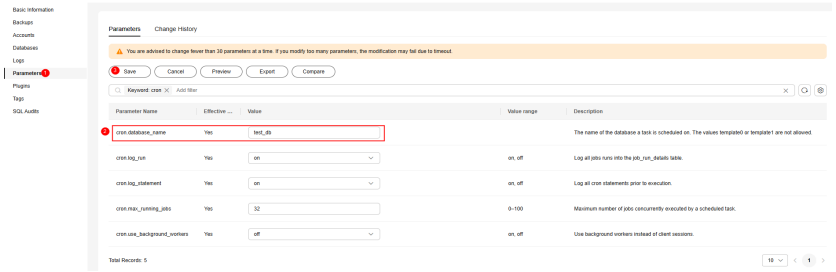
Figure 10-5 Extension added



4. If the extension is not to be installed in the default database postgres, modify the **cron.database_name** parameter. After modifying **cron.database_name**, reboot the instance for the modification to be applied.

   The value of **cron.database_name** must be changed to the name of the database where pg_cron is to be used. For example, if you want to install the extension in the test_db database, change the value of **cron.database_name** to **test_db**.

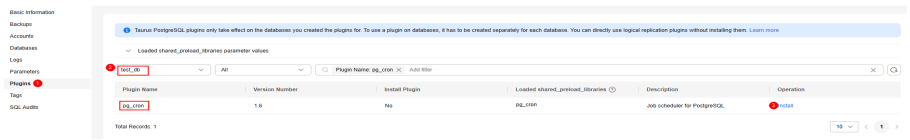
Figure 10-6 Changing the parameter value



5. Install the pg_cron extension.
   - Installing pg_cron on the console

     On the **Plugins** page, select the desired database, search for pg_cron, and click **Install**.

Figure 10-7 Installing pg_cron



- Installing pg_cron using SQL statements

  Log in to the desired database and run the following SQL statement to create the extension:

  ```
  CREATE EXTENSION IF NOT EXISTS pg_cron;
  ```

Basic Usage

Create jobs.

```
-- Job 1: Delete old data at 03:30 a.m. (GMT) every Saturday.
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval '1 week'$
$);
-- Job 2: Run the VACUUM command at 10:00 a.m. (GMT) every day. The job is named nightly-vacuum.
SELECT cron.schedule('nightly-vacuum', '0 10 * * *', 'VACUUM');
```

Advanced Usage (Configuring Scheduled Jobs for Databases Other Than postgres)

Prerequisites: The pg_cron extension has been installed in the postgres database, and the test_db database has been created. Configure scheduled jobs for the test_db database.

1. Log in to the postgres database.
2. Create scheduled jobs.

   ```
   SELECT cron.schedule('create', '10 * * * *', 'create table test (a int);');
   SELECT cron.schedule('insert', '15 * * * *', 'insert into test values(1);');
   SELECT cron.schedule('drop', '20 * * * *', 'drop table test;');
   ```

   NOTE

   Each scheduled job name must be unique. Otherwise, the job will be overwritten.

3. Update the scheduled jobs to run them in the test_db database.

   ```
   UPDATE cron.job SET database = 'test_db' WHERE jobid = 1;
   UPDATE cron.job SET database = 'test_db' WHERE jobid = 2;
   UPDATE cron.job SET database = 'test_db' WHERE jobid = 3;
   ```

   To query the job IDs, run the following commands:

   ```
   postgres=> select * from cron.job;
    jobid |  schedule  |           command           | nodename  | nodeport | database  | username | active |
   jobname
   -------+------------+-----------------------------+-----------+----------+-----------+----------+--------+---------
       1 | 10 * * * * | create table test (a int);  | localhost |     5432 | postgres  | root     | t      | create
       2 | 15 * * * * | insert into test values(1); | localhost |     5432 | postgres  | root     | t      | insert
       3 | 20 * * * * | drop table test;            | localhost |     5432 | postgres  | root     | t      | drop
   ```

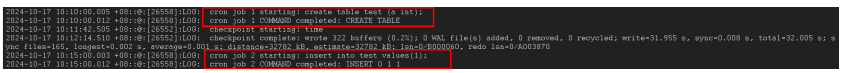
4. In the command output, the value of **database** is **test_db**, indicating that the jobs are executed in the test_db database.

   ```
   postgres=> select * from cron.job;
    jobid |  schedule  |           command           | nodename  | nodeport | database  | username | active |
   jobname
   -------+------------+-----------------------------+-----------+----------+-----------+----------+--------+---------
       1 | 10 * * * * | create table test (a int);  | localhost |     5432 | test_db   | root     | t      | create
       2 | 15 * * * * | insert into test values(1); | localhost |     5432 | test_db   | root     | t      | insert
       3 | 20 * * * * | drop table test;            | localhost |     5432 | test_db   | root     | t      | drop
   ```

5. Check whether the scheduled jobs have been successfully executed.

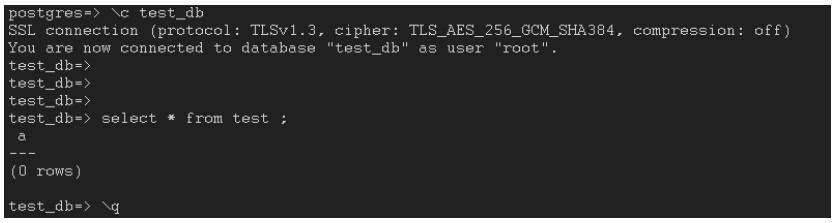
   - Check the logs for successful job execution.

     Figure 10-8 Checking logs

– Check whether the **test** table has been created in the test_db database.

Figure 10-9 Querying the database



6. If you do not want to use the scheduled jobs anymore, delete them based on the result obtained in **4**:
   – Delete a job using the job ID.
   ```
   SELECT cron.unschedule(1);
   ```
   – Delete a job using the job name.
   ```
   SELECT cron.unschedule('create');
   ```

Parameters Related to the pg_cron Extension

Table 10-4 Parameter description

Parameter	Description	Default Value	Reboot Required
cron.database_name	The database in which scheduled job metadata is kept.	postgres	Yes
cron.log_statement	Whether to log all cron statements before running them.	true	Yes
cron.log_run	Whether to log every job that runs in the job_run_details table.	true	Yes
cron.host	The name of the host where scheduled jobs are to be executed.	localhost	Yes
cron.use_background_workers	Whether to use background work processes instead of client sessions to run jobs.	false	Yes
cron.max_running_jobs	The number of jobs that can run concurrently.	5	Yes

10.8 rds_pg_sql_ccl

Introduction

To prevent highly concurrent SQL statements that consume too many resources from causing instance instability, TaurusDB for PostgreSQL provides rds_pg_sql_ccl,

a Huawei-developed extension. ccl is short for concurrent control. SQL statement concurrency control can ensure instance stability, optimize performance, and guarantee resources for critical tasks in the following scenarios:

- When workloads increase sharply, the instance stability is ensured by limiting the execution of a certain type of SQL statements.
- When there are not enough resources, the success of core tasks is ensured by limiting the execution of other SQL statements to reduce resource consumption.

This extension provides two concurrency control methods:

- Method 1: It limits the number of SQL statements that can be executed at the same time. This number is specified by the **rds_pg_sql_ccl.max_concurrent_sql** parameter. The default value is **-1**, indicating that the number is not limited.
- Method 2: It limits the number of a certain type of SQL statements (with the same query ID) that can be executed at the same time. This number is specified by concurrency control rules. For details about concurrency control rules, see below.

To see more extensions supported by TaurusDB for PostgreSQL, go to **Supported Extensions**.

Notes

SQL statement concurrency control rules must be configured based on workloads and resource usage.

Creating Rules

1. In any given database, each rule must have a unique query ID, but rules in different databases can have the same query ID.
2. A rule does not take effect immediately after being created. You need to call the **enable_ccl_rule** function to make it applied.
3. The **get_query_id** function cannot obtain the query IDs of SQL statements using bind variables, and the **add_ccl_rule_by_query** function cannot limit the execution of such SQL statements.
4. You can obtain the query ID of a SQL statement with bind variables using the pg_stat_statements extension. Then, you can use **add_ccl_rule_by_queryid** to create rules. For details, see "Concurrency Control for SQL Statements with Bind Variables".

How Rules Take Effect

1. Method 1 limits the number of SQL statements that can be concurrently executed. Rules creating using this method take effect preferentially. On the basis of method 1, you can use method 2 to further limit concurrent execution of a specific type of SQL statements.
2. After an instance reboot, none of the rules are applied.
3. A rule is only applied to SQL statements executed after the rule was enabled.

Concurrency Control for SQL Statements with Bind Variables

Drivers such as JDBC support prepared statements. They precompile parameterized SQL statements and execute them after parameters are entered. In the pg_stat_statements view, the statements are displayed as bind variables. For SQL statements using bind variables, the query IDs calculated by the kernel are different from those of the same statements using actual parameter values. The concurrent execution of such statements cannot be limited by adding the statements.

Such SQL statements can only be limited by executing them first and then adding concurrency control rules.

1. Execute a SQL statement with bind variables. In this way, the kernel calculates its query ID. An example of the JDBC-based prepared statement program is as follows:
   ```
   String sql = "select pg_sleep(?);";
   PreparedStatement preparedStatement = conn.prepareStatement(sql);
   preparedStatement.setInt(1, 500);
   ResultSet resultSet = preparedStatement.executeQuery();
   ```
2. Query the query ID of the SQL statement in the pg_stat_statements view.
   ```
   select queryid from pg_stat_statements where query like '%select pg_sleep%';
   ```
3. Add a concurrency control rule based on the query ID.
   ```
   select rds_pg_sql_ccl.add_ccl_rule_by_queryid($queryid);
   ```
4. Enable the rule based on the return value (**rule_id**) of the previous SQL statement.
   ```
   select rds_pg_sql_ccl.enable_ccl_rule($rule_id);
   ```
5. Obtain all rules applied from the get_all_enabled_rule view provided by the extension.
   ```
   select * from rds_pg_sql_ccl.get_all_enabled_rule;
   ```

Parameters

Table 10-5 Parameter description

Parameter	Data Type	Default Value	Maximum Value	Minimum Value	Description
rds_pg_sql_ccl.enable_ccl	bool	false	-	-	Whether to enable a concurrency control rule.
rds_pg_sql_ccl.max_enabled_rules	int	5000	500000	0	The number of rules that take effect at the same time.

Parameter	Data Type	Default Value	Maximum Value	Minimum Value	Description
rds_pg_sql_ccl.max_concurrent_sql	int	-1	50000	-1	The number of SQL statements that can be concurrently executed (its priority is higher than that of concurrency control rules). If the value is less than 0, the number of SQL statements is not limited.

Function Interface Description

Table 10-6 Function interface description

Function	Parameter	Return Value	Description
rds_pg_sql_ccl.get_query_id	query_string text, search_path text default 'public'	queryid	Calculates the query ID of a SQL statement.
rds_pg_sql_ccl.add_ccl_rule_by_query	query_string text, max_concurrency int default 0, max_waiting int default 0, search_path text default 'public'	ruleid	Adds a concurrency control rule using SQL statements.

Function	Parameter	Return Value	Description
rds_pg_sql_ccl.add_ccl_rule_by_queryid	query_id bigint, max_concurrency int default 0, max_waiting int default 0, search_path text default 'public'	ruleid	Adds a concurrency control rule based on the query ID.
rds_pg_sql_ccl.enable_ccl_rule	rule_id bigint	bool	Enables a concurrency control rule based on the rule ID.
rds_pg_sql_ccl.disable_ccl_rule	rule_id bigint	bool	Disables a concurrency control rule based on the rule ID.
rds_pg_sql_ccl.disable_all_ccl_rule	-	void	Disables all rules.
rds_pg_sql_ccl.delete_ccl_rule	rule_id bigint	void	Deletes a concurrency control rule based on the rule ID.
rds_pg_sql_ccl.update_ccl_rule	new_rule_id bigint, new_max_concurrency int, new_max_waiting int	void	Updates a concurrency control rule based on the rule ID.

Description of some parameters:

- **max_concurrency**: The maximum number of SQL statements that can be concurrently executed.
- **max_wait**: The maximum waiting time after which new SQL statements of a specified type will fail to be executed when the maximum number of concurrent SQL statements is reached.
- **new_max_concurrency**: The new maximum number of concurrent SQL statements.
- **new_max_wait**: The new maximum waiting time.

View Interface Description

Table 10-7 View interface description

No.	View	Column	Description
1	rds_pg_sql_ccl.get_all_enabled_rule	dbid oid, queryid bigint, max_concurrency int, max_wait int	Displays all concurrency control rules applied.
2	rds_pg_sql_ccl.get_activity_query_status	queryid bigint, wait_start_time timestamptz, pid int, dbid oid	Displays the status of each SQL statement in the current instance, such as the query ID and whether the SQL statement is waiting.
3	rds_pg_sql_ccl.get_current_db_ccl_rule	rule_id bigint, query_id bigint, query_string, max_concurrency int, max_waiting int, search_path text, create_time timestamptz, enabled bool	Displays the concurrency control rules created for the current database (whether applied or not).

10.9 pgAudit

Introduction

Financial institutions, government agencies, and many industries need to keep audit logs to meet regulatory requirements. By using the PostgreSQL Audit Extension (pgAudit) with your TaurusDB for PostgreSQL instance, you can capture detailed records that auditors usually need to meet compliance regulations. For example, you can use pgAudit to track changes made to specific databases and tables, as well as record users who make such changes and many other details.

For more information, see the **official pgAudit documentation**.

Extension Installation and Uninstallation

- Installing the extension
  ```
  SELECT control_extension ('create', 'pgaudit');
  ```


- Deleting the extension
```
SELECT control_extension ('drop', 'pgaudit');
```

How to Use

Configuring the pgAudit

1. First, preload pgAudit on the **Plugins** page of your instance because pgAudit installs event triggers for auditing data definition language (DDL) statements. By default, pgAudit is preloaded. To check whether it is successfully loaded, you can run the following command:
```
show shared_preload_libraries;
                              shared_preload_libraries
-------------------------------------------------------------------------------
 pg_stat_statements,pgaudit,passwordcheck.so,pg_sql_history,auth_delay,pglogical
(1 row)
```
2. After the extension is loaded, install it by referring to **Extension Installation and Uninstallation**.
3. After the extension is installed, enable audit logging.

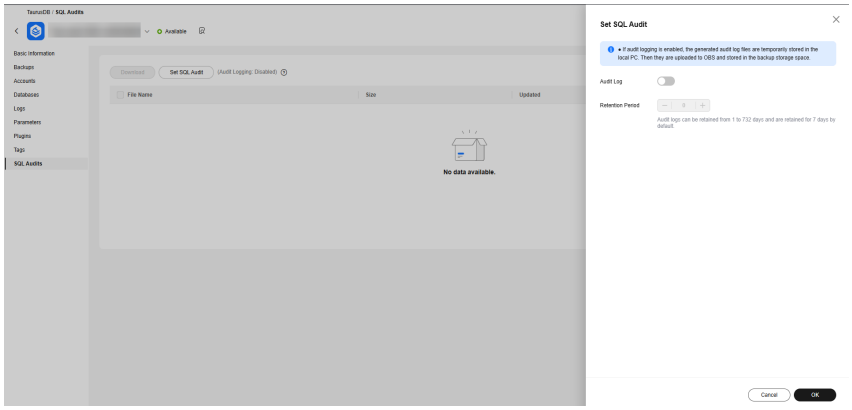
   To enable audit logging, **submit a service ticket** to request required permissions.

   a. On the TaurusDB console, click the DB instance name. On the displayed page, click **SQL Audits**.

   b. Click **Set SQL Audit**.

   c. In the displayed dialog box, toggle on the audit log switch and set the number of days to retain audit logs.

   Figure 10-10 Setting SQL audit



4. Configure parameters.

   Go to the **Parameters** page, search for the **pgaudit.log** parameter (that specifies which types of statements will be logged by session audit logging), and set it to an appropriate value to capture log insertions, updates, deletions, and other changes. The following table explains the values of **pgaudit.log**.

Table 10-8 Parameter description

Value	Description
NONE	(Original value) Specifies that no changes to the database will be recorded.
ALL	Specifies that all changes will be recorded, including READ, WRITE, FUNCTION, ROLE, DDL, and MISC.
DDL	Specifies that all DDL statements (excluding those in the ROLE class) will be recorded.
FUNCTION	Specifies that function calls and DO blocks will be recorded.
MISC	Specifies that commands such as DISCARD, FETCH, CHECKPOINT, VACUUM, and SET will be recorded.
READ	Specifies that SELECT and COPY will be recorded when the source is a relationship (for example, a table) or query.
role	Specifies that statements related to roles and permissions will be recorded, for example, GRANT, REVOKE, CREATE ROLE, ALTER ROLE, and DROP ROLE.
WRITE	Specifies that INSERT, UPDATE, DELETE, TRUNCATE, and COPY will be recorded when the destination is a relationship (table).

The following table lists other parameters related to pgAudit. You can set them on the console as needed.

Table 10-9 Parameter description

Parameter	Description
pgaudit.log	Specifies which types of statements will be logged by session audit logs.
pgaudit.log_catalog	Specifies that session logging should be enabled if all relations in a statement are in pg_catalog.
pgaudit.log_client_authentication	Controls whether to record user authentication information.
pgaudit.log_extra_field	Controls whether to record fields such as PID, IP, user name, and database.
pgaudit.log_file_rotation_age	Sets the rotation interval for separate audit logs.
pgaudit.log_parameter	Specifies that audit logging should include the parameters that were passed with the statement.

Parameter	Description
pgaudit.log_relation	Specifies whether session audit logging should create a separate log entry for each relationship (such as a table and view) referenced in a SELECT or DML statement.
pgaudit.log_rows	Sets the retrieved or affected rows that audit logs should include.
pgaudit.log_write_txid	Controls whether to record the TXID of write operations (such as INSERT and UPDATE).
pgaudit.logstatementonce	Controls whether audit logs include statements, text, and parameters.
pgaudit.log_client	Controls whether audit logs are sent to clients.
pgaudit.log_level	Sets the log level for log entries.
pgaudit.write_into_pg_log_file	Controls whether to write audit information into PostgreSQL run logs.

To display audit logs on your client, configure the following parameters:

- Set both **pgaudit.write_into_pg_log_file** and **pgaudit.log_client** to **on** and select a log level (for example, **notice**) to be displayed on the client based on the value of **pgaudit.log_level**. When you query audit logs on your client again, the logs of the corresponding level are displayed.
- If either **pgaudit.write_into_pg_log_file** or **pgaudit.log_client** is set to **off**, audit logs will not be displayed on the client.
- **pgaudit.log_level** is available only when **pgaudit.log_client** is set to **on**.

11 Security and Encryption

11.1 Database Account Security

Password Strength Requirements

- For information about the database password strength requirements on the Huawei Cloud TaurusDB for PostgreSQL console, see the database configuration table in **Buying a TaurusDB for PostgreSQL Instance**.
- Huawei Cloud TaurusDB for PostgreSQL has a password security policy for user-created database accounts. Passwords must:
  - Consist of at least eight characters.
  - Contain letters, digits, and special characters.
  - Not contain the username.

SSL Encryption

SSL is enabled by default for TaurusDB for PostgreSQL instances and cannot be disabled.

Suggestions for Creating Users

When you run **CREATE USER** or **CREATE ROLE**, you are advised to specify a password expiration time with the **VALID UNTIL 'timestamp'** parameter (**timestamp** indicates the expiration time).

Suggestions for Accessing Databases

When you access a database object, you are advised to specify the schema name of the database object to prevent **trojan-horse attacks**.

Account Description

To provide O&M services, the system automatically creates system accounts when you create TaurusDB for PostgreSQL instances. These system accounts are unavailable to you.

NOTICE

Attempting to delete, rename, and change passwords or permissions for these accounts will result in an error.

- rdsAdmin: management account, which has the superuser permissions and is used to query and modify DB instance information, rectify faults, migrate data, and restore data.
- pg_execute_server_program: account that allows users who run the database to execute programs on the database server to cooperate with COPY and other functions that allow the execution of server programs.
- pg_read_all_settings: account that reads all configuration variables, even those that are usually visible only to the super user.
- pg_read_all_stats: account that reads all **pg_stat_*** views and uses various extension-related statistics, even those that are usually visible only to the super user.
- pg_stat_scan_tables: account that executes a monitoring function that may obtain an **ACCESS SHARE lock** on the table (and may hold the lock for a long time).
- pg_signal_backend: account that sends a signal (for example, a signal for canceling a query operation or an abortion signal) to another backend.
- pg_read_server_files: account that allows a database user to use the COPY and other file access functions to read files from any accessible directory on a server.
- pg_write_server_files: account that allows a database user to use the COPY and other file access functions to write files to any accessible directory on a server.
- pg_monitor: account that reads and executes various monitoring views and functions. It is a member of pg_read_all_settings, pg_read_all_stats, and pg_stat_scan_tables.
- rdsRepl: replication account, which is used to synchronize data from primary DB instances to standby DB instances.
- rdsBackup: backup account, which is used for backend backup.
- rdsMetric: metric monitoring account, which is used by watchdog to collect database status data.

11.2 Resetting the Administrator Password to Restore root Access

Scenarios

You can reset the administrator password only through a primary instance. The new password is applied immediately without rebooting the instance.

If you forget the password of the administrator account **root**, you can reset it.

Precautions

- If the password you provide is regarded as a weak password by the system, you will be prompted to enter a stronger password.
- If you have changed the administrator password of the primary DB instance, the administrator password of the standby DB instance will also be changed.
- The time required for the new password to take effect depends on the amount of service data currently being processed by the primary DB instance.
- To protect against brute force hacking attempts and ensure system security, change your password periodically, such as every three or six months.

Method 1

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, locate the target DB instance and choose **More** > **Reset Password** in the **Operation** column.

Step 5 If you have enabled operation protection, click **Send Code** in the displayed **Identity Verification** dialog box and enter the obtained verification code. Then, click **OK**.

Two-factor authentication improves the security of your account and cloud product. For details about how to enable operation protection, see ***Identity and Access Management User Guide***.

Step 6 Enter and confirm the new password.

NOTICE

Keep this password secure. The system cannot retrieve it.

The password must consist of 8 to 32 characters and contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters (~!@#$%^*-_=+?,). The password must be strong to avoid being easily cracked, as weak passwords will block your operation. Enter a strong password and periodically change it for security reasons.

- To submit the new password, click **OK**.
- To cancel the reset operation, click **Cancel**.

----End

Method 2

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 On the **Basic Information** page, find **Administrator** and click **Reset Password** under it.

Step 6 If you have enabled operation protection, click **Send Code** in the displayed **Identity Verification** dialog box and enter the obtained verification code. Then, click **OK**.

Two-factor authentication improves the security of your account and cloud product. For details about how to enable operation protection, see ***Identity and Access Management User Guide***.

Step 7 Enter and confirm the new password.

NOTICE

Keep this password secure. The system cannot retrieve it.

The password must consist of 8 to 32 characters and contain at least three types of the following characters: uppercase letters, lowercase letters, digits, and special characters (~!@#$%^*-_=+?,). The password must be strong to avoid being easily cracked, as weak passwords will block your operation. Enter a strong password and periodically change it for security reasons.

- To submit the new password, click **OK**.
- To cancel the reset operation, click **Cancel**.

----End

12 Parameters

12.1 Modifying Parameters of a TaurusDB for PostgreSQL Instance

You can modify parameters in a custom parameter template to optimize TaurusDB for PostgreSQL database performance.

You can change parameter values in custom parameter templates only and cannot change parameter values in default parameter templates.

The following are the key points you should know when using parameters:

- Modifying instance parameters: When you modify dynamic parameters on the **Parameters** page of a target DB instance and save the modifications, the modifications take effect immediately regardless of the **Effective upon Reboot** setting. When you modify static parameters on the **Parameters** page of a target DB instance and save the modifications, the modifications take effect only after you manually reboot the target DB instance.
- Modifying parameter template parameters: When you modify parameters in a custom parameter template on the **Parameter Templates** page and save the modifications, the modifications take effect only after you apply the parameter template to DB instances. When you modify static parameters in a custom parameter template on the **Parameter Templates** page and save the modifications, the modifications take effect only after you apply the parameter template to DB instances and manually reboot the DB instances. For operation details, see **Applying a Parameter Template**.

When you modify a parameter, the time when the modification takes effect is determined by the type of the parameter.

The TaurusDB for PostgreSQL console displays the statuses of DB instances that the parameter template applies to. For example, if the DB instance has not yet used the latest modifications made to its parameter template, its status is **Parameter change. Pending reboot**. Manually reboot the DB instance for the latest modifications to take effect for that DB instance.

> **NOTE**
>
> TaurusDB for PostgreSQL has default parameter templates whose parameter values cannot be changed. You can view these parameter values by clicking the default parameter templates. If a custom parameter template is set incorrectly, the database startup may fail. If this happens, you can re-configure the custom parameter template based on the settings of the default parameter template.

Modifying Parameters of a DB Instance

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the target DB instance.

Step 5 In the navigation pane, choose **Parameters**. On the displayed page, modify parameters as required.

Available operations are **Save**, **Cancel**, and **Preview**:

- To save the modifications, click **Save**.
- To cancel the modifications, click **Cancel**.
- To preview the modifications, click **Preview**.

NOTICE

In the **Effective upon Reboot** column:

- If the value is **Yes** and the DB instance status on the **Instances** page is **Parameter change. Pending reboot**, a reboot is required for the modifications to take effect.

  If you have modified parameters of a primary DB instance, you need to reboot the primary DB instance for the modifications to take effect. (For primary/standby DB instances, the parameter modifications are also applied to the standby DB instance.)
- If the value is **No**, the modifications take effect immediately.

After parameters are modified, you can view parameter change history by referring to **Viewing Parameter Change History**.

----End

Modifying a Custom Parameter Template and Applying It to DB Instances

Step 1 Log in to the management console.

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

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

Step 4 Choose **Parameter Templates** in the navigation pane. On the **Custom Templates** page, click the target parameter template.

Step 5 On the **Parameters** page, modify parameters as required.

Available operations are **Save**, **Cancel**, and **Preview**:

- To save the modifications, click **Save**.
- To cancel the modifications, click **Cancel**.
- To preview the modifications, click **Preview**.

Step 6 After the parameter values are modified, you can click **Change History** to view the modification details.

Step 7 Apply the parameter template to your DB instance. For details, see **Applying a Parameter Template**.

Step 8 View the status of the DB instance to which the parameter template was applied.

If the DB instance status is **Parameter change. Pending reboot**, a reboot is required for the modifications to take effect.

If you have modified parameters of a primary DB instance, you need to reboot the primary DB instance for the modifications to take effect. (For primary/standby DB instances, the parameter modifications are also applied to the standby DB instance.)

----End

FAQ

Q: Why did my changes to parameters fail to be applied to my DB instance after I rebooted the instance and the instance status remain **Parameter change. Pending reboot**?

A: If you change specification parameters, such as **work_mem**, **shared_buffers**, and **max_connections**, to large values, the instance may fail to be started. To ensure that the database runs properly, the system automatically rolls back the parameter change when the database startup fails. Check whether the new values you set are within the allowed ranges.

Common Parameters

The modifications of some kernel parameters can be applied only after the instance is rebooted. After you modify the parameters on the console, the message "Parameter change. Pending reboot" is displayed.

Figure 12-1 Modifying parameters

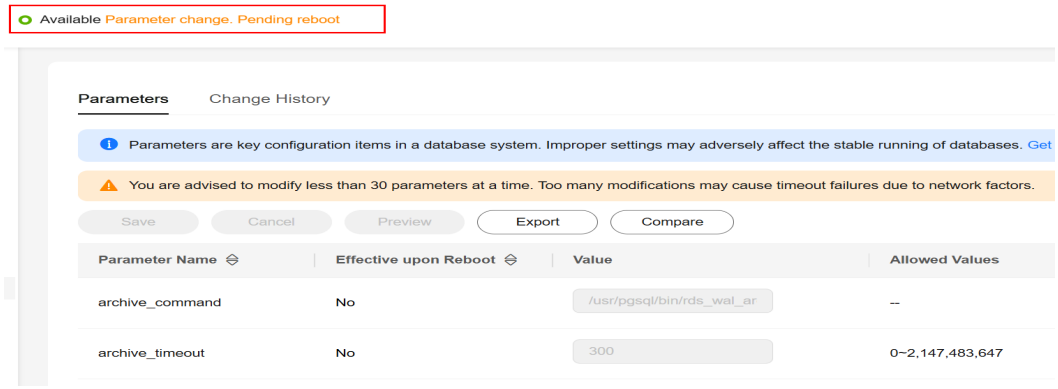


Table 12-1 Common parameters

Parameter	Description
timezone	The time zone for displaying and interpreting time stamps.
wal_level	The level of information written to the WAL.
max_connections	The maximum number of concurrent connections.

12.2 Suggestions on TaurusDB for PostgreSQL Parameter Tuning

Parameters are key configuration items in a database system. Improper parameter settings may adversely affect database performance. This section describes some important parameters for your reference. For details, visit the **PostgreSQL official website**.

For details on how to modify TaurusDB for PostgreSQL parameters on the console, see **Modifying Parameters of a TaurusDB for PostgreSQL Instance**.

Sensitive Parameters

The following parameters can result in system security and stability issues if set improperly:

- The **search_path** parameter must be set to a schema sequence where schemas are separated by commas (,). Ensure that the schemas exist. Otherwise, the database performance will be affected.
- If you enable the parameter **log_duration**, SQL statements containing sensitive information may be recorded in logs. You are advised to disable this parameter.
- **log_min_duration_statement** specifies how many milliseconds a query has to run before it has to be logged. The unit is millisecond. Setting this parameter to **0** means that all statements are recorded. Setting this parameter to **-1**

means that no statement is recorded. For details, see **Viewing and Downloading Slow Query Logs**.

- The **temp_file_limit** parameter limits the total size (in KB) of all temporary files when writing temporary files to the disk is triggered in a session. The value ranges from -1 to 2,147,483,647. The value **-1** indicates that the total size of the temporary files is not limited.
  - This parameter is only available to TaurusDB for PostgreSQL 16.
  - To prevent temporary files from occupying too much disk space and causing service exceptions, do not set this parameter to **-1**.
  - If the parameter value is changed to a larger value for temporary use but is not changed to the original value after the use, the disk space will be continuously used to store temporary files. If the disk space is used up, services will be interrupted and the DB instance will become unavailable.
- The **max_pred_locks_per_transaction** and **max_locks_per_transaction** parameters need to be set based on the values of **max_connections** and **max_prepared_transactions**. Too large values may cause instance unavailability.

Performance Parameters

The following parameters can affect database performance:

- If **log_statement** is set to **ddl**, **mod**, or **all**, the operations for creating and deleting database users (including passwords and other sensitive information) are recorded. This operation affects database performance. Exercise caution when setting this parameter.
- Enabling the following parameters will affect the database performance: **log_hostname**, **log_duration**, **log_connections**, and **log_disconnections**. Exercise caution when enabling these parameters.
- The **shared_buffers** parameter is recommended to be a value ranging from 25% to 40% of the system memory. The maximum value of this parameter cannot exceed 80% of the system memory to avoid affecting database performance.
- The **max_worker_processes** parameter should be set based on the values of **max_parallel_workers** and **max_parallel_workers_per_gather**. If the **max_worker_processes** value is too large, the database performance will be affected.

12.3 Managing Parameter Templates

12.3.1 Creating a Parameter Template

You can use database parameter templates to manage the DB engine configuration. A database parameter template acts as a container for engine configuration values that can be applied to one or more DB instances.

If you create a DB instance without specifying a custom DB parameter template, a default parameter template is used. This default template contains DB engine defaults and system defaults that are configured based on the engine, compute class, and allocated storage of the instance. Default parameter templates cannot

be modified, but you can create your own parameter template to change parameter settings.

NOTICE

Not all of the DB engine parameters in a custom parameter template can be changed.

If you want to use a custom parameter template, you simply create a parameter template and select it when you create a DB instance or apply it to an existing DB instance following the instructions provided in **Applying a Parameter Template**.

When you have already created a parameter template and want to include most of the custom parameters and values from that template in a new parameter template, you can replicate that parameter template following the instructions provided in **Replicating a Parameter Template**.

The following are the key points you should know when using parameters in a parameter template:

- When you change a parameter value in a parameter template that has been applied to a DB instance and save the change, the change takes effect only to the DB instance and does not affect other DB instances.
- The changes to parameter values in a custom parameter template take effect only after you apply the template to DB instances. For details, see **Applying a Parameter Template**.
- When you change dynamic parameter values in parameter templates in batches and save the changes, the changes will take effect only after you apply the parameter templates to DB instances. When you change static parameter values in parameter templates in batches and save the changes, the changes will take effect for DB instances only after you apply the parameter templates to DB instances and manually reboot the DB instances.
- Improper parameter settings may have unintended consequences, including reduced performance and system instability. Exercise caution when modifying database parameters and you need to back up data before modifying parameters in a parameter template. Before applying parameter template changes to a production DB instance, you should try out these changes on a test DB instance.

Procedure

Step 1 **Log in to the management console**.

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

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

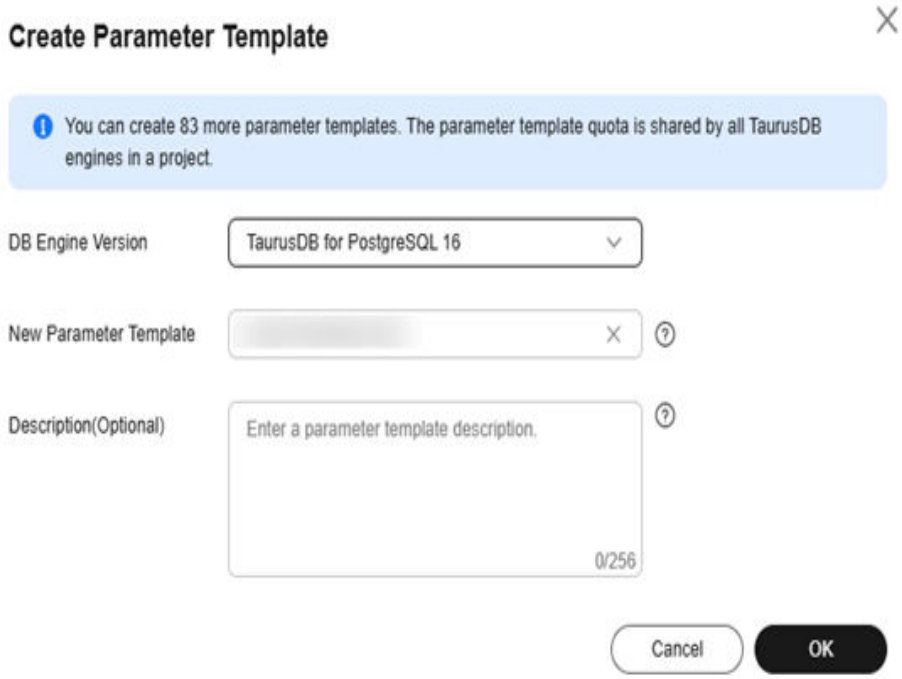
Step 4 On the **Parameter Templates** page, click **Create Parameter Template**.

Step 5 In the displayed dialog box, configure required information and click **OK**.

- Select a DB engine for the parameter template.

- The template name must consist of 1 to 64 characters. It can contain only uppercase letters, lowercase letters, digits, hyphens (-), underscores (_), and periods (.).
- The description consists of a maximum of 256 characters and cannot contain carriage return characters or the following special characters: >!<"&'=

Figure 12-2 Creating a parameter template



----End

12.3.2 Applying a Parameter Template

Scenarios

You can apply parameter templates to DB instances as needed. A parameter template can be applied only to DB instances of the same version.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Parameter Templates** page, apply a default template or a custom template to DB instances.

- If you intend to apply a default parameter template to DB instances, click **Default Templates**, locate the target parameter template, and click **Apply** in the **Operation** column.
- If you intend to apply a custom parameter template to DB instances, click **Custom Templates**, locate the target parameter template, and choose **More** > **Apply** in the **Operation** column.

A parameter template can be applied to one or more DB instances.

Step 5 In the displayed dialog box, select one or more DB instances to which the parameter template will be applied and click **OK**.

After the parameter template is successfully applied, you can view the application records by referring to **Viewing Application Records of a Parameter Template**.

----End

12.3.3 Resetting a Parameter Template

Scenarios

You can reset all parameters in a custom parameter template to their default settings.

Procedure

Step 1 **Log in to the management console**.

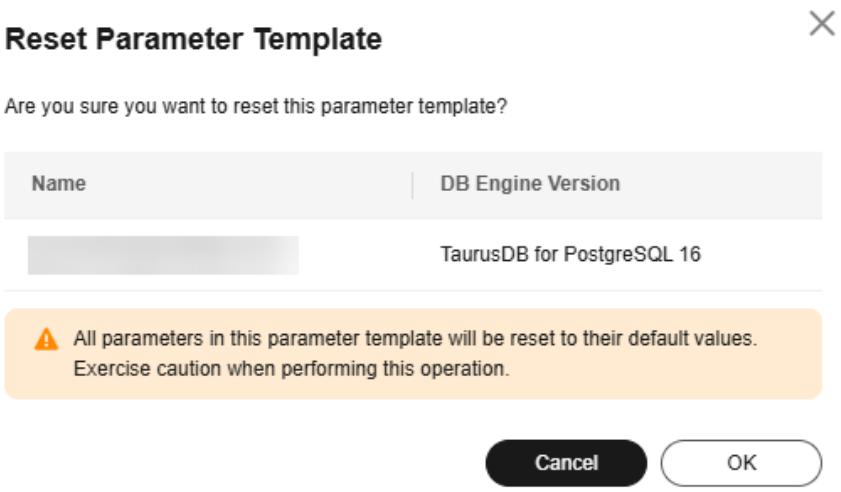
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Parameter Templates** page, click **Custom Templates**. Locate the target parameter template and choose **More** > **Reset** in the **Operation** column.

Step 5 Click **Yes**.

Figure 12-3 Confirming the reset



Step 6 Apply the parameter template to your DB instance. For details, see **Applying a Parameter Template**.

Step 7 View the status of the DB instance to which the parameter template is applied.

If the DB instance status is **Parameter change. Pending reboot**, a reboot is required for the modifications to take effect.

If you have modified parameters of a primary DB instance, you need to reboot the primary DB instance for the modifications to take effect. (For primary/standby DB instances, the parameter modifications are also applied to the standby DB instance.)

----End

12.3.4 Replicating a Parameter Template

Scenarios

You can replicate a parameter template you have created. When you have already created a parameter template and want to include most of the custom parameters and values from that template in a new parameter template, you can replicate that parameter template. You can also export the parameter template to generate a new parameter template for future use.

After a parameter template is replicated, it takes about 5 minutes before the new template is displayed.

Default parameter templates cannot be replicated, but you can create parameter templates based on the default ones.

Procedure

Step 1 **Log in to the management console**.

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

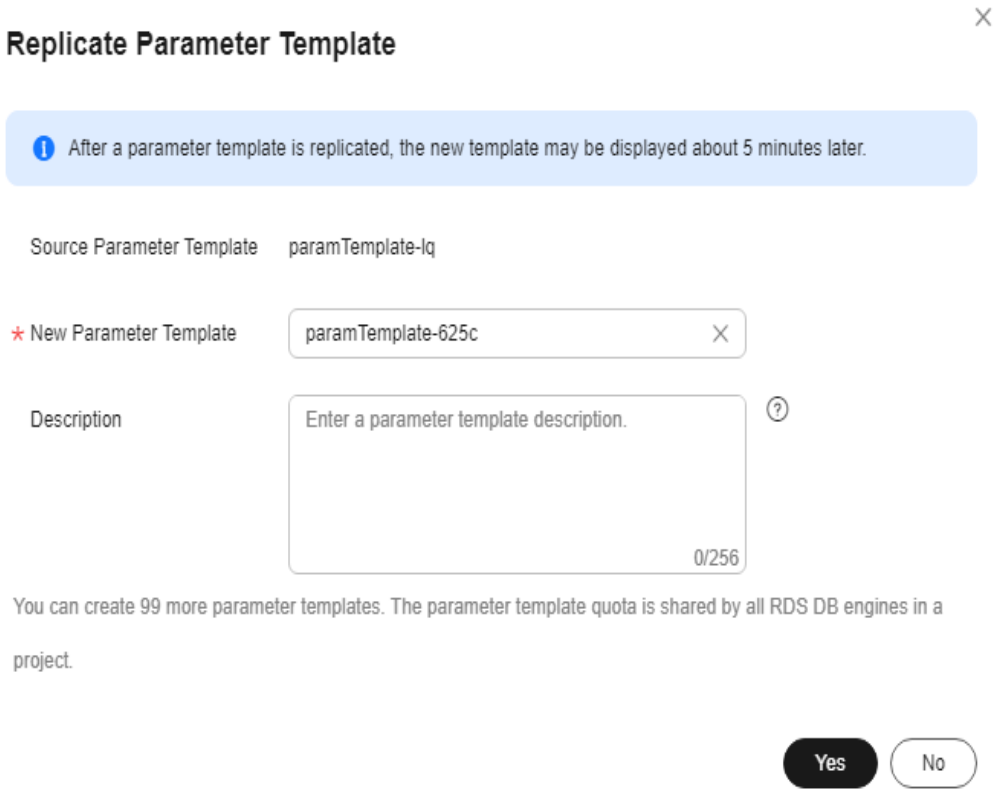
Step 3 Click  in the upper left corner of the page and choose **Databases** > **TaurusDB**.

Step 4 On the **Parameter Templates** page, click **Custom Templates**. Locate the target parameter template and click **Replicate** in the **Operation** column.

Alternatively, click the target DB instance on the **Instances** page. On the **Parameters** page, click **Export** to generate a new parameter template for future use.

Step 5 In the displayed dialog box, configure required information and click **Yes**.

Figure 12-4 Replicating a parameter template



- The template name must consist of 1 to 64 characters. It can contain only uppercase letters, lowercase letters, digits, hyphens (-), underscores (_), and periods (.).
- The description consists of a maximum of 256 characters and cannot contain carriage return characters or the following special characters: >!<"&'=

After the parameter template is replicated, a new template is generated in the list on the **Parameter Templates** page.

----End

12.3.5 Comparing Parameter Templates

Scenarios

You can compare DB instance parameters with a parameter template that uses the same DB engine to understand the differences of parameter settings.

You can also compare default parameter templates that use the same DB engine to understand the differences of parameter settings.

Comparing Instance Parameters with a Parameter Template

Step 1 Log in to the management console.

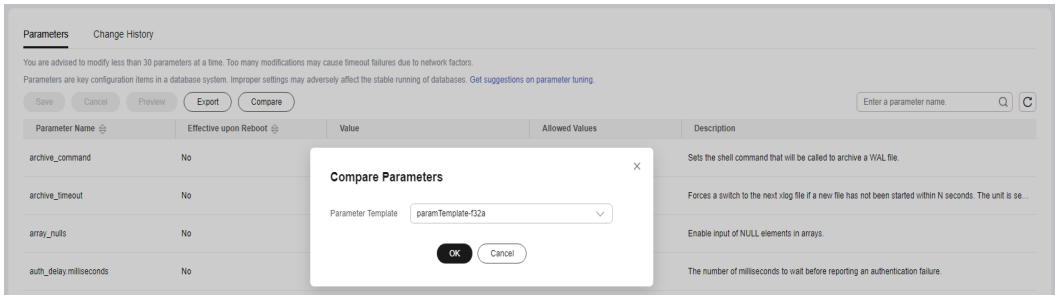
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Parameters**. On the displayed page, click **Compare** above the parameter list.

Figure 12-5 Comparing instance parameters with those in a specified parameter template



Step 6 In the displayed dialog box, select a parameter template to be compared and click **OK**.

- If their settings are different, the parameter names and values of both parameter templates are displayed.
- If their settings are the same, no data is displayed.

----End

Comparing Parameter Templates

Step 1 **Log in to the management console**.

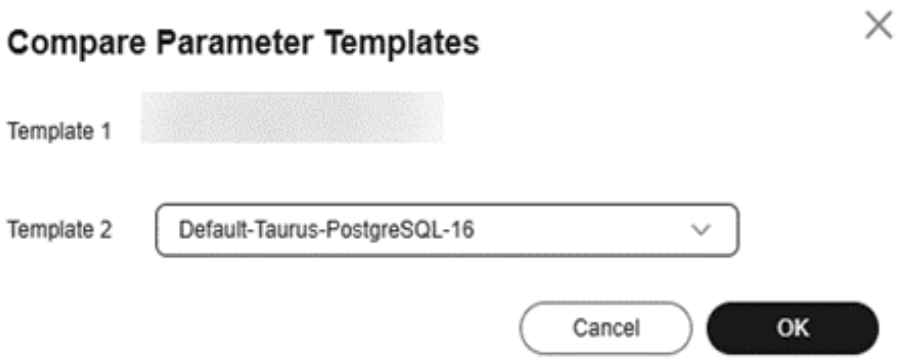
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Parameter Templates** page, click **Custom Templates**. Locate the target parameter template and click **Compare** in the **Operation** column.

Step 5 In the displayed dialog box, select a parameter template that uses the same DB engine as the target template and click **OK**.

Figure 12-6 Selecting a parameter template to be compared



- If their settings are different, the parameter names and values of both parameter templates are displayed.
- If their settings are the same, no data is displayed.

----End

12.3.6 Exporting a Parameter Template

Scenarios

Exporting instance parameters

- You can export parameters of a DB instance as a new parameter template for future use. To apply the exported parameter template to new DB instances, see **Applying a Parameter Template**.
- You can also export the parameter information (including parameter names, values, and descriptions) of a DB instance to a CSV file for viewing and analyzing details.
- You can export a TaurusDB for PostgreSQL parameter template (including parameter names, values, and descriptions) to a CSV file for viewing and analyzing details.

Exporting Instance Parameters

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Parameters**. On the displayed page, click **Export** above the parameter list.

- Exporting to a custom template

  In the displayed dialog box, configure required information and click **OK**.

NOTE

- The template name is case-sensitive and can contain 1 to 64 characters. Only letters, digits, hyphens (-), underscores (_), and periods (.) are allowed.
- The template description consists of a maximum of 256 characters and cannot contain carriage return characters or the following special characters: >!<"&'=

After the parameter template is exported, a new template is generated in the list in the **Custom Templates** tab on the **Parameter Templates** page.

- Exporting to a file

  The parameter template information (parameter names, values, and descriptions) of the DB instance is exported to a CSV file. In the displayed dialog box, enter the file name and click **OK**.

  NOTE

  The file name can contain 4 to 81 characters.

----End

12.3.7 Modifying a Parameter Template Description

Scenarios

You can modify the description of a parameter template you have created.

NOTE

You cannot modify the description of a default parameter template.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Parameter Templates** page, click **Custom Templates**. Locate the target parameter template and click in the **Description** column.

Step 5 Enter a new description and click **OK** to submit the modification or click **Cancel** to cancel the modification.

- The description consists of a maximum of 256 characters and cannot contain carriage return characters or the following special characters: >!<"&'=
- After the modification is successful, you can view the new description in the **Description** column of the parameter template list.

----End

12.3.8 Deleting a Parameter Template

Scenarios

You can delete a custom parameter template that is no longer in use.

NOTICE

- Deleted parameter templates cannot be recovered. Exercise caution when performing this operation.
- Default parameter templates cannot be deleted.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 On the **Parameter Templates** page, click **Custom Templates**. Locate the parameter template to be deleted and choose **More** > **Delete** in the **Operation** column.

Step 5 In the displayed dialog box, click **OK**.

----End

12.3.9 Viewing Parameter Change History

Scenarios

You can view the change history of DB instance parameters or custom parameter templates.

NOTE

The change history for an exported or custom parameter template is initially blank.

Viewing Change History of a DB Instance

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Parameters**. On the displayed page, click **Change History**.

You can view the parameter name, original parameter value, new parameter value, modification status, modification time, application status, and application time.

----End

Viewing Change History of a Parameter Template

Step 1 Log in to the management console.

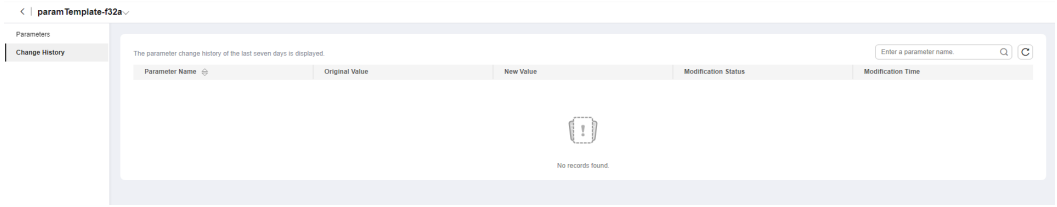
Step 2 Click in the upper left corner and select a region.

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

Step 4 Choose **Parameter Templates** in the navigation pane. On the **Custom Templates** page, click the target parameter template.

Step 5 On the displayed page, choose **Change History** in the navigation pane.

Figure 12-7 Viewing parameter change history



You can view the parameter name, original parameter value, new parameter value, modification status, and modification time.

You can apply the parameter template to DB instances as required by referring to **Applying a Parameter Template**.

----End

12.3.10 Viewing Application Records of a Parameter Template

Scenarios

You can view the application records of a parameter template.

Procedure

Step 1 Log in to the management console.

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

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

Step 4 Choose **Parameter Templates** in the navigation pane.

Step 5 On the **Default Templates** or **Custom Templates** page, locate the target parameter template and choose **More** > **View Application Record** in the **Operation** column.

You can view the name or ID of the DB instance that the parameter template applies to, as well as the application status, application time, and failure cause (if failed).

----End

13 Log Management

13.1 Viewing and Downloading Error Logs

Scenarios

Error logs contain logs generated while the database is running. These can help you analyze problems with the database. You can also download error logs for analysis.

Viewing Log Details

Step 1 Log in to the management console.

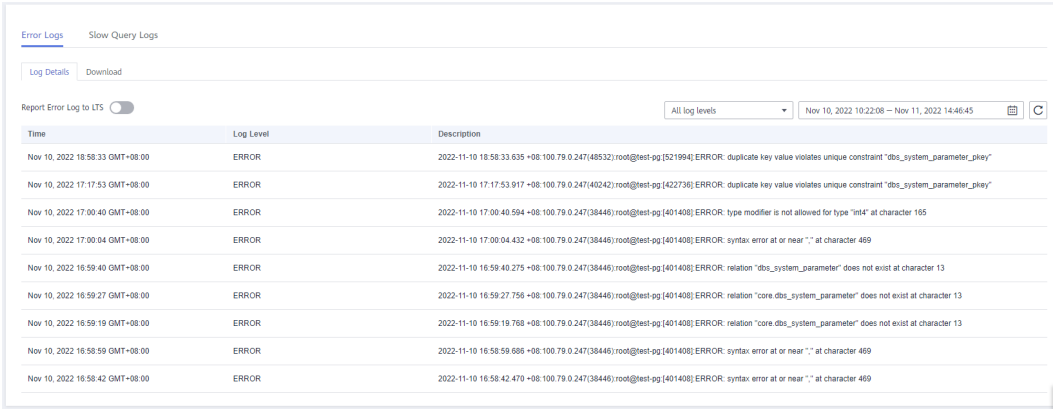
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Logs**. On the **Error Logs** page, click **Log Details** to view details about error logs.

Figure 13-1 Error log details



- You can select a log level in the upper right corner of the log list.

  For TaurusDB for PostgreSQL instances, the following levels of logs are displayed:

  - All log levels
  - ERROR
  - FATAL
  - PANIC

- You can click in the upper right corner to view error logs generated in different time segments.
- If the description of a log is truncated, locate the log and move your pointer over the description in the **Description** column to view details.
- Currently, a maximum of 2,000 error log records can be displayed.

----End

Downloading a Log

Step 1 Log in to the management console.

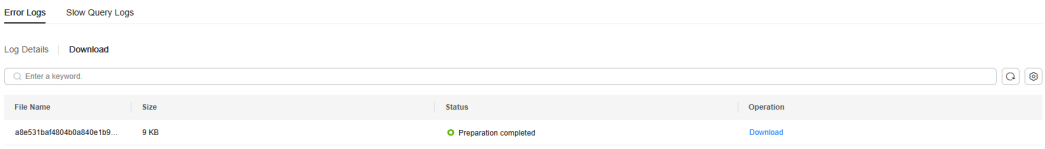
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Logs**. On the **Error Logs** page, click **Download**. In the log list, locate a log whose status is **Preparation completed** and click **Download** in the **Operation** column.

Figure 13-2 Downloading an error log



- It is recommended that a single log file to be downloaded contain a maximum of 10,000 lines and the file size be no more than 10 MB. Otherwise, the log information will be truncated.
- The system automatically loads the downloading preparation tasks. The loading duration is determined by the log file size and network environment.
  - When the log is being prepared for download, the log status is **Preparing**.
  - When the log is ready for download, the log status is **Preparation completed**.
  - If the preparation for download fails, the log status is **Abnormal**.

  Logs in the **Preparing** or **Abnormal** status cannot be downloaded.

- Only the latest log file of no more than 40 MB can be downloaded.
- The download link is valid for 5 minutes. After the download link expires, a message is displayed indicating that the download link has expired. If you need to redownload the log, click **OK**.

----End

Enabling Error Log Reporting to LTS

To use this function, **submit a service ticket** to obtain permissions.

Step 1 **Log in to the management console**.

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

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

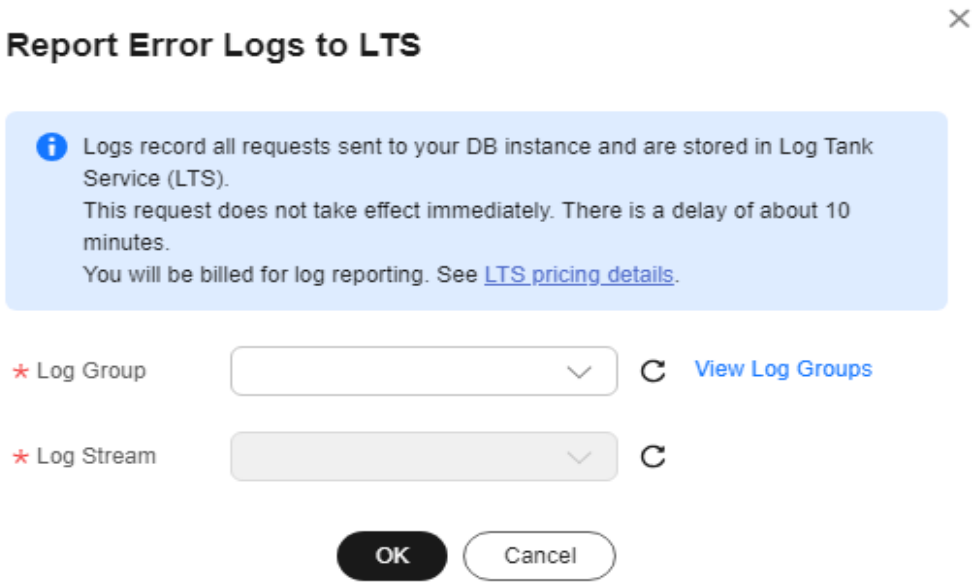
Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, click **Logs**. On the **Error Logs** page, click **Log Details**.

Step 6 Click next to **Report Error Logs to LTS**.

Step 7 Select an LTS log group and log stream and click **OK**.

Figure 13-3 Enabling error log reporting to LTS



----End

13.2 Viewing and Downloading Slow Query Logs

Scenarios

Slow query logs record statements that exceed the **log_min_duration_statement** value. You can view log details and statistics to identify statements that are executing slowly and optimize the statements. You can also download slow query logs for analysis.

TaurusDB for PostgreSQL supports the following statements:

- All
- SELECT
- INSERT
- UPDATE
- DELETE
- CREATE
- DROP
- ALTER
- DO
- CALL
- COPY

Parameter Description

Table 13-1 Parameters related to TaurusDB for PostgreSQL slow queries

Parameter	Description
log_min_duration_statement	Specifies how many milliseconds a query has to run before it has to be logged. If this parameter is set to a smaller value, the number of log records increases, which increases the disk I/O and deteriorates the SQL performance.
log_statement	Specifies the statement type. The value can be **none**, **ddl**, **mod**, or **all**. The default value is **none**. If you change the value to **all**: • The database disk I/O increases, and the SQL performance deteriorates. • The log format changes, and you cannot view slow query logs on the console.

Parameter	Description
log_statement_stats	Specifies whether to generate performance statistics to server logs. The default value is **off**. If you change the value to **on**: • The database disk I/O increases, and the SQL performance deteriorates. • The log format changes, and you cannot view slow query logs on the console.

Showing Original Logs

NOTE

- To disable **Show Original Logs** on the console, contact customer service.
- Original logs will be automatically deleted 30 days later. If the instance is deleted, its logs are also deleted.
- This function takes effect only for slow query logs generated after it is enabled. Historical slow query logs generated before the function is enabled are not displayed in plaintext.

Step 1 Log in to the management console.

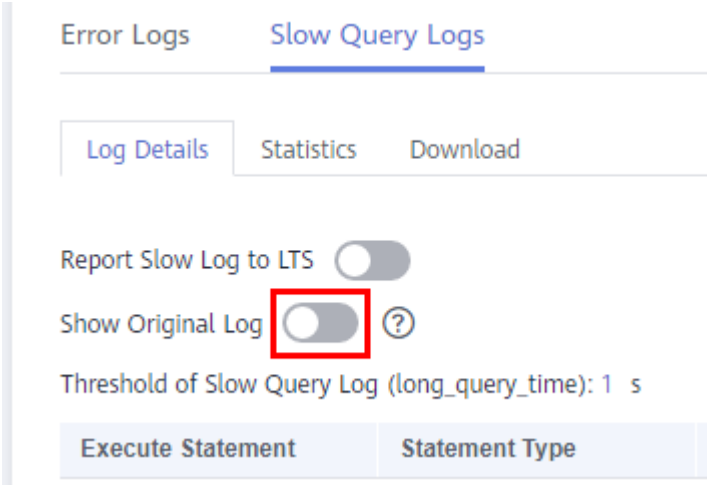
Step 2 Click in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Logs**. On the **Slow Query Logs** page, click **Log Details** and then click on the right of **Show Original Logs**.

Figure 13-4 Enabling Show Original Logs



Step 6 In the displayed dialog box, click **Yes** to enable the display of original slow query logs.

----End

Viewing Log Details

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Logs**. On the **Slow Query Logs** page, click **Log Details** to view details about slow query logs.

- You can view the slow query log records of a specified execution statement type or a specific time period.
- The **log_min_duration_statement** parameter determines when a slow query log is recorded. However, changes to this parameter do not affect already recorded logs. If **log_min_duration_statement** is changed from 1,000 ms to 100 ms, TaurusDB for PostgreSQL starts recording statements that meet the new threshold and still displays the previously recorded logs that do not meet the new threshold. For example, a 1,500 ms SQL statement that was recorded when the threshold was 1,000 ms will not be deleted now that the new threshold is 2,000 ms.
- Currently, a maximum of 2,000 slow log records can be displayed.

----End

Viewing Statistics

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Logs**. On the **Slow Query Logs** page, click **Statistics** to view details.

NOTE

On the **Statistics** page, only one of the SQL statements of the same type is displayed as an example. For example, if two select sleep(N) statements, **select sleep(1)** and **select sleep(2)**, are executed in sequence, only **select sleep(1)** will be displayed.

----End

Downloading a Log

Step 1 Log in to the management console.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Logs**. On the **Slow Query Logs** page, click **Download**. In the log list, locate a log whose status is **Preparation completed** and click **Download** in the **Operation** column.

- It is recommended that a single log file to be downloaded contain a maximum of 10,000 lines and the file size be no more than 10 MB. Otherwise, the log information will be truncated.
- The system automatically loads the downloading preparation tasks. The loading duration is determined by the log file size and network environment.
  - When the log is being prepared for download, the log status is **Preparing**.
  - When the log is ready for download, the log status is **Preparation completed**.
  - If the preparation for download fails, the log status is **Abnormal**.

  Logs in the **Preparing** or **Abnormal** status cannot be downloaded.
- Only the latest log file of no more than 40 MB can be downloaded.
- The download link is valid for 5 minutes. After the download link expires, a message is displayed indicating that the download link has expired. If you need to redownload the log, click **OK**.

----End

Enabling Slow Log Reporting to LTS

To use this function, **submit a service ticket** to obtain permissions.

Step 1 Log in to the management console.

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

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

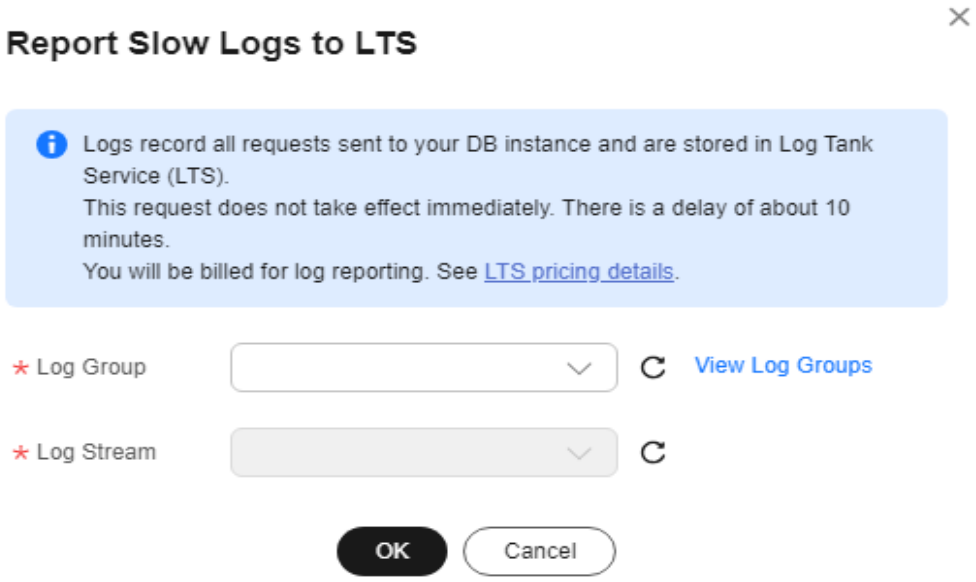
Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, click **Logs**. On the **Slow Query Logs** page, click **Log Details**.

Step 6 Click  next to **Report Slow Logs to LTS**.

Step 7 Select an LTS log group and log stream and click **OK**.

Figure 13-5 Enabling slow log reporting to LTS



----End

13.3 Enabling SQL Audit

Scenarios

After SQL audit is enabled for TaurusDB for PostgreSQL instances, the system records SQL operations and uploads logs every half an hour or when the size of a single record reaches 100 MB. The generated audit logs are stored in OBS. If there is not enough free backup space available for generated audit logs, the additional space required is billed.

Precautions

- SQL audit is disabled for DB instances by default because enabling it increases database loads.
- To ensure good performance, SQL audit uses the Coordinated Universal Time (UTC) format and is not affected by the time zone configuration.
- To enable SQL audit, you need to install the pgAudit extension and set related parameters based on the pgAudit extension usage instruction.

Constraints

Only the latest minor version of TaurusDB for PostgreSQL 16 supports SQL audit. To use this function, **submit a service ticket**.

Performance

The pgAudit impact on database performance depends on how many audit logs there are and how often they are generated. More audit logs mean more impact

on the database performance. pgAudit can decrease the database performance by about 20%. You need to configure parameters for pgAudit based on your workloads to achieve a balance between audit requirements and database performance.

Procedure

Step 1 **Log in to the management console**.

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

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

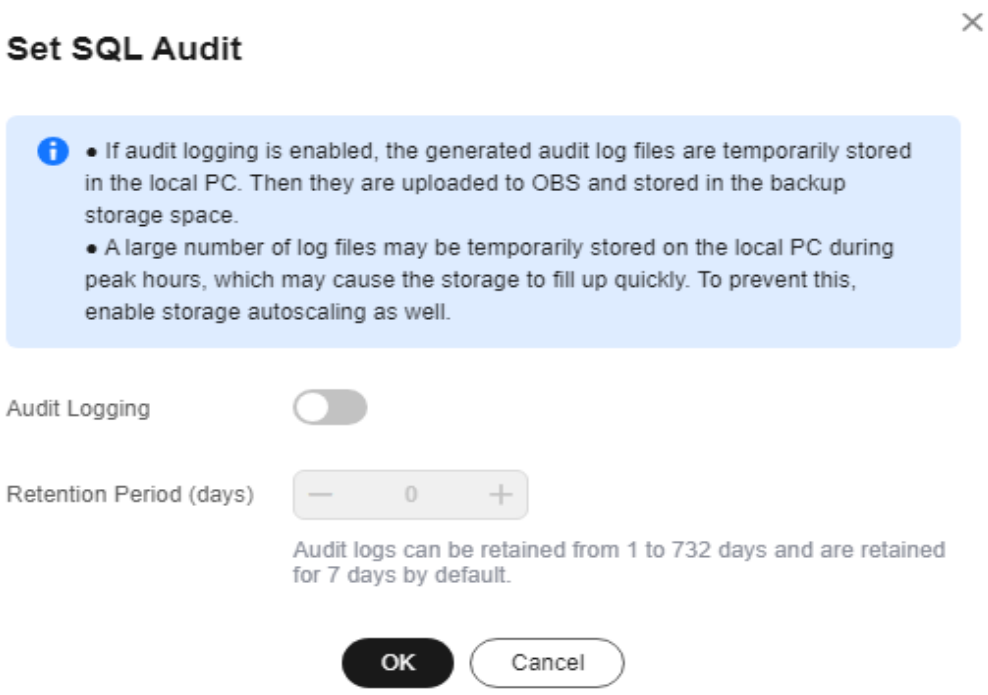
Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **SQL Audits**. On the displayed page, click **Set SQL Audit**.

Step 6 In the displayed dialog box, set the number of days for storing SQL audit logs and click **OK**.

Audit logs can be retained from 1 to 732 days and are retained for 7 days by default.

Figure 13-6 Setting SQL audit

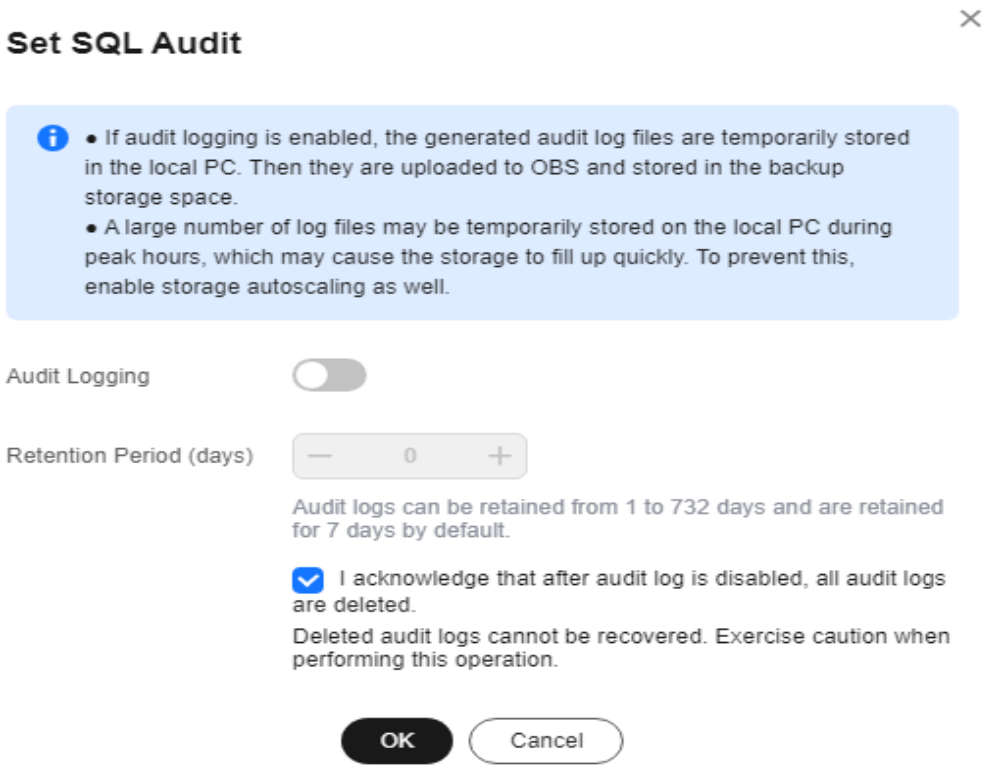


Step 7 To disable SQL audit, toggle off the **Audit Logging** switch, select the confirmation check box, and click **OK**.

NOTICE

After SQL audit is disabled, all audit logs will be deleted immediately and cannot be recovered. Exercise caution when performing this operation.

Figure 13-7 Disabling SQL audit



----End

13.4 Downloading SQL Audit Logs

If you **enable SQL audit**, all SQL operations will be logged, and you can download audit logs to view details. The minimum time unit of audit logs is second.

Procedure

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **SQL Audits**.

Step 6 On the displayed page, select a time range in the upper right corner, select SQL audit logs to be downloaded in the list, and click **Download** above the list to download SQL audit logs in batches.

Alternatively, select an audit log and click **Download** in the **Operation** column to download an individual SQL audit log.

Step 7 The following figure shows the SQL audit log content. For field descriptions, see **Table 13-2**.

Figure 13-8 TaurusDB for PostgreSQL audit logs

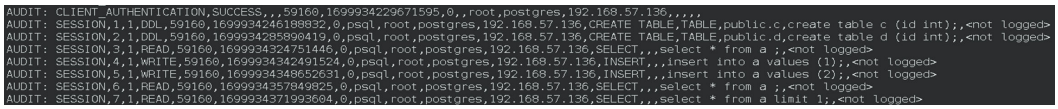


Table 13-2 Audit log field description

Field	Description
AUDIT:	Fixed prefix, which identifies an audit record.
AUDIT_TYPE	Audit type. The value can be **SESSION**, **OBJECT**, or **CLIENT_AUTHENTICATION**.
STATEMENT_ID	Unique statement ID for this session.
SUBSTATEMENT_ID	ID of each substatement in the main statement.
CLASS or AUTHENTICATION_RESULT	Operation type. • **CLASS**: The value depends on the **pgaudit.log** options, and can be **READ** or **ROLE**. • **AUTHENTICATION_RESULT**: The value can be **SUCCESS** or **FAIL**.
PID	Process ID.
STATEMENT_START_TIME	Statement start timestamp, in us.
connection_status	Session status, which is usually the returned error code of a statement. If the statement is successfully executed, the value **0** is returned.
APPLICATION_NAME	Application name, such as **PSQL** and **JDBC**.
USER_NAME	Username for logging in to the database.
DATABASE_NAME	Name of the database that was logged in to.
REMOTE_HOST	IP address of the host used for login.
COMMAND	Type of the SQL command, such as **ALTER TABLE** and **SELECT**.
OBJECT_TYPE	Object type, such as **TABLE**, **INDEX**, and **VIEW**.

Field	Description
OBJECT_NAME	Object name.
STATEMENT	Content of the SQL statement executed at the backend.
PARAMETER	Parameter value.

----End

14 Task Center

14.1 Viewing a Task

You can view the progress and results of scheduled tasks on the **Task Center** page.

Supported Tasks

Table 14-1 Supported tasks

Task Type	Category	Task Name
Instant tasks	Instance creation	Creating a TaurusDB for PostgreSQL instance
	Instance lifecycle	Rebooting a TaurusDB for PostgreSQL instance
		Stopping a TaurusDB for PostgreSQL instance
	Instance modifications	Scaling a TaurusDB for PostgreSQL instance
	Backup and restoration	Restoring to a new TaurusDB for PostgreSQL instance, restoring to an existing TaurusDB for PostgreSQL instance, restoring TaurusDB for PostgreSQL databases, and restoring TaurusDB for PostgreSQL tables
	Parameter configuration	Modifying a TaurusDB for PostgreSQL parameter template
Scheduled tasks	Instance lifecycle	Starting a TaurusDB for PostgreSQL instance

Viewing an Instant Task

Step 1 Log in to the management console.

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

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

Step 4 On the **Task Center** page, locate the target task and view its details.

----End

Viewing a Scheduled Task

Step 1 Log in to the management console.

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

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

Step 4 Choose **Task Center** in the navigation pane. On the **Scheduled Tasks** page, view the task progress and results.

- To identify the target task, you can use the DB instance name/ID or enter the target DB instance ID in the search box in the upper right corner.
- You can view the scheduled tasks in the following statuses:
  - Running
  - Completed
  - Failed
  - Canceled
  - To be executed
  - To be authorized

----End

14.2 Deleting a Task Record

You can delete task records so that they are no longer displayed in the task list. This operation only deletes the task records. It does not delete the DB instances or terminate the tasks that are being executed.

NOTICE

Deleted task records cannot be recovered. Exercise caution when performing this operation.

Deleting an Instant Task Record

Step 1 Log in to the management console.

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

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

Step 4 Choose **Task Center** in the navigation pane. On the displayed **Instant Tasks** page, locate the task record to be deleted and click **Delete** in the **Operation** column. In the displayed dialog box, click **OK**.

You can delete the records of instant tasks in the following statuses:

- Completed
- Failed

----End

Deleting a Scheduled Task Record

Step 1 Log in to the management console.

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

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

Step 4 Choose **Task Center** in the navigation pane. On the **Scheduled Tasks** page, locate the task record to be deleted and check whether the task status is **To be executed** or **To be authorized**.

- If yes, go to **Step 5**.
- If no, go to **Step 6**.

Step 5 Click **Cancel** in the **Operation** column. In the displayed dialog box, click **Yes** to cancel the task. Click **Delete** in the **Operation** column. In the displayed dialog box, click **OK** to delete the task record.

Step 6 Click **Delete** in the **Operation** column. In the displayed dialog box, click **OK** to delete the task record.

You can delete the records of scheduled tasks in the following statuses:

- Completed
- Failed
- Canceled
- To be executed
- To be authorized

----End

15 TaurusDB for PostgreSQL Tags

Scenarios

Tag Management Service (TMS) enables you to use tags on the console to manage resources. TMS works with other cloud services to manage tags. TMS manages tags globally. Other cloud services manage only their own tags.

- Log in to the management console. Click **Service List** and choose **Management & Governance** > **Tag Management Service**. Set predefined tags on the TMS console.
- A tag consists of a key and value. You can add only one value for each key.
- Up to 20 tags can be added for each DB instance.

Adding or Editing a Tag

Step 1 Log in to the management console.

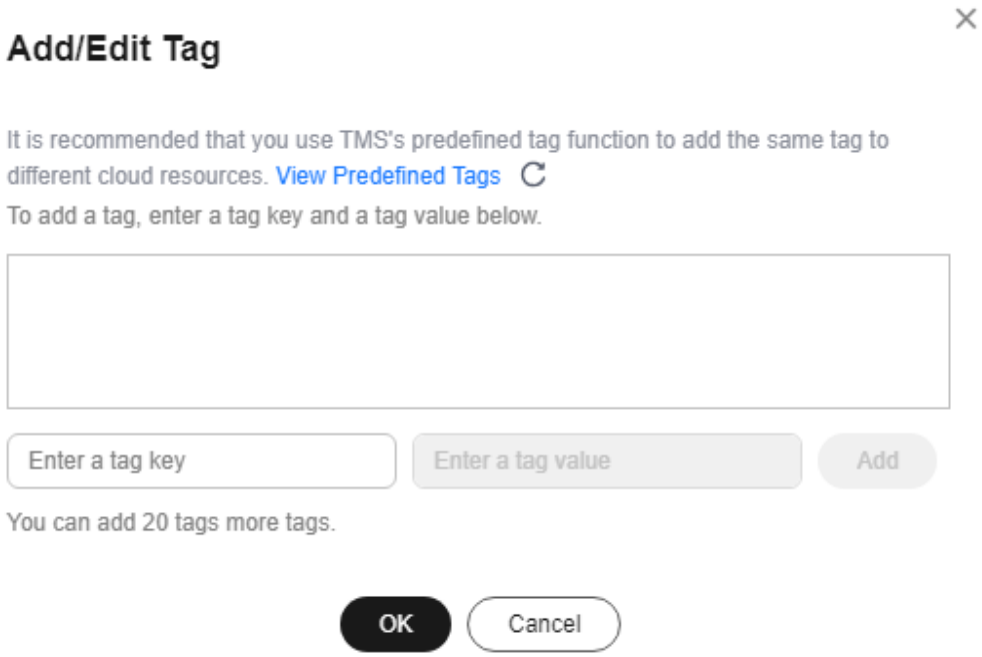
Step 2 Click  in the upper left corner and select a region.

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Tags**. On the displayed page, click **Add/Edit Tag**. In the displayed dialog box, enter a tag key and value, click **Add**, and click **OK**.

Figure 15-1 Adding a tag



- When you enter a tag key and value, the system automatically displays all tags (including predefined tags and resource tags) associated with all instances except the current one.
- The tag key must be unique. It must consist of 1 to 128 characters and can include letters, digits, spaces, and the following characters: _ **.** **:** **=** **+** **-** **@**. It cannot start or end with a space, or start with **_sys_**.
- The tag value (optional) can consist of up to 255 characters and can include letters, digits, spaces, and the following characters: _ **.** **:** **/** **=** **+** **-** **@**.

Step 6 View and manage the tag on the **Tags** page.

----End

Deleting a Tag

Step 1 **Log in to the management console**.

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

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

Step 4 On the **Instances** page, click the instance name.

Step 5 In the navigation pane, choose **Tags**. Select the tag to be deleted and click **Delete**. In the displayed dialog box, click **OK**.

Check that the tag is no longer displayed on the **Tags** page.

----End

16 TaurusDB for PostgreSQL Quotas

What Is a Quota?

A quota is a limit on the quantity or capacity of a certain type of service resources available to you. Examples of TaurusDB for PostgreSQL quotas include the maximum number of DB instances that you can create. Quotas are put in place to prevent excessive resource usage.

If a quota cannot meet your needs, apply for a higher quota.

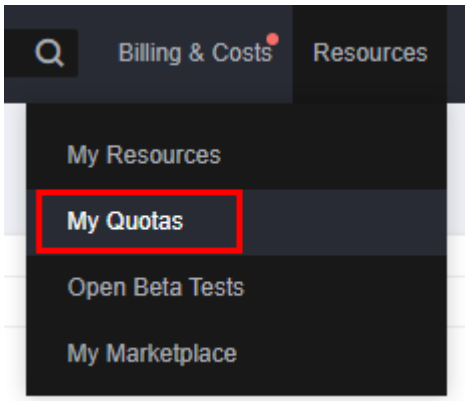
Viewing Quotas

Step 1 Log in to the management console.

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

Step 3 In the upper right corner of the TaurusDB console, choose **Resources** > **My Quotas**.

Figure 16-1 My quotas



Step 4 On the **Quotas** page, view the used and total quotas of each type of resources.

----End

Increasing Quotas

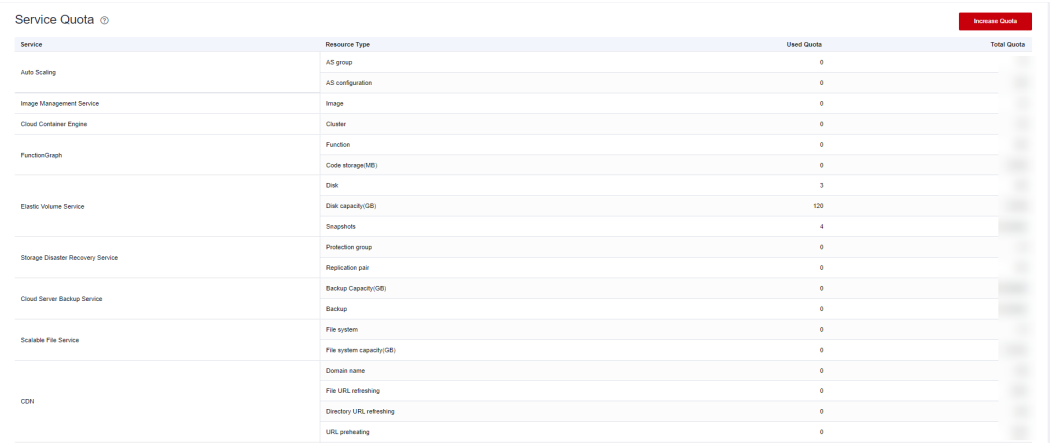
Step 1 Log in to the management console.

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

Step 3 In the upper right corner of the TaurusDB console, choose **Resources** > **My Quotas**.

Step 4 In the upper right corner of the page, click **Increase Quota**.

Figure 16-2 Increasing quotas



Step 5 On the **Create Service Ticket** page, configure parameters as required.

In the **Problem Description** area, fill in the content and reason for adjustment.

Step 6 After all required parameters are configured, select the agreement and click **Submit**.

----End

