ALIBABA CLOUD

Alibaba Cloud

Apsara Stack Enterprise

AnalyticDB for PostgreSQL

User Guide

Product Version: v3.16.2

Document Version: 20220915

Alibaba Cloud

Legal disclaimer

Alibaba Cloud reminds you to carefully read and fully understand the terms and conditions of this legal disclaimer before you read or use this document. If you have read or used this document, it shall be deemed as your total acceptance of this legal disclaimer.

1. You shall download and obtain this document from the Alibaba Cloud website or other Alibaba Cloud-authorized channels, and use this document for your own legal business activities only. The content of this document is considered confidential information of Alibaba Cloud. You shall strictly abide by the confidentiality obligations. No part of this document shall be disclosed or provided to any third party for use without the prior written consent of Alibaba Cloud.

2. No part of this document shall be excerpted, translated, reproduced, transmitted, or disseminated by any organization, company or individual in any form or by any means without the prior written consent of Alibaba Cloud.

3. The content of this document may be changed because of product version upgrade, adjustment, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and an updated version of this document will be released through Alibaba Cloud-authorized channels from time to time. You should pay attention to the version changes of this document as they occur and download and obtain the most up-to-date version of this document from Alibaba Cloud-authorized channels.

4. This document serves only as a reference guide for your use of Alibaba Cloud products and services. Alibaba Cloud provides this document based on the "status quo", "being defective", and "existing functions" of its products and services. Alibaba Cloud makes every effort to provide relevant operational guidance based on existing technologies. However, Alibaba Cloud hereby makes a clear statement that it in no way guarantees the accuracy, integrity, applicability, and reliability of the content of this document, either explicitly or implicitly. Alibaba Cloud shall not take legal responsibility for any errors or lost profits incurred by any organization, company, or individual arising from download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, take responsibility for any indirect, consequential, punitive, contingent, special, or punitive damages, including lost profits arising from the use or trust in this document (even if Alibaba Cloud has been notified of the possibility of such a loss).

5. By law, all the contents in Alibaba Cloud documents, including but not limited to pictures, architecture design, page layout, and text description, are intellectual property of Alibaba Cloud and/or its affiliates. This intellectual property includes, but is not limited to, trademark rights, patent rights, copyrights, and trade secrets. No part of this document shall be used, modified, reproduced, publicly transmitted, changed, disseminated, distributed, or published without the prior written consent of Alibaba Cloud and/or its affiliates. The names owned by Alibaba Cloud shall not be used, published, or reproduced for marketing, advertising, promotion, or other purposes without the prior written consent of Alibaba Cloud. The names owned by Alibaba Cloud include, but are not limited to, "Alibaba Cloud", "Aliyun", "HiChina", and other brands of Alibaba Cloud and/or its affiliates, which appear separately or in combination, as well as the auxiliary signs and patterns of the preceding brands, or anything similar to the company names, trade names, trademarks, product or service names, domain names, patterns, logos, marks, signs, or special descriptions that third parties identify as Alibaba Cloud and/or its affiliates.

6. Please directly contact Alibaba Cloud for any errors of this document.

Document conventions

Style	Description	Example
Danger	A danger notice indicates a situation that will cause major system changes, faults, physical injuries, and other adverse results.	**Danger:** Resetting will result in the loss of user configuration data.
Warning	A warning notice indicates a situation that may cause major system changes, faults, physical injuries, and other adverse results.	**Warning:** Restarting will cause business interruption. About 10 minutes are required to restart an instance.
Notice	A caution notice indicates warning information, supplementary instructions, and other content that the user must understand.	**Notice:** If the weight is set to 0, the server no longer receives new requests.
Note	A note indicates supplemental instructions, best practices, tips, and other content.	**Note:** You can use Ctrl + A to select all files.
>	Closing angle brackets are used to indicate a multi-level menu cascade.	Click **Settings**> **Network**> **Set network type**.
Bold	Bold formatting is used for buttons , menus, page names, and other UI elements.	Click **OK**.
`Courier font`	Courier font is used for commands	Run the `cd /d C:/window` command to enter the Windows system folder.
Italic	Italic formatting is used for parameters and variables.	`bae log list --instanceid` *Instance_ID*
[] or [a|b]	This format is used for an optional value, where only one item can be selected.	`ipconfig [-all|-t]`
{} or {a|b}	This format is used for a required value, where only one item can be selected.	`switch {active|stand}`

Table of Contents

1.What is AnalyticDB for PostgreSQL?

AnalyticDB for PostgreSQL is a distributed cloud database service that uses multiple compute nodes to provide massively parallel processing (MPP) data warehousing.

AnalyticDB for PostgreSQL is developed based on the open source Greenplum database project and enhanced by Alibaba Cloud. This service has the following features:

- Compatible with Greenplum and all tools that support Greenplum.
- Supports Object Storage Service (OSS), JSON, and HyperLogLog, which is a probabilistic algorithm for cardinality estimation.
- Supports SQL:2003-compliant syntax and Online Analytical Processing (OLAP) aggregate functions to provide flexible hybrid analysis.
- Supports both row store and column store to enhance analytics performance.
- Supports data compression to reduce storage costs.
- Provides online scaling and performance monitoring to enable database administrators (DBAs), developers, and data analysts to focus on improving enterprise productivity and creating core business value instead of managing and maintaining large numbers of MPP clusters.

2.Quick start

2.1. Overview

This topic describes all operations that you can perform on an AnalyticDB for PostgreSQL instance, from instance creation to database logon. It provides a quick start guide to the operations on the AnalyticDB for PostgreSQL instance.

For more information, see workflow.

AnalyticDB for PostgreSQL workflow

- Log on to the AnalyticDB for PostgreSQL console

  You can log on to the AnalyticDB for PostgreSQL console.

- Create an instance

  Before you perform other operations, you must first create an AnalyticDB for PostgreSQL instance in the console.

- Configure a whitelist

  Before you use an AnalyticDB for PostgreSQL instance, add IP addresses or CIDR blocks needed to access your database to the whitelist of the instance to improve the security and stability of the database.

- Create an initial account

  After you create an instance, you must create an initial account to log on to the database.

- Connect to a database

  You can use a client that supports PostgreSQL or Greenplum to connect to a database.

2.2. Log on to the AnalyticDB for PostgreSQL console

This topic describes how to log on to the AnalyticDB for PostgreSQL console.

Prerequisites

- Before you log on to the Apsara Uni-manager Management Console, the endpoint of the console is obtained from the deployment staff.
- We recommend that you use Google Chrome.

1. Enter the URL of the Apsara Uni-manager Management Console in the address bar and press the Enter key.
2. Enter your username and password.

   Obtain the username and password that you use to log on to the Apsara Uni-manager Management Console from the operations administrator.

> **Note** The first time that you log on to the Apsara Uni-manager Management Console, you must change the password of your account. For security purposes, your password must meet the minimum complexity requirements. The password must be 10 to 32 characters in length and must contain at least two of the following character types:
>
> - Uppercase or lowercase letters
> - Digits
> - Special characters, including exclamation points (!), at signs (@), number signs (#), dollar signs ($), and percent signs (%)

3. Click **Log On**.
4. If your account has multi-factor authentication (MFA) enabled, perform corresponding operations in the following scenarios:
   - You log on to the Apsara Uni-manager Management Console for the first time after MFA is enabled by the administrator:
     a. On the Bind Virtual MFA Device page, bind an MFA device.
     b. Enter the username and password again as in Step 2 and click **Log On**.
     c. Enter a six-digit MFA verification code and click **Authenticate**.
   - You have enabled MFA and bound an MFA device:

     Enter a six-digit MFA verification code and click **Authenticate**.

     > **Note** For more information, see the *Bind a virtual MFA device to enable MFA* topic in *Apsara Uni-manager Management Console User Guide*.
5. In the top navigation bar, choose **Products > Database Services > AnalyticDB for PostgreSQL**.

2.3. Create an instance

This topic describes how to create an AnalyticDB for PostgreSQL instance in the console. Before you perform other operations, you must first create an AnalyticDB for PostgreSQL instance.

1. Log on to the AnalyticDB for PostgreSQL console.
2. In the upper-right corner of the page, click **Create Instance**.
3. On the **Create AnalyticDB for PostgreSQL Instance** page, configure the parameters described in the following table.

Section	Parameter	Description
	Organization	The organization to which the instance belongs.
	Resource Set	The resource set to which the instance belongs.

Section	Parameter	Description
Region	Region	The region in which you want to create the AnalyticDB for PostgreSQL instance. **Note** If you want to access the AnalyticDB for PostgreSQL instance from an Elastic Compute Service (ECS) instance over a virtual private cloud (VPC), you must create the instance within the same region and zone as the ECS instance.
	Zone	The zone in which you want to create the AnalyticDB for PostgreSQL instance.
Basic Settings	Chip Architecture	The chip architecture of the instance. **Note** This parameter is available only when Instance Resource Type is set to **Storage Included**.
	Instance Resource Type	The resource type of the instance. Select **Storage Included** or **Reserved Storage Mode**.
	Engine Version	The engine version of the instance. **Note** This parameter is available only when Instance Resource Type is set to **Storage Included**.
	Compute Group Specifications	Specifications for each compute group. The storage space and compute capability of a compute group vary based on the specified specifications.
	Compute Groups	The number of compute groups. Instance performance increases with the number of compute groups. **Note** This parameter is available only when Instance Resource Type is set to **Storage Included**.
	Master Nodes	The number of coordinator nodes. It is automatically set to 1. After you create an instance, you can add more coordinator nodes to the instance. For more information, see Change the specifications of an instance. **Note** This parameter is available only when Instance Resource Type is set to **Reserved Storage Mode**.

Section	Parameter	Description
	Compute Nodes	The number of compute nodes. The instance performance increases with the number of compute nodes. Valid values: 2 to 640. **Note** This parameter is available only when Instance Resource Type is set to **Reserved Storage Mode**.
	Compute Node Capacity	The capacity of compute nodes. It is related to the compute node specifications and cannot be modified. The CPU cores, memory size, and storage capacity are in a ratio of 1:8:80. For example, if the compute node specifications are 2 cores and 16 GB, the storage capacity is 160 GB. **Note** This parameter is available only when Instance Resource Type is set to **Reserved Storage Mode**.
	Storage Type	The storage type of the instance. It is automatically set to **Local SSD**. **Note** This parameter is available only when Instance Resource Type is set to **Reserved Storage Mode**.
Network	Network Type	The network type of the AnalyticDB for PostgreSQL instance. Valid values: ○ **Classic Network**: Cloud services in the classic network are not isolated. Unauthorized access to a cloud service is blocked only by the security group or whitelist policy of the service. ○ **VPC**: A VPC helps you build an isolated network environment on Apsara Stack. You can specify custom route tables, CIDR blocks, and gateways within a VPC. We recommend that you select VPC for improved security. You can create a VPC in advance, or change the network type to VPC after the instance is created.
	VPC	The VPC in which you want to create the AnalyticDB for PostgreSQL instance. **Note** This parameter is available only when Network Type is set to **VPC**.

Section	Parameter	Description
	vSwitch	The vSwitch to which the AnalyticDB for PostgreSQL instance is connected. **Note** This parameter is available only when Network Type is set to **VPC**.
	IP Address Whitelist	The IP addresses that are allowed to access the AnalyticDB for PostgreSQL instance.

4. Click **Submit**.

2.4. Configure a whitelist

To ensure a secure and stable database, you must add IP addresses or CIDR blocks that are allowed to access the database to a whitelist.

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Security Controls**.
4. On the **Security Controls** page, perform the following operations:
   - Create a whitelist
     a. Click **Create Whitelist**.

b. In the **Create Whitelist** panel, configure the parameters described in the following table.

Parameter	Description
Whitelist Name	The name of the whitelist. ■ The name can contain lowercase letters, digits, and underscores (_). ■ The name must start with a lowercase letter and end with a lowercase letter or a digit. ■ The name must be 2 to 32 characters in length.
IP Addresses	The IP addresses or CIDR blocks that are allowed to access the instance. ■ Separate multiple IP addresses with commas (,). A maximum of 999 unique IP addresses can be specified. ■ Supported formats are individual IP addresses such as 10.23.12.24 and CIDR blocks such as 10.23.12.24/24. In the preceding example, /24 indicates the subnet mask that specifies the length of the IP address prefix. An IP address prefix can be 1 to 32 bits in length. ■ If you enter CIDR blocks that have a prefix length of 0 such as 0.0.0.0/0 and 127.0.0.1/0, all IP addresses are allowed to access the instance. This poses a high security risk. Proceed with caution. ■ An IP address of 127.0.0.1 indicates that no external IP addresses are allowed to access the instance.

c. Click **OK**.

- Modify a whitelist
  a. Click **Modify** to the right of a whitelist.
  b. In the Modify Whitelist panel, add or remove IP addresses or CIDR blocks in the **IP Addresses** section.

  > **Note** **Whitelist Name** cannot be modified.

  c. Click **OK**.
- Delete a whitelist

  > **Note** The default whitelist cannot be deleted.

  a. Click **Delete** to the right of a whitelist.
  b. In the **Delete Whitelist** message, click **OK**.
- Clear the default whitelist
  a. Click **Clear** to the right of the default whitelist.
  b. In the **Clear Whitelist** message, click **OK**.

  After you clear the default whitelist, it contains only 127.0.0.1.

What's next

- We recommend that you maintain your whitelists on a regular basis to ensure secure access for AnalyticDB for PostgreSQL.
- You can click **Modify** or **Delete** to modify or delete custom whitelists.

2.5. Create a database account

This topic describes how to create a database account for an AnalyticDB for PostgreSQL instance.

Context

AnalyticDB for PostgreSQL provides the following types of database accounts:

- Privileged account: an account that has all permissions on all databases. The first account created for an instance in the console is a privileged account.
- Standard accounts: accounts that have all permissions only on their authorized databases.

> **Note** Permissions include SELECT, INSERT, UPDATE, DELETE, TRUNCATE, REFERENCES, and TRIGGER.

Precautions

- Before you use databases in an AnalyticDB for PostgreSQL instance, you must create a privileged account.
- Standard accounts cannot be created by using the console. For information about how to create standard accounts, see the "Execute SQL statements to create accounts" section of this topic.
- After you create a privileged account for an instance, you cannot delete the privileged account.

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Account Management**.
4. Click **Create Account**.
5. In the **Create Account** panel, configure the parameters described in the following table.

Parameter	Description
Account	The name of the privileged account. ○ The name can contain lowercase letters, digits, and underscores (_). ○ The name must start with a lowercase letter and end with a lowercase letter or a digit. ○ The name cannot start with gp. ○ The name must be 2 to 16 characters in length.

Parameter	Description
New Password	The password of the privileged account. ○ The password must contain at least three of the following character types: uppercase letters, lowercase letters, digits, and special characters. ○ The following special characters are supported: ! @ # $ % ^ & * ( ) _ + - = ○ The password must be 8 to 32 characters in length.
Confirm Password	Enter the password of the account again.

6. Click **OK**.

Notice After the account is created, you can click **Reset Password** in the **Actions** column to modify the account password. To ensure data security, we recommend that you change your password on a regular basis and do not use passwords that you have used before.

Execute SQL statements to create accounts

Before you execute SQL statements to create accounts, you must connect to a database. For more information, see Connect to a database.

- Create a privileged account.

```
CREATE ROLE <Account name> WITH LOGIN ENCRYPTED PASSWORD <Password> RDS_SUPERUSER;
```

Example:

```
CREATE role admin0  WITH LOGIN ENCRYPTED PASSWORD '111111' rds_superuser;
```

- Create a standard account.

```
CREATE ROLE <Account name> WITH LOGIN ENCRYPTED PASSWORD <'Password'>;
```

Example:

```
CREATE role test1 WITH LOGIN ENCRYPTED PASSWORD '111111';
```

2.6. Obtain client tools

The interface protocol of AnalyticDB for PostgreSQL is compatible with Greenplum Community Edition and PostgreSQL 8.2. You can use the Greenplum or PostgreSQL client to connect to AnalyticDB for PostgreSQL.

Note

Apsara Stack is an isolated environment. You must deploy software installation packages to the internal environment.

Graphical client tools

AnalyticDB for PostgreSQL users can directly use client tools that support Greenplum, such as SQL Workbench, Navicat Premium, and Navicat for PostgreSQL.

Command-line client psql (for RHEL 6, RHEL 7, CentOS 6, and CentOS 7)

For Red Hat Enterprise Linux (RHEL) 6, RHEL 7, CentOS 6, and CentOS 7, download the tool from the following links and then decompress the package:

- RHEL 6 or CentOS 6: hybriddb_client_package_el6
- RHEL 7 or CentOS 7: hybriddb_client_package_el7

Command-line client psql (for other Linux systems)

For other Linux systems, perform the following operations to compile the client tools:

1. Obtain the source code by using one of the following methods:
   - Obtain the code from the git directory. You must first install the git tool.

     ```
     git clone https://github.com/greenplum-db/gpdb.git
     cd gpdb
     git checkout 5d870156
     ```

   - Download the code.

     ```
     wget https://github.com/greenplum-db/gpdb/archive/5d87015609abd330c68a5402c1267fc86cbc9e1f.zip
     unzip 5d87015609abd330c68a5402c1267fc86cbc9e1f.zip
     cd gpdb-5d87015609abd330c68a5402c1267fc86cbc9e1f
     ```

2. Use GCC or other compilers to compile the code.

   ```
   ./configure
   make -j32
   make install
   ```

3. Obtain the psql and pg_dump tools from the following paths:

   psql: */usr/local/pgsql/bin/psql*

   pg_dump: */usr/local/pgsql/bin/pg_dump*

Command-line client psql (for Windows and other systems)

For Windows and other systems, go to the Pivotal website to download HybridDB Client.

2.7. Connect to a database

Greenplum Database and AnalyticDB for PostgreSQL are both developed based on PostgreSQL 8.2 and fully compatible with its message protocol. AnalyticDB for PostgreSQL users can use tools that support the PostgreSQL 8.2 message protocol, such as libpq, Java Database Connectivity (JDBC), Open Database Connectivity (ODBC), and psycopg2.

Context

AnalyticDB for PostgreSQL provides psql, a binary program of Red Hat. For more information about the download link, see Obtain the client tool. The Greenplum official website provides an easy-to-install installation package that includes JDBC, ODBC, and libpq. For more information, see Greenplum official documentation.

> **Note**
>
> - Apsara Stack is an isolated environment. To access Apsara Stack, you must prepare the necessary software installation packages in advance.
> - By default, AnalyticDB for PostgreSQL instances can be accessed only by clients that are deployed on Elastic Compute Service (ECS) instances within the same region and zone.

DMS

Data Management (DMS) allows you to manage relational databases such as MySQL, SQL Server, PostgreSQL, PPAS, and Petadata, online transaction processing (OLTP) databases such as PolarDB-X, online analytical processing (OLAP) databases such as AnalyticDB and Data Lake Analytics (DLA), and NoSQL databases such as MongoDB and Redis. DMS offers an integrated solution to manage data, schemas, and servers. You can also use DMS to authorize users, audit security, view BI charts and data trends, track data, and optimize performance.

This section describes how to use DMS to connect to an AnalyticDB for PostgreSQL instance.

1. Log on to the Log on to the AnalyticDB for PostgreSQL console.
2. (Optional)Create an AnalyticDB for PostgreSQL instance. For more information, see Creates an instance.

   If an AnalyticDB for PostgreSQL instance has already been created, skip this step.
3. Find the instance that you want to manage and click its instance ID.
4. (Optional)Create a database account. For more information, see Create an initial account.

   If an initial account has already been created, you can use this account to connect to the instance.
5. In the upper-right corner of the Basic Information page, click **Log On to Database**.
6. In the **Log On to Database Instance** dialog box, specify the **Database Account** and **Database password** parameters, and then click **Login**.

psql

psql is a common tool used together with Greenplum, and provides a variety of command functions. Its binary files are located in the *bin* directory of Greenplum. To use psql, perform the following steps:

1. Use one of the following methods to connect to an instance:
   - Connection string

     ```
     psql "host=yourgpdbaddress.gpdb.rds.aliyuncs.com port=3432 dbname=postgres user=gpdbaccount password=gpdbpassword"
     ```
   - Specified parameters

     ```
     psql -h yourgpdbaddress.gp.aliyun-inc.com -p 3432 -d postgres -U gpdbaccount
     ```

     Parameters:

     - -h: the host address.

- -p: the port number used to connect to the database.
- -d: the name of the database. The default value is postgres.
- -U: the account used to connect to the database.

You can run the `psql --help` command to view more options. You can also run `\?` to view the commands supported in psql.

2. Enter the password to go to the psql shell interface.

```
postgres=>
```

References

- For more information about the Greenplum psql usage, see psql.
- AnalyticDB for PostgreSQL also supports psql for PostgreSQL. Take note of the differences of psql commands between Greenplum and PostgreSQL. For more information, see PostgreSQL 8.3.23 Documentation - psql.

JDBC

Download the official JDBC of PostgreSQL. Then, add it to the environment variables.

Sample code:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class gp_conn {
    public static void main(String[] args) {
        try {
            Class.forName("org.postgresql.Driver");
            Connection db = DriverManager.getConnection("jdbc:postgresql://mygpdbpub.gpdb.r
ds.aliyuncs.com:3432/postgres",
                    "mygpdb", "mygpdb");
            Statement st = db.createStatement();
            ResultSet rs = st.executeQuery(
                    "select * from gp_segment_configuration;");
            while (rs.next()) {
                System.out.print(rs.getString(1));
                System.out.print(" | ");
                System.out.print(rs.getString(2));
                System.out.print(" | ");
                System.out.print(rs.getString(3));
                System.out.print(" | ");
                System.out.print(rs.getString(4));
                System.out.print(" | ");
                System.out.print(rs.getString(5));
                System.out.print(" | ");
                System.out.print(rs.getString(6));
                System.out.print(" | ");
                System.out.print(rs.getString(7));
                System.out.print(" | ");
                System.out.print(rs.getString(8));
                System.out.print(" | ");
                System.out.print(rs.getString(9));
                System.out.print(" | ");
                System.out.print(rs.getString(10));
                System.out.print(" | ");
                System.out.println(rs.getString(11));
            }
            rs.close();
            st.close();
        } catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Python

Python uses psycopg2 to connect to Greenplum and PostgreSQL. Perform the following operations:

1. Install psycopg2. Use one of the following methods to install psycopg2 in CentOS:

- Method 1: Run the following command:

```
yum -y install python-psycopg2
```

- Method 2: Run the following command:

```
pip install psycopg2
```

- Method 3: Run the following source code:

```
yum install -y postgresql-devel*
wget http://initd.org/psycopg/tarballs/PSYCOPG-2-6/psycopg2-2.6.tar.gz
tar xf psycopg2-2.6.tar.gz
cd psycopg2-2.6
python setup.py build
sudo python setup.py install
```

2. Run the following command to set PYTHONPATH and reference it:

```
import psycopg2
 sql = 'select * from gp_segment_configuration;'
 conn = psycopg2.connect(database='gpdb', user='mygpdb', password='mygpdb', host='mygpd
bpub.gpdb.rds.aliyuncs.com', port=3432)
 conn.autocommit = True
 cursor = conn.cursor()
 cursor.execute(sql)
 rows = cursor.fetchall()
 for row in rows:
     print row
 conn.commit()
 conn.close()
```

An output similar to the following one is displayed:

```
(1, -1, 'p', 'p', 's', 'u', 3022, '192.**.**.158', '192.**.**.158', None, None)(6, -1,
'm', 'm', 's', 'u', 3019, '192.**.**.47', '192.**.**.47', None, None)(2, 0, 'p', 'p', '
s', 'u', 3025, '192.**.**.148', '192.**.**.148', 3525, None)(4, 0, 'm', 'm', 's', 'u',
3024, '192.**.**.158', '192.**.**.158', 3524, None)(3, 1, 'p', 'p', 's', 'u', 3023, '19
2.**.**.158', '192.**.**.158', 3523, None)(5, 1, 'm', 'm', 's', 'u', 3026, '192.**.**.1
48', '192.**.**.148', 3526, None)
```

libpq

libpq is the C language interface to AnalyticDB for PostgreSQL. You can use the libpq library to access and manage PostgreSQL databases in a C program. You can find its static and dynamic libraries in the lib directory.

For more information about example programs, see Example Programs.

For more information about libpq, see PostgreSQL 9.4.17 Documentation - Chapter 31. libpq - C Library.

ODBC

PostgreSQL ODBC is an open source version based on the GNU Lesser General Public License (LGPL) protocol. You can download it from the PostgreSQL website.

1. Install the driver.

```
yum install -y unixODBC.x86_64
yum install -y postgresql-odbc.x86_64
```

2. View the driver configurations.

```
cat /etc/odbcinst.ini
```

Sample code:

```
# Example driver definitions
# Driver from the postgresql-odbc package
# Setup from the unixODBC package
[PostgreSQL]
Description = ODBC for PostgreSQL
Driver = /usr/lib/psqlodbcw.so
Setup = /usr/lib/libodbcpsqlS.so
Driver64 = /usr/lib64/psqlodbcw.so
Setup64 = /usr/lib64/libodbcpsqlS.so
FileUsage = 1
# Driver from the mysql-connector-odbc package
# Setup from the unixODBC package
[MySQL]
Description = ODBC for MySQL
Driver = /usr/lib/libmyodbc5.so
Setup = /usr/lib/libodbcmyS.so
Driver64 = /usr/lib64/libmyodbc5.so
Setup64 = /usr/lib64/libodbcmyS.so
FileUsage = 1
```

3. Configure the data source name (DSN). Replace `****` in the following code with the corresponding connection information:

```
[mygpdb]
Description = Test to gp
Driver = PostgreSQL
Database = ****
Servername = ****.gpdb.rds.aliyuncs.com
UserName = ****
Password = ****
Port = ****
ReadOnly = 0
```

4. Test the connectivity.

```
echo "select count(*) from pg_class" | isql mygpdb
```

The following information is returned if the connectivity test passes:

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

Test whether the database runs normally.

```
SELECT count(*) FROM pg_class;
```

The following information is returned if the database runs normally:

```
+---------------------+
| count               |
+---------------------+
| 388                 |
+---------------------+
SQLRowCount returns 1
1 rows fetched
```

5. After the ODBC driver is connected to the database, connect your application to the driver. For more information, see PostgreSQL ODBC driver and psqlODBC HOWTO - C#.

References

- Pivotal Greenplum official documentation
- PostgreSQL psqlODBC
- Compiling psqlODBC on Unix
- Download ODBC connectors
- Download JDBC connectors
- The PostgreSQL JDBC Interface

3.Instances

3.1. Reset the password of an account

If you forget the password of your database account in the AnalyticDB for PostgreSQL console, you can reset the password.

> **Note** To ensure data security, we recommend that you change your password on a regular basis.

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Account Management**.
4. Find an account and click **Reset Password** in the **Actions** column.
5. In the **Modify Account** panel, enter a new password in **New Password** and **Confirm Password**.

   The password must meet the following requirements:

   - The password contains at least three of the following character types: uppercase letters, lowercase letters, digits, and specific special characters.
   - The following special characters are supported:

     ! @ # $ % ^ & * ( ) _ + - =
   - The password is 8 to 32 characters in length.
6. Click **OK**.

3.2. Manage the endpoints of an instance

This topic describes how to view the internal endpoint and port number and how to apply to and release a public endpoint for an AnalyticDB for PostgreSQL instance.

View the internal endpoint and port number

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Database Connection**.
4. In the **Connection Information** section, view the internal endpoint and port number of the instance.

Apply for a public endpoint

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Database Connection**.

4. Click **Apply for Public Endpoint** to the right of **Public Endpoint**.

Release a public endpoint

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Database Connection**.
4. Click **Release Public Endpoint** to the right of **Public Endpoint**.
5. In the **Release Public Endpoint** message, click **OK**.

3.3. Monitoring and alerting

AnalyticDB for PostgreSQL provides the monitoring and alerting feature to help you view the running status of instances.

Procedure

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Monitoring and Alerts**.

   The following metrics are displayed on the **Monitoring and Alerts** page:

   - **Monitoring Summary** tab

     On this tab, you can view **CPU Utilization and Memory Usage (%)**, **Total Connections of Coordinate Node**, **Disk Bandwidth**, and **Disk Usage (MB)**.

   - **Compute Group Monitoring** or **Compute Node Monitoring** tab

     The **Compute Group Monitoring** tab appears if the instance is in storage included mode. The **Compute Node Monitoring** tab appears if the instance is in reserved storage mode.

     On this tab, you can view **CPU Utilization and Memory Usage (%)**, **Write IOPS**, **Read IOPS**, and **Disk Usage (MB)**.

4. (Optional)Select a time range to view metrics within the specified period. The following options are available for **Time Range**:
   - **1 Hour**: You can view metrics within the last hour.
   - **1 Day**: You can view metrics within the last day.
   - **Custom time range**: You can view metrics within a custom time range. The time range cannot be longer than seven days.

3.4. Change the specifications of an instance

This topic describes how to change specifications for an instance in storage included mode or reserved storage mode.

Storage included mode

1. Log on to the AnalyticDB for PostgreSQL console.

2. Find the instance that you want to manage and click **Change Specifications** in the **Actions** column.
3. Select new values for **Compute Group Specifications** and **Compute Groups**.
4. Click **Submit**.

> **Warning** When the specifications of an instance are being changed, no data can be read from or written to the instance. Perform the specification change operation during an appropriate period of time.

Reserved storage mode

Change the specifications of the coordinator node:

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click **Change Master Node Specifications** in the **Actions** column.
3. Select a new value for **Master Nodes**.
4. Click **Submit**.

> **Warning** When the specifications of coordinator nodes are being changed, DDL operations may be blocked. Perform the specification change operation during an appropriate period of time.

Change the specifications of compute nodes:

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click **Change Compute Node Specifications** in the **Actions** column.
3. Select new values for **Compute Group Specifications** and **Compute Nodes**.
4. Click **Submit**.

> **Warning** When the specifications of compute nodes are being changed, no data can be read from or written to the instance. Perform the specification change operation during an appropriate period of time.

3.5. Restart an instance

When the number of connections of an instance reaches the upper limit or the instance has performance issues, you can restart the instance.

Precautions

The restart process takes about 3 to 30 minutes. During the restart period, the instance cannot provide external services. We recommend that you take precautionary measures before you restart the instance. After the instance is restarted and enters the running state, you can access the instance.

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.

3. In the upper-right corner of the page, click **Restart Instance**.
4. In the **Restart Instance** message, click **OK**.

> **Warning** The database service may be interrupted when you restart an instance. Proceed with caution.

3.6. Release an instance

This topic describes how to manually release an AnalyticDB for PostgreSQL instance.

Procedure

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the **Status** section of the **Basic Information** page, click **Release**.
4. In the **Release Instance** message, click **OK**.

> **Warning** After an instance is released, all of its data is deleted. Back up your data before you release an instance.

3.7. Configure SQL audit

AnalyticDB for PostgreSQL allows you to use the SQL audit feature to view SQL details and audit SQL queries on a regular basis. The SQL audit feature does not affect instance performance.

Storage included mode

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane of the Basic Information page, click **Security Controls**.
4. Click the **SQL Audit** tab.
5. (Optional)Click **Enable SQL Audit**.

   If you have enabled SQL audit, skip this step.
6. On the **SQL Audit** tab, query SQL information based on conditions such as the time range, database, user, and keyword.

> **Note** You can click **Disable SQL Audit** to disable the SQL audit feature.

Reserved storage mode

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane of the Basic Information page, click **SQL Audit**.
4. On the **SQL Audit** page, query SQL information based on conditions such as the database, execution duration, time range, database user, source IP address, execution status, operation

category, operation type, and statement.

> **Note** You can click the icon to export SQL information from the current page.

3.8. SSL encryption

To enhance data transmission security, you can enable SSL encryption for your instances. SSL encryption can encrypt network connections at the transport layer. This improves data security and ensures data integrity.

Precautions

SSL encryption is supported only for instances in **reserved storage mode**.

Enable SSL encryption

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Security Controls**.
4. Click the **SSL** tab.
5. On the **SSL** tab, turn on **SSL Encryption**.
6. In the **Enable SSL Encryption** message, click **OK**.

> **Note** If you modify the status of SSL encryption, your instance restarts. We recommend that you perform this operation during off-peak hours.

Update the validity of an SSL certificate

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Security Controls**.
4. Click the **SSL** tab.
5. On the **SSL** tab, click **Update Validity** to the right of **SSL Encryption**.
6. In the **Update SSL Certificate Validity** message, click **OK**.

> **Note** If you modify the status of SSL encryption, your instance restarts. We recommend that you perform this operation during off-peak hours.

Disable SSL encryption

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the left-side navigation pane, click **Security Controls**.
4. Click the **SSL** tab.
5. On the **SSL** tab, turn off **SSL Encryption**.

6. In the **Disable SSL Encryption** message, click **OK**.

> **Note** If you modify the status of SSL encryption, your instance restarts. We recommend that you perform this operation during off-peak hours.

3.9. Update the minor engine version

To better meet your requirements, the minor engine version of AnalyticDB for PostgreSQL is updated on a regular basis. When you create an instance, the latest database engine version is used by default. After a new version is released, you can update your instance to the new version and use its extended features. This topic describes how to update the minor engine version of an instance.

Precautions

When you update the minor engine version of an instance, the instance restarts and is unavailable during the restart. We recommend that you update the minor engine version during off-peak hours.

Procedure

1. Log on to the AnalyticDB for PostgreSQL console.
2. Find the instance that you want to manage and click its ID.
3. In the upper-right corner of the **Basic Information** page, click **Upgrade Minor Version**.
4. In the **Upgrade Minor Version** message, click **OK**.

> **Note** Minor version updates take 3 to 15 minutes to complete. When an instance is being updated, it is unavailable. We recommend that you prepare in advance before you update the instance. After the update is complete, the instance enters the Running state and you can access the instance databases.

5. After you complete the preceding steps, check the state of the instance. If the update is complete, the instance enters the **Running** state. Otherwise, the instance remains in the **Upgrading version** state.

   Before the update, the system checks the minor engine version of your instance. If the latest minor version is used, the system skips the update and restart operations.

3.10. Import data

3.10.1. Import data from or export data to OSS in parallel

AnalyticDB for PostgreSQL allows you to import data from or export data to Object Storage Service (OSS) tables in parallel by using the OSS external table feature (also called gpossext). AnalyticDB for PostgreSQL also supports GZIP compression for OSS external tables to reduce the file size and storage costs. gpossext can read from and write to TEXT and CSV files, even when they are compressed in GZIP packages.

- Create an OSS external table plug-in (oss_ext)

To use an OSS external table, you must first create an OSS external table plug-in in AnalyticDB for PostgreSQL. You must create a plug-in for each database that you need to access.

- To create the plug-in, execute the `CREATE EXTENSION IF NOT EXISTS oss_ext;` statement.
- To delete the plug-in, execute the `DROP EXTENSION IF EXISTS oss_ext;` statement.

- Import data in parallel
  i. Distribute data evenly into multiple files in OSS. We recommend that you set the number of OSS files to an integer that is the multiple of the number of compute nodes in AnalyticDB for PostgreSQL.
  ii. Create a READABLE external table in AnalyticDB for PostgreSQL.
  iii. Execute the following statement to import data in parallel:

```
INSERT INTO <Destination table> SELECT * FROM <External table>
```

> **Note**
> - The data import performance depends on the OSS performance and resources of AnalyticDB for PostgreSQL instances, such as CPU, I/O, memory, and network resources. To ensure the best import performance, we recommend that you use column store and compression when you create a table. For example, you can specify the following clause: `WITH (APPENDONLY=true, ORIENTATION=column, COMPRESSTYPE=zlib, COMPRESSLEVEL=5, BLOCKSIZE=1048576)` . For more information, see CREATE TABLE.
> - To ensure the best import performance, we recommend that you configure OSS and AnalyticDB for PostgreSQL within the same region.

- Export data in parallel
  i. Create a WRITABLE external table in AnalyticDB for PostgreSQL.
  ii. Execute the following statement to export data to OSS in parallel:

```
INSERT INTO <External table> SELECT * FROM <Source table>
```

- Create OSS external tables

> **Note** The syntax to create and use external tables is the same as that of Greenplum Database, except for the syntax of location-related parameters.

```
CREATE [READABLE] EXTERNAL TABLE tablename
( columnname datatype [, ...] | LIKE othertable )
LOCATION ('ossprotocol')
FORMAT 'TEXT'
            [( [HEADER]
               [DELIMITER [AS] 'delimiter' | 'OFF']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
           | 'CSV'
            [( [HEADER]
               [QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE NOT NULL column [, ...]]
               [ESCAPE [AS] 'escape']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
[ ENCODING 'encoding' ]
[ [LOG ERRORS [INTO error_table]] SEGMENT REJECT LIMIT count
       [ROWS | PERCENT] ]
CREATE WRITABLE EXTERNAL TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION ('ossprotocol')
FORMAT 'TEXT'
               [( [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [ESCAPE [AS] 'escape' | 'OFF'] )]
          | 'CSV'
               [([QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE QUOTE column [, ...]] ]
               [ESCAPE [AS] 'escape'] )]
[ ENCODING 'encoding' ]
[ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
ossprotocol:
   oss://oss_endpoint prefix=prefix_name
    id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|
false]
ossprotocol:
   oss://oss_endpoint dir=[folder/[folder/]...]/file_name
    id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|
false]
ossprotocol:
   oss://oss_endpoint filepath=[folder/[folder/]...]/file_name
    id=userossid key=userosskey bucket=ossbucket compressiontype=[none|gzip] async=[true|
false]
```

Parameters

Common parameters

Parameter	Description
Protocol and endpoint	This parameter is in the `protocol name://oss_endpoint` format. The protocol name is oss. oss_endpoint is the domain name that is used to access OSS in a region. **Note** You can access the database from a Virtual Private Cloud (VPC) host by using an internal endpoint that contains "internal" in the name to avoid generating public traffic.
id	The AccessKey ID of the OSS account.
key	The AccessKey secret of the OSS account.
bucket	The bucket where the data file is located. You must use OSS to create the bucket before you import data.

prefix	The prefix of the path name corresponding to the data file. Prefixes are directly matched and cannot be controlled by regular expressions. The prefix, filepath, and dir parameters are mutually exclusive. Only one of the parameters can be specified at a time.
	• If you create a READABLE external table for data import, all OSS files that contain the specified prefix are imported.
	  ◦ If you set prefix to test/filename, the following files are imported:
	  ▪ test/filename
	  ▪ test/filenamexxx
	  ▪ test/filename/aa
	  ▪ test/filenameyyy/aa
	  ▪ test/filenameyyy/bb/aa
	  ◦ If you set prefix to test/filename/, only the following file out of the preceding files is imported:
	  test/filename/aa
	• If you create a WRITABLE external table for data export, each exported file has a unique name based on this parameter.
	
	Note One or more files can be exported for each compute node. The names of exported files are in the `prefix_tablename_uuid.x` format. uuid indicates a timestamp in microseconds as an int64 value. x indicates the node ID. You can use an external table for multiple export operations. Each export operation is assigned a uuid value. The files exported during each operation share a uuid value.

Parameter	Description
dir	The virtual folder path in OSS. The prefix, filepath, and dir parameters are mutually exclusive. Only one of the parameters can be specified at a time. • A folder path must end with a forward slash (/). Example: `test/mydir/`. • If you use this parameter when you create an external table for data import, all files in the specified virtual directory (except for its subdirectories and contained files) are imported. The dir parameter is different from the filepath parameter and does not require you to specify the names of files in a directory. • If you use this parameter when you create an external table for data export, all data is exported as multiple files within the specified directory. The names of exported files are in the `filename.x` format, where x indicates a number. The values of x may not be consecutive.

filepath	The file name that contains a path in OSS. The prefix, filepath, and dir parameters are mutually exclusive. Only one of the parameters can be specified at a time. You can specify only the filepath parameter when you create a READABLE external table for data import.
	• The file name includes the file path, but not the bucket name.
	• The file name that is specified for data import must be in the `filename` or `filename.x` format. The values of x must be consecutive numbers that start from 1.
	For example, if filepath is set to filename and OSS contains the following files, the imported files include filename, filename.1, and filename.2, but not filename.4 because filename.3 does not exist.
	`filename`
	`filename.1`
	`filename.2`
	`filename.4`

Import mode parameters

Parameter	Description
async	Specifies whether to enable asynchronous data import. • By default, asynchronous data import is enabled. You can set async to false or f to disable asynchronous data import. • You can enable the worker thread to load data from OSS to accelerate the import performance. By default, asynchronous data import is used. • Asynchronous data import consumes more hardware resources than normal data import.

Parameter	Description
compressiontype	The compression format of the imported files. Default value: none. Valid values: • none: The import files are not compressed. • gzip: The imported files are compressed in the GZIP format. Only the GZIP format is supported.
compressionlevel	The compression level of the files that are written to OSS. Valid values: 1 to 9. Default value: 6.

Export mode parameters

Parameter	Description
oss_flush_block_size	The size of each data block that is written to OSS. Valid values: 1 to 128. Default value: 32. Unit: MB.
oss_file_max_size	The maximum size for each file that is written to OSS. If the limit is exceeded, subsequent data is written to another file. Valid values: 8 to 4000. Default value: 1024. Unit: MB.
num_parallel_worker	The number of parallel compression threads for data that is written to OSS. Valid values: 1 to 8. Default value: 3.

For data export, take note of the following items:

- WRITABLE is the keyword of the external table for data export. You must specify this keyword when you create an external table.
- Only the prefix and dir parameters are supported for data export. The filepath parameter is not supported.
- You can use the DISTRIBUTED BY clause to write data from compute nodes to OSS based on the specified distribution keys.

Other common parameters

The following table describes the fault-tolerance parameters that can be used for data import and export.

Fault-tolerance parameters

Parameter	Description
oss_connect_timeout	The connection timeout period. Default value: 10. Unit: seconds.
oss_dns_cache_timeout	The Alibaba Cloud DNS timeout period. Default value: 60. Unit: seconds.
oss_speed_limit	The minimum rate tolerated. Default value: 1024 bit/s (1 Kbit/s).

Parameter	Description
oss_speed_time	The maximum amount of time tolerated. Default value: 15. Unit: seconds.

When the default values are used for the preceding parameters, a timeout occurs if the transmission rate is lower than 1 Kbit/s for 15 consecutive seconds. For more information, see Error handling in the C-SDK part of *Object Storage Service Developer Guide*.

The other parameters are compatible with the external table syntax of Greenplum Database. For more information about the syntax, see CREATE EXTERNAL TABLE. The following parameters are included:

- FORMAT: the supported file format, such as TEXT and CSV.
- ENCODING: the data encoding format of a file, such as UTF-8.
- LOG ERRORS refers to improperly imported data that can be ignored and is instead written to error_table. You can also use the count parameter to specify the error reporting threshold.

Examples

1. Create an OSS external table named ossexample to import data.

```
CREATE READABLE EXTERNAL TABLE ossexample
        (date text, time text, open float, high float,
        low float, volume int)
        location('oss://oss-cn-hangzhou.aliyuncs.com
        prefix=osstest/example id=XXX
        key=XXX bucket=testbucket compressiontype=gzip')
        FORMAT 'csv' (QUOTE '''' DELIMITER E'\t')
        ENCODING 'utf8'
        LOG ERRORS INTO my_error_rows SEGMENT REJECT LIMIT 5;
CREATE READABLE EXTERNAL TABLE ossexample
        (date text, time text, open float, high float,
        low float, volume int)
        location('oss://oss-cn-hangzhou.aliyuncs.com
        dir=osstest/ id=XXX
        key=XXX bucket=testbucket')
        FORMAT 'csv'
        LOG ERRORS SEGMENT REJECT LIMIT 5;
CREATE READABLE EXTERNAL TABLE ossexample
        (date text, time text, open float, high float,
        low float, volume int)
        location('oss://oss-cn-hangzhou.aliyuncs.com
        filepath=osstest/example.csv id=XXX
        key=XXX bucket=testbucket')
        FORMAT 'csv'
        LOG ERRORS SEGMENT REJECT LIMIT 5;
```

2. Create an OSS external table to which data is exported.

```
CREATE WRITABLE EXTERNAL TABLE ossexample_exp
        (date text, time text, open float, high float,
        low float, volume int)
        location('oss://oss-cn-hangzhou.aliyuncs.com
        prefix=osstest/exp/outfromhdb id=XXX
        key=XXX bucket=testbucket') FORMAT 'csv'
        DISTRIBUTED BY (date);
CREATE WRITABLE EXTERNAL TABLE ossexample_exp
        (date text, time text, open float, high float,
        low float, volume int)
        location('oss://oss-cn-hangzhou.aliyuncs.com
        dir=osstest/exp/ id=XXX
        key=XXX bucket=testbucket') FORMAT 'csv'
        DISTRIBUTED BY (date);
```

3. Create a heap table named example to which you want to import data.

```
CREATE TABLE example
        (date text, time text, open float,
         high float, low float, volume int)
         DISTRIBUTED BY (date);
```

4. Import data to the example heap table from the ossexample table in parallel.

```
INSERT INTO example SELECT * FROM ossexample;
```

5. Export data from the example heap table to OSS in parallel.

```
INSERT INTO ossexample_exp SELECT * FROM example;
```

6. Execute an execution plan. The following execution plan shows that all compute nodes are involved in the task.

```
EXPLAIN INSERT INTO example SELECT * FROM ossexample;
```

All compute nodes read data from OSS in parallel. AnalyticDB for PostgreSQL performs a redistribution motion operation to compute the data by using a hash algorithm, and then distributes the data to its compute nodes after computing. After a compute node receives data, it performs an INSERT operation to add the data to AnalyticDB for PostgreSQL. The following information is returned:

```
                                        QUERY PLAN
-------------------------------------------------------------------------------------
----------
 Insert (slice0; segments: 4)  (rows=250000 width=92)
   ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..11000.00 rows=2500
00 width=92)
         Hash Key: ossexample.date
         ->  External Scan on ossexample  (cost=0.00..11000.00 rows=250000 width=92)
(4 rows)
```

```
EXPLAIN INSERT INTO ossexample_exp SELECT * FROM example;
```

The following execution plan shows that each compute node directly exports data to OSS without redistributing the data. The following information is returned:

```
                          QUERY PLAN
------------------------------------------------------------------
 Insert (slice0; segments: 3)  (rows=1 width=92)
   ->  Seq Scan on example  (cost=0.00..0.00 rows=1 width=92)
(2 rows)
```

TEXT and CSV format description

The following parameters specify the formats of files read from and written to OSS. You can specify the parameters in the external DDL parameters.

- \n: the line feed for TEXT and CSV files.
- DELIMITER: the delimiter of columns.
  - If the DELIMITER parameter is specified, the QUOTE parameter must also be specified.
  - Recommended column delimiters include commas (,), vertical bars (|), and special characters such as \t.
- QUOTE: encloses user data that contains special characters by column.
  - Strings that contain special characters must be enclosed by QUOTE to differentiate user data from control characters.
  - To optimize the efficiency, it is unnecessary to enclose data such as integers in QUOTE characters.
  - QUOTE characters cannot be the same as those specified by DELIMITER. The default value of QUOTE is a pair of double quotation marks ("").
  - User data that contains QUOTE characters must also contain ESCAPE characters to differentiate user data from machine code.
- ESCAPE: the escape character.
  - Place an escape character before a special character that needs to be escaped to indicate that it is not a special character.
  - If ESCAPE is not specified, the default value is the same as QUOTE.
  - You can also use other characters such as backslashes (\) as ESCAPE characters, which is used by MySQL.

Default control characters for TEXT and CSV files

The following table describes the default control characters for TEXT and CSV files.

Control character	TEXT	CSV
DELIMITER	Tab (\t)	Comma (,)
QUOTE	double quotation mark (")	double quotation mark (")
ESCAPE	N/A	Same as QUOTE
NULL	Backslash plus N (\N)	Empty string without quotation marks

> **Note** All control characters must be single-byte characters.

SDK troubleshooting

The following table lists the error logs generated when an error occurs during the import or export process.

Error log information

Keyword	Description
code	The HTTP status code of the error request.
error_code	The error code returned by OSS.
error_msg	The error message returned by OSS.
req_id	The UUID used to identify the request. If you require assistance in solving a problem, you can submit a ticket that contains the req_id value of the failed request.

You can handle timeout-related errors by using parameters related to oss_ext.

References

- CREATE EXTERNAL TABLE
- CREATE TABLE

3.10.2. Import data from MySQL

You can use the mysql2pgsql tool to migrate tables from MySQL to AnalyticDB for PostgreSQL, Greenplum Database, PostgreSQL, or Postgres Plus Advanced Server (PPAS).

Background information

mysql2pgsql connects to both the source MySQL database and the destination AnalyticDB for PostgreSQL database. The tool retrieves the data that you want to export from the source MySQL database, and uses a COPY statement to import the data to the destination database. This tool supports simultaneous data import over multiple threads. Each worker thread imports data from specific database tables.

You can download the binary installation package of mysql2pgsql.

You can view instructions on source code compilation of mysql2pgsql.

Procedure

1. Modify the my.cfg configuration file to configure the connection information of source and destination databases.

i. Modify the connection information of the source MySQL database.

> **Note** You must have the read permissions on all user tables.

```
[src.mysql]
host = "192.168.1.1"
port = "3306"
user = "test"
password = "test"
db = "test"
encodingdir = "share"
encoding = "utf8"
```

ii. Modify the connection information of the destination PostgreSQL, PPAS, or AnalyticDB for PostgreSQL database.

> **Note** You must have the write permissions on the destination table.

```
[desc.pgsql]
connect_string = "host=192.168.1.2 dbname=test port=3432  user=test password=pgsql"
```

2. Use mysql2pgsql to import data.

```
./mysql2pgsql -l <tables_list_file> -d -n -j <number of threads> -s <schema of target a
ble>
```

Parameters

Parameter	Description

-l	Optional. This parameter is used to specify a text file that contains tables to be synchronized. If you do not specify this parameter, all the tables in the database that is specified in the configuration file are synchronized. `<tables_list_file>` specifies the name of a file that contains a collection of tables to be synchronized and conditions for table queries. You can specify the file content in the following format:
	`table1 : select * from table_big where column1 < '2016-08-05'`
	`table2 :`
	`table3`
	`table4: select column1, column2 from tableX where column1 != 10`
	`table5: select * from table_big where column1 >= '2016-08-05'`

-d	Optional. This parameter indicates the table creation DDL statement that creates the destination table but does not synchronize data.
-n	Optional. This parameter must be used together with -d to specify that the table partitioning definition is not included in the DDL statement.

Parameter	Description
-j	Optional. This parameter is used to specify the number of threads that are used for data synchronization. If you do not specify this parameter, five concurrent threads are used.
-s	Optional. This parameter is used to specify the schema of the destination table. Only a single schema can be specified at a time by the command. If you do not specify this parameter, the data is imported to the table under the public schema.

Typical usage

Full database migration

1. Run the following command to obtain the DDL statements of the corresponding destination table:

   ```
   ./mysql2pgsql -d
   ```

2. Create a table in the destination database based on these DDL statements with the distribution key information added.
3. Run the following command to synchronize all tables:

   ```
   ./mysql2pgsql
   ```

   This command migrates the data from all MySQL tables in the database that is specified in the configuration file to the destination database. By default, five concurrent threads are used to read and import data from involved tables.

Partial table migration

1. Create a file named tab_list.txt and enter the following content:

   ```
   t1
   t2 : select * from t2 where c1 > 138888
   ```

2. Run the following command to synchronize the specified t1 and t2 tables. For the t2 table, only data that meets the c1 > 138888 condition is migrated.

   ```
   ./mysql2pgsql -l tab_list.txt
   ```

3.10.3. Import data from PostgreSQL

You can use the pgsql2pgsql tool to migrate tables across AnalyticDB for PostgreSQL, Greenplum Database, PostgreSQL, and Postgres Plus Advanced Server (PPAS).

Context

pgsql2pgsql supports the following features:

- Full migration across PostgreSQL, PPAS, Greenplum Database, and AnalyticDB for PostgreSQL.
- Full migration and incremental migration from PostgreSQL or PPAS (version 9.4 or later) to AnalyticDB for PostgreSQL or PPAS.

You can perform the following operations in the dbsync project library:

- Download the binary installation package of pgsql2pgsql.
- View instructions on source code compilation of pgsql2pgsql.

Procedure

1. Modify the my.cfg configuration file to configure the connection information of source and destination databases.
   i. Modify the connection information of the source PostgreSQL database.

   > **Note** In the connection information of the source PostgreSQL database, we recommend that you set the user to the owner of the source database.

   ```
   [src.pgsql]
   connect_string = "host=192.168.0.1 dbname=test port=3432  user=test password=pgsql"
   ```

   ii. Modify the connection information of the self-managed PostgreSQL database.

   ```
   [local.pgsql]
   connect_string = "host=192.168.0.2 dbname=test port=3432  user=test2 password=pgsql
   "
   ```

   iii. Modify the connection information of the destination PostgreSQL database.

   > **Note** You must have the write permissions on the destination table.

   ```
   [desc.pgsql]
   connect_string = "host=192.168.0.2 dbname=test port=3432  user=test3 password=pgsql
   "
   ```

   > **Note**
   > - If you want to synchronize incremental data, you must have the permissions to create replication slots in the source database.
   > - PostgreSQL 9.4 and later support logic flow replication. Therefore, source databases of these versions support incremental data migration. A kernel supports logic flow replication only after you configure the following parameters:
   >
   >   wal_level = logical
   >
   >   max_wal_senders = 6
   >
   >   max_replication_slots = 6

2. Use pgsql2pgsql to perform full database migration.

   ```
   ./pgsql2pgsql
   ```

   By default, the migration program migrates the table data of all users from the source PostgreSQL database to the destination PostgreSQL database.

3. View the status information.

   You can view the status information in a single migration process by connecting to the self-managed database. The status information is stored in the db_sync_status table and includes the start and end time of full data migration, the start time of incremental data migration, and the status of incremental data synchronization.

3.10.4. Use the COPY statement to import data

You can use the `\copy` command to import data from local text files to AnalyticDB for PostgreSQL instances. Local text must be properly formatted and include necessary delimiters such as commas (,), semicolons (;), or special characters.

Context

- Large amounts of data cannot be written in parallel because the `\copy` command writes data in series by using the coordinator node. If you need to import large amounts of data in parallel, you can use the data import method based on Object Storage Service (OSS).
- The `\copy` command is a psql instruction. If you use the database statement `COPY` instead of the `\copy` command, be aware that only stdin is supported. The COPY statement does not support files because the root user does not have the superuser permissions to perform operations on files.
- AnalyticDB for PostgreSQL also allows you to use Java Database Connectivity (JDBC) to execute the COPY statement. The CopyIn method is encapsulated within JDBC. For more information, see Interface CopyIn.
- For more information about how to use the COPY statement, see COPY.

Procedure

1. Execute the following COPY statement to import data:

```
\COPY table [(column [, ...])] FROM {'file' | STDIN}
     [ [WITH]
       [OIDS]
       [HEADER]
       [DELIMITER [ AS ] 'delimiter']
       [NULL [ AS ] 'null string']
       [ESCAPE [ AS ] 'escape' | 'OFF']
       [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
       [CSV [QUOTE [ AS ] 'quote']
            [FORCE NOT NULL column [, ...]]
       [FILL MISSING FIELDS]
       [[LOG ERRORS [INTO error_table] [KEEP]
       SEGMENT REJECT LIMIT count [ROWS | PERCENT] ]
\COPY {table [(column [, ...])] | (query)} TO {'file' | STDOUT}
      [ [WITH]
        [OIDS]
        [HEADER]
        [DELIMITER [ AS ] 'delimiter']
        [NULL [ AS ] 'null string']
        [ESCAPE [ AS ] 'escape' | 'OFF']
        [CSV [QUOTE [ AS ] 'quote']
             [FORCE QUOTE column [, ...]] ]
      [IGNORE EXTERNAL PARTITIONS ]
```

4.Databases

4.1. Overview

Basically, you can use AnalyticDB for PostgreSQL in the same manner as you use Greenplum Database. They have the same schemas, supported data types, and user permission requirements. Except for specific operations that are exclusive to Greenplum Database such as those on distribution keys and append-optimized (AO) tables, all operations are the same as those of the open source PostgreSQL.

References

- Pivotal Greenplum official documentation
- GP 4.3 Best Practice
- Greenplum数据分布黄金法则

4.2. Create a database

After you connect to an AnalyticDB for PostgreSQL instance, you can execute SQL statements to create databases.

As in PostgreSQL, you can execute SQL statements to create databases in AnalyticDB for PostgreSQL. For example, after the **psql** client is used to connect to AnalyticDB for PostgreSQL, execute the following statement:

```
CREATE DATABASE mygpdb;
```

View database information.

```
\c mygpdb
```

The following information is returned:

```
You are now connected to database "mygpdb" as user "mygpdb".
```

4.3. Create a distribution key

AnalyticDB for PostgreSQL is a distributed database service where data is distributed across all the data nodes. You must create distribution keys to distribute data in AnalyticDB for PostgreSQL. Distribution keys are vital to query performance. Distribution keys are used to ensure even data distribution. Appropriate selection of keys can significantly improve query performance.

Specify a distribution key

In AnalyticDB for PostgreSQL, the data of a table can be distributed across all compute nodes in hash or random mode. When you create a table, you must specify a distribution key. Imported data is distributed to specific compute nodes based on the hash value calculated by using the distribution key.

```
CREATE TABLE vtbl(id serial, key integer, value text, shape cuboid, location geometry, comm
ent text) DISTRIBUTED BY (key);
```

If you do not specify a distribution key in the DISTRIBUTED BY clause, AnalyticDB for PostgreSQL distributes data by id field by using the round-robin algorithm.

Rules for selecting a distribution key

- Select evenly distributed columns or multiple columns to prevent data skew.
- Select fields that are commonly used for connection operations, especially for highly concurrent statements.
- Select conditional columns that feature high concurrency queries and high filterability.
- Do not use random distribution.

4.4. Construct data

In some test scenarios, you must construct data to fill the database.

1. Create a function that generates random strings.

```
CREATE OR REPLACE FUNCTION random_string(integer) RETURNS text AS $body$
SELECT array_to_string(array
                       (SELECT substring('0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghij
klmnopqrstuvwxyz'
                                         FROM (ceil(random()*62))::int
                                         FOR 1)
                        FROM generate_series(1, $1)), '');
$body$
LANGUAGE SQL VOLATILE;
```

2. Create a partition key.

```
CREATE TABLE tbl(id serial, KEY integer, locate geometry, COMMENT text) distributed by
(key);
```

3. Construct data.

```
INSERT INTO tbl(KEY, COMMENT, locate)
 SELECT
     KEY,
     COMMENT,
     ST_GeomFromText(locate) AS locate
 FROM
     (SELECT
         (a + 1) AS KEY,
         random_string(ceil(random() * 24)::integer) AS COMMENT,
         'POINT(' || ceil(random() * 36 + 99) || ' ' || ceil(random() * 24 + 50) || ')'
AS locate
     FROM
         generate_series(0, 99999) AS a)
     AS t;
```

4.5. Query data

This topic describes the query statements and how to view the query plans.

Sample query statement

```
SELECT * FROM tbl WHERE key = 751;
```

The following information is returned:

```
  | id  | key | value | shape        | locate                                          | comment
|
  +-----+-----+-------+--------------+-------------------------------------------------+--------
--------+
  | 751 | 751 | red   | 0101000000000000000C05B4000000000000004A40 | B9hPhjeNWPqV |
  (1 row)
  Time: 513.101 ms
```

View a query plan

```
EXPLAIN SELECT * FROM tbl WHERE key = 751;
```

The following information is returned:

```
  Gather Motion 1:1  (slice1; segments: 1)  (cost=0.00..1519.28 rows=1 width=53)
    ->  Seq Scan on tbl  (cost=0.00..1519.28 rows=1 width=53)
          Filter: key = 751
  Settings:  effective_cache_size=8GB; gp_statistics_use_fkeys=on
  Optimizer status: legacy query optimizer
```

4.6. Manage extensions

You can use extensions to expand database features. AnalyticDB for PostgreSQL enables you to manage extensions.

Supported extensions

AnalyticDB for PostgreSQL supports the following extensions:

- PostGIS: processes geographic data.
- MADlib: provides a machine learning function library.
- fuzzystrmatch: implements fuzzy match of strings.
- orafunc: provides compatibility with some Oracle functions.
- oss_ext: reads data from Object Storage Service (OSS).
- HyperLogLog: collects statistics.
- PL/Java: compiles user-defined functions (UDFs) in PL/Java.
- pgcrypto: provides cryptographic functions.
- IntArray: provides integer array-related functions, operators, and indexes.

Create an extension

Execute the following statements to create an extension:

```
CREATE EXTENSION <extension name>;
CREATE SCHEMA <schema name>;
CREATE EXTENSION IF NOT EXISTS <extension name> WITH SCHEMA <schema name>;
```

> **Note**
>
> Before you create a MADlib extension, you must create a plpythonu extension.
>
> ```
> CREATE EXTENSION plpythonu;
> CREATE EXTENSION madlib;
> ```

Delete an extension

Execute the following statements to delete an extension.

```
DROP EXTENSION <extension name>;
DROP EXTENSION IF EXISTS <extension name> CASCADE;
```

> **Note** If an object depends on an extension that you want to delete, you must add the CASCADE keyword to delete the object.

4.7. Manage users and permissions

This topic describes how to manage users and permissions in AnalyticDB for PostgreSQL.

Manage users

The system prompts you to specify an initial username and password when you create an instance. This initial user is the root user. After the instance is created, you can use the root user to connect to a database. The system also creates superusers such as aurora and replicator for internal management.

You can run the `\du+` command to view the information of all the users after you connect to the database by using the client tool of PostgreSQL or Greenplum. Sample output:

```
                                   List of roles
  Role name   |            Attributes             | Member of |  Description
--------------+-----------------------------------+-----------+---------------
 root_user    |                                   |      | rds_superuser
 ...
```

AnalyticDB for PostgreSQL does not provide superuser permissions but offers a similar role RDS_SUPERUSER. This is consistent with the permission system of ApsaraDB RDS for PostgreSQL. The root user, such as root_user in the preceding sample output, has the permissions of the RDS_SUPERUSER role. This permission attribute can be identified only by viewing the user description.

The root user has the following permissions:

- Create databases and users and perform operations such as LOGIN, excluding the SUPERUSER permissions.
- View and modify the data tables of other users and perform operations such as SELECT, UPDATE, DELETE, and changing owners.
- View the connection information of other users, cancel their SQL statements, and kill their connections.
- Create and delete plug-ins.
- Create other users with RDS_SUPERUSER permissions. Example:

```
CRATE ROLE root_user2 RDS_SUPERUSER LOGIN PASSWORD 'xyz' ;
```

Manage permissions

You can manage permissions at the database, schema, and table levels. For example, if you want to grant read permissions on a table to a user and revoke write permissions, you can execute the following statements:

```
GRANT SELECT ON TABLE t1 TO normal_user1;
REVOKE UPDATE ON TABLE t1 FROM normal_user1;
REVOKE DELETE ON TABLE t1 FROM normal_user1;
```

4.8. Manage real-time materialized views

AnalyticDB for PostgreSQL provides real-time materialized views. Compared with non-real-time materialized views, real-time materialized views can implement automatic refresh in response to data changes without the need to execute REFRESH statements.

A real-time materialized view can be automatically updated with changes made to its base tables. Real-time materialized views of AnalyticDB for PostgreSQL support only statement-level automatic refresh. When an INSERT, COPY, UPDATE, or DELETE statement is executed on a base table, real-time materialized views created on this base table are updated in real time to ensure strong data consistency.

> **Note** In this mode, the write performance of base tables may be reduced. We recommend that you do not create more than five real-time materialized views on the same base table.

For more information about materialized views, see CREATE MATERIALIZED VIEW.

Statement-level refresh

When a statement is executed on a base table, real-time materialized views created on this base table are updated in real time to ensure data consistency. Such statements include INSERT, COPY, UPDATE, and DELETE. Real-time materialized views are updated based on the following logic:

- The database engine first updates base tables, and then updates their materialized views. If base tables fail to be updated, no data changes are made to their materialized views.
- If materialized views fail to be updated, no data changes are made to their base tables and an error is returned for the executed statement.

For an explicit transaction that starts with the BEGIN TRANSACTION statement and ends with the COMMIT statement, after base tables are updated, data changes are also made to their materialized views in this transaction.

- If AnalyticDB for PostgreSQL uses the default READ COMMITTED isolation level, updates of materialized views are invisible to other transactions before this transaction is committed.
- If this transaction is rolled back, base tables and their materialized views are also rolled back.

Limits

AnalyticDB for PostgreSQL imposes the following limits on the query statements that are used to create real-time materialized views:

- If a query statement contains JOIN operations, only INNER JOIN can be used to join tables. OUTER JOIN or SELF JOIN is not supported.
- Query statements can contain most filtering and projection operations.
- Only the following aggregate operations are supported in a query statement: COUNT, SUM, AVG, MAX, and MIN. The HAVING clause is not supported.
- Complex statements such as those that contain subqueries and common table expressions (CTEs) are not supported.

After you create a real-time materialized view on a base table, DDL statements that you execute on the base table are subject to the following limits:

- When you execute the TRUNCATE statement on the base table, the real-time materialized view is not synchronously updated. You must manually refresh the real-time materialized view or create another materialized view.
- You must specify the CASCADE option to execute the DROP TABLE statement on the base table.
- ALTER TABLE statements on the base table cannot be used to delete or modify fields referenced by the materialized view.

Real-time materialized views are subject to the following limits:

- Real-time materialized views are supported for standard and partitioned heap tables, but not for append-optimized (AO) tables.

Scenarios

We recommend that you use real-time materialized views in the following scenarios:

- The number of rows or columns in the query results is much smaller than that of the base tables. For example, the query statement may contain a WHERE clause that effectively narrows down the results or may contain an aggregate function that consolidates multiple values into a single value.
- Large amounts of computations are required to obtain the query results of the following operations:
  - Semi-structured data analysis
  - Aggregate operations that take a long time to complete
- Base tables are not frequently changed.

Real-time materialized views can be used in all scenarios where materialized views are suitable. Compared with non-real-time materialized views, real-time materialized views are highly consistent with their base tables. When a base table changes, its real-time materialized views synchronously change with minimal performance cost. However, when you use non-real-time materialized views, you must manually update them each time their base tables change. Therefore, when large amounts of data are changed on a base table or a streaming update is performed, real-time materialized views have great advantages over non-real-time materialized views.

Disadvantages of real-time materialized views

Real-time materialized views are similar to indexes maintained in real time. They can significantly optimize query performance but also reduce write performance.

When you create a real-time materialized view that contains only a single table, the write performance of the related database is reduced because the data in the materialized view must be synchronously updated. The write latency can be up to three times longer compared with writing data to the base table but not updating the materialized view. We recommend that you do not create more than five real-time materialized views on the same base table.

Batch data writes help reduce the maintenance overhead of real-time materialized views. When you execute COPY or INSERT statements, we recommend that you increase the number of batch processed rows within a single statement. This significantly reduces the maintenance overhead of real-time materialized views.

When the query statement used to create a real-time materialized view contains a JOIN clause to join two tables, you must optimize the write performance of the real-time materialized view. If you do not have relevant experience or encounter low performance when you test the execution, we recommend that you use a real-time materialized view that contains only a single table. The following suggestions are available in scenarios where two tables are joined:

- Use the join key of each base table as their respective distribution keys.
- Create an index for the join key of each base table.

Create or delete a real-time materialized view

- Execute the following `CREATE INCREMENTAL MATERIALIZED VIEW` statement to create a real-time materialized view named `mv`:

  ```
  CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM base WHERE id > 40;
  ```

- Execute the following `DROP MATERIALIZED VIEW` statement to delete the `mv` materialized view:

  ```
  DROP MATERIALIZED VIEW mv;
  ```

Examples

1. Execute the following statement to create a base table:

   ```
   CREATE TABLE test (a int, b int) DISTRIBUTED BY (a);
   ```

2. Execute the following statement to create a real-time materialized view:

   ```
   CREATE INCREMENTAL MATERIALIZED VIEW mv AS SELECT * FROM TEST WHERE b > 40 DISTRIBUTED
   BY (a);
   ```

3. Execute the following statement to insert data to the base table:

```
INSERT INTO test VALUES (1, 30), (2, 40), (3, 50), (4, 60);
```

4. Execute the following statement to view data in the base table:

```
SELECT * FROM test;
```

The following information is returned:

```
 a | b
---+----
 1 | 30
 2 | 40
 3 | 50
 4 | 60
(4 rows)
```

5. Execute the following statement to view data in the materialized view:

```
SELECT * FROM mv;
```

The following information is returned, which indicates that the materialized view is updated:

```
 a | b
---+----
 3 | 50
 4 | 60
(2 rows)
```

4.9. Manage JSON data

JSON has become a basic data type in the Internet and IoT fields. For more information about JSON, visit JSON official website. PostgreSQL provides full support for JSON. AnalyticDB for PostgreSQL is optimized by Alibaba Cloud to support the JSON type based on the PostgreSQL syntax.

Check whether the current version supports JSON

Execute the following statement to check whether the current version supports JSON:

```
SELECT '""'::json;
```

If the following information is returned, the JSON type is supported and the instance is ready for use. If the operation fails, restart the instance.

```
 json
------
 ""
(1 row)
```

If the following information is returned, the JSON type is not supported:

```
ERROR:  type "json" does not exist
LINE 1: SELECT '""'::json;
               ^
```

The preceding statement converts data from the string type to the JSON type. PostgreSQL supports operations on JSON data based on this conversion.

JSON conversion in the database

Database operations include reading and writing. The written data is typically converted from the string type to the JSON type. The contents of a string must meet the JSON standard, such as strings, digits, arrays, and objects. Example:

String

```
SELECT '"hijson"'::json;
```

The following information is returned:

```
 json
-------
 "hijson"
(1 row)
```

`::` is used for explicit type conversion in PostgreSQL, Greenplum, and AnalyticDB for PostgreSQL. The database calls the input function of the JSON type during the conversion. Therefore, JSON format check is performed. Example:

```
SELECT '{hijson:1024}'::json;
```

The following error information is returned:

```
ERROR:  invalid input syntax for type json
LINE 1: SELECT '{hijson:1024}'::json;
               ^
DETAIL:  Token "hijson" is invalid.
CONTEXT:  JSON data, line 1: {hijson...
```

In the preceding example, `hijson` must be enclosed in double quotation marks (`"`) because JSON requires the KEY value to be a string. A syntax error is returned when `{hijson:1024}` is entered.

Apart from explicit type conversion, database records can also be converted to JSON.

Typically, JSON is not used for a string or a digit, but an object that contains one or more key-value pairs. AnalyticDB for PostgreSQL can support most JSON scenarios after data is converted from the string type to the object type. Example:

```
SELECT row_to_json(row('{"a":"a"}', 'b'));
```

The following information is returned:

```
           row_to_json
---------------------------------
 {"f1":"{\"a\":\"a\"}","f2":"b"}
(1 row)
```

```
SELECT row_to_json(row('{"a":"a"}'::json, 'b'));
```

The following information is returned:

```
        row_to_json
---------------------------
 {"f1":{"a":"a"},"f2":"b"}
(1 row)
```

You can see the differences between the string and JSON. The entire record is converted to the JSON type.

JSON data types

- Object

  The object is the most frequently used data type in JSON. Example:

  ```
  SELECT '{"key":"value"}'::json;
  ```

  ```
        json
  -----------------
   {"key":"value"}
  (1 row)
  ```

- Integer and floating point number

  JSON supports only three data types for numeric values: integer, floating-point number, and constant expression. AnalyticDB for PostgreSQL supports the three types.

  ```
  SELECT '1024'::json;
  ```

  The following information is returned:

  ```
    json
  ------
   1024
  (1 row)
  ```

  ```
  SELECT '0.1'::json;
  ```

  The following information is returned:

  ```
    json
  ------
   0.1
  (1 row)
  ```

  The following information is required in special situations:

```
SELECT '1e100'::json;
```

The following information is returned:

```
 json
-------
 1e100
(1 row)
```

```
SELECT '{"f":1e100}'::json;
```

The following information is returned:

```
     json
-------------
 {"f":1e100}
(1 row)
```

Extra-long numbers are also supported. Example:

```
SELECT '9223372036854775808'::json;
```

The following information is returned:

```
         json
---------------------
 9223372036854775808
(1 row)
```

- Array

```
SELECT '[[1,2], [3,4,5]]'::json;
```

The following information is returned:

```
        json
-----------------
 [[1,2], [3,4,5]]
(1 row)
```

Operators

Operators supported by JSON

```
SELECT oprname,oprcode FROM pg_operator WHERE oprleft = 3114;
```

The following information is returned:

```
 oprname |          oprcode
---------+---------------------------
 ->      | json_object_field
 ->>     | json_object_field_text
 ->      | json_array_element
 ->>     | json_array_element_text
 #>      | json_extract_path_op
 #>>     | json_extract_path_text_op
(6 rows)
```

Basic usage

```
SELECT '{"f":"1e100"}'::json -> 'f';
```

The following information is returned:

```
 ?column?
----------
 "1e100"
(1 row)
```

```
SELECT '{"f":"1e100"}'::json ->> 'f';
```

The following information is returned:

```
 ?column?
----------
 1e100
(1 row)
```

```
SELECT '{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}'::json#>array['f4','f6'];
```

The following information is returned:

```
 ?column?
-----------
 "stringy"
(1 row)
```

```
SELECT '{"f2":{"f3":1},"f4":{"f5":99,"f6":"stringy"}}'::json#>'{f4,f6}';
```

The following information is returned:

```
 ?column?
-----------
 "stringy"
(1 row)
```

```
SELECT '{"f2":["f3",1],"f4":{"f5":99,"f6":"stringy"}}'::json#>>'{f2,0}';
```

The following information is returned:

```
 ?column?
----------
 f3
(1 row)
```

JSON functions

Supported JSON functions

```
\df *json*
```

The following information is returned:

```
                                                    List of functions
   Schema   |           Name            | Result data type |                     Argument da
ta types                    |  Type
------------+---------------------------+------------------+--------------------------------
----------------------------+--------
 pg_catalog | array_to_json             | json             | anyarray
| normal
 pg_catalog | array_to_json             | json             | anyarray, boolean
| normal
 pg_catalog | json_array_element        | json             | from_json json, element_index
integer                     | normal
 pg_catalog | json_array_element_text   | text             | from_json json, element_index
integer                     | normal
 pg_catalog | json_array_elements       | SETOF json       | from_json json, OUT value json
| normal
 pg_catalog | json_array_length         | integer          | json
| normal
 pg_catalog | json_each                 | SETOF record     | from_json json, OUT key text,
OUT value json              | normal
 pg_catalog | json_each_text            | SETOF record     | from_json json, OUT key text,
OUT value text              | normal
 pg_catalog | json_extract_path         | json             | from_json json, VARIADIC path_
elems text[]                | normal
 pg_catalog | json_extract_path_op      | json             | from_json json, path_elems tex
t[]                         | normal
 pg_catalog | json_extract_path_text    | text             | from_json json, VARIADIC path_
elems text[]                | normal
 pg_catalog | json_extract_path_text_op | text             | from_json json, path_elems tex
t[]                         | normal
 pg_catalog | json_in                   | json             | cstring
| normal
 pg_catalog | json_object_field         | json             | from_json json, field_name tex
t                           | normal
 pg_catalog | json_object_field_text    | text             | from_json json, field_name tex
t                           | normal
 pg_catalog | json_object_keys          | SETOF text       | json
| normal
 pg_catalog | json_out                  | cstring          | json
| normal
 pg_catalog | json_populate_record      | anyelement       | base anyelement, from_json jso
n, use_json_as_text boolean | normal
```

```
n, use_json_as_text boolean | normal
 pg_catalog | json_populate_recordset   | SETOF anyelement | base anyelement, from_json jso
n, use_json_as_text boolean | normal
 pg_catalog | json_recv                 | json             | internal
| normal
 pg_catalog | json_send                 | bytea            | json
| normal
 pg_catalog | row_to_json               | json             | record
| normal
 pg_catalog | row_to_json               | json             | record, boolean
| normal
 pg_catalog | to_json                   | json             | anyelement
| normal
(24 rows)
```

Basic usage

```
SELECT array_to_json('{{1,5},{99,100}}'::int[]);
```

The following information is returned:

```
  array_to_json
------------------
 [[1,5],[99,100]]
(1 row)
```

```
SELECT row_to_json(row(1,'foo'));
```

The following information is returned:

```
     row_to_json
---------------------
 {"f1":1,"f2":"foo"}
(1 row)
```

```
SELECT json_array_length('[1,2,3,{"f1":1,"f2":[5,6]},4]');
```

The following information is returned:

```
 json_array_length
-------------------
                 5
(1 row)
```

```
SELECT * FROM json_each('{"f1":[1,2,3],"f2":{"f3":1},"f4":null,"f5":99,"f6":"stringy"}')
q;
```

The following information is returned:

```
 key |   value
-----+-----------
 f1  | [1,2,3]
 f2  | {"f3":1}
 f4  | null
 f5  | 99
 f6  | "stringy"
(5 rows)
```

```
SELECT json_each_text('{"f1":[1,2,3],"f2":{"f3":1},"f4":null,"f5":"null"}');
```

The following information is returned:

```
  json_each_text
-------------------
 (f1,"[1,2,3]")
 (f2,"{""f3"":1}")
 (f4,)
 (f5,null)
(4 rows)
```

```
SELECT json_array_elements('[1,true,[1,[2,3]],null,{"f1":1,"f2":[7,8,9]},false]');
```

The following information is returned:

```
  json_array_elements
-----------------------
 1
 true
 [1,[2,3]]
 null
 {"f1":1,"f2":[7,8,9]}
 false
(6 rows)
```

```
CREATE TYPE jpop AS (a text, b int, c timestamp);
```

```
SELECT * FROM json_populate_record(null::jpop,'{"a":"blurfl","x":43.2}', false) q;
```

The following information is returned:

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

```
SELECT * FROM json_populate_recordset(null::jpop,'[{"a":"blurfl","x":43.2},{"b":3,"c":"20
12-01-20 10:42:53"}]',false) q;
```

The following information is returned:

```
   a    | b |            c
--------+---+--------------------------
 blurfl |   |
        | 3 | Fri Jan 20 10:42:53 2012
(2 rows)
```

Code examples

Create a table

Create a table and insert data into the table.

```
CREATE TABLE tj(id serial, ary int[], obj json, num integer);
INSERT INTO tj(ary, obj, num) VALUES('{1,5}'::int[], '{"obj":1}', 5);
```

Query information from the table.

```
SELECT row_to_json(q) FROM (select id, ary, obj, num from tj) AS q;
```

The following information is returned:

```
               row_to_json
-----------------------------------------
 {"f1":1,"f2":[1,5],"f3":{"obj":1},"f4":5}
(1 row)
```

Insert data into the table.

```
INSERT INTO tj(ary, obj, num) VALUES('{2,5}'::int[], '{"obj":2}', 5);
```

Query information from the table.

```
SELECT row_to_json(q) FROM (select id, ary, obj, num from tj) AS q;
```

The following information is returned:

```
               row_to_json
-----------------------------------------
 {"f1":1,"f2":[1,5],"f3":{"obj":1},"f4":5}
 {"f1":2,"f2":[2,5],"f3":{"obj":2},"f4":5}
(2 rows)
```

Join multiple tables

Create a table and insert data into the table.

```
CREATE TABLE tj2(id serial, ary int[], obj json, num integer);
INSERT INTO tj2(ary, obj, num) VALUES('{2,5}'::int[], '{"obj":2}', 5);
```

Query information from the table.

```
SELECT * FROM tj, tj2 where tj.obj->>'obj' = tj2.obj->>'obj';
```

The following information is returned:

```
 id |  ary  |    obj     | num | id |  ary  |    obj     | num
----+-------+------------+-----+----+-------+------------+-----
  2 | {2,5} | {"obj":2} |   5 |  1 | {2,5} | {"obj":2} |   5
(1 row)
```

Query information from the table.

```
SELECT * FROM tj, tj2 WHERE json_object_field_text(tj.obj, 'obj') = json_object_field_text(
tj2.obj, 'obj');
```

The following information is returned:

```
 id |  ary  |    obj     | num | id |  ary  |    obj     | num
----+-------+------------+-----+----+-------+------------+-----
  2 | {2,5} | {"obj":2} |   5 |  1 | {2,5} | {"obj":2} |   5
(1 row)
```

Use the JSON function index

Create a table and insert data into the table.

```
CREATE TEMP TABLE test_json (
       json_type text,
       obj json
);
INSERT INTO test_json VALUES('aa', '{"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}');
INSERT INTO test_json VALUES('cc', '{"f7":{"f3":1},"f8":{"f5":99,"f6":"foo"}}');
```

Query information from the table.

```
SELECT obj->'f2' FROM test_json WHERE json_type = 'aa';
```

The following information is returned:

```
 ?column?
----------
 {"f3":1}
(1 row)
```

Create an index in the table.

```
CREATE INDEX i ON test_json (json_extract_path_text(obj, '{f4}'));
```

Query information from the table.

```
SELECT * FROM test_json where json_extract_path_text(obj, '{f4}') = '{"f5":99,"f6":"foo"}';
```

The following information is returned:

```
 json_type |                 obj
-----------+-----------------------------------------
 aa        | {"f2":{"f3":1},"f4":{"f5":99,"f6":"foo"}}
(1 row)
```

Note JSON data cannot be used as the distribution key and does not support JSON aggregate functions.

The following example demonstrates how to use Python to access the database:

```
#! /bin/env python
import time
import json
import psycopg2
def gpquery(sql):
    conn = None
    try:
        conn = psycopg2.connect("dbname=sanity1x2")
        conn.autocommit = True
        cur = conn.cursor()
        cur.execute(sql)
        return cur.fetchall()
    except Exception as e:
        if conn:
            try:
                conn.close()
            except:
                pass
            time.sleep(10)
        print e
    return None
def main():
    sql = "select obj from tj;"
    #rows = Connection(host, port, user, pwd, dbname).query(sql)
    rows = gpquery(sql)
    for row in rows:
        print json.loads(row[0])
if __name__ == "__main__":
    main()
```

4.10. Use HyperLogLog

AnalyticDB for PostgreSQL is highly optimized by Alibaba Cloud. It has the features of Greenplum Database and supports HyperLogLog. AnalyticDB for PostgreSQL is suited for industries such as Internet advertising and estimation analysis that require quick estimation of business metrics such as page views (PVs) and unique visitors (UVs).

Create a HyperLogLog plug-in

Execute the following statement to create a HyperLogLog plug-in:

```
CREATE EXTENSION hll;
```

Basic types

- Execute the following statement to create a table that contains the hll field:

```
CREATE TABLE agg (id int primary key,userids hll);
```

- Execute the following statement to convert int to hll_hashval:

```
SELET 1::hll_hashval;
```

Basic operators

- The hll type supports =, !=, <>, ||, and #.

```
SELECT hll_add_agg(1::hll_hashval) = hll_add_agg(2::hll_hashval);
SELECT hll_add_agg(1::hll_hashval) || hll_add_agg(2::hll_hashval);
SELECT #hll_add_agg(1::hll_hashval);
```

- The hll_hashval type supports =, !=, and <>.

```
SELECT 1::hll_hashval = 2::hll_hashval;
SELECT 1::hll_hashval <> 2::hll_hashval;
```

Basic functions

- Hash functions such as hll_hash_boolean, hll_hash_smallint, and hll_hash_bigint.

```
SELECT hll_hash_boolean(true);
SELECT hll_hash_integer(1);
```

- hll_add_agg: converts the int format to the hll format.

```
SELECT hll_add_agg(1::hll_hashval);
```

- hll_union: aggregates the hll fields.

```
SELECT hll_union(hll_add_agg(1::hll_hashval),hll_add_agg(2::hll_hashval));
```

- hll_set_defaults: sets the precision.

```
SELECT hll_set_defaults(15,5,-1,1);
```

- hll_print: displays debugging information.

```
SELECT hll_print(hll_add_agg(1::hll_hashval));
```

Examples

1. Create a test table.

```
CREATE TABLE access_date (acc_date date unique, userids hll);
```

2. Insert test data into the table.

```
INSERT INTO access_date SELECT current_date, hll_add_agg(hll_hash_integer(user_id)) FRO
M generate_series(1,10000) t(user_id);
INSERT INTO access_date SELECT current_date-1, hll_add_agg(hll_hash_integer(user_id)) F
ROM generate_series(5000,20000) t(user_id);
INSERT INTO access_date SELECT current_date-2, hll_add_agg(hll_hash_integer(user_id)) F
ROM generate_series(9000,40000) t(user_id);
```

3. View the test results of the HyperLogLog plug-in.

```
select #userids from access_date where acc_date=current_date;
```

The following information is returned:

```
     ?column?
------------------
 9725.85273370708
(1 row)
```

```
select #userids from access_date where acc_date=current_date-1;
```

The following information is returned:

```
     ?column?
------------------
 14968.6596883279
(1 row)
```

```
select #userids from access_date where acc_date=current_date-2;
```

The following information is returned:

```
     ?column?
------------------
 29361.5209149911
(1 row)
```

4.11. Use the CREATE LIBRARY statement

AnalyticDB for PostgreSQL introduces the CREATE LIBRARY and DROP LIBRARY statements to help you import custom software packages.

Syntax

```
CREATE LIBRARY library_name LANGUAGE [JAVA] FROM oss_location OWNER ownername
CREATE LIBRARY library_name LANGUAGE [JAVA] VALUES file_content_hex OWNER ownername
DROP LIBRARY library_name
```

Parameters

Parameter	Description
library_name	The name of the library that you want to install. If the library that you want to install has the same name as an existing library, you must delete the existing library before you install the new one.
LANGUAGE [JAVA]	The programming language that you want to use. Only PL/Java is supported.
oss_location	The location of the package. You can specify the Object Storage Service (OSS) bucket and object names. Only one object can be specified and the specified object cannot be a compressed file. Sample format: `oss://oss_endpoint filepath= [folder/[folder/]...]/file_name id=userossid key=userosskey bucket=ossbucket`
file_content_hex	The content of the file. The byte stream is in hexadecimal notation. For example, 73656c6563742031 indicates the hexadecimal byte stream of "select 1". You can use this syntax to import packages without the need to use OSS.
ownername	The user.
DROP LIBRARY	Deletes a library.

Examples

- Example 1: Install a JAR package named analytics.jar.

```
create library example language java from 'oss://oss-cn-hangzhou.aliyuncs.com filepath=an
alytics.jar id=xxx key=yyy bucket=zzz';
```

- Example 2: Import the file content with the byte stream in hexadecimal notation.

```
create library pglib LANGUAGE java VALUES '73656c6563742031' OWNER "myuser";
```

- Example 3: Delete a library.

```
drop library example;
```

- Example 4: View installed libraries.

```
select name, lanname from pg_library;
```

4.12. Create and use a PL/Java UDF

AnalyticDB for PostgreSQL allows you to compile and upload JAR software packages that are written in PL/Java language, and use these JAR packages to create user-defined functions (UDFs). AnalyticDB for PostgreSQL supports PL/Java 1.5.0 and JVM 1.8. This topic describes how to create a PL/Java UDF. For more information about PL/Java examples, see PL/Java code. For more information about the compiling method, see PL/Java documentation.

Procedure

1. In AnalyticDB for PostgreSQL, execute the following statement to create a PL/Java extension. You need only to execute the statement once for each database.

```
create extension pljava;
```

2. Compile the UDF based on your business requirements. For example, you can use the following code to compile the Test.java file:

```
public class Test
 {
     public static String substring(String text, int beginIndex,
                int endIndex)
             {
                     try {
                         Process process = null;
                         process = Runtime.getRuntime().exec("echo Test running");
                     } catch (Exception e) {
                        return "" + e;
                     }
                     return text.substring(beginIndex, endIndex);
             }
 }
```

3. Compile the manifest.txt file:

```
Manifest-Version: 1.0
Main-Class: Test
Specification-Title: "Test"
Specification-Version: "1.0"
Created-By: 1.7.0_99
Build-Date: 01/20/2016 21:00 AM
```

4. Run the following commands to compile and package the program:

```
javac Test.java
 jar cfm analytics.jar manifest.txt Test.class
```

5. Upload the analytics.jar file generated in Step 4 to Object Storage Service (OSS) by using the following OSS console command:

```
osscmd put analytics.jar oss://zzz
```

6. In AnalyticDB for PostgreSQL, execute the CREATE LIBRARY statement to import the file to AnalyticDB for PostgreSQL.

```
create library example language java from 'oss://oss-cn-hangzhou.aliyuncs.com filepath=
analytics.jar id=xxx key=yyy bucket=zzz';
```

> **Note** You can only use the filepath variable in the CREATE LIBRARY statement to import files one at a time. The CREATE LIBRARY statement also supports byte streams to import files without the need to use OSS. For more information, see Use the CREATE LIBRARY statement.

7. In AnalyticDB for PostgreSQL, execute the following statements to create and use the UDF:

```
create table temp (a varchar) distributed randomly;
 insert into temp values ('my string');
 create or replace function java_substring(varchar, int, int) returns varchar as 'Test.
substring' language java;
 select java_substring(a, 1, 5) from temp;
```

5.Table

5.1. Create a table

You can create tables within your databases.

Syntax

The following statement shows how to create a table. Not all clauses are required. Use the clauses that can fulfill your business needs.

```
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (
[ { column_namedata_type [ DEFAULT default_expr ]
   [column_constraint [ ... ]
[ ENCODING ( storage_directive [,...] ) ]
]
   | table_constraint
   | LIKE other_table [{INCLUDING | EXCLUDING}
                      {DEFAULTS | CONSTRAINTS}] ...}
   [, ... ] ]
   )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH ( storage_parameter=value [, ... ] )
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
   [ PARTITION BY partition_type (column)
       [ SUBPARTITION BY partition_type (column) ]
          [ SUBPARTITION TEMPLATE ( template_spec ) ]
       [...]
    ( partition_spec )
        | [ SUBPARTITION BY partition_type (column) ]
          [...]
    ( partition_spec
      [ ( subpartition_spec
           [(...)]
         ) ]
    )
```

Definition of the column_constraint clause:

```
[CONSTRAINT constraint_name]
   NOT NULL | NULL
   | UNIQUE [USING INDEX TABLESPACE tablespace]
            [WITH ( FILLFACTOR = value )]
   | PRIMARY KEY [USING INDEX TABLESPACE tablespace]
                 [WITH ( FILLFACTOR = value )]
   | CHECK ( expression )
   | REFERENCES table_name [ ( column_name [, ... ] ) ]
            [ key_match_type ]
            [ key_action ]
```

Definition of the storage_directive clause of columns:

```
COMPRESSTYPE={ZLIB | QUICKLZ | RLE_TYPE | NONE}
 [COMPRESSLEVEL={0-9} ]
 [BLOCKSIZE={8192-2097152} ]
```

Definition of the storage_parameter clause of tables:

```
APPENDONLY={TRUE|FALSE}
BLOCKSIZE={8192-2097152}
ORIENTATION={COLUMN|ROW}
CHECKSUM={TRUE|FALSE}
COMPRESSTYPE={ZLIB|QUICKLZ|RLE_TYPE|NONE}
COMPRESSLEVEL={0-9}
FILLFACTOR={10-100}
OIDS[=TRUE|FALSE]
```

Definition of the table_constraint clause:

```
[CONSTRAINT constraint_name]
UNIQUE ( column_name [, ... ] )
       [USING INDEX TABLESPACE tablespace]
       [WITH ( FILLFACTOR=value )]
| PRIMARY KEY ( column_name [, ... ] )
              [USING INDEX TABLESPACE tablespace]
              [WITH ( FILLFACTOR=value )]
| CHECK ( expression )
| FOREIGN KEY ( column_name [, ... ] )
         REFERENCES table_name [ ( column_name [, ... ] ) ]
         [ key_match_type ]
         [ key_action ]
         [ key_checking_mode ]
```

Valid values of key_match_type:

```
  MATCH FULL
| SIMPLE
```

Valid values of key_action:

```
  ON DELETE
| ON UPDATE
| NO ACTION
| RESTRICT
| CASCADE
| SET NULL
| SET DEFAULT
```

Valid values of key_checking_mode:

```
    DEFERRABLE
  | NOT DEFERRABLE
  | INITIALLY DEFERRED
  | INITIALLY IMMEDIATE
```

Valid values of partition_type:

```
    LIST
  | RANGE
```

Definition of the partition_specification clause:

```
partition_element [, ...]
```

Definition of the partition_element clause:

```
   DEFAULT PARTITION name
  | [PARTITION name] VALUES (list_value [,...] )
  | [PARTITION name]
     START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]
     [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]
     [ EVERY ([datatype] [number | INTERVAL] 'interval_value') ]
  | [PARTITION name]
     END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]
     [ EVERY ([datatype] [number | INTERVAL] 'interval_value') ]
[ WITH ( partition_storage_parameter=value [, ... ] ) ]
[ TABLESPACE tablespace ]
```

Definition of the subpartition_spec or template_spec clause:

```
subpartition_element [, ...]
```

Definition of the subpartition_element clause:

```
   DEFAULT SUBPARTITION name
  | [SUBPARTITION name] VALUES (list_value [,...] )
  | [SUBPARTITION name]
     START ([datatype] 'start_value') [INCLUSIVE | EXCLUSIVE]
     [ END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE] ]
     [ EVERY ([datatype] [number | INTERVAL] 'interval_value') ]
  | [SUBPARTITION name]
     END ([datatype] 'end_value') [INCLUSIVE | EXCLUSIVE]
     [ EVERY ([datatype] [number | INTERVAL] 'interval_value') ]
[ WITH ( partition_storage_parameter=value [, ... ] ) ]
[ TABLESPACE tablespace ]
```

Definition of the storage_parameter clause:

```
APPENDONLY={TRUE|FALSE}
BLOCKSIZE={8192-2097152}
ORIENTATION={COLUMN|ROW}
CHECKSUM={TRUE|FALSE}
COMPRESSTYPE={ZLIB|QUICKLZ|RLE_TYPE|NONE}
COMPRESSLEVEL={1-9}
FILLFACTOR={10-100}
OIDS[=TRUE|FALSE]
```

Parameters

The table describes the key parameters for creating a table.

Table creation parameters

Parameter	Description
TABLE_NAME	The name of the table that you want to create.
column_name	The name of a column that you want to create in the table.
data_type	The data type of the column. For columns that contain textual data, set the data type to VARCHAR or TEXT. We recommend that you do not use the CHAR type.
DEFAULT default_expr	Specifies a default value for the column. The system assigns this default value to all columns that do not have a value. The default value can be a variable-free expression. Subqueries or cross-references to other columns in the table are not allowed. The data type of the default expression must match the data type of the column. If a column does not have a default value, this parameter is left empty.
ENCODING storage_directive	Specifies the compression type and block size for the column data. This clause is valid only for append-optimized, column-oriented tables. Column compression settings are inherited from the table level to the partition level and then to the sub-partition level. The lowest-level settings have the highest priority.
INHERITS	Configures all columns in the new table to automatically inherit a parent table. You can use INHERITS to create a persistent relationship between the new child table and its parent table. Schema modifications to the parent table are applied to the child table as well. When the parent table is also scanned, the data of the child table is scanned as well.
LIKE other_table	Specifies a table from which the new table automatically copies all column names, data types, NOT NULL constraints, and distribution policies. Storage properties such as append-optimized or partition structure are not copied. Unlike INHERITS, the new table is completely decoupled from the original table after the new table is created.

Parameter	Description
CONSTRAINT constraint_name	Configures a column or table constraint. If a constraint is violated, the constraint name is displayed in the error message. Constraint names can be used to communicate helpful information to client applications.Constraint names that contain spaces must be enclosed by double quotation marks ('').
WITH (storage_option=value)	Configures storage options for the table or its indexes.
ON COMMIT	The operation that the system performs on the temporary tables at the end of a transaction. Valid values: • **PRESERVE ROWS**: No special action is taken. The data is retained after the transaction is complete. The data is released only when the session is disconnected. • **DELETE ROWS**: All rows in the temporary table are deleted. • **DROP**: The temporary table is deleted.
TABLESPACE tablespace	Specifies the name of the tablespace in which you want to create the new table. If this parameter is not specified, the default tablespace of the database is used.
DISTRIBUTED BY	Configures the distribution policy for the database. • DISTRIBUTED BY (column, [...]): specifies the distribution key. The system distributes data based on the distribution key. To evenly distribute data, you must set the distribution key to the primary key of the table or a unique column or a set of columns. • DISTRIBUTED RANDOMLY: randomly distributes data. **Note** We recommend that you do not use random distribution.
PARTITION BY	Configures a partition key to partition the table. Partitioning large tables improves data access efficiency. To partition a table is to create a top-level (parent) table and multiple lower-level (child) tables. After a partition table is created, its parent table is always empty. Data is stored in the lowest-level child tables. In a multi-level partition table, data is stored only in the lowest-level sub-partitions. Valid values: RANGE, LIST, and a combination of the two.
SUBPARTITION BY	Configures a multi-level partition table.
SUBPARTITION TEMPLATE	Specifies a sub-partition template to create sub-partitions (lower-level child tables). This ensures that all parent partitions have the same sub-partition structure.

Examples

Create a table and configure a distribution key. By default, a primary key is used as a distribution key in AnalyticDB for PostgreSQL.

```
CREATE TABLE films (
code        char(5) CONSTRAINT firstkey PRIMARY KEY,
title       varchar(40) NOT NULL,
did         integer NOT NULL,
date_prod   date,
kind        varchar(10),
len         interval hour to minute
);
CREATE TABLE distributors (
did    integer PRIMARY KEY DEFAULT nextval('serial'),
name   varchar(40) NOT NULL CHECK (name <> '')
);
```

Create a compressed table and configure a distribution key.

```
CREATE TABLE sales (txn_id int, qty int, date date)
WITH (appendonly=true, compresslevel=5)
DISTRIBUTED BY (txn_id);
```

Use sub-partition templates of each level and the default partition to create a three-level partition table.

```
CREATE TABLE sales (id int, year int, month int, day int,
region text)
DISTRIBUTED BY (id)
PARTITION BY RANGE (year)
  SUBPARTITION BY RANGE (month)
    SUBPARTITION TEMPLATE (
       START (1) END (13) EVERY (1),
       DEFAULT SUBPARTITION other_months )
  SUBPARTITION BY LIST (region)
    SUBPARTITION TEMPLATE (
       SUBPARTITION usa VALUES ('usa'),
       SUBPARTITION europe VALUES ('europe'),
       SUBPARTITION asia VALUES ('asia'),
       DEFAULT SUBPARTITION other_regions)
( START (2008) END (2016) EVERY (1),
  DEFAULT PARTITION outlying_years);
```

5.2. Principles and scenarios of row store, column store, heap tables, and AO tables

AnalyticDB for PostgreSQL supports row store, column store, heap tables, and append-optimized (AO) tables. This topic describes their principles and scenarios.

Row store and column store

Comparison

Dimension	Row store	Column store
Definition	Row store stores data in the form of rows. Each row is a tuple. To read a column, you must deform all of the columns that precede it. Therefore, the costs for accessing the first and the last columns are different.	Column store stores data as columns. Each column corresponds to a file or a batch of files. The cost of reading each column is the same. However, if you want to read multiple columns, you must access multiple files. The more columns you access, the higher the overheads are.
Compression ratio	Low.	High.
Cost of reading a column	The higher the column number, the higher the cost of reading the column.	Same.
Vector computing and JIT architecture	Not suitable. Not suitable for batch computing.	Suitable. More efficient when you access and collect statistics of a batch of data.
Scenarios	If you need to perform a large number of update and delete operations due to online transaction processing (OLTP) requirements such as when you query table details where multiple columns are returned, you can use row store. You can use partition tables if you have complex requirements. For example, if you want to partition data based on time, you can use row store to query the details of recent data and use column store to obtain more statistics from historical data.	You can use column store if you need data statistics because of online analytical processing (OLAP) requirements. If you want a higher compression ratio, you can use column store.

Heap tables

A heap table is heap storage. All changes to the heap table generate redo logs that can be used to restore data by point in time. However, heap tables cannot implement logical incremental backup because all data blocks in the tables can be changed and it is inconvenient to record the position by using heap storage.

Commit and redo logs are used to ensure reliability when transactions are finished. You can also implement redundancy by using secondary nodes through redo logs.

AO tables

AO tables are used to append data for storage. When you delete the updated data, you can use another bitmap file to mark the row that you want to delete and then use the offset of the bit to determine whether the row was deleted.

When the transaction is finished, you must call the fsync() function to record the offset of the data block that performs the last write operation. Even if the data block contains only a single record, a new data block is appended for the next transaction. The data block is synchronized to the secondary node for data redundancy.

AO tables are not suitable for small transactions because the fsync() function is called at the end of each transaction. This data block is not reused even if space is left.

AO tables are suitable for OLAP scenarios, batch data writing, high compression ratio, and logical backup that supports incremental backup. During backup, you need only to record the offset from the backup and the bitmap deletion mark for each full backup.

Use scenarios of heap tables

- When multiple small transactions are handled, use a heap table.
- When you need to restore data by point in time, use a heap table.

Use scenarios of AO tables

- When you want to use column store, use an AO table.
- When data is written in batches, use an AO table.

5.3. Enable the column store and compression features

If you want to improve performance, speed up data import, or reduce costs for tables that have infrequent updates and multiple fields, we recommend that you use column store and compression. These features increase the compression ratio by up to threefold to ensure high performance and speed up data import.

To enable the column store and compression features, you must specify the column store and compression options when you create a table. For example, you can add the following clause to the CREATE statement to enable these features. For more information about the table creation syntax, see Create a table.

```
with (APPENDONLY=true, ORIENTATION=column, COMPRESSTYPE=zlib, COMPRESSLEVEL=5, BLOCKSIZE=10
48576, OIDS=false)
```

> **Note** AnalyticDB for PostgreSQL supports only zlib and RLE_TYPE compression algorithms. If you specify the quicklz algorithm, it is automatically converted to zlib.

5.4. Add a field to a column-oriented table and set the default value

This topic describes how to add a field to a column-oriented table and set the default value for the field, and how to use the ANALYZE statement to view the impact of updated data on the size of the column-oriented table.

Context

In a column-oriented table, each column is stored as a file, and two columns in the same row correspond to each other by using an offset. For example, if you add two fields of the INT8 type, you can quickly locate column B from column A by using the offset.

When you add a field, append-optimized (AO) tables are not rewritten. If an AO table contains the records of deleted data, the added field must be filled with the deleted records before the offset can be used.

Procedure

1. Create three append-optimized column-oriented (AOCO) tables.

```
CREATE TABLE tbl1 (id int, info text) WITH (appendonly=true, blocksize=8192, compressty
pe=none, orientation=column);
CREATE TABLE tbl2 (id int, info text) WITH (appendonly=true, blocksize=8192, compressty
pe=none, orientation=column);
CREATE TABLE tbl3 (id int, info text) WITH (appendonly=true, blocksize=8192, compressty
pe=none, orientation=column);
```

2. Insert 10 million entries to the first two tables and 20 million entries to the third one.

```
INSERT INTO tbl1 SELECT generate_series(1,10000000),'test';
INSERT INTO tbl2 SELECT generate_series(1,10000000),'test';
INSERT INTO tbl3 SELECT generate_series(1,20000000),'test';
```

3. Analyze the tables and query the table sizes.

   Execute the following statements to analyze each table:

```
ANALYZE tbl1;
ANALYZE tbl2;
ANALYZE tbl3;
```

   Execute the following statements to query the size of each table:

```
SELECT pg_size_pretty(pg_relation_size('tbl1'));
```

   The following information is returned:

```
 pg_size_pretty
----------------
 88 MB
(1 row)
```

```
SELECT pg_size_pretty(pg_relation_size('tbl2'));
```

The following information is returned:

```
 pg_size_pretty
----------------
 88 MB
(1 row)
```

```
SELECT pg_size_pretty(pg_relation_size('tbl3'));
```

The following information is returned:

```
 pg_size_pretty
----------------
 173 MB
(1 row)
```

4. Update all the data in the first table. Query the table size after the update. The size is twice as large as the size before the update.

   Execute the following statement to update all the data in the first table:

   ```
   UPDATE tbl1 SET INFO='test';
   ```

   Execute the following statement to analyze the first table again:

   ```
   ANALYZE tbl1;
   ```

   Execute the following statement to query the size of the first table:

   ```
   SELECT pg_size_pretty(pg_relation_size('tbl1'));
   ```

   The following information is returned:

   ```
    pg_size_pretty
   ----------------
    173 MB
   (1 row)
   ```

5. Add fields to the three tables and set the default values.

   ```
   ALTER TABLE tbl1 ADD column c1 int8 default 1;
   ALTER TABLE tbl2 ADD column c1 int8 default 1;
   ALTER TABLE tbl3 add column c1 int8 default 1;
   ```

6. Analyze the tables and query the table sizes.

   Execute the following statements to analyze each table:

```
ANALYZE tbl1;
ANALYZE tbl2;
ANALYZE tbl3;
```

Execute the following statements to query the size of each table:

```
SELECT pg_size_pretty(pg_relation_size('tbl1'));
```

The following information is returned:

```
 pg_size_pretty
----------------
 325 MB
(1 row)
```

```
SELECT pg_size_pretty(pg_relation_size('tbl2'));
```

The following information is returned:

```
 pg_size_pretty
----------------
 163 MB
(1 row)
```

```
SELECT pg_size_pretty(pg_relation_size('tbl3'));
```

The following information is returned:

```
 pg_size_pretty
----------------
 325 MB
(1 row)
```

When you add fields to the AO tables, the number of entries in the existing tables prevails. Even if all the entries are deleted, new fields are added to the original data.

5.5. Configure table partitions

For fact tables and large-sized tables in a database, we recommend that you configure table partitions.

Configure table partitions

You can use the table partitioning feature to add and remove data based on table partitions on a regular basis. You can use the `ALTER TABLE DROP PARTITION` statement to remove all the data in a partition, and use the `ALTER TABLE EXCHANGE PARTITION` statement to import data to a new data partition.

AnalyticDB for PostgreSQL supports range partitioning, list partitioning, and composite partitioning. Range partitioning supports only the numeric or datetime data types of fields.

The following example shows how to use range partitioning in a table:

```
CREATE TABLE LINEITEM (
L_ORDERKEY              BIGINT NOT NULL,
L_PARTKEY               BIGINT NOT NULL,
L_SUPPKEY               BIGINT NOT NULL,
L_LINENUMBER    INTEGER,
L_QUANTITY              FLOAT8,
L_EXTENDEDPRICE FLOAT8,
L_DISCOUNT              FLOAT8,
L_TAX                   FLOAT8,
L_RETURNFLAG    CHAR(1),
L_LINESTATUS    CHAR(1),
L_SHIPDATE              DATE,
L_COMMITDATE    DATE,
L_RECEIPTDATE   DATE,
L_SHIPINSTRUCT  CHAR(25),
L_SHIPMODE              CHAR(10),
L_COMMENT               VARCHAR(44)
)  WITH  (APPENDONLY=true, ORIENTATION=column, COMPRESSTYPE=zlib, COMPRESSLEVEL=5, BLOCKSIZ
E=1048576, OIDS=false) DISTRIBUTED BY (l_orderkey)
PARTITION BY RANGE (L_SHIPDATE) (START (date '1992-01-01') INCLUSIVE END (date '2000-01-01'
) EXCLUSIVE EVERY (INTERVAL '1 month' ));
```

Principles of table partitioning

The purpose of partitioning is to minimize the amount of data to be scanned during a query. Therefore, partitions must be associated with the query conditions.

- Principle 1: Select the fields that are related to the query conditions to configure partitions and reduce the amount of data to be scanned.
- Principle 2: If multiple query conditions exist, configure sub-partitions to further reduce the amount of data to be scanned.

5.6. Configure the sort key

A sort key is an attribute of a table. Data on disks is stored in the order of the sort key.

Context

Sort keys have two major advantages:

- Speed up and optimize column-store operations. The min and max meta information the system collects seldom overlaps with each other, which features good filterability.
- Eliminate the need to perform ORDER BY and GROUP BY operations. The data directly read from the disk is ordered as required by the sorting conditions.

Create a table

```
Command:     CREATE TABLE
Description: define a new table
Syntax:
CREATE [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name (
[ { column_name data_type [ DEFAULT default_expr ]     [column_constraint [ ... ]
[ ENCODING ( storage_directive [,...] ) ]
]
   | table_constraint
   | LIKE other_table [{INCLUDING | EXCLUDING}
                      {DEFAULTS | CONSTRAINTS}] ...}
   [, ... ] ]
   [column_reference_storage_directive [, ] ]
   )
   [ INHERITS ( parent_table [, ... ] ) ]
   [ WITH ( storage_parameter=value [, ... ] )
   [ ON COMMIT {PRESERVE ROWS | DELETE ROWS | DROP} ]
   [ TABLESPACE tablespace ]
   [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
   [ SORTKEY (column, [ ... ] )]
   [ PARTITION BY partition_type (column)
       [ SUBPARTITION BY partition_type (column) ]
          [ SUBPARTITION TEMPLATE ( template_spec ) ]
       [...]
    ( partition_spec )
        | [ SUBPARTITION BY partition_type (column) ]
          [...]
    ( partition_spec
      [ ( subpartition_spec
           [(...)]
         ) ]
)
```

Examples:

```
create table test(date text, time text, open float, high float, low float, volume int) with
(APPENDONLY=true,ORIENTATION=column) sortkey (volume);
```

Sort the table

```
VACUUM SORT ONLY [tablename]
```

Modify the sort key

This statement only modifies the catalog and does not sort data. You must execute the `VACUUM SORT ONLY` statement to sort the table.

```
ALTER [[GLOBAL | LOCAL] {TEMPORARY | TEMP}] TABLE table_name SET SORTKEY (column, [ ... ] )
```

Examples:

```
alter table test set sortkey (high,low);
```

6.Best practices

6.1. Configure memory and load parameters

To improve database stability, you must configure memory and load parameters.

Background information

AnalyticDB for PostgreSQL is a massively parallel processing (MPP) database service with high computational and resource requirements. AnalyticDB for PostgreSQL consumes all of its resources. This allows AnalyticDB for PostgreSQL to reach high processing speeds but makes it very easy to exceed its limits.

When CPU, network, or hard disk resources are exhausted, you may encounter hardware bottlenecks. However, databases may not respond if memory resources are overused.

How to avoid OOM errors

Out of memory (OOM) indicates that the system is unable to provide sufficient memory requested by a process. The following prompt appears when OOM errors occur:

```
Out of memory (seg27 host.example.com pid=47093) VM Protect failed to allocate 4096 bytes,
0 MB available
```

Causes

OOM errors may occur due to the following causes:

- Database nodes do not have sufficient memory.
- Kernel parameters related to the memory of the operating system are incorrectly configured.
- Data skew has occurred. This causes a compute node to request a large amount of memory.
- Query skew has occurred. For example, if the grouping fields of specific aggregate or window functions are not distribution keys, the data must be redistributed. After the data is redistributed, data is skewed on a specific computer node and the node may request a large amount of memory.

Solutions

1. Modify the queries to request less memory.
2. Use a resource queue of AnalyticDB for PostgreSQL to limit the number of concurrent queries. Reduce the number of concurrent queries that are executed on the instance to reduce the overall memory requested by the system.
3. Reduce the number of compute nodes deployed on a host. For example, you can deploy 8 compute nodes instead of 16 on a host that has 128 GB of memory to allow each compute node to utilize twice as much memory.
4. Increase the memory of a host.
5. Set the gp_vmem_protect_limit parameter to limit the maximum VMEM that can be used by a single compute node. The memory size of a single host and the number of compute nodes deployed on the host determine the maximum memory size that a single compute node can use on average.

6. For SQL statements that have unpredictable memory usage, you can set the statement_mem parameter in the session to limit the memory usage of a single SQL statement. This prevents a single SQL statement from consuming all available memory.
7. Set the statement_mem parameter at the database level to apply the parameter to all the sessions in the database.
8. Use the resource queue of AnalyticDB for PostgreSQL to limit the maximum memory usage of the resource group. Add database users to the resource group to limit the total memory used by these users.

Configure memory-related parameters

You can properly configure the operating system, database parameters, and resource queue to reduce the probability of OOM.

When you calculate the average memory usage of a single compute node on a single host, you must consider both the primary and secondary compute nodes. If the cluster encounters a host failure, the system switches the service from primary compute nodes to the corresponding secondary compute nodes. At this time, the number of compute nodes on the host is greater than usual. Therefore, you must consider the amount of resources occupied by the secondary compute nodes during failover.

The following tables describe how to configure parameters for the operating system kernel and database to avoid OOM.

The following table describes the parameter configuration of the operating system kernel.

Operating system kernel parameters

Parameter	Description
huge page	Do not configure the huge page parameter of the system. AnalyticDB for PostgreSQL does not support the latest version of PostgreSQL and therefore does not support the huge page feature. The huge page parameter locks a part of the allocated memory. Database nodes are not able to use this part of the memory.
vm.overcommit_memory	If you use the swap space, set this parameter to 2. If you do not use the swap space, set this parameter to 0. Valid values: • 0: The requested memory space cannot exceed the difference between the total memory and the resident set size (RSS). An error is returned only when the memory space has been exceeded. • 1: Most processes use the malloc() function to request memory, but do not use all of the requested memory. When this parameter is set to 1, the memory requested by the malloc() function is allocated in all circumstances unless the memory is not sufficient. • 2: The swap space is also considered when the system calculates the memory space that can be requested. You can request a large amount of memory even if the swap space is triggered.

Parameter	Description
overcommit_ratio	The larger the value is, the more memory the process can request. However, less space is reserved for the operating system. For more information about the formula that is used to calculate the memory parameters, see the "Examples to calculate the memory parameters" section of this topic. When this parameter is set to 2, the memory that can be requested cannot exceed swap + memory × overcommit_ratio.

The following Database parameters table describes the parameter configuration of the database.

Database parameters

Parameter	Description
gp_vmem_protect_limit	The maximum amount of memory that all processes can request on each node. If the value is too large, it may result in a system OOM error or even more serious problems. If the value is too small, SQL statements may not be executed even when the system has enough memory.
runaway_detector_activation_percent	Default value: 90. Unit: %. When the amount of memory used by a compute node exceeds runaway_detector_activation_percent × gp_vmem_protect_limit/100, the query is terminated to prevent OOM. The query remains stopped until the memory usage reaches a value lower than runaway_detector_activation_percent × gp_vmem_protect_limit/100. You can use the gp_toolkit.session_level_memory_consumption view to observe the memory usage of each session and runaway information.

Parameter	Description
statement_mem	The maximum amount of memory that a single SQL statement can request. When the maximum memory is exceeded, spill files are created. Default value: 125. Unit: MB. We recommend that you set this parameter based on the following formula: `(gp_vmem_protect_limit * 0.9) / max_expected_concurrent_queries` **Note** • You can specify the statement_mem parameter in a session. If few concurrent queries exist and if the session needs to execute a query that requires a large amount of memory, you must specify this parameter in the session. • The statement_mem parameter is suitable for limiting memory usage in low concurrency scenarios. If you use statement_mem to limit the memory for high concurrency scenarios, each query is allocated with a very small amount of memory. As a result, the performance of a small number of queries with high memory requirements in high concurrency scenarios is affected. We recommend that you use the resource queue to limit the maximum memory usage in high concurrency scenarios.
gp_workfile_limit_files_per_query	The maximum number of spill files that can be created by each query. When the amount of memory requested by the query exceeds the statement_mem limit, spill files (also known as work files) are created. Spill files are similar to the swap space of the operating system. When the number of spill files used exceeds the limit, the query is terminated. The default value is 0, which indicates that an unlimited number of spill files can be created.

Parameter	Description
gp_workfile_compress_algorithm	The compression algorithm for spill files. Valid values: NONE and ZLIB. This parameter can optimize storage space or I/O by sacrificing CPU performance. You can set this parameter when the disk space is insufficient or when the spill files encounter a write bottleneck.

Examples to calculate the memory parameters

The following environment configurations are used in the examples:

- Host configuration:

```
Total RAM = 256GB
SWAP = 64GB
```

- Four hosts, each deployed with eight primary compute nodes and eight secondary compute nodes.

  If a host fails, the eight primary compute nodes are distributed to the remaining three hosts. A single host can be deployed with a maximum of three extra primary compute nodes from the failed host. A single host can be deployed with a maximum of 11 primary compute nodes.

1. Calculate the total memory allocated to AnalyticDB for PostgreSQL by the operating system.

   Reserve 7.5 GB and 5% of memory for the operating system and calculate the available memory for all applications. Then, divide the available memory by an empirical coefficient of 1.7.

```
gp_vmem = ((SWAP + RAM) - (7.5 GB + 0.05 × RAM))/1.7
        = ((64 + 256) - (7.5 + 0.05 * 256)) / 1.7
        = 176
```

2. Use an empirical coefficient of 0.026 to calculate an ideal value for the overcommit_ratio parameter.

```
vm.overcommit_ratio = (RAM - (0.026 * gp_vmem)) / RAM
                    = (256 - (0.026 * 176)) / 256
                    = .982
Set vm.overcommit_ratio to 98.
```

3. Calculate the gp_vmem_protect_limit parameter by dividing gp_vmem by maximum_acting_primary_segments. The maximum_acting_primary_segments parameter indicates the number of primary compute nodes to be run on each host after one host fails.

```
gp_vmem_protect_limit calculation
gp_vmem_protect_limit = gp_vmem / maximum_acting_primary_segments
                      = 176 / 11
                      = 16GB
                      = 16384MB
```

Configure resource queues

You can use resource queues of AnalyticDB for PostgreSQL to limit the number of concurrent queries and the total memory usage. While a query is being executed, it is added to the corresponding queue and the resources used are recorded in the queue. The resource limit of the queue applies to all sessions in the queue.

The resource queue in AnalyticDB for PostgreSQL is similar to cgroup in Linux.

The following syntax demonstrates how to create a resource queue:

```
Command:     CREATE RESOURCE QUEUE
Description: create a new resource queue for workload management
Syntax:
CREATE RESOURCE QUEUE name WITH (queue_attribute=value [, ... ])
where queue_attribute is:
   ACTIVE_STATEMENTS=integer
        [ MAX_COST=float [COST_OVERCOMMIT={TRUE|FALSE}] ]
        [ MIN_COST=float ]
        [ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
        [ MEMORY_LIMIT='memory_units' ]
|  MAX_COST=float [ COST_OVERCOMMIT={TRUE|FALSE} ]
        [ ACTIVE_STATEMENTS=integer ]
        [ MIN_COST=float ]
        [ PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX} ]
        [ MEMORY_LIMIT='memory_units' ]
```

The table describes the parameters for creating the resource queue.

Resource queue creation parameters

Parameter	Description
ACTIVE_STATEMENTS	The number of SQL statements that can be concurrently executed (in the active state). A value of -1 indicates that an unlimited number of SQL statements can be concurrently executed.

Parameter	Description

MEMORY_LIMIT 'memory_units kB, MB or GB'	The maximum memory usage allowed by all SQL statements in the resource queue. A value of -1 indicates unlimited memory usage. However, it is easy to trigger OOM errors because it is limited by the database or system parameters mentioned in the preceding sections.
	The memory usage of SQL statements is limited by resource queues and parameters.
	• When the gp_resqueue_memory_policy parameter is set to none, the limit is the same as that in versions of Greenplum Database earlier than 4.1.
	• When the gp_resqueue_memory_policy parameter is set to auto and you have specified the statement_mem parameter for a session or at the database level, the allowed memory of a single query exceeds the MEMORY_LIMIT value of the resource queue.
	Example:
	`SET statement_mem='2GB';`
	`SELECT * FROM my_big_table WHERE column='value' ORDER BY id;`
	`RESET statement_mem;`
	• The system parameter max_statement_mem can limit the maximum memory usage at the compute node level. The memory requested by a single query cannot exceed max_statement_mem.
	You can modify the statement_mem parameter at the session level, but do not modify the max_statement_mem parameter. We recommend that you specify max_statement_mem in the following formula:
	`(seghost_physical_memory) /`
	`(average_number_concurrent_queries)`
	• When the gp_resqueue_memory_policy parameter is set to eager_free, the database allocates the memory by stage. For example, if a query requests 1 GB of memory in total but needs only 100 MB during each stage, the database allocates 100 MB of memory to the query. You can use eager_free to reduce the possibility of insufficient memory for the query.

MAX_COST float	The maximum cost of the queries that can be concurrently executed by the resource group. The cost is the estimated total cost in the SQL execution plan. The value of the parameter can be specified as a floating-point number such as 100.0 or an exponent such as 1e+2. A value of -1 indicates that the cost is not limited.
COST_OVERCOMMIT boolean	Specifies whether the limit of max_cost can be exceeded while the system is idle. A value of TRUE indicates that the limit can be exceeded.
MIN_COST float	When the resources requested exceed the limit, the queries are queued. However, if the cost of a query is less than the min_cost value, the query can be executed without being queued.

Parameter	Description
PRIORITY={MIN|LOW|MEDIUM|HIGH|MAX}	The priority of the current resource queue. When resources are insufficient, CPU resources are allocated to the resource queue that has a higher priority. The SQL statements in the resource queue that has a higher priority can obtain CPU resources first. We recommend that you allocate users that initiate queries with high real-time requirements to resource queues that have higher priorities. This parameter is similar to the time slice policy that is used in CPU resource groups in a Linux cgroup to schedule real-time and common tasks.

Example of modifying resource queue limits:

```
ALTER RESOURCE QUEUE myqueue WITH (MAX_COST=-1.0, MIN_COST= -1.0);
```

Example of putting a user in a resource queue:

```
ALTER ROLE sammy RESOURCE QUEUE poweruser;
```

Resource queue parameters

Parameter	Description
gp_resqueue_memory_policy	The memory management policy of the resource queue.
gp_resqueue_priority	Specifies whether to enable query prioritization. Valid values: • ON: enables query prioritization. • OFF: disables query prioritization. If query prioritization is disabled, the existing priority settings are not evaluated.
gp_resqueue_priority_cpucores_per_segment	The number of CPU cores allocated to each compute node. For example, if an 8-core host is configured with two primary compute nodes, you can set the parameter to 4. If no other nodes exist on the primary node, set the parameter to 8. When the CPU is preempted, the SQL statements that are executed in a higher-priority resource group are allocated with CPU resources first.
gp_resqueue_priority_sweeper_interval	The interval at which CPU utilization is recalculated for all active statements. The share value is calculated when the SQL statement is executed. You can calculate the share value based on the priority and gp_resqueue_priority_cpucores_per_segment. The smaller the interval and the more frequent the calculation, the better the results achieved by the priority settings and the larger the overhead.

Tips for configuring resource queues:

- We recommend that you create a resource queue for each user.

The default resource queue of AnalyticDB for PostgreSQL is pg_default. If no queue is created, all users are assigned to pg_default. This operation is not recommended. We recommend that you create a resource queue for each user. Typically, a database user corresponds to a business operation. Different database users may correspond to different business operations or users, such as business users, analysts, developers, and DBAs.

- We recommend that you do not use superusers to execute queries.

  Queries initiated by superusers are limited only by the preceding parameters and not by the resource queue. If you want to use resource queues to limit the use of resources, we recommend that you do not use superusers to execute queries.

- ACTIVE_STATEMENTS indicates the SQL statements that can be concurrently executed within a resource queue. When the cost of a query is lower than the minimum cost specified by min_cost, the query can be executed without being queued.
- You can specify the MEMORY_LIMIT parameter to set the allowed maximum memory usage of all the SQL statements in a resource queue. The statement_mem parameter has a higher priority that can break through the limit of resource queues.

  > **Note** The memory of all resource queues cannot exceed gp_vmem_protect_limit.

- You can distinguish different business operations by configuring the priorities of resource queues.

  For example, you can give report-related business operations a top priority while giving common business operations and analysis lower priorities. In this case, you can create three resource queues that have the max, high, and medium priorities, respectively.

- If the number of resources requested at different periods of time varies, you can run the crontab command to periodically adjust the limits of resource queues based on usage patterns.

  For example, you can give the queue of analysis a higher priority in the day and the queue of report-related business operations a higher priority at night. AnalyticDB for PostgreSQL does not allow you to configure resource limits by period of time. Therefore, you can only externally deploy tasks by using the ALTER RESOURCE QUEUE statement.

- You can use the view provided by gp_toolkit to view the resource usage of the resource queues.

```
gp_toolkit.gp_resq_activity
gp_toolkit.gp_resq_activity_by_queue
gp_toolkit.gp_resq_priority_backend
gp_toolkit.gp_resq_priority_statement
gp_toolkit.gp_resq_role
gp_toolkit.gp_resqueue_status
```