Alibaba Cloud AnalyticDB for PostgreSQL

Data Access

Issue: 20200623

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 due to product version upgrades, adjustments, or other reasons. Alibaba Cloud reserves the right to modify the content of this document without notice and the updated versions of this document will be occasionally released through Alibaba Cloud-authorized channels. You shall 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 the document in the context that Alibaba Cloud products and services are provided on an "as is", "with all faults" and "as available" basis. 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 bear any liability for any errors or financial losses incurred by any organizations, companies, or individuals arising from their download, use, or trust in this document. Alibaba Cloud shall not, under any circumstances, bear responsibility for any indirect, consequential, exemplary, incidental, 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 contact Alibaba Cloud directly if you discover any errors in this document.

Document conventions

Style	Description	Example
	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.
	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.
	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.
	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]`

Style	Description	Example
{} or {a|b}	This format is used for a required value, where only one item can be selected.	switch {active|stand}

Contents

1 Introduction to data migration and synchronization solutions

This topic introduces the data migration and synchronization solutions supported by AnalyticDB for PostgreSQL. You can migrate or synchronize data smoothly between AnalyticDB for PostgreSQL databases and other types of data sources without interruptions to your business. These data sources include ApsaraDB RDS for MySQL, Apsara PolarDB for MySQL, ApsaraDB RDS for PostgreSQL, ApsaraDB RDS for PPAS, MaxCompute, Greenplum Database, user-created MySQL, user-created PostgreSQL, and Amazon Redshift.

The following table lists data migration and synchronization scenarios supported by AnalyticDB for PostgreSQL and related operations.

Operation	Type	Scenario
Import or export OSS data by using OSS external tables	Data migration	Use OSS external tables to migrate data between an AnalyticDB for PostgreSQL instance and OSS.
Use Data Integration to migrate and batch synchronize data	Data synchronization or migration	Use Data Integration to migrate or synchronize data between an AnalyticDB for PostgreSQL instance and a heterogeneous data source within a few minutes.
Use the \COPY command	Data migration	Use the `\COPY` command to migrate data in local text files to an AnalyticDB for PostgreSQL instance.
Use rds_dbsync to migrate or synchronize data from a MySQL database to an AnalyticDB for PostgreSQL database	Data synchronization or migration	Use the mysql2pgsql function of the rds_dbsync tool to synchronize the data of tables from a local MySQL database to an AnalyticDB for PostgreSQL database.
Use rds_dbsync to migrate or synchronize data from a PostgreSQL database to an AnalyticDB for PostgreSQL database	Data synchronization or migration	Use the pgsql2pgsql function of the rds_dbsync tool to synchronize data between AnalyticDB for PostgreSQL databases, Greenplum Databases, PostgreSQL databases, or PPAS databases.

1. Data migration: Data in a database instance or local data is migrated to an AnalyticDB for PostgreSQL instance.

2. Data synchronization: Data in a database is synchronized to an AnalyticDB for PostgreSQL instance in real time.

2 Use Data Integration to migrate and batch synchronize data

Data Integration is a reliable, secure, cost-effective, elastic, and scalable data synchronization platform provided by Alibaba Cloud. It supports data storage across heterogeneous systems and offers offline (both full and incremental) data access channels in diverse network environments for more than 20 types of data sources. For more information, visit #unique_6.

Scenarios

- Export data from AnalyticDB for PostgreSQL to other data sources and process the data as needed.
- Import processed data from other data sources to AnalyticDB for PostgreSQL.

In both the preceding scenarios, you can execute a synchronization task in Data Integration. For more information about how to create and configure a synchronization task including the data source and whitelist, visit the DataWorks documentation. The following sections detail how to import data to and export data from AnalyticDB for PostgreSQL.

Preparations

Complete the following preparations for Data Integration:

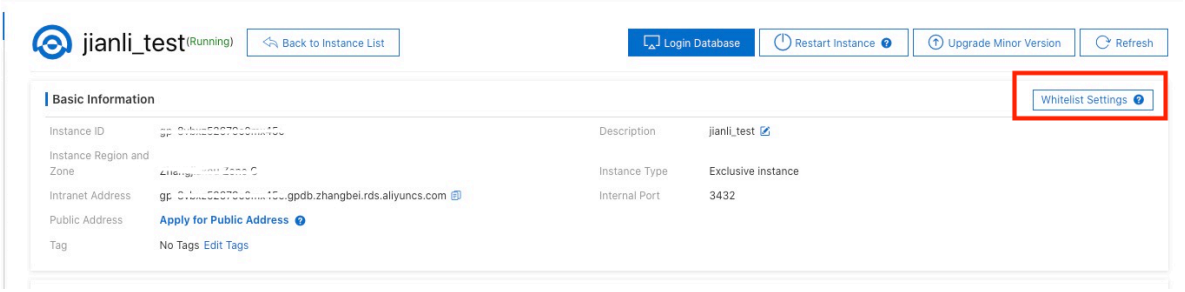
1. Register an Alibaba Cloud account.
2. Activate MaxCompute. After a default MaxCompute data source is generated, log on to the DataWorks console by using your Alibaba Cloud account.
3. Create a workspace. In the workspace, you can complete your workflow and maintain both your data and tasks.

> **Note:**
> If you want to create a synchronization task in Data Integration by using the credentials of your RAM user, you must grant the required permissions to your RAM user. For more information, visit Create a RAM user.

Complete the following preparations for AnalyticDB for PostgreSQL:

1. If you want to import data, use the psql CLI client to create both the destination database and table on your AnalyticDB for PostgreSQL instance.

2. If you want to export data, configure an **IP address whitelist** for your AnalyticDB for PostgreSQL instance. For more information, visit Add whitelist.



Import data

Add a data source in the DataWorks console. For more information, visit Configure data sources.

Configure a synchronization task.

The synchronization task is used to synchronize data from a data source to AnalyticDB for PostgreSQL. You can configure the synchronization task in **wizard** or **script** mode.

- To configure a synchronization task in wizard mode, follow these steps:

  1. Create a data synchronization node.
  2. Specify a data source.
  3. Specify a data destination. This must your AnalyticDB for PostgreSQL instance.
  4. Configure the mappings between fields from the source and destination tables.
  5. Specify a maximum transmission rate and dirty data check rules.
  6. Configure scheduling attributes.

Note:

For more information, visit Configure a synchronization task in wizard mode.

- To configure a synchronization task in script mode, follow these steps:

  1. Create a data synchronization node.
  2. Import a template.
  3. Configure a reader for the synchronization task.
  4. Configure a writer for the synchronization task. The writer must be your AnalyticDB for PostgreSQL instance.
  5. Configure the mappings between fields from the source and destination tables.
  6. Specify a maximum transmission rate and dirty data check rules.
  7. Configure scheduling attributes.

   **Note:**

  For more information, visit Configure a synchronization task in script mode.

Export data

The data export procedure is similar to the data import procedure, except that you must specify AnalyticDB for PostgreSQL as the data source. For more information, visit Add an AnalyticDB for PostgreSQL connection.

References

For more information, visit the DataWorks documentation.

3 Use the \COPY command

This topic describes how to run the \COPY command on the psql CLI client to import and export data between your computer and an AnalyticDB for PostgreSQL instance. Only .txt files can be imported from your computer to an AnalyticDB for PostgreSQL instance. In addition, they must be formatted. For example, they must use commas (,), semicolons (;), or other special characters as delimiters.

Notice:

- The \COPY command can only be executed by the coordinator node that is used to process serial data writes. It cannot be used to write a large volume of data in parallel. If you want to write a large volume of data in parallel, we recommend that you first import the data from your computer to Alibaba Cloud Object Storage Service (OSS) and then import the data from OSS to the AnalyticDB for PostgreSQL instance.
- The \COPY command is executed on the psql CLI client. If you execute the COPY statement but not the \COPY command, only standard input streams are supported and files are not supported because the root user does not have the superuser permissions to manage files.

The following example shows how to run the \COPY command on the psql CLI client to import data from your computer to an AnalyticDB for PostgreSQL instance:

```
\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] ]
```

The following example shows how to run the \COPY command to export data from an AnalyticDB for PostgreSQL instance to your computer:

```
\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 ]
```

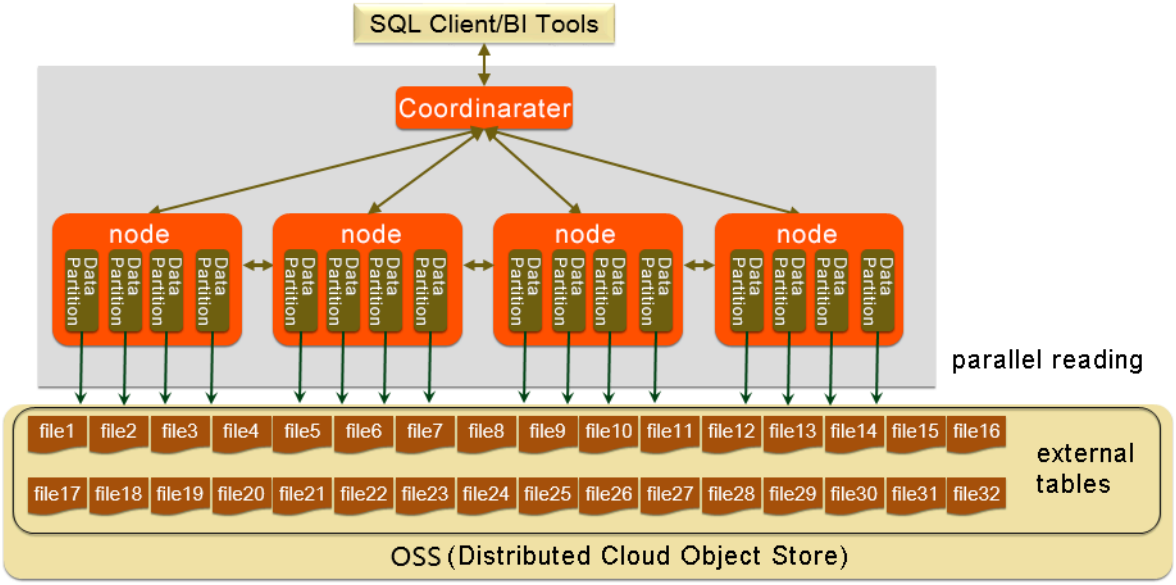
Notice:

- You can execute the COPY statement in AnalyticDB for PostgreSQL by using JDBC, in which the CopyIn method is encapsulated. For more information, visit Interface CopyIn.
- For more information about the COPY SQL statement, visit COPY.

4 Import or export OSS data by using OSS external tables

Object Storage Service (OSS) is a storage service that enables you to store any amount of data in Alibaba Cloud. AnalyticDB for PostgreSQL can import and export OSS data in parallel by using OSS external tables known as the gpossext function. AnalyticDB for PostgreSQL can also compress OSS external table files in the GZIP format to reduce the storage space and costs.

Currently, gpossext can read data from and write data to both gzip-compressed and uncompressed TEXT and CSV files.



This topic includes the following sections:

- Operations
- Parameters
- Examples
- Precautions
- TEXT and CSV formats
- SDK troubleshooting
- Instructions
- References

Operations

When you use OSS external tables in AnalyticDB for PostgreSQL, you may need to perform the following operations:

- Create an OSS external table plug-in (oss_ext)
- Import data in parallel
- Export data in parallel
- Create the OSS external table syntax

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 want 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

Perform the following steps to import data:

1. Distribute data evenly among multiple files in OSS.

   Notice:

   All compute nodes of an AnalyticDB for PostgreSQL instance read data in parallel from the data files stored in OSS by using a round-robin algorithm. To increase read efficiency, we recommend that the number of data files in OSS be an integer multiple of the number of compute nodes in your AnalyticDB for PostgreSQL instance.
2. Create a readable external table for each database in your AnalyticDB for PostgreSQL instance.
3. Execute the following statement to import data in parallel:

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

Export data in parallel

Perform the following steps to export data:

1. Create a writable external table for each database in your AnalyticDB for PostgreSQL instance.

2. Execute the following statement to export data to OSS in parallel:

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

Create the OSS external table syntax

Execute the following statements to create the OSS external table syntax:

```
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

This section describes the parameters used in various operations including:

- Common parameters
- Import mode parameters
- Export mode parameters
- Other parameters

Common parameters

- Protocol and endpoint: the communication protocol and endpoint address in the " Protocol name://oss_endpoint" format. The protocol name is oss and oss_endpoint indicates the domain name of the OSS region.

  Notice:

  If you access your AnalyticDB for PostgreSQL instance from a host that is housed on Alibaba Cloud, we recommend that you use an internal endpoint containing "internal" in the name to avoid incurring Internet traffic.
- id: the AccessKey ID of the OSS account.
- key: the AccessKey secret of the OSS account.
- bucket: the bucket where the data files you want to access are stored. It must be an existing bucket in OSS.
- prefix: the prefix in the name of the directory used to store the data files you want to access. The prefix is directly matched and cannot be controlled by a regular expression.

The prefix, filepath, and dir parameters are mutually exclusive. Only one of them can be specified at a time.

- If you specify the prefix parameter when you create a readable external table for data import, all data files with the specified prefix in their names are imported from OSS.
  - If you set the prefix parameter to test/filename, all of the following data files will be imported:
    - test/filename
    - test/filenamexxx
    - test/filename/aa
    - test/filenameyyy/aa
    - test/filenameyyy/bb/aa
  - If you set the prefix parameter to test/filename/, only the following file out of the preceding files will be imported:
    - test/filename/aa
- If you specify the prefix parameter when you create a writable external table for data export, a unique name with the specified prefix is generated for each data file exported to OSS.

- dir: the virtual directory in OSS. The prefix, filepath, and dir parameters are mutually exclusive. Only only one of them can be specified at a time.
  - The specified directory must end with a forward slash (/), such as `test/mydir/`.
  - If you specify this parameter when you create an external table for data import, all data files stored within the directory are imported. However, the data files stored in other subdirectory levels are not imported. Unlike the filepath parameter, the dir parameter does not require the data files stored in it to follow specific naming conventions.
  - If you specify this parameter when you create an external table for data export, all data is exported as multiple files to the specified directory. The exported data files are named in the `filename.x` format, where x indicates a number. The values of x may be out of order.
- filepath: the file name used to filter the OSS data files you want to import. The file name contains the directory name. The prefix, filepath, and dir parameters are mutually

exclusive. Only one of them can be specified at a time. In addition, you can only specify the filepath parameter when you create a readable external table for data import.

- The file name specified by this parameter must contain the directory name but not the bucket name.
- Only the files named as filename or in the filename.x format are imported. In addition, the values of x must be consecutive numbers starting from 1. Assume that the following data files are stored in OSS and the filepath parameter is set to filename:

```
filename
filename.1
filename.2
filename.4,
```

Then, the filename, filename.1, and filename.2 files are imported. The filename.4 file is not imported because there is not a file named filename.3.

Import mode parameters

- async: specifies whether to import data asynchronously.
  - You can enable an auxiliary thread to expedite the import of data from OSS.
  - The asynchronous mode is enabled by default. If you want to disable it, set the async parameter to false or f.
  - The asynchronous mode consumes more hardware resources than the normal data import mode.
- compressiontype: the format used to compress imported data files.
  - If you retain the default value none, imported files are not compressed.
  - If you set this parameter to gzip, imported files are compressed in the GZIP format. Currently, only the GZIP format is supported.
- compressionlevel: the level of compression for data files written to OSS. Valid values: 1 to 9. Default value: 6.

Export mode parameters

- oss_flush_block_size: the buffer size for a single data flush to OSS. Unit: MB. Valid values : 1 to 128. Default value: 32.
- oss_file_max_size: the maximum size of a data file allowed to be written to OSS. If a data file reaches the maximum size, the data that remains is written to other data files. Unit: MB. Valid values: 8 to 4000. Default value: 1024.

- num_parallel_worker: the maximum number of threads that are allowed to run in parallel to compress the data written to OSS. Valid values: 1 to 8. Default value: 3.
- compressiontype: the compression format of exported data files.
  - If you retain the default value none, imported files are not compressed.
  - If you set this parameter to gzip, exported files are compressed in the GZIP format. Currently, only the GZIP format is supported.

Take note of the following points when you configure export mode parameters:

- You must specify the WRITABLE keyword when creating an external table for data export.
- 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 key.

Other parameters

The following error-tolerance parameters can be used for data import and export:

- oss_connect_timeout: the connection timeout period. Unit: seconds. Default value: 10.
- oss_dns_cache_timeout: the DNS timeout period. Unit: seconds. Default value: 60.
- oss_speed_limit: the minimum data transmission rate. Unit: byte/s. Default value: 1024.
- oss_speed_time: the maximum wait period during which the data transmission rate can be lower than its minimum value. Unit: seconds. Default value: 15.

If you retain the default values of the preceding parameters, a timeout is triggered after the transmission rate is less than 1 KB/s for 15 continuous seconds. For more information, see OSS SDK troubleshooting.

AnalyticDB for PostgreSQL also supports parameters that are compatible with the Greenplum external table syntax. For more information, visit CREATE EXTERNAL TABLE in Greenplum Database Documentation. These parameters include:

- FORMAT: the supported file format, such as TEXT or CSV.
- ENCODING: the data encoding format of a file, such as UTF-8.
- LOG ERRORS: indicates that the clause can ignore imported erroneous data and write the data to error_table. You can also use the count parameter to specify the error reporting threshold.

 **Note:**

- You can use LOG ERRORS to record information about the rows that fail to be imported in the profile that is associated with internal tables.

```
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 SEGMENT REJECT LIMIT 5;
```

- You can use the gp_read_error_log('external_table_name') function to obtain information about the rows that fail to be imported.

```
select * from gp_read_error_log('external_table_name');
```

- After the external table is deleted, the internal file is also deleted. You can also use the gp_truncate_error_log('external_table_name') function to delete the internal file.

```
select gp_truncate_error_log('external_table_name');
```

- Only AnalyticDB for PostgreSQL V4.3 allows you to use LOG ERRORS INTO error_table to specify the error table.

```
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;
```

Examples

```
# Create an OSS external table for data import.
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 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;
# Create an OSS external table for data export.
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);
# Create a heap table for data loading.
create table example
        (date text, time text, open float,
         high float, low float, volume int)
         DISTRIBUTED BY (date);
# Import data to the example heap table from the ossexample table in parallel.
insert into example select * from ossexample;
# Export data from the example heap table to OSS in parallel.
insert into ossexample_exp select * from example;
# In the following execution plan, all compute nodes participate in data import.
# 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.
explain insert into example select * from ossexample;
                                  QUERY PLAN
-------------------------------------------------------------------------------------------
 Insert (slice0; segments: 4)  (rows=250000 width=92)
   ->  Redistribute Motion 4:4  (slice1; segments: 4)  (cost=0.00..11000.00 rows=250000
width=92)
         Hash Key: ossexample.date
         ->  External Scan on ossexample  (cost=0.00..11000.00 rows=250000 width=92)
(4 rows)
# In the following query plan, each compute node directly exports local data to OSS
without redistributing the data.
explain insert into ossexample_exp select * from example;
                     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)
```

Precautions

- AnalyticDB for PostgreSQL and Greenplum use similar syntax to create and use external tables, but differ in regard to location-related parameters.

- Data import performance depends on the OSS performance as well as available resources of your AnalyticDB for PostgreSQL instance, such as CPU, I/O, memory, and network resources. To maximize the import performance, we recommend that you enable column-oriented storage 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 in Greenplum Database Documentation.
- We recommend that OSS and AnalyticDB for PostgreSQL instances be in the same region for the best import performance.

TEXT and CSV formats

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

- \n: the string used as a line delimiter or line break for TEXT and CSV files.
- DELIMITER: the string used to delimit columns.
  - If you specify the DELIMITER parameter, you must also specify the QUOTE parameter.
  - Recommended column delimiters include commas (,), vertical bars (|), \t, and other special characters.
- QUOTE: a pair of characters used to enclose user data.
  - The pair of characters specified by the QUOTE parameter is used to distinguish user data from control characters.
  - For the sake of efficient coding, it is not required to enclose data such as integers in QUOTE characters.
  - QUOTE cannot be the same string specified in 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 the user data from code for the machine.
- ESCAPE: the escape character used to distinguish data.
  - Escape characters are placed before characters that otherwise have a special meaning.
  - If ESCAPE is not specified, its default value is the same as QUOTE.
  - You can also use other characters as escape characters, such as the '\' used by MySQL by default.

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

When errors occur during the import or export process, the error log contains the following information:

- 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 that identifies the request. You can provide the req_id to OSS development engineers for further help.

For more information, see OSS error response. Timeout errors can be handled using oss_ext related parameters.

Instructions

If the data import process takes an abnormally extended period of time, see the descriptio ns on import performance in the "Precautions" section.

References

- OSS domain names
- Object Storage Service documentation
- OSS SDK troubleshooting
- OSS error response
- CREATE EXTERNAL TABLE in Greenplum Database Documentation
- CREATE TABLE in Greenplum Database Documentation

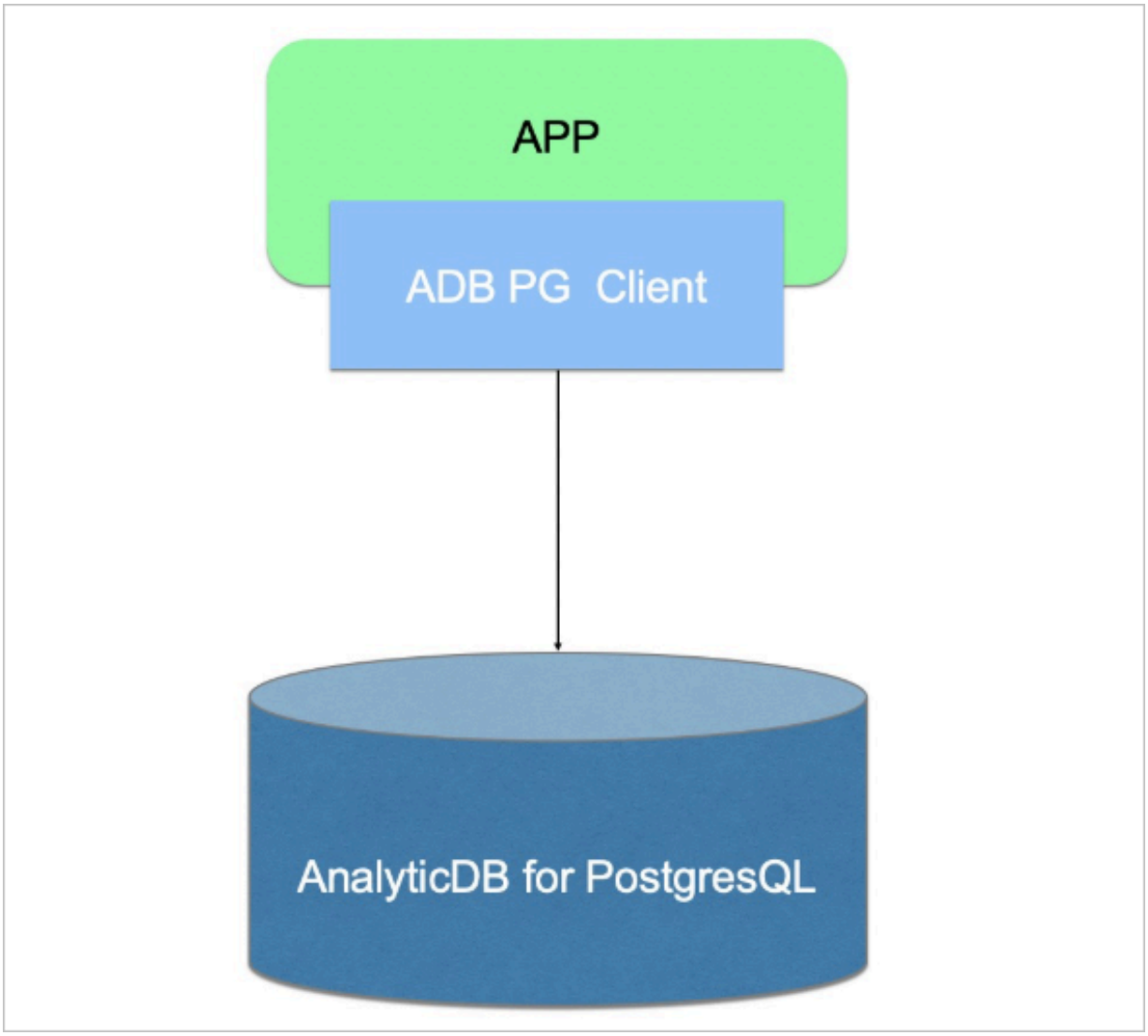
5 Use the Client SDK

This topic describes how to develop an efficient data write program by using the Client SDK. This program enables you to write data into AnalyticDB for PostgreSQL by calling API operations.

The Client SDK supports both custom development and integration of data write programs. A custom data write program developed by using the Client SDK simplifies the data write process with its internal mechanisms such as parallel processing and performs multiple times better than COPY and INSERT statements. You do not need to worry about connection pool or cache issues.

Note:

The Client SDK used with AnalyticDB for PostgreSQL is designed for efficient data writes. It does not read or process raw data.



Maven repositories

You can use Maven to configure the version of the Client SDK. The configuration of Maven is as follows:

```
<dependency>
  <groupId>com.alibaba.cloud.analyticdb</groupId>
  <artifactId>adb4pgclient</artifactId>
  <version>1.0.0</version>
</dependency>
```

Note:

- The Client SDK used with AnalyticDB for PostgreSQL depends on the following packages: druid (version 1.1.17), PostgreSQL JDBC (version 42.2.5), commons-lang3 (version 3.4), slf4j-api (version 1.7.24), and slf4j-log4j12 (version 1.7.24).
- If the system prompts version conflicts with the Client SDK, you must verify that you are using the correct versions of these packages.

Related operations

Table 5-1: Operations used for the DatabaseConfig class

Operation	Description
setHost(String adbHost)	Specifies the endpoint used to connect to the AnalyticDB for PostgreSQL database.
setPort(int port)	Specifies the port used to connect to the AnalyticDB for PostgreSQL database. The default port is 3432.
setDatabase(String database)	Specifies the name of the AnalyticDB for PostgreSQL database.
setUser(String username)	Specifies the username used to log on to the AnalyticDB for PostgreSQL database.
setPassword(String pwd)	Specifies the password used to log on to the AnalyticDB for PostgreSQL database.
addTable(List<String> table, String schema)	Specifies the tables into which you want to insert data. This operation can be called multiple times. Each time you can only specify tables that use the same schema. However, after you use the DatabaseConfig class to create an Adb4pgClient object, this operation no longer takes effect.
setColumns(List<String> columns, String tableName, String schemaName)	Specifies the columns you want to insert into a table. If you want to insert all columns into a table, set the columns parameter to columnList.add("*") for that table. You must specify columns for all tables listed in the table parameter. Otherwise, the operation fails the system check.
setInsertIgnore(boolean insertIgnore)	Specifies whether to ignore the rows with conflicted primary keys. This operation takes effect on all specified tables. Default value: true.
setEmptyAsNull(boolean emptyAsNull)	Specifies whether to set empty values to null. This operation takes effect on all specified tables. Default value: false.

Operation	Description
setParallelNumber(int parallelNumber)	Specifies the maximum number of concurrent threads used to write data into the AnalyticDB for PostgreSQL database. This operation takes effect on all specified tables. Default value: 4. In normal cases, we recommend that you retain the default value.
setLogger(Logger logger)	Specifies the logger used by the Adb4pgClient object. In AnalyticDB for PostgreSQL, slf4j.Logger is used.
setRetryTimes(int retryTimes)	Specifies the maximum number of retries to commit data in the event of an error. Default value: 3.
setRetryIntervalTime(long retryIntervalTime)	Specifies the interval between every two retries. Unit: milliseconds. Default value: 1000.
setCommitSize(long commitSize)	Specifies the volume of data automatically committed. Unit: bytes. Default value: 1000000. In normal cases, we recommend that you retain the default value.

Table 5-2: Operations used for the Row class

Operation	Description
setColumn(int index, Object value)	Specifies the value of a column in the row. You can call this operation multiple times based on the sequence of columns in the row to specify multiple columns. This operation does not support the reuse of row objects. Each data record must be associated with a specific row object.
setColumnValues(List<Object> values)	Specifies the values of multiple columns in the row.
updateColumn(int index, Object value)	Updates the value of a column in the row. This operation allows you to reuse a row object by updating the data in it.

Table 5-3: Operations used for the Adb4pgClient class

Operation	Description
addRow(Row row, String tableName, String schemaName)/addRows(List<Row> rows, String tableName, String schemaName)	Inserts one or more row-formatted data records into a table. The data records are stored in the buffer of the Client SDK until they are committed. If the volume of data records stored in the buffer reaches the value of the commitSize parameter, the system automatically commits the data records when you call the addRow or addRows operation. If the automatic commit fails, the system reports an error that contains the failed data records. You must handle the error based on the error information.
addMap(Map<String, String> dataMap, String tableName, String schemaName)/addMaps(List<Map<String, String>> dataMaps, String tableName, String schemaName)	Inserts one or more map-formatted data records into a table. The data records are stored in the buffer of the Client SDK. If the volume of data records stored in the buffer reaches the value of the commitSize parameter, the system automatically commits the data records when you call the addMap or addMaps operation. If the automatic commit fails, the system reports an error that contains the failed data records. You must handle the error based on the error information.
commit()	Commits the data records stored in the buffer of the Client SDK. If the commit fails, the system reports an error that contains the failed statements. You must handle the error based on the error information.
TableInfo getTableInfo(String tableName, String schemaName)	Obtains the schema of a table.
List<ColumnInfo> getColumnInfo(String tableName, String schemaName)	Obtains the columns of a table. You can call the columnInfo.isNullable() operation to check whether the ColumnInfo class can be null.

Operation	Description
stop()	Releases the internal thread pools and resources used by the Adb4pgClient object after you no longer need it. If data records in the buffer of the Client SDK are not committed when you call this operation, the system reports an error. To forcibly release the internal thread pools and resources, call the forceStop() operation.
forceStop()	Forcibly releases the internal thread pools and resources used by the Adb4pgClient object. After you call this operation, data records that have not been committed in the buffer of the Client SDK are lost. Therefore, we recommend that you do not call this operation unless necessary.
Connection getConnection() throws SQLException	Retrieves the connection with the AnalyticDB for PostgreSQL database from the connection pool of the Adb4pgClient object. The retrieved connection works the same as a Java Database Connectivity (JDBC) connection. You can use the retrieved connection to perform write operations except COPY. **Note:** You must release the resources such as ResultSet, Statement, and Connection used by the Adb4pgClient object after you no longer need it.

Table 5-4: Operations used for the ColumnInfo class

Operation	Description
boolean isNullable()	Specifies whether the ColumnInfo class can be null.

Error code	HTTP status code	Description
COMMIT_ERROR_DATA_LIST	101	The error returned because some data records have failed to be committed. **Note:** You can call the e.getErrData() operation to obtain the list of failed data records (List<String>). This error may be returned for the addMap, addMaps, addRow, addRows, or commit operation. You must handle this error for each operation separately.
COMMIT_ERROR_OTHER	102	The error returned because an exception other than a commit failure has occurred.
ADD_DATA_ERROR	103	The error returned because data records have failed to be added.
CREATE_CONNECTION_ERROR	104	The error returned because a connection has failed to be established.
CLOSE_CONNECTION_ERROR	105	The error returned because a connection has failed to be terminated.
CONFIG_ERROR	106	The error returned because the configuration of the DatabaseConfig class has failed.
STOP_ERROR	107	The error returned because the Adb4pgClient object has failed to be stopped.
OTHER	999	The default error returned.

Sample code

```
public class Adb4pgClientUsage {
    public void demo() {
```

```
        DatabaseConfig databaseConfig = new DatabaseConfig();
        //Should set your database real host or url
        databaseConfig.setHost("100.100.100.100");
        //Should set your database real port
        databaseConfig.setPort(8888);
        //Specify the username used to log on to your AnalyticDB for PostgreSQL database.
        databaseConfig.setUser("your user name");
        //Specify the password used to log on to your AnalyticDB for PostgreSQL database.
        databaseConfig.setPassword("your password");
       //Specify the name of your AnalyticDB for PostgreSQL database.
        databaseConfig.setDatabase("your database name");
        //Specify the tables into which you want to insert data.
        List<String> tables = new ArrayList<String>();
        tables.add("your table name 1");
        tables.add("your table name 2");

        //You can call the addTable operation to specify tables that use the same schema.
However, after you use the DatabaseConfig class to create an Adb4pgClient object, the
specified tables cannot be changed.
        //By default, if the table schema name parameter is set to null, the public schema is
used.
        databaseConfig.addTable(tables, "table schema name");

        //Specify the columns you want to insert into a table.
        List<String> columns = new ArrayList<String>();
        columns.add("column1");
        columns.add("column2");
        //If you want to insert all columns into a table, set the columns parameter to
columns.add("*") for that table.
        databaseConfig.setColumns(columns, "your table name 1", "table schema name");
        databaseConfig.setColumns(Collections.singletonList("*"),"your table name 2", "table
 schema name");


        //If the value of column is empty, set null
        databaseConfig.setEmptyAsNull(false);
        //Specify whether to ignore the rows with conflicted primary keys.
        databaseConfig.setInsertIgnore(true);
        //Specify to retry up to three times if data fails to be committed into AnalyticDB for
PostgreSQL.
        databaseConfig.setRetryTimes(3);
        //Specify a 1000-millisecond retry interval.
        databaseConfig.setRetryIntervalTime(1000);
        //Specify to initialize the Adb4pgClient object. After the Adb4pgClient object is
initialized, the configuration of the DatabaseConfig class cannot be modified.
        Adb4pgClient adbClient = new Adb4pgClient(databaseConfig);

        //Specify to store data records to the buffer of the Client SDK and commit them
simultaneously. For more information, see the "Precautions" section.
        for (int i = 0; i < 10; i++) {
            //Add row(s) to buffer. One instance for one record
            Row row = new Row(columns.size());
            //Set column
            //the column index must be same as the sequence of columns
            //the column value can be any type, internally it will be formatted according to
column type
            row.setColumn(0, i); // Number value
            row.setColumn(1, "string value"); // String value
            //If the volume of data records in the buffer reaches the upper limit, the system
automatically commits them when you call the addRow or addMap operation.
            //If the automatic commit fails, the system returns the "COMMIT_ERROR_DATA_LIST
" error.
            adbClient.addRow(row, "your table name 1", "table schema name");
        }
```

```
        Row row = new Row();
        row.setColumn(0, 10); // Number value
        row.setColumn(1, "2018-01-01 08:00:00"); // Date/Timestamp/Time value
        adbClient.addRow(row, "your table name 1", "table schema name");
        //Update column. A Row object can be reused.
        row.updateColumn(0, 11);
        row.updateColumn(1, "2018-01-02 08:00:00");
        adbClient.addRow(row, "your table name 1", "table schema name");

        //Add map(s) to buffer
        Map<String, String> rowMap = new HashMap<String, String>();
        rowMap.put("t1", "12");
        rowMap.put("t2", "string value");
        //If you want to commit more than one data record, we recommend that you store
these data records to the buffer of the Client SDK and commit them at a time.
        adbClient.addMap(rowMap, "your table name 2", "table schema name");

        //Commit buffer to ADS
        //Buffer is cleaned after successfully commit to ADS
        try {
            adbClient.commit();
        } catch (Exception e) {
            // TODO: Handle exception here
        } finally {
            adbClient.stop();
        }
    }

}
```

Precautions

- AnalyticDB for PostgreSQL is not thread-safe. If you call more than one thread, make sure that each thread maintains its own Adb4pgClient object.

  Notice:
  If multiple threads share one Adb4pgClient object, security issues arise and data write performance is constrained.

- A data record is only considered to be written into AnalyticDB for PostgreSQL after it is committed.
- If the Adb4pgClient object reports an error, you must handle the error based on the error information. If a data record cannot be committed, you can commit it again. You also have the option to skip the failed data record after you note it down.
- When you write data into AnalyticDB for PostgreSQL, the data records may be stored to the buffer and batch committed. This increases CPU utilization. Therefore, we recommend that you maintain an appropriate number of threads and watch the garbage collection status of your application.
- We recommend that you commit 10,000 data records at a time.

- You must complete the configuration of the DatabaseConfig class before you create an Adb4pgClient object. After the Adb4pgClient object is created, the configuration of the DatabaseConfig class cannot be modified.
- The Client SDK aims to optimize data writes (INSERT statements). To optimize the execution of other SQL statements, you can call the getConnection() operation to establish a JDBC connection.

6 Import data

6.1 Synchronize data from ApsaraDB RDS for SQL Server instance to an AnalyticDB for PostgreSQL instance

Data Transmission Service (DTS) allows you to synchronize Data from ApsaraDB RDS for SQL Server instance to an AnalyticDB for PostgreSQL instance for centralized analysis of enterprise Data.

Prerequisites

- #unique_29(SQL Server 2008 R2, 2012, 2016, or 2017).
- Create an AnalyticDB for PostgreSQL instance.
- The tables to be synchronized from the ApsaraDB RDS for SQL Server instance must contain primary keys.
- The AnalyticDB for PostgreSQL to be synchronized in the destination table instance must have primary keys or unique indexes.

Precautions

- DTS uses read and write resources of the source and destination databases during full data migration. This may increase the database load. If the database performance is unfavorable, the specification is low, or the data volume is large, database services may become unavailable. For example, DTS occupies a large amount of read and write resources in the following cases: a large number of slow SQL queries are performed on the source database, the tables have no primary keys, or a deadlock occurs in the destination database. Before you migrate data, evaluate the performance of the source and destination databases. We recommend that you migrate data during off-peak hours. For example, you can migrate data when the CPU usage of the source and destination databases is less than 30%.
- To ensure that the displayed latency is accurate, DTS adds a heartbeat table(named `dts_log_heart_beat`) to the source database.

Limits

- A data synchronization task can synchronize data with only one database. If you want to synchronize data from multiple databases, you must create a data synchronization task for each database.

- Initial schema synchronization is not supported. Before configuring data synchronization task, create a structure in the destination AnalyticDB for PostgreSQL. For more information, see Preparations.
- DTS can only synchronize DML operations, such as INSERT, UPDATE, and DELETE operations. DTS cannot synchronize DDL operations. During data synchronization, if the source table schema is modified (for example, ADD COLUMN), an error occurs during data synchronization task.

  Note:
  In this case, you need to change the schema of the destination table to the same source table, and then restart the synchronization task.
- DTS does not support the synchronize of data of the following types: TIMESTAMP, CURSOR, ROWVERSION, hive ID, SQL_VARIANT, SPATIAL GEOMETRY, SPATIAL GEOMETRY, and TABLE.

Permissions required for database accounts

Database	Required permission	Authorization method
ApsaraDB RDS for SQL Server instance	The owner permission of the source database.	modify account permissions
AnalyticDB for PostgreSQL instance	• The LOGIN permission. • The SELECT, CREATE, INSERT, UPDATE, and DELETE permissions of the destination table. • The CONNECT and CREATE permissions of the destination database. • The CREATE permissions of the destination Schema. • Copy permissions (memory-based batch copy). **Note:** You can also use the initial account of the AnalyticDB for PostgreSQL instance.	manage user permissions

Preparations

In this synchronization scenario , you need to referring the following table to create a database and tables in the destination AnalyticDB for PostgreSQL instance based on the

schemas of the objects in the source ApsaraDB RDS for SQL Server instance. For related syntax, see SQL syntax.

Warning:
This scenario uses data synchronization between heterogeneous databases. Data types do not have one-to-one correspondence. We recommend that you evaluate the impacts of data type mappings on your business.

Table 6-1: Supported types and mappings

ApsaraDB RDS for SQL Server data type	AnalyticDB for PostgreSQL data type
INT	INTEGER
SMALLINT	SMALLINT
TINYINT	SMALLINT
BIGINT	BIGINT
BIT	BIT
DECIMAL	DECIMAL
NUMERIC	DECIMAL
CHAR	CHARACTER[(N)]
VARCHAR	CHARACTER[(N)]
NCHAR	CHARACTER[(N)]
TEXT/NTEXT	Text
FLOAT	DOUBLE
REAL	REAL
DATE	DATE
DATETIME/DATETIME2	TIMESTAMP WITHOUT TIME ZONE
DATETIMEOFFSET	TIMESTAMP WITH TIME ZONE
SMALLDATETIME	TIMESTAMP WITHOUT TIME ZONE
TIME	TIME WITHOUT TIME ZONE
BINARY/VARBINARY	bytea
IMAGE	bytea
MONEY/SMALLMONEY	DECIMAL
UNIQUEIDENTIFIER	CHARACTER VARING

ApsaraDB RDS for SQL Server data type	AnalyticDB for PostgreSQL data type
XML format	XML format

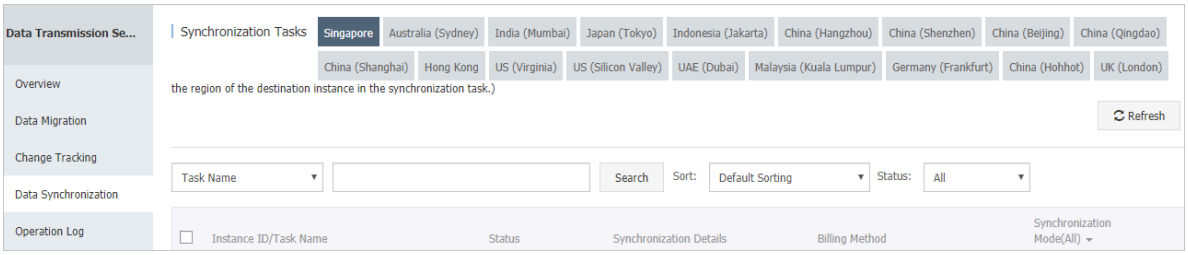
Procedure

1. Purchase a data synchronization instance. For more information, see #unique_31.

   Note:

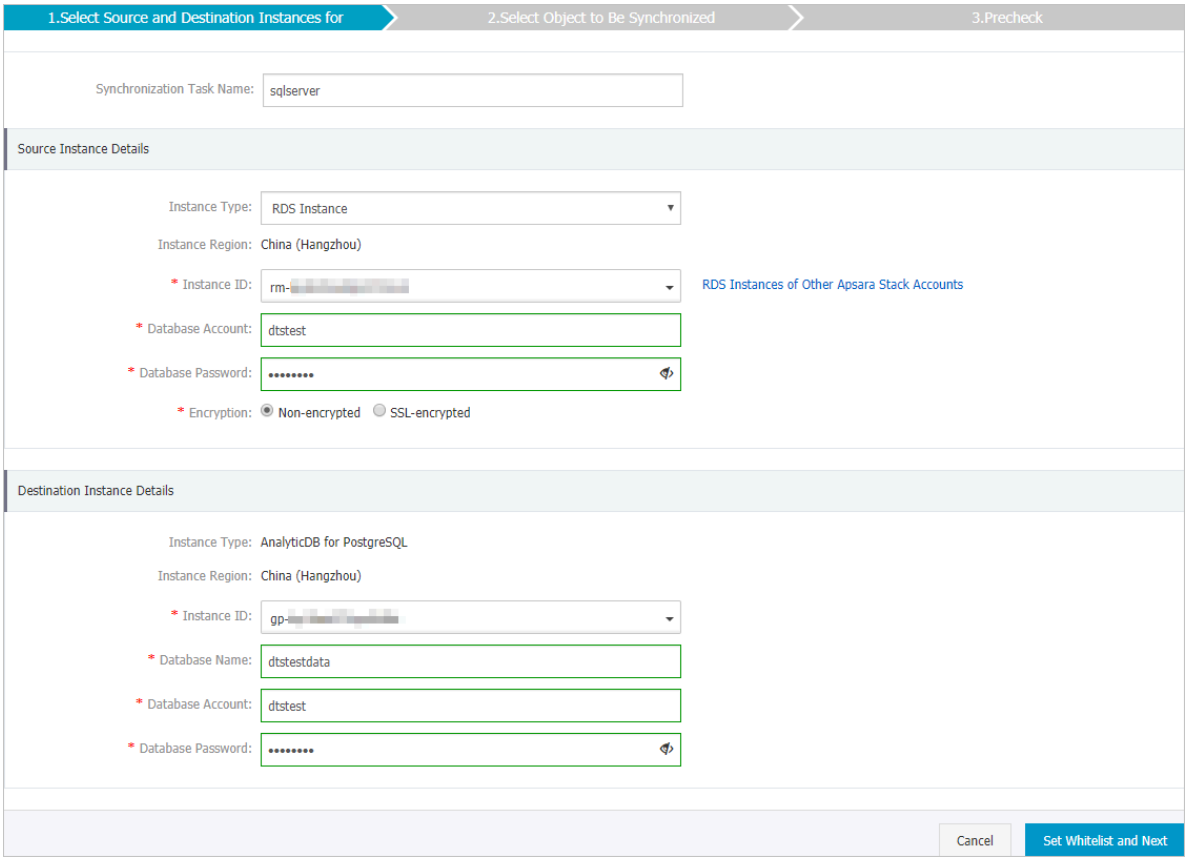
   On the buy page, set Source Instance to **SQLServer**, Target Instance to **AnalyticDB for PostgreSQL**, and Synchronization Topology to **One-Way Synchronization**.

2. Log on to the DTS console.
3. In the left-side navigation pane, click **Data Synchronization**.
4. At the top of the **Synchronization Tasks** page, select the region where the destination instance resides.



5. Find the data synchronization instance and click **Configure Synchronization Channel** in the Actions column.

6. Configure the source and destination instances.



Section	Parameter	Description
N/A	Synchronization Task Name	DTS automatically generates a task name. We recommend that you specify an informative name for easy identification. You do not need to use a unique task name.
Source Instance Details	Instance Type	Select **RDS Instance**.
	Instance Region	The region of the source instance that you select when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the source RDS instance.
	Database Account	Enter the database account of the RDS instance. For permission requirements, see Permissions required for database accounts.
	Database Password	Enter the password for the destination database account.

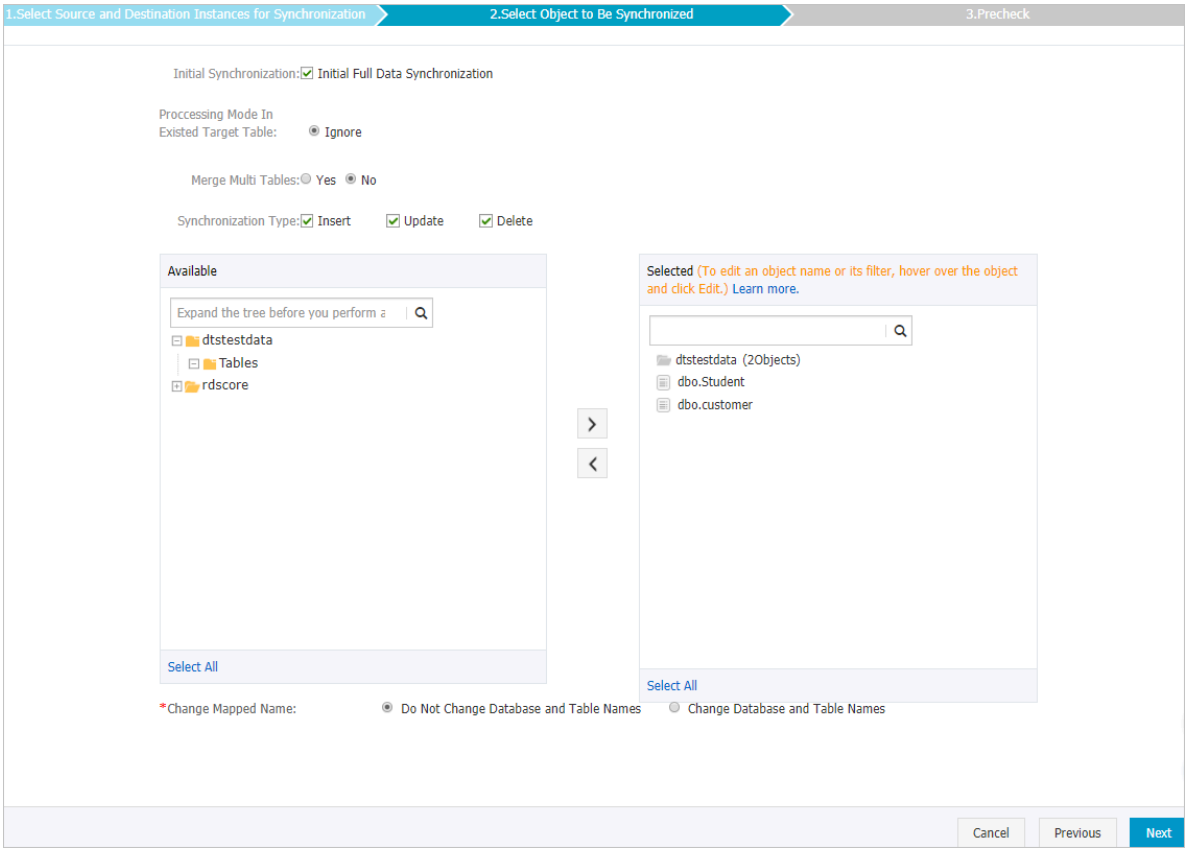
Section	Parameter	Description
	Encryption	Select **Non-encrypted** or **SSL-encrypted**. If set to **SSL-encrypted**, you must enable SSL encryption for the RDS instance. For more information, see #unique_32. **Note:** The **Encryption** parameter is available only for Mainland China.
Destination Database	Instance Type	The value of this parameter is set to **AnalyticDB for PostgreSQL** and cannot be changed.
	Instance Region	The region of the destination instance. The region is the same as the destination region that you selected when you purchased the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the destination AnalyticDB for PostgreSQL instance.
	Database Name	Enter the name of the destination database.
	Database Account	Enter the database account of the AnalyticDB for PostgreSQL instance. For permission requirements, see Permissions required for database accounts.
	Database Password	Enter the password for the destination database account.

7. In the lower-right corner of the page, click **Set Whitelist and Next**.

Note:

The IP addresses of DTS servers are automatically added to the whitelists of the RDS instance and AnalyticDB for PostgreSQL instance . This ensures that DTS servers can connect to the source and destination instances.

8. Configure the synchronization policy and objects.

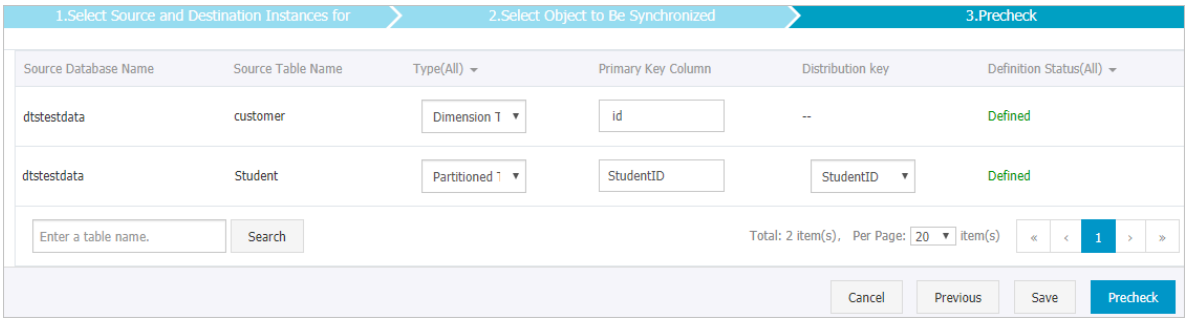


Parameter	Description
Initial Synchronization	By default, you need to select **initial full data synchronization**. After the PreCheck, DTS synchronizes the historical data of the required objects from the source instance to the destination instance. The data is the basis for subsequent incremental data synchronization. **Note:** **Initial schema synchronization** is not supported.
Processing Mode In Existed Target Table	Since the related data table has been created in the target AnalyticDB for PostgreSQL in Preparations. it is fixed as **Ignore**: skips the **Schema Name Conflict** item during the precheck.

Parameter	Description
Merge Multi Tables	• Select **yes**: In OLTP scenarios, Sharding is usually implemented to speed up the response to business tables. In AnalyticDB for PostgreSQL, a single data table can store a large amount of data, making it easier to use single-table query. In such scenarios, you can use the multi-table merge feature of DTS to synchronize multiple tables with the same table structure (that is, each table Shard) in the source database to the same table in the AnalyticDB for PostgreSQL.  **Note:** - After selecting multiple tables, you can change them to the same table name in the AnalyticDB for PostgreSQL by using the object name mapping feature. For more information about the object name mapping feature, see #unique_33. - You must add `__dts_data_source` column (tpye is text) to store the data source. DTS will write column values in the format of `<dts Data Synchronization instance ID order:< source database name>. <source Schema name>. <source table name>` to distinguish the source of the table. For example `dts********:dtstestdata.testschema.customer1`. - You can merge the data source columns based on tasks rather than tables. To merge data of some tables but not some other tables, you need to create two data synchroniz ation task. • Select **no**: the default option.
Synchronization Type	Select the type of operation to be synchronized based on the business. The default status is selected.

Parameter	Description
Objects to be synchronized	Select tables from the **Available** section and click the right arrow ( ) icon to add the tables to the **Selected** section. You can select tables and databases as the objects to be synchronized. **Note:** • You can only select one database or multiple tables in one database. If you want to synchronize data from multiple databases, you must create a data synchronization task for each database. • After an object is synchronized to the destination database, the name of the object remains unchanged. You can change the names of the objects that are synchronized to the destination database by using the object name mapping feature. For more information, see #unique_33. • If the configuration **multi-table merging** for **yes**, after selecting multiple tables in the source database, you need to change them to the same table name in the AnalyticDB for PostgreSQL by using the object name mapping function.

9. According to the table structure defined in AnalyticDB for PostgreSQL inPreparations, set the table type, primary key columns, and distributed key configuration.



Note:

For more information about the primary key columns and distributed keys, see table constraint definitions and table distribution key definition.

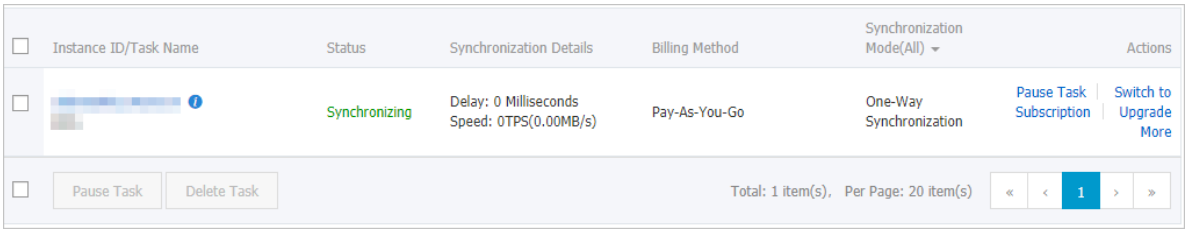
10. In the lower-right corner of the page, click **Precheck**.

Note:

- Before you can start the data synchronization task, a precheck is performed. You can start the data synchronization task only after the task passes the precheck.
- If the task fails to pass the precheck, click the icon next to each failed item to view details. Troubleshoot the issues based on the causes and run the precheck again.

11. Close the **Precheck** dialog box after the following message is displayed: **The precheck is passed.** Then, the data synchronization task starts.
12. Wait until the initial synchronization is complete and the data synchronization task is in the **Synchronizing** state.

    You can view the status of the data synchronization task on the **Synchronization Tasks** page.



6.2 Use DTS to synchronize data from an ApsaraDB RDS for MySQL instance to an AnalyticDB for PostgreSQL instance

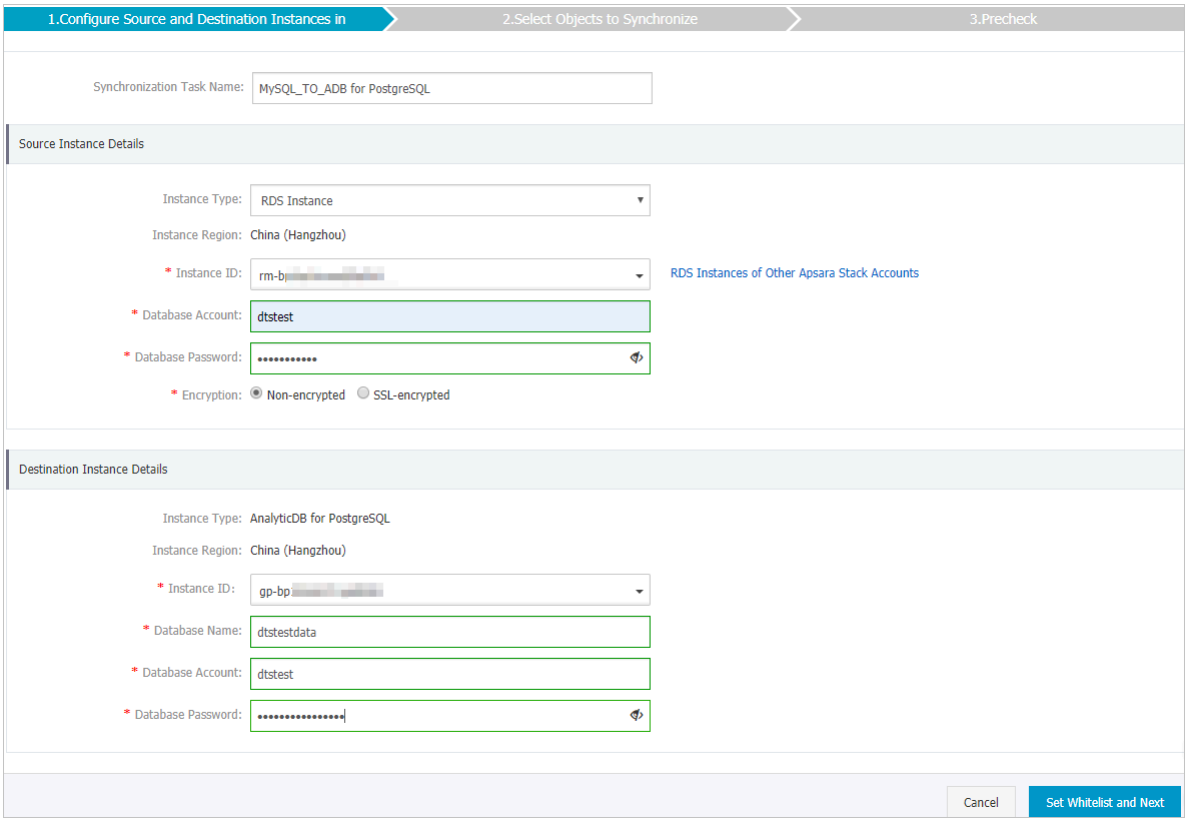
This topic describes how to use Data Transmission Service (DTS) to synchronize data from an ApsaraDB RDS for MySQL instance to an AnalyticDB for PostgreSQL instance. DTS facilitates data transmission and the centralized analysis of enterprise data.

Prerequisites

- The tables to be synchronized from the ApsaraDB RDS for MySQL instance contain primary keys.
- An AnalyticDB for PostgreSQL instance is created. For more information, see Create an instance.

Procedure

1. Configure the source and destination instances.



Section	Parameter	Description
N/A	Synchronization Task Name	DTS automatically generates a task name. We recommend that you use an informative name for easy identification. You do not need to use a unique task name.
Source Instance Details	Instance Type	Select **RDS Instance**.
	Instance Region	The region of the source instance. The value is the same as that you selected when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the source RDS instance.

Section	Parameter	Description
	Database Account	Enter the database account for the source ApsaraDB RDS for MySQL instance. **Note:** If the database type of the source RDS instance is **MySQL 5.5** or **MySQL 5.6**, you do not need to configure the **database account** and **database password**.
	Database Password	Enter the password for the database account.
	Encryption	Select **Non-encrypted** or **SSL-encrypted**. If you want to select **SSL-encrypted**, you must enable SSL encryption for the RDS instance before configuring the data synchronization task. For more information, see Configure SSL encryption for an RDS for MySQL instance . **Note:** The **Encryption** parameter is available only in mainland China and Hong Kong(China).
Destination Instance Details	Instance Type	The value of this parameter is set to **AnalyticDB for PostgreSQL** and cannot be changed.
	Instance Region	The region of the destination instance. The value is the same as that you selected when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the destination AnalyticDB for PostgreSQL instance.
	Database Name	The name of the destination database.
	Database Account	Enter the database account for the destination AnalyticDB for PostgreSQL instance. **Note:** The database account must have the SELECT, INSERT, UPDATE, DELETE, COPY, TRUNCATE, and ALTER TABLE permissions.

Section	Parameter	Description
	Database Password	Enter the password for the database account.

2. In the lower-right corner of the page, click **Set Whitelist and Next**.

> **Note:**
> The CIDR blocks of DTS servers are automatically added to the whitelists of the source and destination instances. This ensures that DTS servers can connect to the source and destination instances.

6.3 Use DTS to synchronize data from an Apsara PolarDB for MySQL cluster to an AnalyticDB for PostgreSQL instance

This topic describes how to use Data Transmission Service (DTS) to synchronize data from an Apsara PolarDB for MySQL cluster to an AnalyticDB for PostgreSQL instance. DTS facilitates data transmission and the centralized analysis of enterprise data.

Prerequisites

- The binary logging feature for the Apsara PolarDB for MySQL cluster is enabled. For more information, see Enable binlogging.
- The tables to be synchronized from the Apsara PolarDB for MySQL cluster contain primary keys.
- An AnalyticDB for PostgreSQL instance is created. For more information, see Create an instance.

Notes

- DTS uses read and write resources of the source and destination databases during initial full data synchronization. This may increase the database load. If the database performance is unfavorable, the specification is low, or the data volume is large, database services may become unavailable. For example, DTS occupies a large amount of read and write resources in the following cases: a large number of slow SQL queries are performed on the source database, the tables have no primary keys, or a deadlock occurs in the destination database. Before synchronizing data, you must evaluate the performance of the source and destination databases. We recommend that you synchronize data during off-peak hours. For example, you can synchronize data when the CPU usage of the source and destination databases is less than 30%.

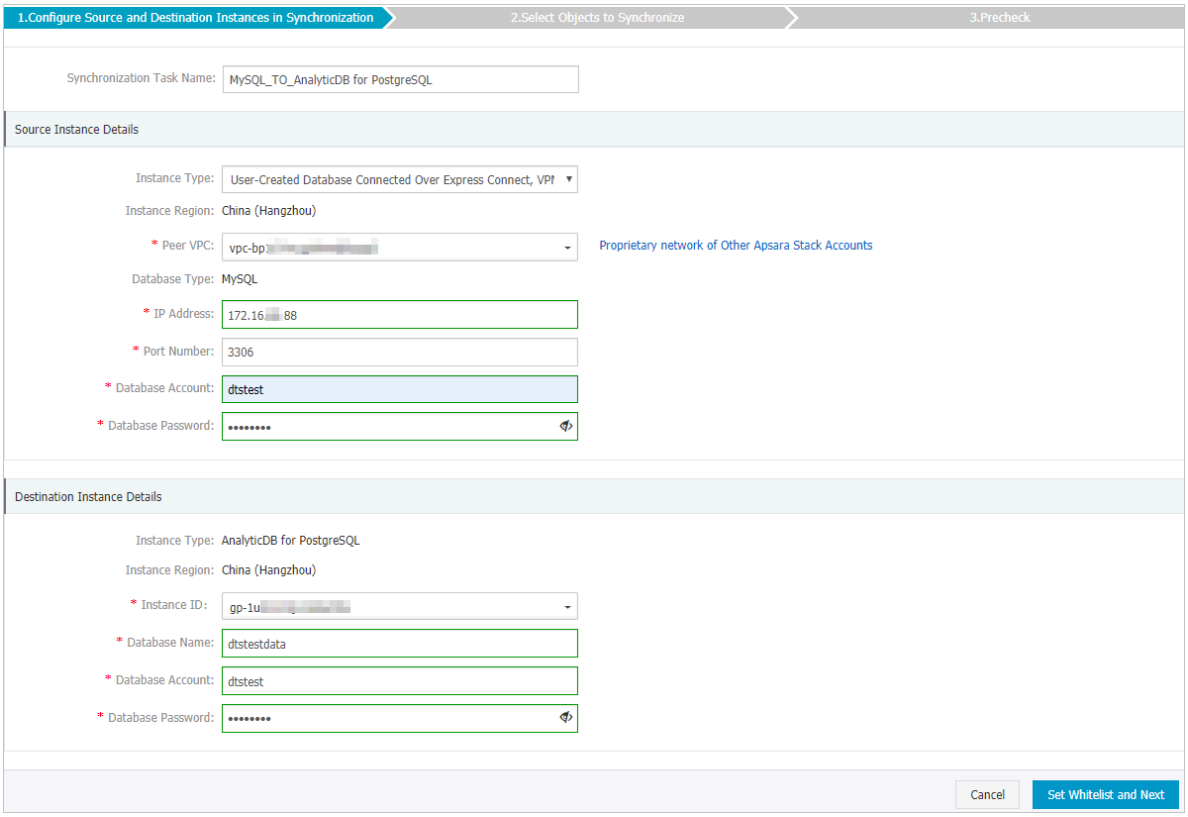
- During initial full data synchronization, concurrent INSERT operations cause segments in the tables of the destination instance. After initial full data synchronization, the tablespace of the destination instance is larger than that of the source cluster.

Term mappings

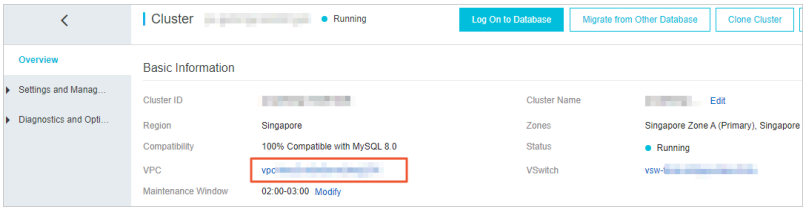
Term in Apsara PolarDB for MySQL	Term in AnalyticDB for PostgreSQL
Database	Schema
Table	Table

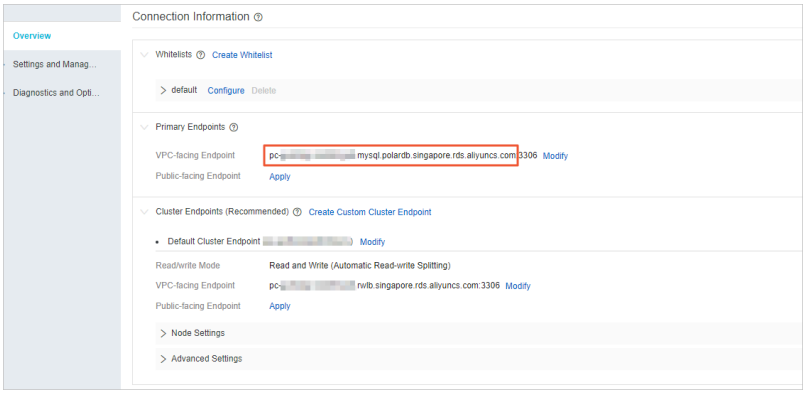
Procedure

1. Configure the source and destination instances.



Section	Parameter	Description
N/A	Synchronization Task Name	DTS automatically generates a task name. We recommend that you use an informative name for easy identification. You do not need to use a unique task name.

Section	Parameter	Description
Source Instance Details	Instance Type	Select **User-Created Database Connected Over Express Connect, VPN Gateway, or Smart Access Gateway**. **Note:** You cannot select Apsara PolarDB for MySQL cluster as the instance type. To synchronize data from a Apsara PolarDB for MySQL cluster, you can select User-Created Database Connected Over Express Connect, VPN Gateway, or Smart Access Gateway.
	Instance Region	The region of the Apsara PolarDB for MySQL cluster. The value is the same as that you selected when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Peer VPC	Select the ID of the VPC where the Apsara PolarDB for MySQL cluster resides. To obtain the VPC ID, you can log on to the Apsara PolarDB console and click the cluster ID. On the Overview page that appears, you can view the ID of the VPC where the cluster resides in the **Basic Information** section. 
	Database Type	The value of this parameter is set to **MySQL** and cannot be changed.

Section	Parameter	Description
	IP Address	Enter the private IP address of the Apsara PolarDB for MySQL cluster. In this example, enter **172.16.20.36**. You can obtain the private IP address by pinging the **VPC-facing endpoint** of the Apsara PolarDB for MySQL cluster. 
	Port Number	Enter the port number of the Apsara PolarDB for MySQL cluster. The default port number is **3306**.
	Database Account	Enter the database account for the Apsara PolarDB for MySQL cluster. **Note:** The account must have the REPLICATION SLAVE permission, the REPLICATION CLIENT permission, the SHOW VIEW permission, and the permission to perform SELECT operations on the required objects.
	Database Password	Enter the password for the database account.
Destination Instance Details	Instance Type	The value of this parameter is set to **AnalyticDB for PostgreSQL** and cannot be changed.
	Instance Region	The region of the destination instance. The value is the same as that you selected when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the destination AnalyticDB for PostgreSQL instance.
	Database Name	Enter the name of the destination database in the AnalyticDB for PostgreSQL instance.

Section	Parameter	Description
	Database Account	Enter the database account for the destination AnalyticDB for PostgreSQL instance. **Note:** The database account must have the ALL permission for the objects to be synchronized.
	Database Password	Enter the password for the database account.

2. In the lower-right corner of the page, click **Set Whitelist and Next**.

Note:
The CIDR blocks of DTS servers are automatically added to the whitelists of the Apsara PolarDB for MySQL cluster and the AnalyticDB for PostgreSQL instance. This ensures that DTS servers can connect to the source cluster and destination instance.

6.4 Use DTS to synchronize data from an ECS-based user-created MySQL database to an AnalyticDB for PostgreSQL instance

This topic describes how to use Data Transmission Service (DTS) to synchronize data from an ECS-based user-created MySQL database to an AnalyticDB for PostgreSQL instance. DTS facilitates data transmission and the centralized analysis of enterprise data.

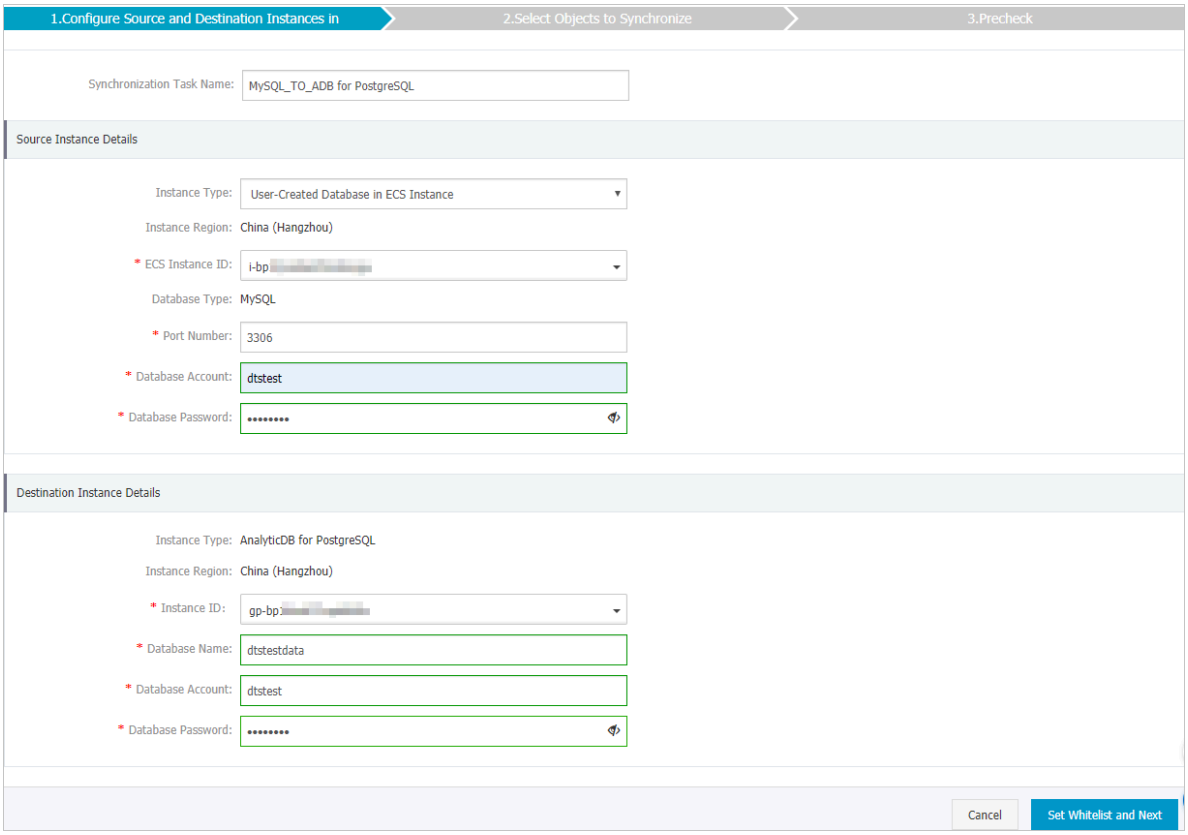
Prerequisites

- The version of the user-created MySQL database is 5.1, 5.5, 5.6, 5.7, or 8.0.
- The binary logging feature is enabled for the source database. A database account is created for the data synchronization task. For more information, see #unique_37.

  Note:
  The account must have the REPLICATION SLAVE permission, the REPLICATION CLIENT permission, the SHOW VIEW permission, and the permission to perform SELECT operations on the required objects.
- The tables to be synchronized from the source database contain primary keys.
- An AnalyticDB for PostgreSQL instance is created. For more information, see Create an instance.

Procedure

1. Configure the source and destination instances.



Section	Parameter	Description
N/A	Synchronization Task Name	DTS automatically generates a task name. We recommend that you use an informative name for easy identification. You do not need to use a unique task name.
Source Instance Details	Instance Type	Select **User-Created Database in ECS Instance**.
	Instance Region	The region of the source instance. The value is the same as that you selected when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the ECS instance that is connected to the user-created MySQL database.
	Database Type	The value of this parameter is set to **MySQL** and cannot be changed.
	Port Number	Enter the service port number of the user-created MySQL database. The default port number is **3306**.

Section	Parameter	Description
	Database Account	Enter the account for the user-created MySQL database. **Note:** The account must have the REPLICATION SLAVE permission, the REPLICATION CLIENT permission, the SHOW VIEW permission, and the permission to perform SELECT operations on the required objects.
	Database Password	Enter the password for the database account.
Destination Instance Details	Instance Type	The value of this parameter is set to **AnalyticDB for PostgreSQL** and cannot be changed.
	Instance Region	The region of the destination instance. The value is the same as that you selected when purchasing the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the destination AnalyticDB for PostgreSQL instance.
	Database Name	Enter the name of the destination database.
	Database Account	Enter the database account for the destination AnalyticDB for PostgreSQL instance. **Note:** The database account must have the SELECT, INSERT, UPDATE, DELETE, COPY, TRUNCATE, and ALTER TABLE permissions.
	Database Password	Enter the password for the database account.

2. In the lower-right corner of the page, click **Set Whitelist and Next**.

Note:

The CIDR blocks of DTS servers are automatically added to the inbound rule of the source ECS instance and the whitelist of the destination AnalyticDB for PostgreSQL instance. This ensures that DTS servers can connect to the source and destination instances.

6.5 Use DTS to synchronize data from a user-created MySQL database connected by a leased line, VPN Gateway, or Smart Access Gateway to an AnalyticDB for PostgreSQL instance

This topic describes how to use Data Transmission Service (DTS) to synchronize data from a user-created MySQL database connected by a leased line, VPN Gateway, or Smart Access Gateway to an AnalyticDB for PostgreSQL instance. DTS facilitates data transmission and the centralized analysis of enterprise data.

Prerequisites

- The version of the user-created MySQL database is 5.1, 5.5, 5.6, 5.7, or 8.0.
- The tables to be synchronized from the source database contain primary keys.
- The binary logging feature is enabled for the source database. A database account is created for the data synchronization task. For more information, see #unique_37.

  Note:
  The account must have the REPLICATION SLAVE permission, the REPLICATION CLIENT permission, the SHOW VIEW permission, and the permission to perform SELECT operations on the required objects.
- The on-premises network to which the user-created MySQL database belongs is connected to Alibaba Cloud VPC over Express Connect, VPN Gateway, or Smart Access Gateway. DTS is allowed to access the network to which Express Connect, VPN Gateway, or Smart Access Gateway belongs. For more information, see #unique_39.

  Note:
  For more information about how to connect a VPC to an on-premises data center, see #unique_40.
- An AnalyticDB for PostgreSQL instance is created. For more information, see Create an instance.

Precautions

DTS uses read and write resources of the source and destination databases during initial full data synchronization. This may increase the database load. If the database performanc e is unfavorable, the specification is low, or the data volume is large, database services may become unavailable. For example, DTS occupies a large amount of read and write resources in the following cases: a large number of slow SQL queries are performed on the

source database, the tables have no primary keys, or a deadlock occurs in the destination database. Before you synchronize data, evaluate the performance of the source and destination databases. We recommend that you synchronize data during off-peak hours. For example, you can synchronize data when the CPU usage of the source and destination databases is less than 30%.

Limits

- You can select only tables as the objects to be synchronized.
- You cannot synchronize the following types of data: BIT, VARBIT, GEOMETRY, ARRAY, UUID, TSQUERY, TSVECTOR, and TXID_SNAPSHOT.
- We recommend that you do not use gh-ost or pt-online-schema-change to perform DDL operations on objects during data synchronization. Otherwise, data synchronization may fail.

SQL operations that can be synchronized

- DML operations: INSERT, UPDATE, and DELETE
- DDL operations: ADD COLUMN and RENAME COLUMN

> **Note:**
> The CREATE TABLE operation is not supported. To synchronize data from a new table, you must add the table to the selected objects. For more information, see #unique_41.

Supported synchronization topologies

- One-way one-to-one synchronization
- One-way one-to-many synchronization
- One-way many-to-one synchronization

Term mappings

Term in MySQL	Term in AnalyticDB for PostgreSQL
Database	Schema
Table	Table

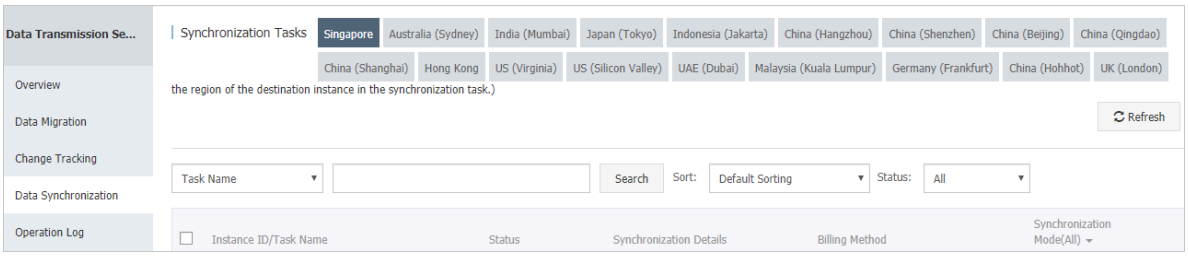
Procedure

1. Purchase a data synchronization instance. For more information, see #unique_31.

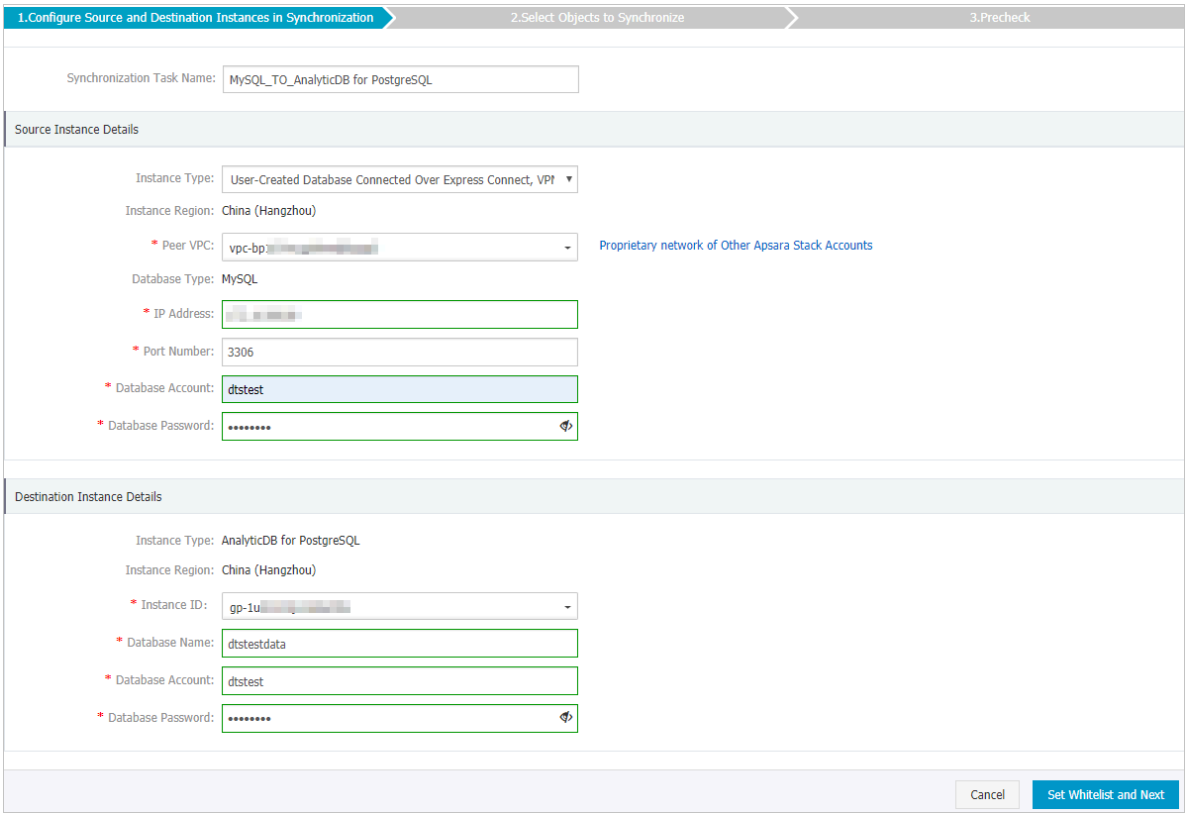
> **Note:**

On the buy page, set Source Instance to **MySQL**, set Target Instance to **AnalyticDB for PostgreSQL**, and set Synchronization Topology to **One-Way Synchronization**.

2. Log on to the DTS console.
3. In the left-side navigation pane, click **Data Synchronization**.
4. At the top of the **Synchronization Tasks** page, select the region where the destination instance resides.



5. Find the data synchronization instance and click **Configure Synchronization Channel** in the Actions column.
6. Configure the source and destination instances.



Section	Parameter	Description
N/A	Synchronization Task Name	DTS automatically generates a task name. We recommend that you specify an informative name for easy identification. You do not need to use a unique task name.

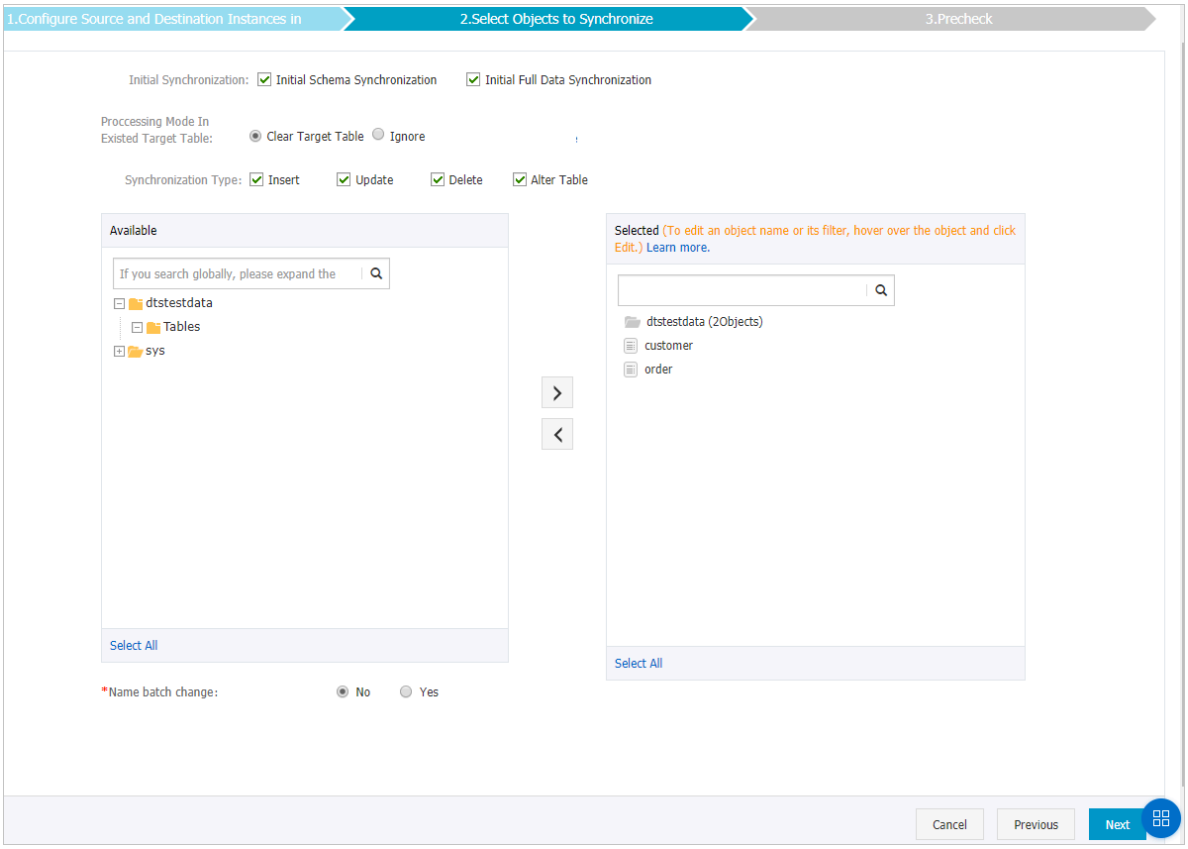
Section	Parameter	Description
Source Instance Details	Instance Type	Select **User-Created Database Connected over Express Connect, VPN Gateway, or Smart Access Gateway**.
	Instance Region	The region of the source instance. The region is the same as the source region that you selected when you purchased the data synchronization instance. You cannot change the value of this parameter.
	Peer VPC	Select the ID of the VPC that is connected to the user-created MySQL database.
	Database Type	The value of this parameter is set to **MySQL** and cannot be changed.
	IP Address	Enter the server IP address of the user-created MySQL database.
	Port Number	Enter the service port number of the user-created MySQL database. The default port number is **3306**.
	Database Account	Enter the account of the user-created MySQL database. **Note:** The account must have the REPLICATION SLAVE permission, the REPLICATION CLIENT permission, the SHOW VIEW permission, and the permission to perform SELECT operations on the required objects.
	Database Password	Enter the password for the source database account.
Destination Instance Details	Instance Type	The value of this parameter is set to **AnalyticDB for PostgreSQL** and cannot be changed.
	Instance Region	The region of the destination instance. The region is the same as the destination region that you selected when you purchased the data synchronization instance. You cannot change the value of this parameter.
	Instance ID	Select the ID of the destination AnalyticDB for PostgreSQL instance.
	Database Name	Enter the name of the destination database.

Section	Parameter	Description
	Database Account	Enter the database account of the destination AnalyticDB for PostgreSQL instance. **Note:** The account must have the SELECT, INSERT, UPDATE, DELETE, COPY, TRUNCATE, and ALTER TABLE permissions.
	Database Password	Enter the password for the destination database account.

7. In the lower-right corner of the page, click **Set Whitelist and Next**.

Note:
The CIDR blocks of DTS servers are automatically added to the whitelist of the destination AnalyticDB for PostgreSQL instance. This ensures that DTS servers can connect to the destination AnalyticDB for PostgreSQL instance.

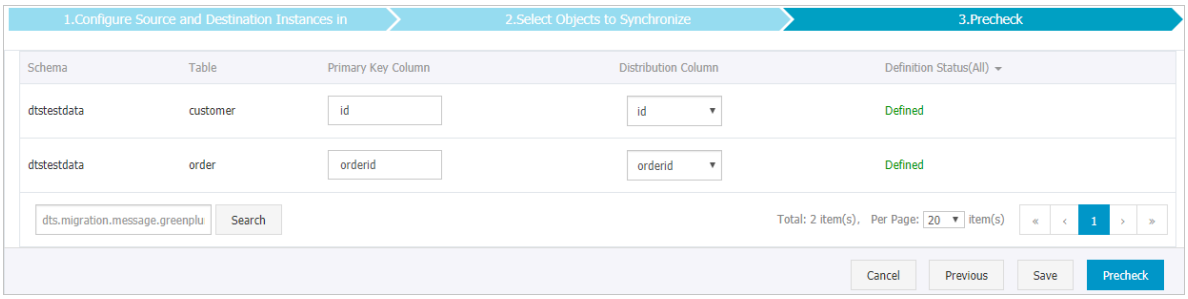
8. Configure the synchronization policy and objects.



Section	Parameter	Description
Synchronization policy	Initial Synchronization	You must select both **Initial Schema Synchronization** and **Initial Full Data Synchronization** in most cases. After the precheck, DTS synchronizes the schemas and data of the required objects from the source instance to the destination instance. The schemas and data are the basis for subsequent incremental synchronization.

Section	Parameter	Description
	Processing Mode In Existed Target Table	• **Clear Target Table** Skips the **Schema Name Conflict** item during the precheck. Clears the data in the destination table before initial full data synchronization. If you want to synchronize your business data after testing the data synchronization task, you can select this mode. • **Ignore** Skips the **Schema Name Conflict** item during the precheck. Adds data to the existing data during initial full data synchronization. If you want to synchronize data from multiple tables to one table, you can select this mode.
	Synchronization Type	Select the types of operations that you want to synchronize based on your business requirements. • **Insert** • **Update** • **Delete** • **Alter Table**
Objects to be synchronized	N/A	Select tables from the **Available** section and click the  icon to move the tables to the **Selected** section. **Note:** • You can select only tables as the objects to be synchronized. • You can change the names of the columns that are synchronized to the destination database by using the object name mapping feature. For more information about how to use this feature, see #unique_33.

9. Specify the primary key column and distribution column of the table that you want to synchronize to the AnalyticDB for PostgreSQL instance.



Note:

The page in this step appears only if you select **Initial Schema Synchronization**. For more information about primary key columns and distribution columns, see Table constraints and Partition keys.

10. In the lower-right corner of the page, click **Precheck**.

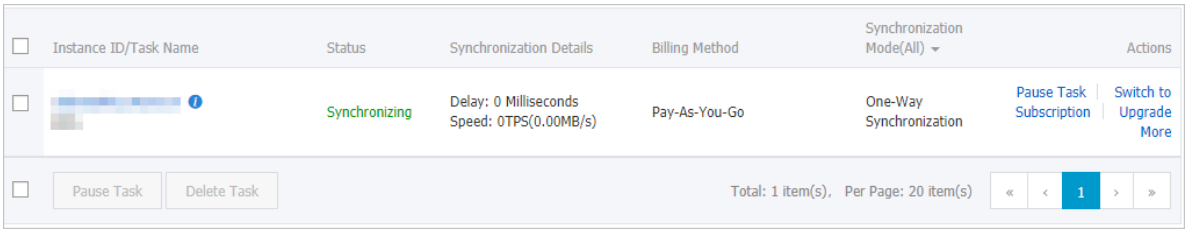
Note:

- Before you can start the data synchronization task, a precheck is performed. You can start the data synchronization task only after the task passes the precheck.
- If the task fails to pass the precheck, click the icon next to each failed item to view details. Troubleshoot the issues based on the causes and run the precheck again.

11. Close the **Precheck** dialog box after the following message is displayed: **The precheck is passed.** Then, the data synchronization task starts.

12. Wait until the initial synchronization is complete and the data synchronization task is in the **Synchronizing** state.

You can view the status of the data synchronization task on the **Synchronization Tasks** page.



6.6 Use rds_dbsync to migrate or synchronize data from a MySQL database to an AnalyticDB for PostgreSQL database

rds_dbsync

rds_dbsync is an open source tool used to migrate or synchronize data. You can use the mysql2pgsql function of this tool to migrate data from a MySQL database to an AnalyticDB for PostgreSQL database, Greenplum Database, PostgreSQL database, or PPAS database without having to store data. This tool connects to both the source MySQL database and the destination database, retrieves the data you want to export from the source MySQL database, and then 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 of some database tables.

Parameter configuration

Modify the my.cfg file and configure the connection information of both the source and destination databases.

- The connection information of the source MySQL database is as follows:

  Notice:
  You must have read permission on all user tables in the source MySQL database.

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

- The connection information of the destination database (such as PostgreSQL, PPAS, or AnalyticDB for PostgreSQL) is as follows:

  Notice:
  You must have write permission on the destination table in the destination database.

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

mysql2pgsql usage

The usage of mysql2pgsql is described as follows:

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

Parameters

- -l: optional. This parameter specifies a text file that contains tables whose data needs to be synchronized. If you do not specify this parameter, the data of all tables within the source database is synchronized. <tables_list_file> specifies the name of a file that contains a collection of tables to be synchronized and conditions for table queries. The content format is as follows:

  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 that only the DDL statement used to create the destination table is generated and data is not synchronized.
- -n: optional. This parameter indicates that the definitions of table partitions are not included in the DDL statement. It must be used with the -d parameter.
- -j: optional. This parameter indicates the number of threads concurrently used to synchronize data. If you do not specify this parameter, five concurrent threads are used.
- -s: optional. This parameter indicates the schema of the destination table. Set it to public.

Typical usage

Full database migration

Follow these steps:

1. Run the following command to obtain the DDL statement used to create a table in the destination database:

   ```
   ./mysql2pgsql -d
   ```
2. Use the DDL statement to create a table in the destination database, and then specify a distribution key for the table.

3. Run the following command to synchronize data:

```
./mysql2pgsql
```

This command migrates the data of all tables from the source MySQL database to the destination database. By default, five concurrent threads are used to concurrently read and import data.

Partial table migration

1. Create a file named tab_list.txt and add the following content to it:

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

2. Run the following command to synchronize data of the t1 and t2 tables (note that for the t2 table, only data that meets the "c1 > 138888" criterion is migrated):

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

Download and instructions

- To download the binary installation package of mysql2pgsql, click here.
- To view the instructions on source code compilation of mysql2pgsql, click here.

6.7 Use rds_dbsync to migrate or synchronize data from a PostgreSQL database to an AnalyticDB for PostgreSQL database

rds_dbsync is an open source tool. You can use the pgsql2pgsql function of this tool to migrate data of tables from an AnalyticDB for PostgreSQL database, Greenplum Database, PostgreSQL database, or PPAS database to another AnalyticDB for PostgreSQL database, Greenplum Database, PostgreSQL database, or PPAS database.

Features of pgsql2pgsql

pgsql2pgsql provides the following features:

- Full data migration from a PostgreSQL database, PPAS database, Greenplum Database, or AnalyticDB for PostgreSQL database to another PostgreSQL database, PPAS database , Greenplum Database, or AnalyticDB for PostgreSQL database
- Full and incremental data migration from a PostgreSQL or PPAS database (9.4 or later) to another PostgreSQL or PPAS database

Parameter configuration

Modify the my.cfg file and configure the connection information of both the source and destination databases.

- The connection information of the source database is as follows:

  Notice:
  In the connection information of the source database, we recommend that you set the user to the owner of the source database.

  ```
  [src.pgsql]
  connect_string = "host=192.168.1.1 dbname=test port=3432 user=test password=pgsql
  "
  ```

- The connection information of the local temporary database is as follows:

  ```
  [local.pgsql]
  connect_string = "host=192.168.1.2 dbname=test port=3432 user=test2 password=
  pgsql"
  ```

- The connection information of the destination database is as follows:

  Notice:
  You must have write permission on the destination table in the destination database.

  ```
  [desc.pgsql]
  connect_string = "host=192.168.1.3 dbname=test port=3432 user=test3 password=
  pgsql"
  ```

Notice:

- If you want to synchronize incremental data, you must have permission 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. The kernel supports logic flow replication only after you configure the following parameters:

  ```
  wal_level = logical
  max_wal_senders = 6
  max_replication_slots = 6
  ```

pgsql2pgsql usage

Full database migration

Run the following command to perform a full database migration:

```
./pgsql2pgsql
```

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

Status query

You can connect to the local temporary database and view the status of a migration task. 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.

Download and instructions

- To download the binary installation package of rds_dbsync, click here.
- To view the instructions on source code compilation of rds_dbsync, click here.