Relational Database Service

Best Practices

Issue	01
Date	2025-11-11

HUAWEI CLOUD COMPUTING TECHNOLOGIES CO., LTD.

Trademarks and Permissions


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

Notice

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

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

Contents

1 RDS for MySQL 1
1.1 Description of innodb_flush_log_at_trx_commit and sync_binlog 1
2 RDS for PostgreSQL 3
2.1 User-Defined Data Type Conversion 3
2.2 Best Practices for Using PoWA 4
2.2.1 Overview 4
2.2.2 Supported Performance Metrics 5
2.2.2.1 Database Performance Metrics 6
2.2.2.2 Instance Performance Metrics 8
2.2.3 PoWA Deployment 12
2.2.3.1 Deploying PoWA for an RDS for PostgreSQL Instance 12
2.2.3.2 Deploying PoWA on a Self-Managed PostgreSQL Instance 14
2.2.4 Viewing Metric Details on PoWA 17
3 RDS for SQL Server 21
3.1 Migrating Data from a Self-Managed SQL Server Database on an ECS to an RDS for SQL Server DB Instance 21
3.2 Modifying Parameters of RDS for SQL Server Instances 25
3.3 Using the Import and Export Function to Migrate Data from a Local Database to an RDS for SQL Server DB Instance 27
3.4 Shrinking an RDS for SQL Server Database 31

1 RDS for MySQL

1.1 Description of innodb_flush_log_at_trx_commit and sync_binlog

The **innodb_flush_log_at_trx_commit** and **sync_binlog** are key parameters for controlling the disk write policy and data security of RDS for MySQL. Different parameter values have different impacts on performance and security.

Table 1-1 Parameter description

Parameter	Allowed Values	Description
innodb_flush_log_at_trx_commit	0, 1, and 2	Controls the balance between strict ACID compliance for commit operations, and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. The default value is **1**. For details, see **Parameter Description**.
sync_binlog	0 to 4, 294, 967, 295	Sync binlog (RDS for MySQL flushes binary logs to disks or relies on the OS).

Parameter Description

- **innodb_flush_log_at_trx_commit**:
  - **0**: The log buffer is written out to the log file once per second and the flush to disk operation is performed on the log file, but nothing is done at a transaction commit.
  - **1**: The log buffer is written out to the log file at each transaction commit and the flush to disk operation is performed on the log file.

  - **2**: The log buffer is written out to the file at each commit, but the flush to disk operation is not performed on it. However, the flushing on the log file takes place once per second.

    NOTE

    - A value of **0** is the fastest choice but less secure. Any mysqld process crash can erase the last second of transactions.
    - A value of **1** is the safest choice because in the event of a crash you lose at most one statement or transaction from the binary log. However, it is also the slowest choice.
    - A value of **2** is faster and more secure than **0**. Only an operating system crash or a power outage can erase the last second of transactions.

- **sync_binlog=1 or N**

  By default, the binary log is not every time synchronized to disk. In the event of a crash, the last statement in the binary log may get lost.

  To prevent this issue, you can use the **sync_binlog** global variable (**1** is the safest value, but also the slowest) to synchronize the binary log to disk after N binary log commit groups.

Recommended Configurations

Table 1-2 Recommended configurations

innodb_flush_log_at_trx_commit	sync_binlog	Description
1	1	High data security and strong disk write capability
1	0	High data security and insufficient disk write capability. Standby lagging behind or no replication is allowed.
2	0/N (0 < N < 100)	Low data security. A small amount of transaction log loss and replication delay is allowed.
0	0	Limited disk write capability. No replication or long replication delay is allowed.

NOTE

- When both **innodb_flush_log_at_trx_commit** and **sync_binlog** are set to **1**, the security is the highest but the write performance is the lowest. In the event of a crash you lose at most one statement or transaction from the binary log. This is also the slowest choice due to the increased number of disk writes.
- When **sync_binlog** is set to *N* (*N* > 1) and **innodb_flush_log_at_trx_commit** is set to **2**, the RDS for MySQL write operation achieves the optimal performance.

2 RDS for PostgreSQL

2.1 User-Defined Data Type Conversion

Description

There are three data type conversion modes for PostgreSQL: implicit conversion, assignment conversion, and explicit conversion. They correspond to i (Implicit), a (Assignment), and e (Explicit) in the pg_cast system catalog.

- Implicit conversion: a conversion from low bytes to high bytes of the same data type, for example, from **int** to **bigint**
- Assignment conversion: a conversion from high bytes to low bytes of the same data type, for example, from **smallint** to **int**
- Explicit conversion: a conversion between different data types

How to Use

1. Before converting data types, you can run the following command to check whether RDS for PostgreSQL supports data type conversion:

```
select * from pg_catalog.pg_cast ;
oid  | castsource | casttarget | castfunc | castcontext | castmethod
-------+------------+------------+----------+-------------+------------
 11277 |         20 |         21 |      714 | a           | f
 11278 |         20 |         23 |      480 | a           | f
 11279 |         20 |        700 |      652 | i           | f
 11280 |         20 |        701 |      482 | i           | f
......
```

2. Run the following command to check whether **int4** can be converted to **text**:

```
select * from pg_catalog.pg_cast where castsource = 'int4'::regtype and casttarget = 'bool'::regtype;
  oid  | castsource | casttarget | castfunc | castcontext | castmethod
-------+------------+------------+----------+-------------+------------
 11311 |         23 |         16 |     2557 | e           | f
(1 row)
```

The conversion is supported, and the conversion type is implicit conversion.

If no built-in conversion functions are available, customize a conversion function to support the conversion. For details, see **User-Defined Data Type Conversion**.

User-Defined Data Type Conversion

- Use double colons (::) to perform a forcible conversion.

```
select '10'::int,'2023-10-05'::date;
 int4 |    date
------+------------
   10 | 2023-10-05
(1 row)
```

- Use the CAST function to convert the type.

```
select CAST('10' as int),CAST('2023-10-05' as date);
 int4 |    date
------+------------
   10 | 2023-10-05
(1 row)
```

- Customize a data type conversion.

  For details, see **https://www.postgresql.org/docs/14/sql-createcast.html**.

  NOTICE

  Adding a custom type conversion will affect the existing execution plans of RDS for PostgreSQL. Therefore, customizing type conversions are not recommended.

  – Conversion between time and character types

    CREATE CAST(varchar as date) WITH INOUT AS IMPLICIT;

  – Conversion between boolean types and numeric types

    create cast(boolean as numeric) with INOUT AS IMPLICIT;

  – Conversion between numeric types and character types

    create cast(varchar as numeric) with INOUT AS IMPLICIT;

  Example: Convert **text** to **date**.

```
create or replace function public.text_to_date(text) returns date as
$$
  select to_date($1,'yyyy-mm-dd');
$$
language sql strict;

create cast (text as date) with function public.text_to_date(text) as implicit;

select text '2023-09-09' + 1;
  ?column?
------------
 2023-09-10
(1 row)
```

2.2 Best Practices for Using PoWA

2.2.1 Overview

PoWA is an open-source system used to monitor the performance of RDS for PostgreSQL databases. It consists of the PoWA-archivist, PoWA-collector, and PoWA-web components and obtains performance data through other extensions installed in the RDS for PostgreSQL databases. The key components are as follows:

- PoWA-archivist: the PostgreSQL extension for collecting performance data obtained by other extensions.
- PoWA-collector: the daemon that gathers performance metrics from remote PostgreSQL instances on a dedicated repository server.
- PoWA-web: the web-based user interface displaying performance metrics collected by the PoWA-collector.
- Other extensions: the sources of performance metric data. They are installed on the target PostgreSQL database.
- PoWA: the system name.

Security Risk Warning

The following security risks may exist during PoWA deployment and configuration.

- (Remote mode) When configuring instance performance metric information to be collected in powa-repository, you need to enter the IP address, **root** username, and connection password of the target instance. You can query related information in the **powa_servers** table. The connection password is displayed in plaintext.
- In the **PoWA-collector** configuration file, the powa-repository connection information does not contain the connection password. It means that the powa-repository connection configuration item for PoWA-collector must be trust.
- In the **PoWA-web** configuration file, the **root** username and connection password of powa-repository (remote mode) or DB instance (local mode) are optional and stored in plaintext.

Before using the PoWA, you need to be aware of the preceding security risks. For details about how to harden security, see the **official PoWA documentation**.

Other Supported Extensions

In addition to mandatory extensions pg_stat_statements, btree_gist, and PoWA, the following extensions are used for collecting extended performance metrics:

- pg_qualstats
- pg_stat_kcache
- pg_wait_sampling
- pg_track_settings
- hypopg

Each of the extensions can collect different extended performance metrics.

2.2.2 Supported Performance Metrics

2.2.2.1 Database Performance Metrics

General Overview

Figure 2-1 General Overview

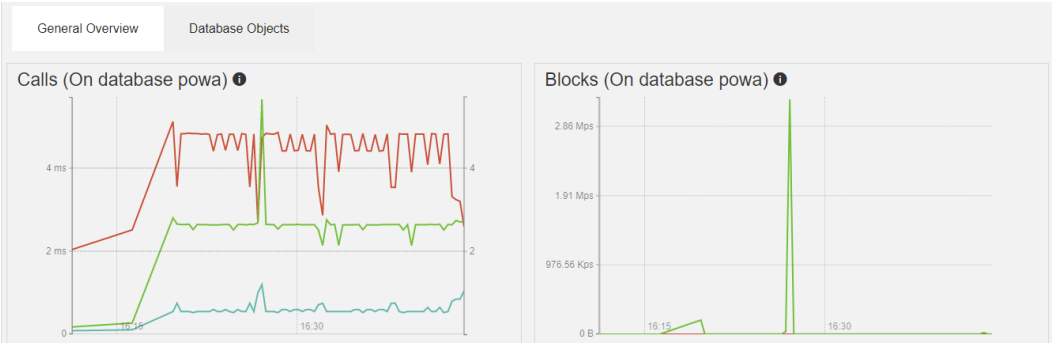


Table 2-1 Calls field description

Field	Description
Queries per sec	Number of queries executed per second
Runtime per sec	Total duration of queries executed per second
Avg runtime	Average query duration

Table 2-2 Blocks field description

Field	Description
Total shared buffers hit	Amount of data found in shared buffers
Total shared buffers miss	Amount of data found in OS cache or read from disk

Database Objects

Figure 2-2 Database Objects

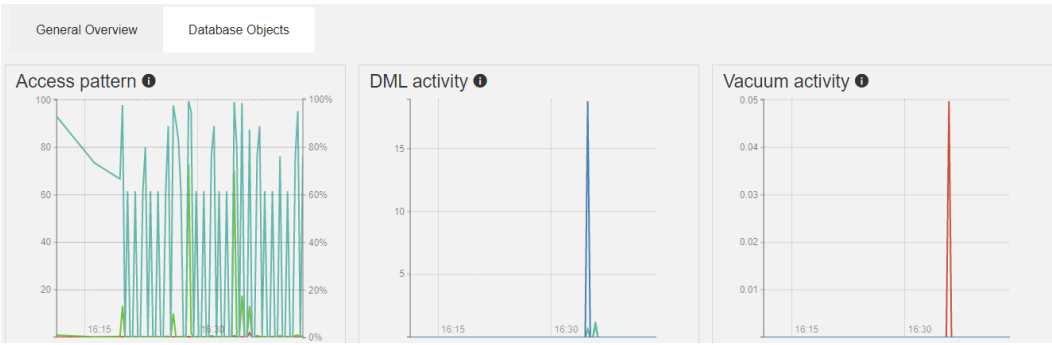


Table 2-3 Access pattern field description

Field	Description
Index scans ratio	Ratio of index scans to sequential scans
Index scans	Number of index scans per second
Sequential scans	Number of sequential scans per second

Table 2-4 DML activity field description

Field	Description
Tuples inserted	Number of tuples inserted per second
Tuples updated	Number of tuples updated per second
Tuples HOT updated	Number of heap-only tuples (HOT) updated per second
Tuples deleted	Number of tuples deleted per second

Table 2-5 Vacuum activity field description

Field	Description
# Vacuum	Number of vacuums per second
# Autovacuum	Number of autovacuums per second
# Analyze	Number of analyses per second
# Autoanalyze	Number of autoanalyses per second

Details for all databases

Figure 2-3 Details for all databases

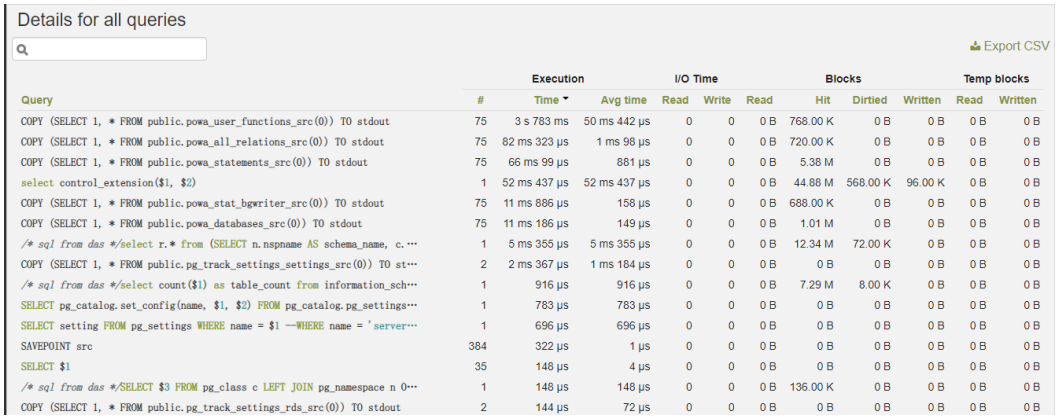


Table 2-6 Details for all databases field description

Field	Description
Query	SQL statement to be executed
(Execution) #	Number of times that the SQL statement is executed
(Execution) Time	Total execution time of the SQL statement
(Execution) Avg time	Average time for executing the SQL statement
(I/O Time) Read	Read I/O wait time
(I/O Time) Write	Write I/O wait time
(Blocks) Read	Number of disk read pages
(Blocks) Hit	Number of hit pages in the shared buffer
(Blocks) Dirtied	Number of dirty pages
(Blocks) Written	Number of disk write pages
(Temp blocks) Read	Number of disk temporary read pages
(Temp blocks) Write	Number of disk temporary write pages

2.2.2.2 Instance Performance Metrics

General Overview

Figure 2-4 General Overview metrics

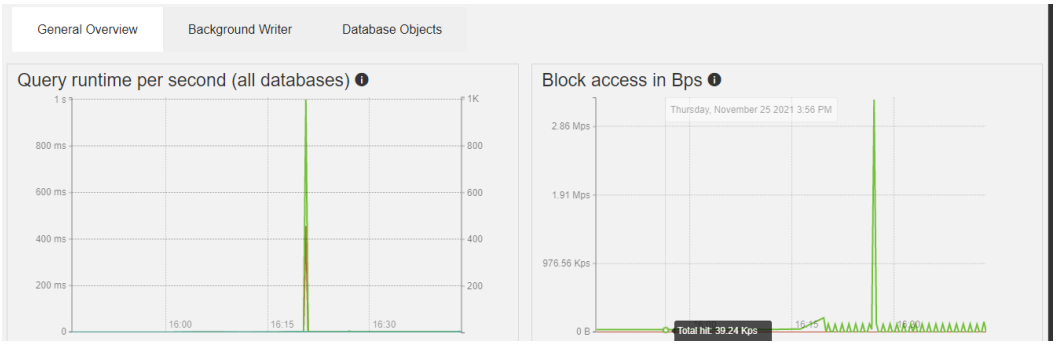


Table 2-7 Query runtime per second (all databases) field description

Field	Description
Queries per sec	Number of queries executed per second
Runtime per sec	Total duration of queries executed per second
Avg runtime	Average query duration

Table 2-8 Block access in Bps field description

Field	Description
Total hit	Amount of data found in shared buffers
Total read	Amount of data found in OS cache or read from disk

Background Writer

Figure 2-5 Background Writer metrics

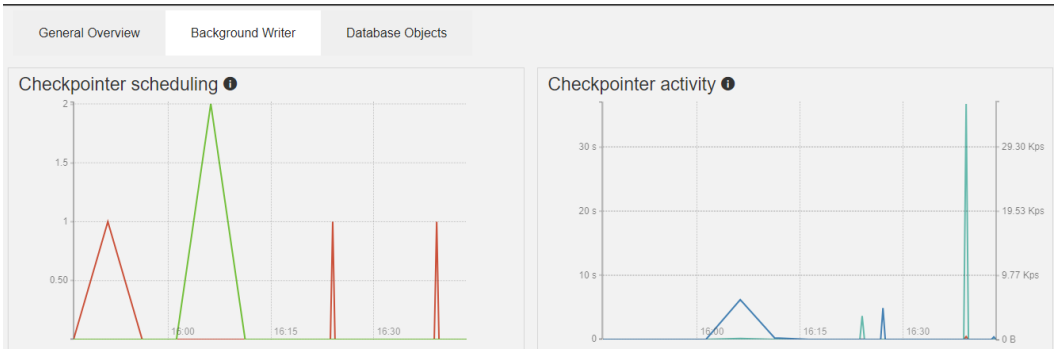


Table 2-9 Checkpointer scheduling field description

Field	Description
of requested checkpoints	Number of requested checkpoints that have been performed
of scheduled checkpoints	Number of scheduled checkpoints that have been performed

Table 2-10 Checkpointer activity field description

Field	Description
Buffers alloc	Number of buffers allocated
Sync time	Total amount of time that has been spent in the portion of checkpoint processing where files are synchronized to disk, in milliseconds
Write time	Total amount of time that has been spent in the portion of checkpoint processing where files are written to disk, in milliseconds

Table 2-11 Background writer field description

Field	Description
Maxwritten clean	Number of times the background writer stopped a cleaning scan because it had written too many buffers
Buffers clean	Number of buffers written by the background writer

Table 2-12 Backends field description

Field	Description
Buffers backend fsync	Number of times a backend had to execute its own fsync call (normally the background writer handles those even when the backend does its own write)
Buffers backend	Number of buffers written directly by a backend

Database Objects

Figure 2-6 Database Objects metrics

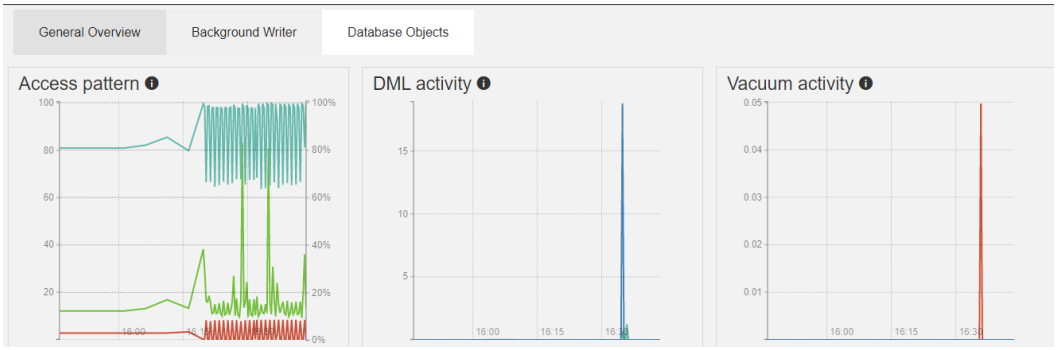


Table 2-13 Access pattern field description

Field	Description
Index scans ratio	Ratio of index scans to sequential scans
Index scans	Number of index scans per second
Sequential scans	Number of sequential scans per second

Table 2-14 DML activity field description

Field	Description
Tuples inserted	Number of tuples inserted per second
Tuples updated	Number of tuples updated per second
Tuples HOT updated	Number of heap-only tuples (HOT) updated per second
Tuples deleted	Number of tuples deleted per second

Table 2-15 Vacuum activity field description

Field	Description
# Vacuum	Number of vacuums per second
# Autovacuum	Number of autovacuums per second
# Analyze	Number of analyses per second
# Autoanalyze	Number of autoanalyses per second

Details for all databases

Figure 2-7 Details for all databases metrics

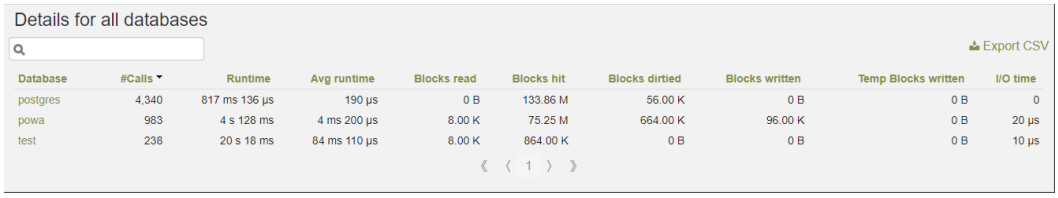


Table 2-16

Field	Description
Database	Database name
#Calls	Total number of executed SQL statements
Runtime	Total runtime of the SQL statement
Avg runtime	Average runtime of the SQL statement
Blocks read	Number of pages read from the disk
Blocks hit	Number of hit pages in the shared buffer
Blocks dirtied	Number of dirty pages

Field	Description
Blocks written	Number of disk write pages
Temp Blocks written	Number of disk temporary write pages
I/O time	I/O wait time

2.2.3 PoWA Deployment

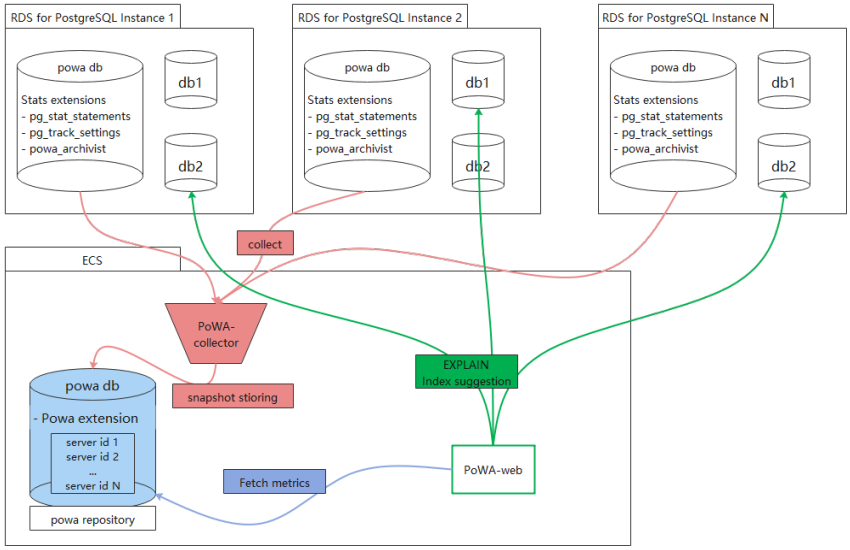
2.2.3.1 Deploying PoWA for an RDS for PostgreSQL Instance

To remotely deploy PoWA, there must be an ECS with PoWA-archivist, PoWA-collector, and PoWA-web installed on it. This section describes how to install PoWA-archivist, PoWA-collector, and PoWA-web.

Architecture

The remote deployment architecture is as follows:

Figure 2-8 Remote deployment architecture



Preparations

- An RDS for PostgreSQL 12.6 instance has been created.
- An ECS has been created and associated with an EIP. In this example, the ECS uses the CentOS 8.2 64-bit image.

Installing Python3

PoWA-collector and PoWA-web must be installed in a Python3 environment. You can use pip3 to install them to facilitate the installation. In this example, Python 3.6.8 is installed on the ECS by default. The latest PoWA version fails to be

installed. For details about how to install the latest version, see **Installing Python 3.9.9**.

Installing PoWA-archivist

1. Run the **wget** command to obtain the PoWA-archivist source code.
   ```
   wget https://github.com/powa-team/powa-archivist/archive/refs/tags/REL_4_1_2.tar.gz
   ```
2. Decompress the downloaded **REL_4_1_2.tar.gz** package.
3. Install PoWA-archivist to the decompressed directory.
   ```
   make && make install
   ```

Installing PoWA-collector and PoWA-web

1. Switch to the RDS for PostgreSQL database user. Take user **postgres** as an example.
   ```
   su - postgres
   ```
2. Install PoWA-collector and PoWA-web. psycopg2 is mandatory for installing them.
   ```
   pip install psycopg2
   pip install powa-collector
   pip install powa-web
   ```

After the installation is complete, check the following path tree. If the following information is displayed, the PoWA-collector and PoWA-web have been installed.

```
/home/postgres/.local/bin
├── powa-collector.py
├── powa-web
└── __pycache__
```

Creating the PoWA Extension

Step 1 Log in to the powa database of the RDS for PostgreSQL instance as user **root**. (If the powa database does not exist, create it first.)

Step 2 Create the PoWA extension in the powa database.

```
select control_extension('create', 'pg_stat_statements');
select control_extension('create', 'btree_gist');
select control_extension('create', 'powa');
```

----End

FAQs

Q: What should I do if the error message "python setup.py build_ext --pg-config / path/to/pg_config build" is displayed when the **pip install psycopg2** command is executed?

A: You need to add the **bin** and **lib** paths of RDS for PostgreSQL to environment variables and run the **pip install psycopg2** command.

Installing Python 3.9.9

1. Prepare the environment.

   Perform the following operations in sequence. Otherwise, Python 3.9.9 may fail to be installed (for example, SSL component dependency fails). As a result, PoWA-collector and PoWA-web fail to be installed.


```
yum install readline* -y
yum install zlib* -y
yum install gcc-c++ -y
yum install sqlite* -y
yum install openssl-* -y
yum install libffi* -y
```

2. Install Python 3.9.9.

   a. Run the following commands as user **root**:

   ```
   mkdir env
   cd env
   wget https://www.python.org/ftp/python/3.9.9/Python-3.9.9.tgz
   tar -xzvf Python-3.9.9.tgz
   cd Python-3.9.9
   ./configure --prefix=/usr/local/python3.9.9
   make && make install
   ```

   b. Create a soft link.

   ```
   ln -s /usr/local/python3.9.9/bin/python3.9 /usr/bin/python
   ln -s /usr/local/python3.9.9/bin/pip3.9 /usr/bin/pip
   ```

3. Check whether the installation is successful.

   a. Verify the installation, especially the SSL function.

   ```
   [root@ecs-ad4d Python-3.9.9]# python
   Python 3.9.9 (main, Nov 25 2021, 12:36:32)
   [GCC 8.4.1 20200928 (Red Hat 8.4.1-1)] on linux
   Type "help", "copyright", "credits" or "license" for more information.
   import ssl
   import urllib.request
   context = ssl._create_unverified_context()
   urllib.request.urlopen('https://www.example.com/',context=context).read()
   ```

   b. If any command output is displayed, the installation is successful. Run the following command to exit:

   ```
   quit()
   ```

2.2.3.2 Deploying PoWA on a Self-Managed PostgreSQL Instance

This section describes how to deploy PoWA on a self-managed PostgreSQL database built on an ECS.

Preparations

There is a self-managed PostgreSQL instance:

- Version: PostgreSQL 12.6
- Administrator account: **postgres**
- PostgreSQL dedicated storage database: powa-repository
- Data path: **/home/postgres/data**

Deploying PoWA

Step 1 Add **pg_stat_statements** to **shared_preload_libraries** in the **/home/postgres/data/postgresql.conf** file.

```
shared_preload_libraries = 'pg_stat_statements'         # (change requires restart)
```

Step 2 Restart the database.

```
pg_ctl restart -D /home/postgres/data/
```


Step 3 Log in to the database as the **postgres** user, create database **powa**, and install related extensions.

NOTICE

The created database must be named **powa**. Otherwise, an error is reported and certain functions do not take effect while PoWA is running.

```
[postgres@ecs-ad4d ~]$ psql -U postgres -d postgres
psql (12.6)
Type "help" for help.
postgres=# create database powa;
CREATE DATABASE
postgres=# \c powa
You are now connected to database "powa" as user "postgres".
powa=# create extension pg_stat_statements ;
CREATE EXTENSION
powa=# create extension btree_gist ;
CREATE EXTENSION
powa=# create extension powa;
CREATE EXTENSION
```

Step 4 Configure the instance whose performance metrics need to be collected.

1. Add the instance information.

```
powa=# select powa_register_server(
    hostname => '192.168.0.1',
    alias => 'myInstance',
    port => 5432,
    username => 'user1',
    password => '**********',
    frequency => 300);
     powa_register_server
    ----------------------
     t
    (1 row)
```

2. Obtain the information about the target instance from the **powa_servers** table.

```
powa=# select * from powa_servers;
id | hostname | alias | port | username |  password  | dbname | frequency | powa_coalesce | retention |
allow_ui_connection |version
----+---------------+------------+------+----------+------------+--------+-----------+---------------+-----------
+------------------
0 |           | <local>|  0 |          |            |        |        -1 |   100         | 00:00:00  | t           |
1 | 192.168.0.1 | myInstance | user1 | 5432     | ********** | powa   |       300 |           100 | 1 day       |
t             |
(2 rows)
```

NOTICE

The preceding operations involve important privacy information such as the IP address, **root** account, and plaintext password of the target instance.

Before using this extension, assess its security risks.

----End

Configuring PoWA-collector

Start the PoWA-collector.


```
cd /home/postgres/.local/bin
./powa-collector.py &
```

When PoWA-collector starts, it searches for configuration files in the following sequence:

1. /etc/powa-collector.conf
2. ~/.config/powa-collector.conf
3. ~/.powa-collector.conf
4. ./powa-collector.conf

The configuration files must contain the following options:

- **repository.dsn**: URL. It is used to notify PoWA-collector of how to connect to the dedicated storage database (powa-repository).
- **debug**: Boolean type. It specifies whether to enable PoWA-collector in debugging mode.

Take the configuration file **./powa-collector.conf** as an example.

```
{
   "repository": {
   "dsn": "postgresql://postgres@localhost:5432/powa"
   },
   "debug": true
}
```

No password is configured in the PoWA-collector configuration. Therefore, you need to set the connection policy in the **pg_hba.conf** file of the **powa-repository** database to **trust** (password-free connection).

Configuring PoWA-web

Start the PoWA-web.

```
cd /home/postgres/.local/bin
./powa-web &
```

When PoWA-web starts, it searches for configuration files in the following sequence:

1. /etc/powa-web.conf
2. ~/.config/powa-web.conf
3. ~/.powa-web.conf
4. ./powa-web.conf

Take the configuration file **./powa-web.conf** as an example.

```
# cd /home/postgres/.local/bin
# vim ./powa-web.conf
# Write the configuration information and save it.
servers={
     'main': {
       'host': 'localhost',
       'port': '5432',
       'database': 'powa',
       'username': 'postgres',
       'query': {'client_encoding': 'utf8'}
     }
}
cookie_secret="SECRET_STRING"
```


In this section, the connection policy in the **pg_hab.conf** file of the powa-repository database is set to **trust** (password-free connection). Therefore, the password is not configured.

2.2.4 Viewing Metric Details on PoWA

After PoWA is deployed and PoWA-collector and PoWA-web are started, you can log in to PoWA using a browser to view metric details of the monitored instances.

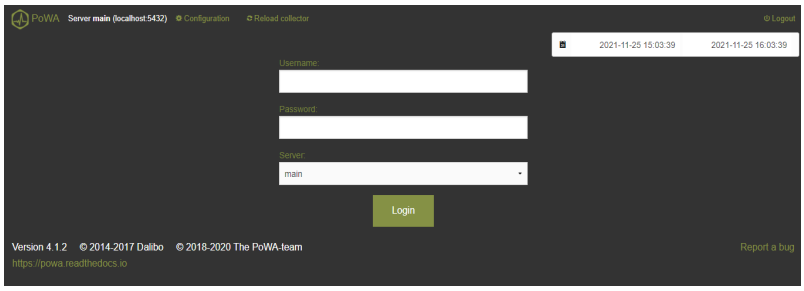
Accessing PoWA

Step 1 Use a browser to access PoWA.

NOTE

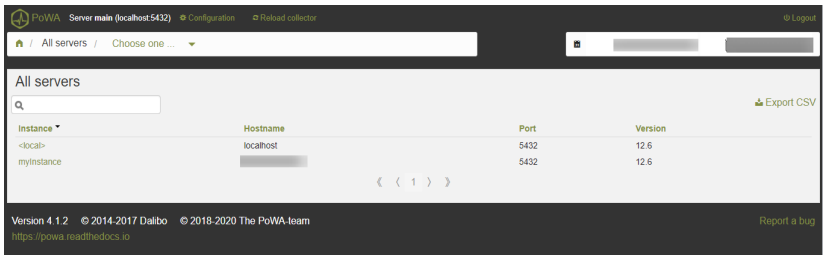
- There is no port option in the **powa-web.conf** file. The default value **8888** is used.
- URL: **http://***ECS_IP_address*:**8888/**

Figure 2-9 Accessing PoWA



Step 2 Enter the username and password, and click **Login**.

Figure 2-10 PoWA home page



In this example, PoWA collects information about two PostgreSQL instances.

- **<local>**: PostgreSQL database built on an ECS, which is used as the powa-repository database.
- **myinstance**: RDS for PostgreSQL instance, which is used as the target for performance data collection. (**myinstance** is the instance alias registered in powa-repository.)

Step 3 Click an instance to view its performance metrics.

----End

Viewing Metric Details

PoWA can collect and display various performance metrics. The following describes how to view the slow SQL metric.

Step 1 Log in to the management console.

Step 2 Click ☰ in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 3 On the **Instances** page, locate the target DB instance and click **Log In** in the **Operation** column.

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

Step 4 On the displayed login page, enter the username and password and click **Log In**.

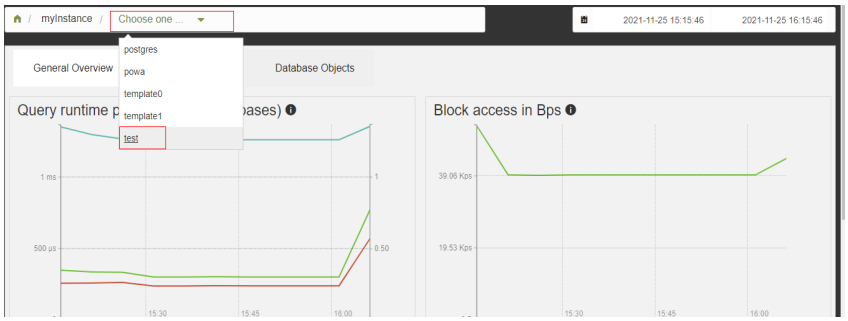
Step 5 In the database list of the **Home** page, click **Create Database**.

Step 6 On the displayed page, enter a database name (for example, **test**) and select a character set.

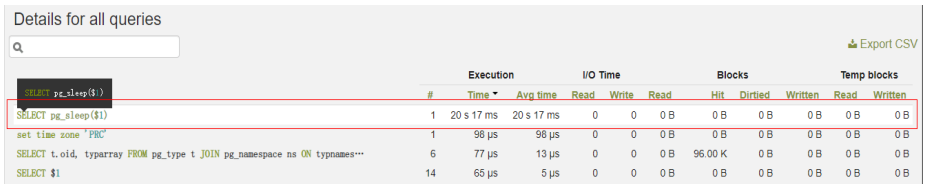
Step 7 Choose **SQL Operations** > **SQL Query** to execute a slow SQL statement, for example, **SELECT pg_sleep($1)**, in the **test** database.

Step 8 After about 5 minutes, on the PoWA home page, select the target DB instance and select the **test** database.

Figure 2-11 PoWA home page



In **Details for all queries**, check that the execution time of the **SELECT pg_sleep($1)** statement is 20s.



----End

Installing Other Extensions to Collect Performance Metrics

The following steps take pg_track_settings as an example.

Step 1 Log in to the management console.

Step 2 Click ☰ in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 3 On the **Instances** page, locate the target DB instance and click **Log In** in the **Operation** column.

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

Step 4 On the displayed login page, enter the username and password and click **Log In**.

Step 5 Select the **powa** database and run the following SQL command to create pg_track_settings:

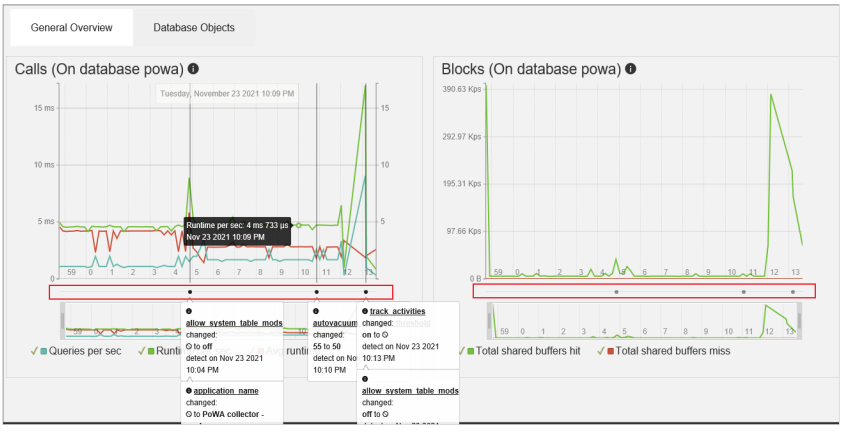
```
select control_extension('create', 'pg_track_settings');
```

Step 6 Create a PostgreSQL database (powa-repository) on the ECS, and install and activate pg_track_settings to collect performance metrics.

```
# pg_track_settings
cd /home/postgres/env
wget https://github.com/rjuju/pg_track_settings/archive/refs/tags/2.0.1.tar.gz
mv 2.0.1.tar.gz pg_track_settings.2.0.1.tar.gz
tar -xzvf pg_track_settings.2.0.1.tar.gz
cd pg_track_settings-2.0.1
make && make install
# powa-repository
psql -d powa
powa=# create extension pg_track_settings ;
CREATE EXTENSION
# Activate pg_track_settings for the target instance.
dbpowa=# select powa_activate_extension(1, 'pg_track_settings');
powa_activate_extension
-------------------------
t
(1 row)
```

Step 7 Verify the pg_track_settings extension.

Change the value of the **autovacuum_analyze_threshold** parameter on the target instance to **55**. The default value is **50**. After about 5 minutes, you can view the modification record on the PoWA page, as shown in the following figure:



The contents in the three boxes in the preceding figure include:

- The time when pg_track_settings was activated and the database parameter value at that time.

- The time when the **autovacuum_analyze_threshold** parameter was modified, its original value, and new value.
- The time when pg_track_settings was canceled and the database parameter value at that time.

----End

3 RDS for SQL Server

3.1 Migrating Data from a Self-Managed SQL Server Database on an ECS to an RDS for SQL Server DB Instance

Scenarios

- You have created a Microsoft SQL Server database on an ECS.
- The self-managed SQL Server database version on the ECS cannot be later than the version of the RDS for SQL Server DB instance.
- You have installed the SQL Server Management Studio (SSMS).

Procedure

Step 1 Create an ECS.

NOTE

The ECS and the RDS DB instance must be in the same region and VPC.

Step 2 Install Microsoft SQL Server 2008, 2012, or 2014 on the ECS.

NOTE

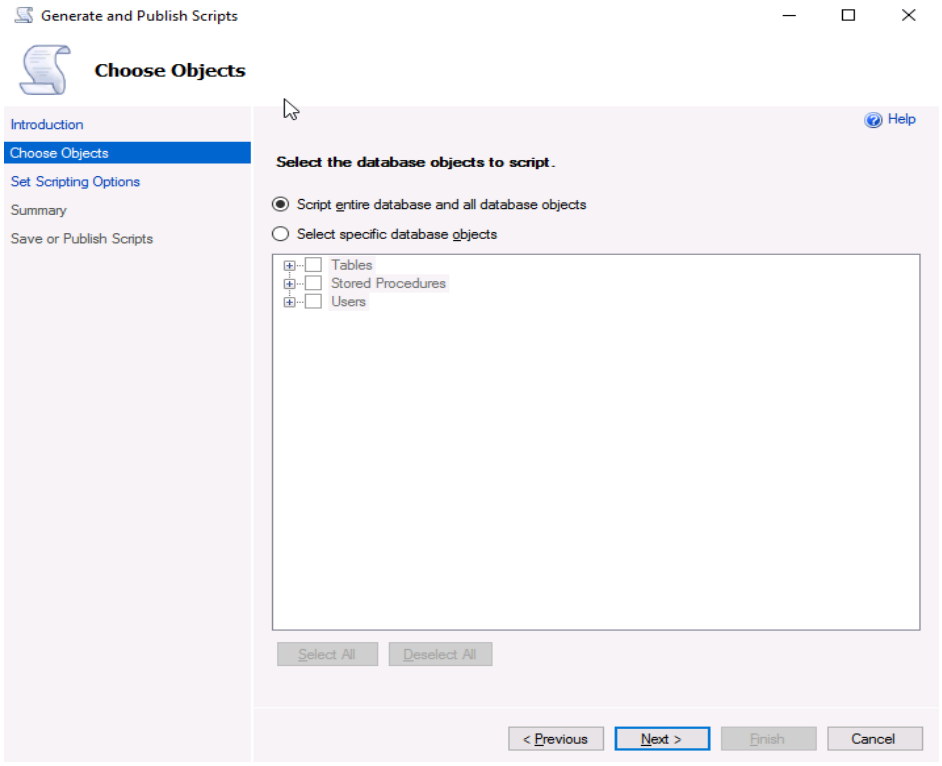
The Microsoft SQL Server installed on the ECS must be Standard or Enterprise Edition. It is recommended that the Microsoft SQL Server version be the same as the RDS DB instance version.

Step 3 Upload a local .bak file to the ECS and use Microsoft SQL Server to restore the local file to the RDS DB instance.

Step 4 Use the script generation tool provided by Microsoft SQL Server to generate a database structure script.

1. Right-click the database whose schema script needs to be generated and choose **Tasks** > **Generate Scripts**.
2. On the **Choose Objects** page, choose database objects to script, as shown in **Figure 3-1**. Then, click **Next**.

Figure 3-1 Choosing objects

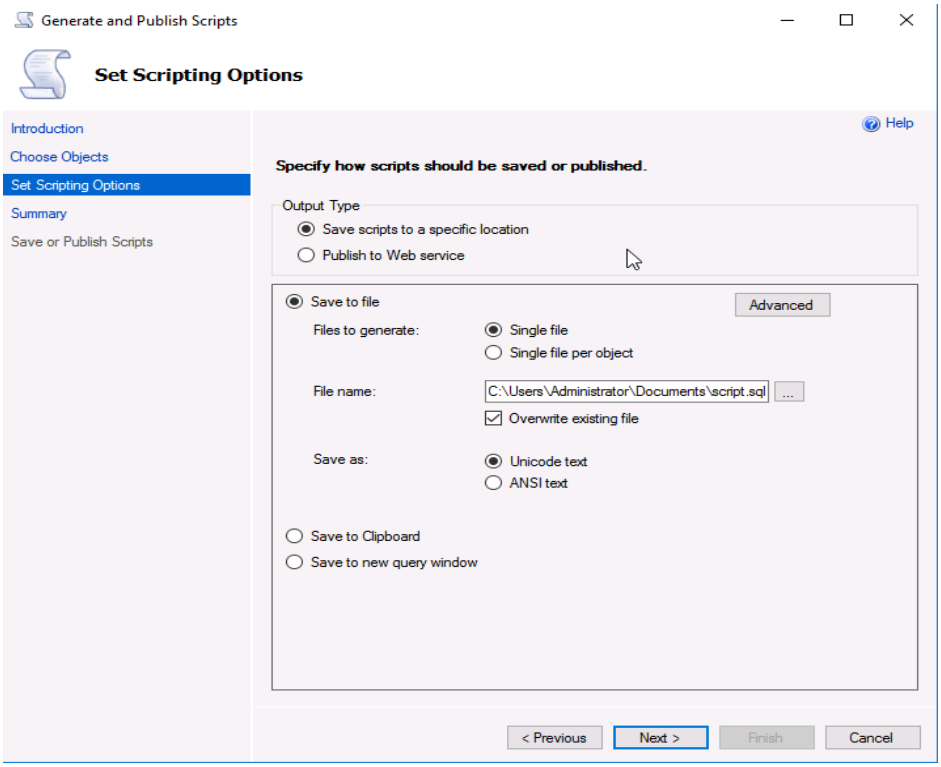


3. On the **Set Scripting Options** page, specify a directory for saving the script.

   NOTE

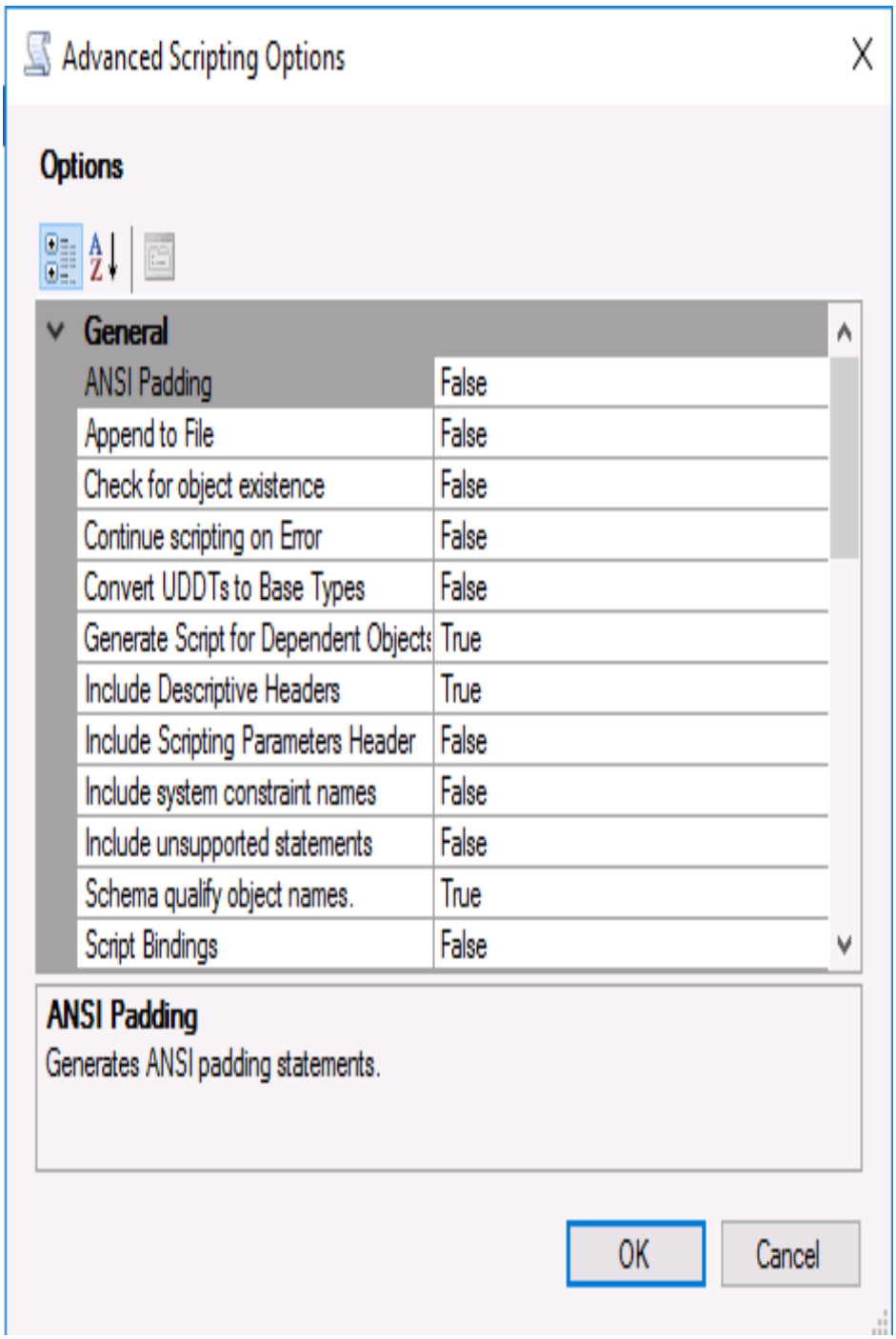
   You are advised to save the script locally and generate an SQL script for execution.

Figure 3-2 Specifying a directory for saving the script



4. Click **Advanced**. In the displayed **Advanced Scripting Options** dialog box, specify scripting options for triggers, indexes, unique keys, the primary key, and server version. Then, click **OK**.

Figure 3-3 Specifying advanced scripting options



NOTE

Generate Script for Dependent Objects indicates the script data type option.

5. Click **Next** to generate the script.

Step 5 Use the SSMS client to connect to the RDS DB instance and open the generated SQL script.

NOTE

You need to create an empty database, and then use the script to create structures in the database.

Step 6 Use the import and export function provided by Microsoft SQL Server to migrate data.

1. Right-click the database where data is to be imported and choose **Tasks** > **Import Data**.
2. Click **Next**.
3. On the **Choose a Data Source** page, select a data source and click **Next**.
4. On the **Choose a Destination** page, select a destination database and click **Next**.
   - **Destination**: Select **SQL Server Native Client** (depending on your destination database type).
   - **Server name**: Enter the IP address and port number of the destination DB instance.
   - **Authentication**: Select **Use SQL Server Authentication**. Then, set **User name** to **rdsuser**, and **Password** to the password of **rdsuser**.
   - **Database**: Select the destination database where data is to be imported.
5. Select **Copy data from one or more tables or views** and click **Next**.
6. On the **Select Source Tables and Views** page, select the tables and views that you want to copy. Then, click **Edit Mappings**. In the displayed dialog box, select **Enable identity insert** and edit mappings based on your requirements.
7. Click **Next**.
8. Select **Run immediately** and click **Next**.
9. Click **Finish** to import data. You can view the progress. About 4,000 rows can be processed per second.

----End

3.2 Modifying Parameters of RDS for SQL Server Instances

You can modify parameters in custom parameter templates.

Each DB instance is assigned with a group of parameters when it is being created and modifications to these parameters do not affect other DB instances.

Procedure

Step 1 Log in to the management console.

Step 2 Click  in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 3 On the **Instances** page, click the target DB instance.

Step 4 On the **Parameters** page, modify the parameters as required.

NOTE

- You cannot modify parameters in a default parameter template.
  - Each Microsoft SQL Server version has a unique default parameter template.
  - To apply a default parameter template to the current DB instance, choose **Parameter Templates** page in the navigation pane on the left, locate the target template on the **Default Templates** page, and click **Apply** in the **Operation** column.
- You can modify parameters in a custom template.
  - To create a custom template, choose **Parameter Templates** page in the navigation pane on the left, click **Create Parameter Template** on the **Custom Templates** page, and configure required information in the displayed dialog box. Then, click **OK**.
  - After you save modifications to the parameters in the custom template, you can apply this parameter template to multiple DB instances running corresponding versions.

You can modify the parameters listed in **Table 3-1** to improve DB instance performance.

Table 3-1 Parameters

Parameter	Description	Application Scenario
max degree of parallelism	Specifies the maximum degree of parallelism option. When an RDS for SQL Server DB instance runs on a computer with more than one microprocessor or CPU, RDS for SQL Server detects the best degree of parallelism (the number of processors used to run a single statement) for each parallel plan execution. The default value is **0**.	• If the DB instance is used for querying results, set the parameter to **0**. • If the DB instance is used for operations such as inserting, updating, and deleting, set the parameter to **1**.
max server memory (MB)	Specifies the server memory option. It is used to reconfigure the amount of memory (in MB) in the buffer pool used by a Microsoft SQL Server DB instance.	You are advised to retain the default value for this parameter. If you want to modify this parameter, the value of this parameter must be: • No less than 2 GB. • Not greater than 95% of the maximum memory of the DB instance.

Parameter	Description	Application Scenario
user connections	Specifies the maximum number of simultaneous user connections allowed on Microsoft SQL Server. Default value: **1000**	• If this parameter is set to **0**, the number of connections to the DB instance is not limited. • Allowed values: Values excluding 1 to 10

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

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

----End

3.3 Using the Import and Export Function to Migrate Data from a Local Database to an RDS for SQL Server DB Instance

Scenarios

- You have created a local Microsoft SQL Server database.
- The local database version cannot be later than the version of the destination RDS for SQL Server DB instance.
- You want to migrate only tables instead of the whole database.

Procedure

Step 1 Log in to the management console.

Step 2 Click ☰ in the upper left corner of the page and choose **Databases** > **Relational Database Service**.

Step 3 On the **Instances** page, click the instance name.

Step 4 In the navigation pane on the left, choose **Connectivity & Security**.

Step 5 In the **Connection Information** area, click **Bind** next to the **EIP** field.

Step 6 In the displayed dialog box, select an EIP and click **Yes**.

Step 7 Install the SSMS client locally and use the EIP to connect to the RDS DB instance.

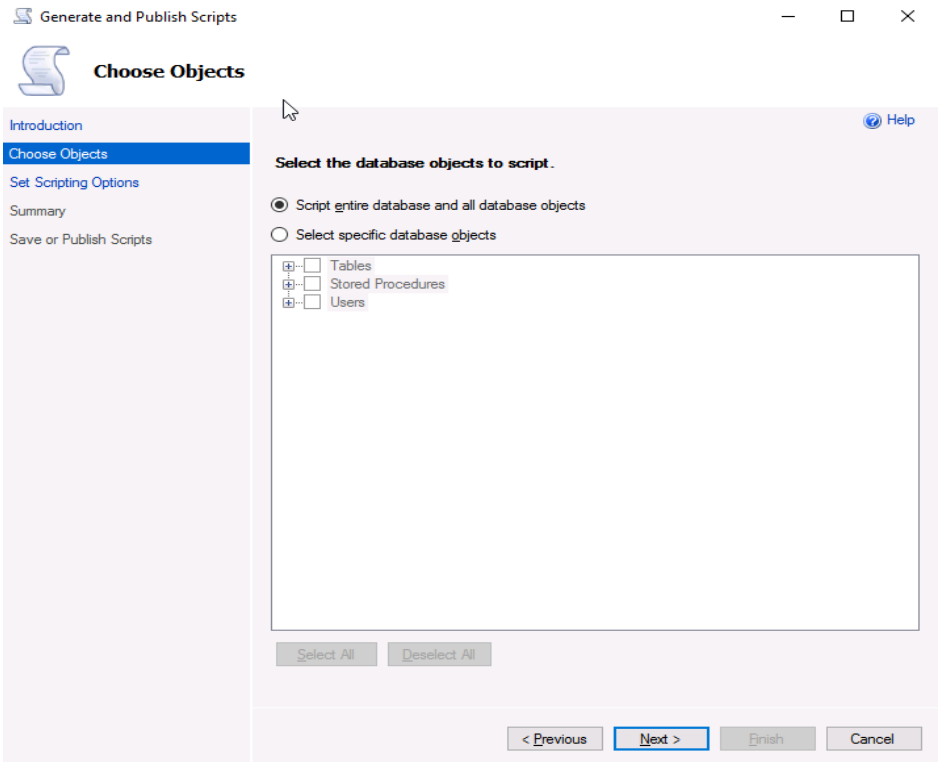
NOTE

Click **here** to download the SSMS client.

Step 8 Use the script generation tool provided by Microsoft SQL Server to generate a database structure script.

1. Right-click the database whose schema script needs to be generated and choose **Tasks** > **Generate Scripts**.
2. On the **Choose Objects** page, choose database objects to script, as shown in **Figure 3-4**. Then, click **Next**.

Figure 3-4 Choosing objects

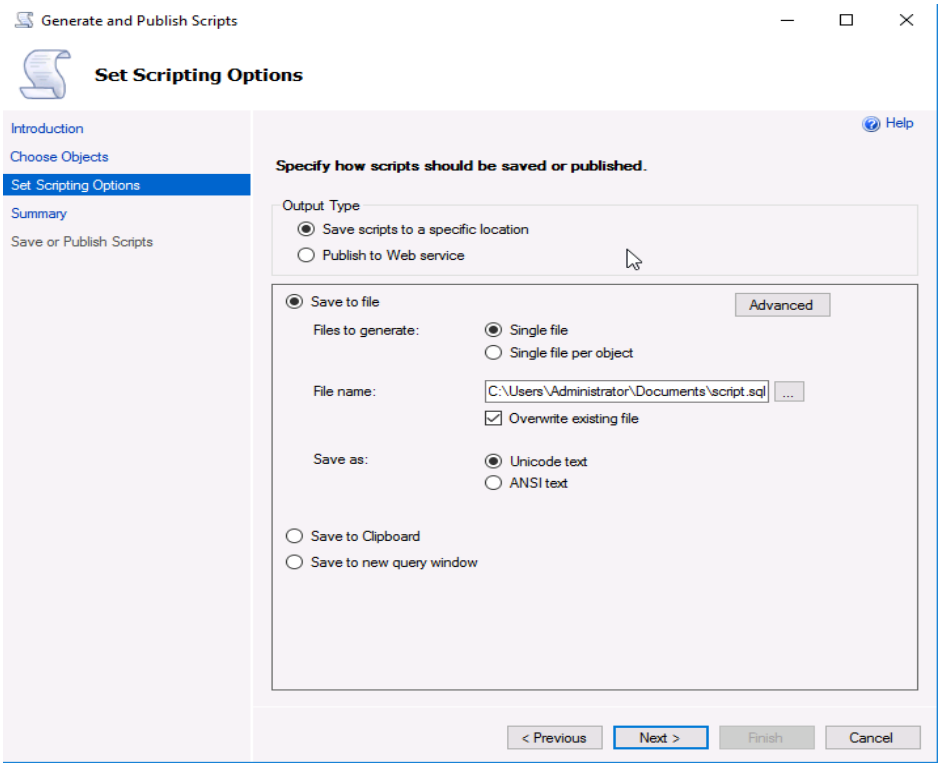


3. On the **Set Scripting Options** page, specify a directory for saving the script.

NOTE

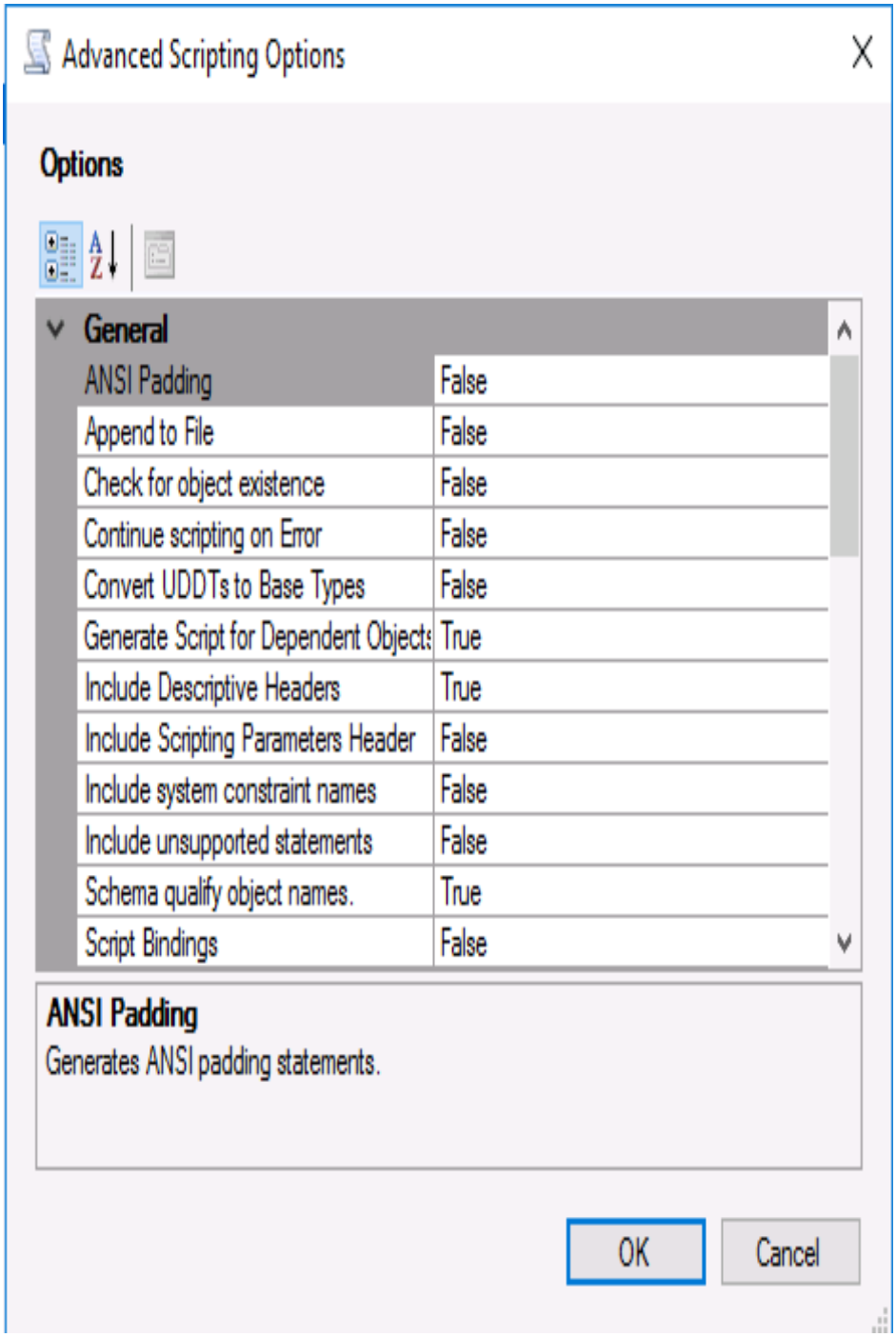
You are advised to save the script locally and generate an SQL script for execution.

Figure 3-5 Specifying a directory for saving the script



4. Click **Advanced**. In the displayed **Advanced Scripting Options** dialog box, specify scripting options for triggers, indexes, unique keys, the primary key, and server version. Then, click **OK**.

Figure 3-6 Specifying advanced scripting options



NOTE

Generate Script for Dependent Objects indicates the script data type option.

5. Click **Next** to generate the script.

Step 9 Use the SSMS client to connect to the RDS DB instance and open the generated SQL script.

NOTE

You need to create an empty database, and then use the script to create structures in the database.

Step 10 Use the import and export function provided by Microsoft SQL Server to migrate data.

1. Right-click the database where data is to be imported and choose **Tasks** > **Import Data**.
2. Click **Next**.
3. On the **Choose a Data Source** page, select a data source and click **Next**.
4. On the **Choose a Destination** page, select a destination database and click **Next**.
   - **Destination**: Select **SQL Server Native Client** (depending on your destination database type).
   - **Server name**: Enter the IP address and port number of the destination DB instance.
   - **Authentication**: Select **Use SQL Server Authentication**. Then, set **User name** to **rdsuser**, and **Password** to the password of **rdsuser**.
   - **Database**: Select the destination database where data is to be imported.
5. Select **Copy data from one or more tables or views** and click **Next**.
6. On the **Select Source Tables and Views** page, select the tables and views that you want to copy. Then, click **Edit Mappings**. In the displayed dialog box, select **Enable identity insert** and edit mappings based on your requirements.
7. Click **Next**.
8. Select **Run immediately** and click **Next**.
9. Click **Finish** to import data. You can view the progress. About 4,000 rows can be processed per second.

----End

3.4 Shrinking an RDS for SQL Server Database

Scenarios

You can use a stored procedure to shrink the size of the data and log files in a specified RDS for SQL Server database.

Prerequisites

An RDS for SQL Server DB instance has been connected. Connect to the DB instance through the Microsoft SQL Server client.

Constraints

- The database can be shrunk only when the database file size exceeds 50 MB. Otherwise, the following message is displayed:

```
Cannot shrink file '2' in database 'master' to 6400 pages as it only contains 2
```


- Transactions running at the row version control-based isolation level may prevent shrinking operations. To solve this problem, perform the following steps:
  a. Terminate the transactions that prevent shrinking.
  b. Terminate the shrinking operation. All completed tasks will be retained.
  c. Do not perform any operations and wait until the blocking transactions are complete.

Best Practices

When you plan to shrink a database, consider the following:

- A shrink operation is most effective after an operation that creates lots of unused space, such as a database reboot.
- Most databases require some free space to be available for regular day-to-day operations. If you shrink a database repeatedly and notice that the database size grows again, this indicates that the space that was shrunk is required for regular operations. In these cases, repeatedly shrinking the database is a wasted operation.
- A shrink operation does not preserve the fragmentation state of indexes in the database, and generally increases fragmentation to a degree. This is another reason not to repeatedly shrink the database.

Procedure

Step 1 Run the following command to shrink a database:

EXEC [master].[dbo].[rds_shrink_database] @DBName='*myDbName***';**

Table 3-2 Parameter description

Parameter	Description
myDbName	Name of the database to be shrunk. If this parameter is not specified, all databases are shrunk by default.

The following figure shows the execution result set. Each result corresponds to the information about each file in the specified database (or all databases).

Figure 3-7 Result set

Results | Messages

	DbId	FileId	CurrentSize	MinimumSize	UsedPages	EstimatedPages
1	1	1	688	512	512	512

Table 3-3 Result set parameter description

Column Name	Description
DbId	Database ID of the current shrink file.
FileId	File ID of the current shrink file.
CurrentSize	Number of 8 KB pages occupied by the file.
MinimumSize	Minimum number of 8 KB pages occupied by the file. The value indicates the minimum size or the initial size of the file.
UsedPages	Number of 8 KB pages used by the file.
EstimatedPages	Number of 8 KB pages that the database engine estimates the file can be shrunk to.

Step 2 After the command is successfully executed, the following information is displayed:

```
RDS_Process_Successful: Shrink Database Done.
```

----End

Fault Rectification

If the file size does not change after the database is shrunk, run the following SQL statement to check whether the file has sufficient available space:

SELECT name, size/128.0 - CAST(FILEPROPERTY(name, 'SpaceUsed') AS int)/ 128.0 AS AvailableSpaceInMB FROM sys.database_files;

Example

1. Run the following command to shrink the **dbtest2** database:

   EXEC [master].[dbo].[rds_shrink_database] @DBName = 'dbtest2';

   The command output is as follows.

   Figure 3-8 Execution result

   ```
   [Shrink Start] Date and time: 2020-03-19 15:51:07

   Start to shrink files in database [dbtest2], current file id is 1...
   DBCC execution completed. If DBCC printed error messages, contact your system administrator.
   Shrink file (id: 1) in database [dbtest2] done!

   Start to shrink files in database [dbtest2], current file id is 2...
   DBCC execution completed. If DBCC printed error messages, contact your system administrator.
   Shrink file (id: 2) in database [dbtest2] done!

   [Shrink End] Date and time: 2020-03-19 15:51:08

   HW_RDS_Process_Successful : Shrink Database done.
   ```

2. Run the following command to shrink all databases:

   EXEC [master].[dbo].[rds_shrink_database];

