ALIBABA CLOUD

Alibaba Cloud

ApsaraDB for RDS

Proprietary AliPG

Document Version: 20201127

Alibaba Cloud

Legal disclaimer

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

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

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

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

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

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

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

Document conventions

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

Table of Contents

1.AliPG benefits

This topic describes the background information and benefits of AliPG.

Background information

PostgreSQL is an advanced open source enterprise-grade database service. PostgreSQL was listed in the DB-Engines Ranking in 2017 and 2018. In 2019, PostgreSQL even won the O'Reilly Open Source Convention award (OSCON).

Alibaba Cloud offers two PostgreSQL-compatible database services that run AliPG: ApsaraDB RDS for PostgreSQL and ApsaraDB MyBase for PostgreSQL. AliPG is a unified database engine that is developed by Alibaba Cloud. Since its commercial rollout in 2015, AliPG has been running stably for years to process a large number of workloads within Alibaba Group and on the cloud. AliPG supports the following major PostgreSQL versions: 9.4, 10, 11, and 12.

Alibaba Cloud database services that run AliPG

- ApsaraDB RDS for PostgreSQL
- ApsaraDB MyBase for PostgreSQL

Supported PostgreSQL versions

- 12
- 11
- 10
- 9.4

Benefits

AliPG is developed based on insights into the industry requirements. AliPG aims to help customers expand business boundaries.

AliPG has the following benefits over the PostgreSQL Community edition:

- Faster
  - Image recognition and vector similarity-based search run tens of thousands of times faster on AliPG than on the PostgreSQL Community edition. For more information, see Image recognition, face recognition, similarity-based retrieval, and similarity-based audience spotting.
  - Real-time marketing and user profiling run thousands of times faster on AliPG than on the PostgreSQL Community edition. For more information, see Real-time precision marketing (user selection).
  - The GIS-based Mod operator processes mobile objects 50 times faster than the Mod operator that runs based on the open source PostGIS. For more information, see Overview.
- More stable

  AliPG uses the Platform as a Service (PaaS) architecture. This architecture allows you to transform traditional software from license-based service to subscription-based service. You can manage a large amount of metadata, optimize connections, and better isolate resources. You can also use tens of thousands of schemas per RDS instance.
- More secure

- AliPG is certified based on leading national and international security standards, which empowers enterprises to increase institutional security scores in the financing and listing phases.
- AliPG provides the following security enhancements:
  - Encrypts sensitive data that contains passwords. The sensitive data includes dynamic views, shared memory, the dblink plug-in, historical commands, and audit logs.
  - Fixes bugs in the functions that you call in the PostgreSQL Community edition.
  - Supports fully encrypted databases. For more information, see Create a fully encrypted database on an ApsaraDB RDS for PostgreSQL instance.
  - Supports the semi-synchronous mode. This mode allows you to configure the following protection levels for your RDS instance: maximum protection, maximum availability, and maximum performance. For more information, see Set the protection level of an ApsaraDB RDS for PostgreSQL instance.
  - Supports the failover slot function. This function prevents primary/secondary switchovers from affecting the reliability of logical replication. For more information, see Use the failover slot feature for logical subscriptions.

- More flexible and controllable (For more information, see What is ApsaraDB for MyBase?)
  - AliPG grants you the permissions to manage the operating systems on hosts in dedicated ApsaraDB MyBase for PostgreSQL clusters. This allows you to manage your dedicated ApsaraDB MyBase for PostgreSQL clusters based on your business requirements.
  - AliPG allows you to customize overcommit ratios in the development, test, and staging environments. For example, you can configure 128 CPU cores for a host that provides only 64 CPU cores. This allows you to exclusively occupy resources in the production system to reduce the overall costs.

2.Release notes of minor AliPG versions

This topic describes the release notes of minor AliPG versions.

PostgreSQL 12

20200830

- New features:
  - The Ganos plug-in is upgraded to version 3.0.
  - The sql_firewall plug-in is supported. It allows you to prevent an injection of malicious SQL statements.
  - The pg_bigm plug-in is supported. It allows you to implement fuzzy match.
  - The TimescaleDB plug-in is supported. It allows you to process time series data.
- Bugs fixed:
  - The bug that prevents the backend from identifying the `rds_` prefix in parameters is fixed.
  - The bug that prevents you from properly creating and using the pg_cron plug-in is fixed.
  - The bug that prevents you from loading the RDKit plug-in due to the lack of dependencies is fixed.

20200421

- New features:
  - AliPG is upgraded to ensure compatibility with PostgreSQL 12.2 of the Community edition.
  - The Ganos plug-in is upgraded to version 2.7.
  - The HLL plug-in of version 2.14 is supported.
  - The PL/Proxy plug-in of version 2.9.0 is supported.
  - The tsm_system_rows plug-in of version 1.0 is supported.
  - The tsm_system_time plug-in of version 1.0 is supported.
  - The smlar plug-in of version 1.0 is supported.
  - The tds_fdw plug-in of version 1.0 is supported.
- Bug fixed:

  The bug that causes RDS instances to restart due to logical subscription timeout is fixed.

20200221

- New features:
  - The reservation for a specific number of connections is supported for the rds_superuser role. If you are authorized with the rds_superuser role, you can connect to an RDS instance to troubleshoot issues even if the number of established connections to the RDS instance reaches the upper limit.
  - The wal2json plug-in is supported.
  - The Ganos plug-in is upgraded to version 2.6.
- Bugs fixed:

  Some permission-related bugs are fixed.

20191230

New features:

- The pg_roaringbitmap, RDKit, mysql_fdw, and Ganos plug-ins are supported.
- The permissions to publish all tables at a time and create subscriptions are granted to privileged accounts.

PostgreSQL 11

20200830

- New features:
  - The Ganos plug-in is upgraded to version 3.0.
  - The sql_firewall plug-in is supported. It allows you to prevent an injection of malicious SQL statements.
  - The pg_bigm plug-in is supported. It allows you to implement fuzzy match.
  - The ZomboDB plug-in is supported. It allows you to implement text search and analysis.
- Bugs fixed:
  - The bug that prevents the backend from identifying the `rds_` prefix in parameters is fixed.
  - The bug that causes a secondary RDS instance to exit because the failover slot has the same name as the streaming replication slot is fixed.
  - The bug that prevents you from properly creating and using the pg_cron plug-in is fixed.
  - The bug in a global variable of the PASE plug-in is fixed.

20200610

New features:

- The TimescaleDB plug-in of version 1.7.1 is supported.
- The pageinspect plug-in is supported for the rds_superuser role.
- The rds_superuser role is authorized to grant the replication permissions to other users.

20200511

- New feature:

  The Ganos plug-in is upgraded to version 2.8.
- Bug fixed:

  The bug that causes the PASE plug-in to slowly run INSERT statements on HNSW indexes is fixed.

20200421

New features:

- The failover slot function is supported. For more information, see Use the failover slot feature for logical subscriptions.
- The PL/Proxy plug-in of version 2.9.0 is supported.
- The tsm_system_rows plug-in of version 1.0 is supported.
- The tsm_system_time plug-in of version 1.0 is supported.
- The smlar plug-in of version 1.0 is supported.

20200402

- New features:
  - The HLL plug-in of version 2.14 is supported. It allows ApsaraDB RDS for PostgreSQL to support the HLL data type, respond to queries in milliseconds, and analyze approximate data at low costs and high speeds. For example, you can query page views (PVs) and unique visitors (UVs) in real time and determine whether the analyzed approximate data contains the specified characteristic tags.
  - The oss_fdw plug-in of version 1.1 is supported. It allows you to store infrequently requested historical data to Object Storage Service (OSS) buckets. This reduces storage costs.
  - The tds_fdw plug-in of version 2.0.1 is supported. It allows you to initiate requests on an ApsaraDB RDS for PostgreSQL instance to query data from a Sybase or SQL Server database without the need to perform extract, transform and load (ETL) operations. It also allows you to migrate data between an ApsaraDB RDS for PostgreSQL instance and a Sybase or SQL Server database.
- Upgraded plug-ins:
  - The Ganos plug-in is upgraded to version 2.7.
  - The wal2json plug-in is upgraded to version 2.2.
- Performance optimization:

  The `shutdown -m fast` command is optimized.

20191218

New features:

- The PASE plug-in is supported. It allows you to create indexes that are used to recognize images.
- The permissions to publish all tables at a time and create subscriptions are granted to privileged accounts.

PostgreSQL 10

20200830

- New features:
  - The Ganos plug-in is upgraded to version 3.0.
  - The sql_firewall plug-in is supported. It allows you to prevent an injection of malicious SQL statements.
  - The pg_bigm plug-in is supported. It allows you to implement fuzzy match.
- Bug fixed:

  The bug that prevents you from properly creating and using the pg_cron plug-in is fixed.

20200212

- New features:
  - The reservation for a specific number of connections is supported for the rds_superuser role. If you are authorized with the rds_superuser role, you can connect to an RDS instance to troubleshoot issues even if the number of established connections to the RDS instance reaches the upper limit.
  - The Ganos plug-in is upgraded to version 2.6.
- Bug fixed:

  The bug that causes long waits in streaming replication is fixed.

20190703

- New features:
  - AliPG is upgraded to version 10.9.
  - The change from synchronous replication to asynchronous replication is supported when the ongoing replication times out.
- Bug fixed:
  - The bug that prevents you from properly creating the pg_hint_plan plug-in is fixed.
  - The bug that causes failures in external RUM indexing is fixed.

PostgreSQL 9.4

20200623

- New features:
  - The wal2json plug-in is upgraded to version 2.2.
  - The xml2 plug-in of version 1.0 is supported.
- Bug fixed:

  The bug that causes memory exhaustion when the wal2json plug-in runs is fixed.

20200210

The reservation for a specific number of connections is supported for the rds_superuser role. If you are authorized with the rds_superuser role, you can connect to an RDS instance to troubleshoot issues even if the number of established connections to the RDS instance reaches the upper limit.

20190601

AliPG is upgraded to version 9.4.19.

3.Functional modules of AliPG

This topic describes the functional modules that are provided by AliPG.

Overview

Category	Functional module	Description
Account permission	rds_superuser	The rds_superuser role is a type of intermediate account between standard accounts and superuser accounts. The accounts that are assigned the rds_superuser role are called privileged accounts. To ensure security on the cloud, AliPG does not provide superuser accounts. However, AliPG provides the rds_superuser role. The rds_superuser role does not have the sensitive security permissions that superuser accounts have. The rds_superuser role has the permissions to create and delete plug-ins, create and delete standard and privileged accounts, access and manage all of the tables that are created by standard accounts, and close connections.
Spatio-temporal database engine	Ganos	Ganos is a spatio-temporal database engine that is developed by Alibaba Cloud. Ganos provides a series of data types, functions, and stored procedures to efficiently store, index, query, analyze, and compute spatio-temporal data.
External data reads and writes	oss_fdw	The oss_fdw plug-in supports data migration and hot and cold data separation. You can load data from Alibaba Cloud Object Storage Service (OSS) buckets to RDS instances. You can also write data from RDS instances to OSS buckets.
Concurrency control (CCL)	pg_concurrency_control	The pg_concurrency_control plug-in controls the concurrency of transactions, SQL queries, stored procedures, and data manipulation language (DML) operations. This plug-in allows you to customize large queries, which expedites the execution of highly concurrent workloads.
Failover slot	Failover Slot	In open source PostgreSQL, logical slots cannot be automatically switched over to the new primary RDS instance in the event of a primary/secondary switchover. As a result, logical subscriptions are disconnected. In AliPG, the failover slot feature allows ApsaraDB RDS to synchronize all logical slots from the primary RDS instance to the secondary RDS instance. This prevents the disconnection of logical subscriptions.
Bitmap function extension	varbitx	The varbit plug-in of open source PostgreSQL supports only simple BIT-type operation functions. The varbit plug-in is an extension of the varbit plug-in. The varbit plug-in is provided in AliPG to support more BIT-type operations in more scenarios. These scenarios include real-time user profile recommendation, access control advertising, and ticketing.

Category	Functional module	Description
Vector search	PASE	PASE is a high-performance vector search index plug-in that is developed for AliPG. The PASE plug-in uses IVFFlat and Hierarchical Navigable Small World (HNSW), which are well-developed, stable, and efficient approximate nearest neighbor (ANN) search algorithms. These algorithms are used to query vectors from PostgreSQL databases at high speeds. The PASE plug-in does not support the extraction or output of feature vectors. You must retrieve the feature vectors of the entities that you want to query. The PASE plug-in only implements a similarity search among a large number of vectors that are identified based on retrieved feature vectors.
Log query	log_fdw	The log_fdw plug-in queries logs from external tables.
Availability	Protection levels	You can set the protection level of an ApsaraDB RDS instance based on your business requirements. This setting allows you to improve the availability and performance of your RDS instance.
Security	Security hardening	A security hardening module is built-in to improve custom views, enhance the security of the functions that you call, prevent security traps, and avoid the security vulnerabilities that are detected in open source PostgreSQL.

References

Plug-ins supported

4.Plug-ins supported

本文列出RDS PostgreSQL各版本支持的插件及其版本。

Note 如果您对插件有需求或建议，请 提交工单 。

插件名	12	11	10	9.4
address_standardizer	2.5.4	2.5.4	2.5.4	无
address_standardizer_data_us	2.5.4	2.5.4	2.5.4	无
aggs_for_arrays	无	无	1.3.1	无
ali_decoding	无	无	0.0.1	0.0.1
pg_bigm	1.2	1.2	1.2	无
btree_gin	1.3	1.3	1.2	1.0
btree_gist	1.5	1.5	1.5	1.0
chkpass	无	无	1.0	1.0
citext	1.6	1.5	1.4	1.0
cube	1.4	1.4	1.2	1.0
dblink	1.2	1.2	1.2	1.1
dict_int	1.0	1.0	1.0	1.0
earthdistance	1.1	1.1	1.1	1.0
fuzzystrmatch	1.1	1.1	1.1	1.0
ganos_address_standardizer	3.2	3.2	3.2	无
ganos_address_standardizer_data_us	3.2	3.2	3.2	无
ganos_geometry	3.2	3.2	3.2	无
ganos_geometry_sfcgal	3.2	3.2	3.2	无
ganos_geometry_topology	3.2	3.2	3.2	无

插件名	12	11	10	9.4
ganos_networking	3.2	3.2	3.2	无
ganos_pointcloud	3.2	3.2	3.2	无
ganos_pointcloud_geometry	3.2	3.2	3.2	无
ganos_raster	3.2	3.2	3.2	无
ganos_spatialref	3.2	3.2	3.2	无
ganos_tiger_geocoder	3.2	3.2	3.2	无
ganos_trajectory	3.2	3.2	3.2	无
hll	2.14	2.14	无	无
hstore	1.6	1.5	1.4	1.3
imgsmlr	无	无	无	1.0
intagg	1.1	1.1	1.1	1.0
intarray	1.2	1.2	1.2	1.0
isn	1.2	1.2	1.1	1.0
jsonbx	无	无	无	1.0
log_fdw	无	1.0	无	无
ltree	1.1	1.1	1.1	1.0
mysql_fdw	1.1	无	1.0	无
ogr_fdw	无	1.0	1.0	无
orafce	无	3.8	3.6	3.6
oss_fdw	无	1.1	1.1	1.1
pase	无	0.0.1	无	无
pg_awr	无	无	无	1.0
pg_buffercache	1.3	1.3	1.3	1.0
pg_concurrency_control	无	1.0	1.0	1.0
pg_cron	1.1	1.1	1.1	无

插件名	12	11	10	9.4
pg_hint_plan	无	无	1.3.0	1.1.3
pg_pathman	无	1.5	1.5	无
pg_prewarm	1.2	1.2	1.1	1.0
pg_roaringbitmap	0.5.0	无	无	无
pg_sphere	无	无	1	1
pg_stat_statements	1.7	1.6	1.6	1.2
pg_trgm	1.4	1.4	1.3	1.1
pgcrypto	1.3	1.3	1.3	1.1
pgrouting	无	2.6.2	2.6.2	2.0.0
pgrowlocks	1.2	1.2	1.2	1.1
pgstattuple	1.5	1.5	1.5	1.2
plcoffee	无	无	1.4.2	1.4.2
plls	无	无	1.4.2	1.4.2
plperl	1.0	1.0	1.0	1.0
plpgsql	1.0	1.0	1.0	1.0
plproxy	2.9.0	2.9.0	2.8.0	无
pltcl	无	无	无	1.0
plv8	无	2.3.13	1.4.2	1.4.2
postgis	2.5.4	2.5.4	2.5.4	2.2.8
postgis_sfcgal	2.5.4	2.5.4	2.5.4	无
postgis_tiger_geocoder	2.5.4	2.5.4	2.5.4	2.2.8
postgis_topology	2.5.4	2.5.4	2.5.4	2.2.8
postgres_fdw	1.0	1.0	1.0	1.0
q3c	无	无	1.5.0	1.5.0
rdkit	3.8	无	无	3.4

插件名	12	11	10	9.4
rum	无	1.3	1.3	无
smlar	1.0	1.0	1.0	1.0
sslinfo	1.2	1.2	1.2	1.0
tablefunc	1.0	1.0	1.0	1.0
tds_fdw	2.0.1	2.0.1	无	无
timescaledb	1.7.1	1.7.1	1.7.1	无
tsearch2	无	无	无	1.0
tsm_system_rows	1.0	1.0	无	无
tsm_system_time	1.0	1.0	无	无
unaccent	1.1	1.1	1.1	1.0
uuid-ossp	无	无	1.1	1.0
varbitx	无	1.0	1.0	1.0
wal2json	2	2.2	无	无
zhparser	无	1.0	1.0	1.0
zombodb	无	4.0	无	无

5.Query vertical industry-specific data

5.1. Use the TimescaleDB plug-in

The TimescaleDB plug-in is introduced to ApsaraDB RDS for PostgreSQL instances. The plug-in supports automatic sharding, fast writes, retrieval, and near real-time aggregation of time series data.

The TimescaleDB plug-in supported by ApsaraDB RDS for PostgreSQL is an open-source edition and may not support some advanced features because of issues such as licenses. For more information, see TimescaleDB.

Prerequisites

Your RDS instance runs PostgreSQL 11.

> **Note** Some existing users may have created the TimescaleDB plug-in. If the following message appears after you upgrade the kernel version:
>
> ```
> ERROR: could not access file "$libdir/timescaledb-1.3.0": No such file or directory
> ```
>
> Execute the following SQL statement in the target database to update the plug-in:
>
> ```
> alter extension timescaledb update;
> ```

Create the TimescaleDB plug-in

Use the pgAdmin client to connect to your RDS instance. Then execute the following statement to create the TimescaleDB plug-in:

```
CREATE EXTENSION IF NOT EXISTS timescaledb CASCADE;
```

Create a time series table

1. Execute the following statement to create a standard table named conditions:

   ```
   CREATE TABLE conditions (
    time      TIMESTAMPTZ      NOT NULL,
    location  TEXT             NOT NULL,
    temperature DOUBLE PRECISION  NULL,
    humidity    DOUBLE PRECISION  NULL
   );
   ```

2. Execute the following statement to create a time series table:

   ```
   SELECT create_hypertable('conditions', 'time');
   ```

> **Note** For more information, see Create a Hypertable.

Insert data into a hypertable

You can execute standard SQL statements to insert data into a hypertable. Example:

```
INSERT INTO conditions(time, location, temperature, humidity)
 VALUES (NOW(), 'office', 70.0, 50.0);
```

You can also insert multiple rows of data into a hypertable at a time. Example:

```
INSERT INTO conditions
 VALUES
  (NOW(), 'office', 70.0, 50.0),
  (NOW(), 'basement', 66.5, 60.0),
  (NOW(), 'garage', 77.0, 65.2);
```

Retrieve data

You can run advanced SQL queries to retrieve data. Example:

```
--Query data that was collected at 15-minute intervals for the last 3 hours and was sorted from time and tem
perature dimensions.
SELECT time_bucket('15 minutes', time) AS fifteen_min,
  location, COUNT(*),
  MAX(temperature) AS max_temp,
  MAX(humidity) AS max_hum
 FROM conditions
 WHERE time > NOW() - interval '3 hours'
 GROUP BY fifteen_min, location
 ORDER BY fifteen_min DESC, max_temp DESC;
```

You can also use built-in functions to analyze and query data. Examples:

```
--Query the median.
SELECT percentile_cont(0.5)
 WITHIN GROUP (ORDER BY temperature)
 FROM conditions;
```

```
--Query the moving average.
SELECT time, AVG(temperature) OVER(ORDER BY time
    ROWS BETWEEN 9 PRECEDING AND CURRENT ROW)
  AS smooth_temp
 FROM conditions
 WHERE location = 'garage' and time > NOW() - interval '1 day'
 ORDER BY time DESC;
```

5.2. Use the smlar plug-in

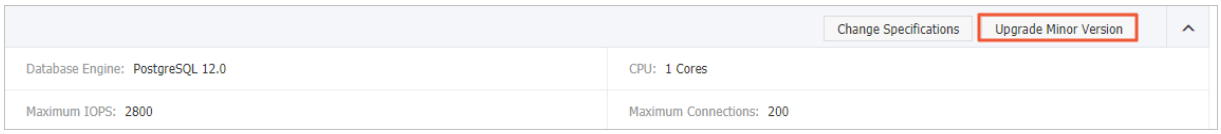
This topic describes the smlar plug-in. This allows you to calculate the similarity between two arrays of the same data type.

Prerequisites

The instance runs one of the following PostgreSQL versions:

- PostgreSQL 12 (kernel version 20200421 and later)
- PostgreSQL 11 (kernel version 20200402 and later)

> **Note** To view the kernel version, perform the following steps: Log on to the ApsaraDB for RDS console, find the target RDS instance, and navigate to the **Basic Information** page. Then, in the **Configuration Information** section, check whether the **Upgrade Minor Version** button exists. If the button exists, click it to view the kernel version. If the button does not exist, it indicates that you are already using the latest kernel version. For more information, see Upgrade the kernel version of an ApsaraDB RDS for PostgreSQL instance.



Context

The smlar plug-in provides multiple functions to calculate the similarity between two arrays of the same data type. It also provides parameters to control the similarity calculation methods. All built-in data types are supported.

Function description

- float4 smlar(anyarray, anyarray)

  Calculates the similarity between two arrays of the same data type.

- float4 smlar(anyarray, anyarray, bool useIntersect)

  Calculates the similarity between two arrays of composite data types. The composite data type is defined as follows:

  ```
  CREATE TYPE type_name AS (element_name anytype, weight_name FLOAT4);
  ```

When the useIntersect parameter is set to true, only the parts that contain duplicate elements are calculated. When the useIntersect parameter is set to false, all elements are calculated.

- float4 smlar(anyarray a, anyarray b, text formula)

  Calculates the similarity between two arrays of the same data type. The arrays are specified by the formula parameter.

  The predefined variables for formula are described as follows:

  - N.i: The number of common elements in the two arrays.
  - N.a: The number of distinct elements in array a.
  - N.b: The number of distinct elements in array b.

- float4 set_smlar_limit(float4)

  Sets the smlar.threshold parameter.

- float4 show_smlar_limit()

  Displays the smlar.threshold parameter value.

- anyarray % anyarray

  Returns true if the similarity between arrays is greater than the smlar.threshold parameter value. Otherwise, returns false.

- text[] tsvector2textarray(tsvector)

  Converts the tsvector type to the text type.

- anyarray array_unique(anyarray)

  Sorts the elements (excluding duplicate elements) in an array.

- float4 inarray(anyarray, anyelement)

  Returns 1 if the anyelement parameter value exists in the anyarray parameter value. otherwise, returns 0.

- float4 inarray(anyarray, anyelement, float4, float4)

  Returns the third parameter value if anyelement exists in anyarray. Otherwise, returns the fourth parameter value.

For more information about parameter descriptions and supported data types, visit smlar.

Use smlar

- After you have connected to an instance, execute the following statement to create a smlar plug-in:

  ```
  testdb=> create extension smlar;
  ```

- Execute the following statements to use basic functions of smlar:

```
testdb=> SELECT smlar('{1,4,6}'::int[], '{5,4,6}' );
 smlar
----------
 0.666667
(1 row)
testdb=> SELECT smlar('{1,4,6}'::int[], '{5,4,6}', 'N.i / sqrt(N.a * N.b)' );
 smlar
----------
 0.666667
(1 row)
```

- Execute the following statement to remove smlar:

```
testdb=> drop extension smlar;
```

5.3. Use the PASE plug-in

This topic describes how to use the PostgreSQL ANN search extension (PASE) plug-in to search for vectors in RDS PostgreSQL.

Prerequisites

Your ApsaraDB RDS PostgreSQL instance runs PostgreSQL 11.

Background information

Representation learning is a typical artificial intelligence (AI) technology in the deep learning discipline. It has developed rapidly over the recent years and is used in various businesses such as advertising, face scan payment, image recognition, and voice recognition. This technology enables data to be embedded into high-dimensional vectors and allows you to query data by using the vector search approach.

PASE is a high-performance vector search index plug-in developed for PostgreSQL. It uses two well-developed, stable, and efficient approximate nearest neighbor (ANN) search algorithms, IVFFlat and Hierarchical Navigable Small World (HNSW), to query vectors from PostgreSQL databases at high speeds. PASE does not support the extraction or output of feature vectors. You must retrieve the feature vectors of the entities you want to query. PASE only implements a similarity search among a large amount of vectors identified based on retrieved feature vectors.

Intended audience

This topic does not explain the terms related to machine learning in detail. Before you read this topic, you must understand the basics of machine learning and search technologies.

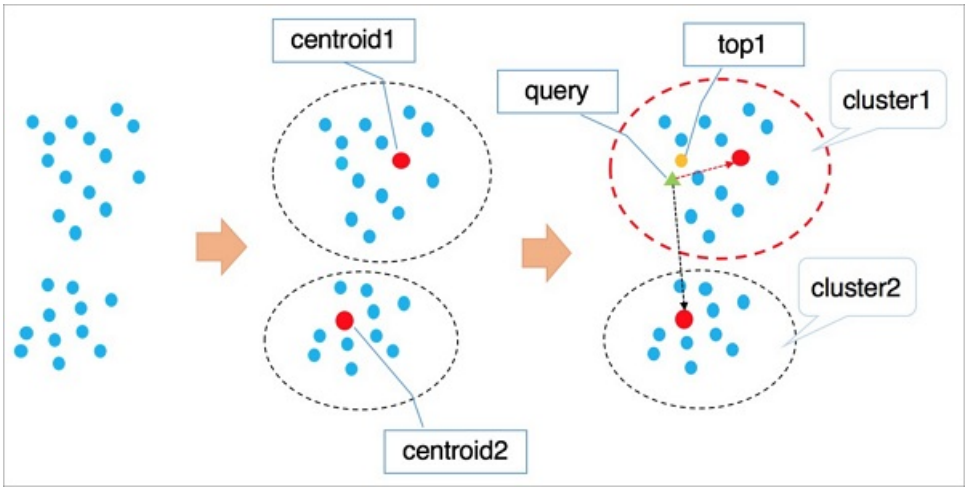
Algorithms used by PASE

- IVFFlat

  IVFFlat is a simplified version of the IVFADC algorithm. It is suitable for businesses that require high precision but can tolerate up to 100 milliseconds taken for queries. IVFFlat has the following advantages compared with other algorithms:

- If the vector to query is one of the candidate datasets, IVFFlat delivers 100% recall.
- IVFFlat uses a simple structure to create an index fast and occupy less storage space.
- You can specify a centroid for clustering and can control precision by reconfiguring parameters.
- You can control the accuracy of IVFFlat by reconfiguring its interpretable parameters.

The following figure shows how IVFFlat works.



The procedure is described as follows:

i. IVFFlat uses a clustering algorithm such as k-means to divide vectors in the high-dimensional data space into clusters based on implicit clustering properties. Each cluster has a centroid.

ii. IVFFlat traverses the centroids of all clusters to identify the n centroids nearest to the target vector you want to query.

iii. IVFFlat traverses and sorts all vectors in the clusters to which the identified n centroids belong, and then obtains the nearest k vectors.

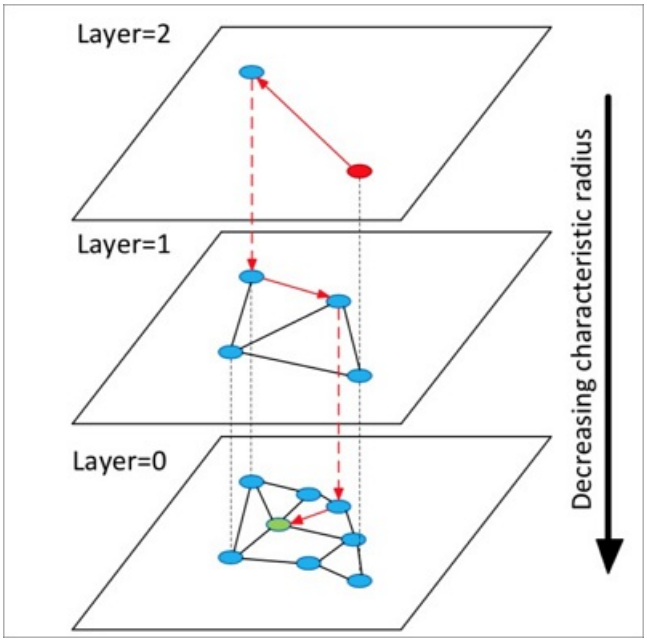
> **Note**
>
> - When IVFFlat attempts to identify the nearest n centroids, it skips the clusters located far away to expedite the query. However, IVFFlat cannot ensure that all the similar k vectors are included in the clusters to which the identified n centroids belong. As a result, precision may decrease. You can use the variable n to control precision. A larger n value indicates higher precision but more computing workloads.
> - In the first phase, IVFFlat works the same as IVFADC. The main differences between them lie in the second phase. In the second phase, IVFADC uses product quantization to avoid traversal computing workloads. This is faster but has lower precision. Whereas, IVFFlat uses brute-force computing to ensure precision and allows you to control computing workloads.

- HNSW

  HNSW is a graph-based ANN algorithm suitable for queries among tens of millions or more vector datasets that take 10 milliseconds or less.

  HNSW searches among proximity graph neighbors for similar vectors. When the data volume is large, HNSW significantly improves performance compared to other algorithms. However, HNSW requires the storage of proximity graph neighbors, which occupy storage space. In addition, the precision of HNSW cannot be increased by reconfiguring parameters after reaching a certain level.

The following figure shows how HNSW works.



The procedure is described as follows:

i. HNSW builds a hierarchical structure consisting of multiple layers (graphs). Each layer is a panorama and skip list of its lower layer.

ii. HNSW randomly selects an element from the top layer to start a search.

iii. HNSW identifies the neighbors of the selected element and adds the identified neighbors to a fixed-length dynamic list based on their distances to the selected element. HNSW continues to identify the neighbors of each neighbor included in the list and adds the identified neighbors to the list. Every time when HNSW adds a neighbor to the list, it re-sorts the neighbors in the list and only retains the first k neighbors. If the list changes, HNSW continues to search until the list reaches its final state. Then, HNSW uses the first element in the list as the start for a search in the lower layer.

iv. HNSW repeats the third step until it enters the bottom layer.

> **Note** HNSW constructs a multi-layer structure by using the Navigable Small World (NSW) algorithm designed to construct single-layer structures. The employment of an approach for selecting proximity graph neighbors enables HNSW to deliver higher query speedup than clustering algorithms.

IVFFlat and HNSW are each suitable for specific businesses. For example, IVFFlat is suitable for image comparison at high precision, and HNSW is suitable for searches with recommended recall. More industry-leading algorithms will be integrated into PASE.

Procedure

1. Execute the following statement to create the PASE plug-in:

```
CREATE EXTENSION pase;
```

2. Use one of the following construction methods to calculate vector similarity:
   - PASE-type-based construction

Example:

```
SELECT ARRAY[2, 1, 1]::float4[] <? > pase(ARRAY[3, 1, 1]::float4[]) AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <? > pase(ARRAY[3, 1, 1]::float4[], 0) AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <? > pase(ARRAY[3, 1, 1]::float4[], 0, 1) AS distance;
```

> **Note**
>
> - <? > is a PASE-type operator, which specifies to calculate similarity between the vectors to the left and right of a specific element. The vector to the left must use the float4[] data type and the vector to the right must use the PASE data type.
> - The PASE data type is defined in the PASE plug-in and can contain up to three constructors. Take the `float4[], 0, 1` part in the preceding third statement as an example: The first parameter specifies the vector to the right with the float4[] data type. The second parameter does not serve a special purpose, so you can set it to 0. The third parameter specifies the similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product (also referred to as inner product) method.
> - The vector to the left must have the same number of dimensions as the vector to the right. Otherwise, the system reports similarity calculation errors.

- String-based construction

  Example:

```
SELECT ARRAY[2, 1, 1]::float4[] <? > '3,1,1'::pase AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <? > '3,1,1:0'::pase AS distance;
SELECT ARRAY[2, 1, 1]::float4[] <? > '3,1,1:0:1'::pase AS distance;
```

> **Note** The string-based construction method differs from the PASE-type-based construction in the following aspect: The string-based construction method uses colons (:) to separate parameters. Take the `3,1,1:0:1` part in the preceding third statement as an example: The first parameter specifies the vector to the right. The second parameter does not serve a special purpose, so you can set it to 0. The third parameter specifies the similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product (also referred to as the inner product) method.

3. Use IVFFlat or HNSW to create an index.

> **Note** If you use the Euclidean distance method to calculate vector similarity, the original vector does not need to be processed. If you use the dot product or cosine method to calculate vector similarity, the original vector must be normalized. For example, if the original vector is $x_1, x_2, x_3, \ldots, x_n$, it must meet the following formula:
>
> $x_1^2 + x_2^2 + x_3^2 + \cdots + x_n^2 = 1$. In this example, the dot product is the same as the cosine value.

- IVFFlat

  Example:

```
CREATE INDEX ivfflat_idx ON vectors_table
USING
 pase_ivfflat(vector)
WITH
 (clustering_type = 1, distance_type = 0, dimension = 256, base64_encoded = 0, clustering_params = "
10,100");
```

  The following table describes the parameters in the IVFFlat index.

Parameter	Description
clustering_type	The type of clustering operation IVFFlat performs on vectors. This parameter is mandatory. Valid values: ■ 0: external clustering. An external centroid file is loaded, which is specified by the **clustering_params** parameter. ■ 1: internal clustering. The k-means clustering algorithm is used, which is specified by the **clustering_params** parameter. If you are using PASE for the first time, we recommend that you use internal clustering.
distance_type	The method to calculate vector similarity. Default value: 0. Valid values: ■ 0: Euclidean distance. ■ 1: dot product. This method requires the normalization of vectors. The order of dot products is opposite to the order of Euclidean distances. Currently, only the Euclidean distance method is supported. Dot products can only be calculated after vectors are normalized. For more information, see Appendixes.
dimension	The number of dimensions. This parameter is mandatory.

Parameter	Description
base64_encoded	Specifies whether to use Base64 encoding. Default value: 0. Valid values: ■ 0: specifies to use the float4[] data type to represent the vector type. ■ 1: specifies to use the Base64-encoded float[] data type to represent the vector type.
clustering_params	For external clustering, this parameter specifies the directory of the external centroid file to use. For internal clustering, this parameter specifies the clustering algorithm to use. The format is as follows: `clustering_sample_ratio,k`. This parameter is mandatory. ■ clustering_sample_ratio: the sampling fraction with 1000 as its denominator. The value of this field is an integer within the (0, 1000] range. For example, if you set this field to 1, the system samples data from the dynamic list based on the 1/1000 sampling ratio before it performs k-means clustering. A larger value indicates higher query accuracy but slower index creation. We recommend that the total number of data records sampled do not exceed 100,000. ■ k: the number of centroids. A larger value indicates higher query accuracy but slower index creation. We recommend that you set this field to a value within the [100, 1000] range.

- HNSW

  Example:

```
CREATE INDEX hnsw_idx ON vectors_table
USING
 pase_hnsw(vector)
WITH
 (dim = 256, base_nb_num = 16, ef_build = 40, ef_search = 200, base64_encoded = 0);
```

The following table describes the parameters in the HNSW index.

Parameter	Description
dim	The number of dimensions. This parameter is mandatory.
base_nb_num	The number of neighbors to identify for an element. This parameter is mandatory. A larger value indicates higher query accuracy, but slower index creation and more storage space occupied. We recommend that you set this parameter to a value within the [16, 128] range.
ef_build	The heap length to use during index creation. This parameter is mandatory. A longer heap length indicates higher query accuracy but slower index creation. We recommend that you set this parameter to a value within the [40, 400] range.

Parameter	Description
ef_search	The heap length to use during query. This parameter is mandatory. A longer heap length indicates higher query accuracy but poorer query performance. You can specify this parameter when initiating the query request. Default value: 200.
base64_encoded	Specifies whether to use Base64 encoding. Default value: 0. Valid values: ■ 0: specifies to use the float4[] data type to represent the vector type. ■ 1: specifies to use the Base64-encoded float[] data type to represent the vector type.

4. Use one of the following indexes to query a vector:
   - IVFFlat index

     Example:

     ```
     SELECT id, vector <#> '1,1,1'::pase as distance
     FROM vectors_ivfflat
     ORDER BY
     vector <#> '1,1,1:10:0'::pase
     ASC LIMIT 10;
     ```

     > **Note**
     > - <#> is an operator used by the IVFFlat index.
     > - You must execute the ORDER BY statement to make the IVFFlat index take effect. The IVFFlat index allows vectors to be sorted in ascending order.
     > - The PASE data type requires three parameters to specify a vector. These parameters are separated with colons (:). For example, `1,1,1:10:0` includes three parameters: The first parameter specifies the vector to query. The second parameter specifies the query efficiency of IVFFlat with a value range of (0, 1000], in which a larger value indicates higher query accuracy but poorer query performance. The third parameter specifies the vector similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product (also referred to as inner product) method. The dot product method requires the normalization of vectors. The order of dot products is opposite to the order of Euclidean distances.

   - HNSW

     Example:

     ```
     SELECT id, vector <? > '1,1,1'::pase as distance
     FROM vectors_ivfflat
     ORDER BY
     vector <? > '1,1,1:100:0'::pase
     ASC LIMIT 10;
     ```

> **Note**
>
> - <? > is an operator used by the HNSW index.
> - You must execute the ORDER BY statement to make the HNSW index take effect. The HNSW index allows vectors to be sorted in ascending order.
> - The PASE data type requires three parameters to specify a vector. These parameters are separated with colons (:). For example, `1,1,1:10:0` includes three parameters: The first parameter specifies the vector to query. The second parameter specifies the query efficiency of HNSW with a value range of (0, ∞), in which a larger value indicates higher query accuracy but poorer query performance. We recommend that you set the second parameter to 40 and then test the value by small increases until you find the most suitable value for your business. The third parameter specifies the vector similarity calculation method, where the value 0 represents the Euclidean distance method and the value 1 represents the dot product (also referred to as inner product) method. The dot product method requires the normalization of vectors. The order of dot products is opposite to the order of Euclidean distances.

Appendixes

- Calculate the dot product of a vector.

  For this example, use the HNSW index to create a function:

```
CREATE OR REPLACE FUNCTION inner_product_search(query_vector text, ef integer, k integer, table_nam
e text) RETURNS TABLE (id integer, uid text, distance float4) AS $$
BEGIN
  RETURN QUERY EXECUTE format('
  select a.id, a.vector <? > pase(ARRAY[%s], %s, 1) AS distance from
  (SELECT id, vector FROM %s ORDER BY vector <? > pase(ARRAY[%s], %s, 0) ASC LIMIT %s) a
  ORDER BY distance DESC;', query_vector, ef,  table_name,  query_vector, ef, k);
END
$$
LANGUAGE plpgsql;
```

  > **Note** The dot product of a normalized vector is the same as its cosine value. Therefore, you can also follow this example to calculate the cosine value of a vector.

- Create the IVFFlat index from an external centroid file.

  This is an advanced feature. You must upload an external centroid file to the specified directory of the server and use this file to create the IVFFlat index. For more information, see the parameters of the IVFFlat index. The file format is as follows:

```
Number of dimensions|Number of centroids|Centroid vector dataset
```

  Example:

```
3|2|1,1,1,2,2,2
```

References

- Product Quantization for Nearest Neighbor Search

  Herv´e J´egou, Matthijs Douze, Cordelia Schmid. Product quantization for nearest neighbor search.

- Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs

  Yu.A.Malkov, D.A.Yashunin. Efficient and robust approximate nearest neighbor search using Hierarchical Navigable Small World graphs.

5.4. Use the roaringbitmap plug-in

This topic describes how to use the roaringbitmap plug-in provided by ApsaraDB RDS for PostgreSQL to improve query performance.

Prerequisites

Your RDS instance runs PostgreSQL 12.

Context

The Roaring bitmap algorithm divides 32-bit integers into 2^{16} chunks. Each chunk stores the 16 most significant digits and uses a container to store the 16 least significant digits. A Roaring bitmap stores containers in a dynamic array as primary indexes. Two types of containers are available: array containers for sparse chunks and bitmap containers for dense chunks. An array container can store up to 4,096 integers. A bitmap container can store more than 4,096 integers.

Roaring bitmaps can use this storage structure to rapidly retrieve specific values. Additionally, Roaring bitmaps provide bitwise operations such as AND, OR, and XOR between the two types of containers. Therefore, Roaring bitmaps can deliver excellent storage and computing performance.

Procedure

1. Create a plug-in. Example:

   ```
   CREATE EXTENSION roaringbitmap;
   ```

2. Create a table with roaringbitmap data. Example:

   ```
   CREATE TABLE t1 (id integer, bitmap roaringbitmap);
   ```

3. Call the rb_build function to insert roaringbitmap data. Example:

   ```
   -- Set the bit value of an array to 1. INSERT INTO t1 SELECT 1,RB_BUILD(ARRAY[1,2,3,4,5,6,7,8,9,200]);
   -- Set the bit values of multiple elements to 1 and aggregate the bit values into a Roaring bitmap. INSER
   T INTO t1 SELECT 2,RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
   ```

4. Perform bitwise operations such as OR, AND, XOR, and ANDNOT. Example:

   ```
   SELECT RB_OR(a.bitmap,b.bitmap) FROM (SELECT bitmap FROM t1 WHERE id = 1) AS a,(SELECT bitmap F
   ROM t1 WHERE id = 2) AS b;
   ```

5. Perform bitwise aggregate operations such as OR, AND, XOR, and BUILD to generate a new Roaring bitmap. Example:

```
SELECT RB_OR_AGG(bitmap) FROM t1;
SELECT RB_AND_AGG(bitmap) FROM t1;
SELECT RB_XOR_AGG(bitmap) FROM t1;
SELECT RB_BUILD_AGG(e) FROM GENERATE_SERIES(1,100) e;
```

6. Calculate the cardinality of the Roaring bitmap. The cardinality is the number of bits that are set to 1 in the Roaring bitmap. Example:

```
SELECT RB_CARDINALITY(bitmap) FROM t1;
```

7. Obtain the subscripts of the bits that are set to 1. Example:

```
SELECT RB_ITERATE(bitmap) FROM t1 WHERE id = 1;
```

Bitmap calculation functions

Function	Input	Output	Description	Example
rb_build	integer[]	roaringbitmap	Creates a Roaring bitmap from an integer array.	rb_build('{1,2,3,4,5}')
rb_and	roaringbitmap,roaringbitmap	roaringbitmap	Performs an AND operation.	rb_and(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_or	roaringbitmap,roaringbitmap	roaringbitmap	Performs an OR operation.	rb_or(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_xor	roaringbitmap,roaringbitmap	roaringbitmap	Performs an XOR operation.	rb_xor(rb_build('{1,2,3}'),rb_build('{3,4,5}'))

Function	Input	Output	Description	Example
rb_andnot	roaringbitmap,roaringbitmap	roaringbitmap	Performs an ANDNOT operation.	rb_andnot(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_cardinality	roaringbitmap	integer	Calculates the cardinality.	rb_cardinality(rb_build('{1,2,3,4,5}'))
rb_and_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an AND operation on two Roaring bitmaps.	rb_and_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_or_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an OR operation on two Roaring bitmaps.	rb_or_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_xor_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an XOR operation on two Roaring bitmaps.	rb_xor_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))

Function	Input	Output	Description	Example
rb_andnot_cardinality	roaringbitmap,roaringbitmap	integer	Calculates the cardinality from an ANDNOT operation on two Roaring bitmaps.	rb_andnot_cardinality(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_is_empty	roaringbitmap	boolean	Checks whether a Roaring bitmap is empty.	rb_is_empty(rb_build('{1,2,3,4,5}'))
rb_equals	roaringbitmap,roaringbitmap	boolean	Checks whether two Roaring bitmaps are the same.	rb_equals(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_intersect	roaringbitmap,roaringbitmap	boolean	Checks whether two Roaring bitmaps intersect.	rb_intersect(rb_build('{1,2,3}'),rb_build('{3,4,5}'))
rb_remove	roaringbitmap,integer	roaringbitmap	Removes an offset from a Roaring bitmap.	rb_remove(rb_build('{1,2,3}'),3)
rb_flip	roaringbitmap,integer,integer	roaringbitmap	Flips specific offsets in a Roaring bitmap.	rb_flip(rb_build('{1,2,3}'),2,3)

Function	Input	Output	Description	Example
rb_minimum	roaringbitmap	integer	Returns the smallest offset in a Roaring bitmap. If the Roaring bitmap is empty, the value -1 is returned.	rb_minimum(rb_build('{1,2,3}'))
rb_maximum	roaringbitmap	integer	Returns the largest offset in a Roaring bitmap. If the Roaring bitmap is empty, the value 0 is returned.	rb_maximum(rb_build('{1,2,3}'))
rb_rank	roaringbitmap,integer	integer	Returns the number of elements that are smaller than or equal to a specified offset in a Roaring bitmap.	rb_rank(rb_build('{1,2,3}'),3)
rb_iterate	roaringbitmap	setof integer	Returns a list of offsets from a Roaring bitmap.	rb_iterate(rb_build('{1,2,3}'))

Bitmap aggregate functions

Function	Input	Output	Description	Example
rb_build_agg	integer	roaringbitmap	Creates a Roaring bitmap from a group of offsets.	rb_build_agg(1)
rb_or_agg	roaringbitmap	roaringbitmap	Performs an OR aggregate operation.	rb_or_agg(rb_build('{1,2,3}'))

Function	Input	Output	Description	Example
rb_and_agg	roaringbitmap	roaringbitmap	Performs an AND aggregate operation.	`rb_and_agg(rb_build('{1,2,3}'))`
rb_xor_agg	roaringbitmap	roaringbitmap	Performs an XOR aggregate operation.	`rb_xor_agg(rb_build('{1,2,3}'))`
rb_or_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an OR aggregate operation on two Roaring bitmaps.	`rb_or_cardinality_agg(rb_build('{1,2,3}'))`
rb_and_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an AND aggregate operation on two Roaring bitmaps.	`rb_and_cardinality_agg(rb_build('{1,2,3}'))`
rb_xor_cardinality_agg	roaringbitmap	integer	Calculates the cardinality from an XOR aggregate operation on two Roaring bitmaps.	`rb_xor_cardinality_agg(rb_build('{1,2,3}'))`

5.5. Use the varbitx plug-in

The varbit plug-in that is provided in the PostgreSQL Community edition supports only simple BIT-type operation functions. The varbitx plug-in is an extension of the varbit plug-in. The varbitx plug-in is provided in ApsaraDB RDS for PostgreSQL to support more BIT-type operations in more scenarios. These scenarios include real-time user profile recommendation, access control advertising, and ticketing.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 11
- PostgreSQL 10

Functions supported

Function	Description
get_bit (varbit a, int b, int c) returns varbit	Obtains a specified number c of bits that start at position b and returns a VARBIT-type string. For example, get_bit('111110000011', 3, 5) returns 11000.
set_bit_array (varbit a, int b, int c, int[] d) returns varbit	Changes the values of the bits that are specified by the subscript array d to values b (0 or 1), and fills the bits that exceed the original length with values c (0 or 1). For example, set_bit_array('111100001111', 0, 1, array[1,15]) returns 1011000011111110.
bit_count (varbit a, int b, int c, int d) returns int	Counts the number of values b (0 or 1) that start at position c among a specified number d of bits. The bits that are beyond the specified length are not counted. For example, bit_count('1111000011110000', 1, 5, 4) returns 1.
bit_count (varbit a, int b) returns int	Counts the total number of values b (0 or 1). For example, bit_count('1111000011110000', 1) returns 8.
bit_fill (int a, int b) returns varbit	Fills a specified number b of bits with values a (0 or 1). For example, bit_fill(0,10) returns 0000000000.
bit_rand (int a, int b, float c) returns varbit	Obtains a random percentage c of bits from a specified number a of bits, and randomly fills the obtained bits with values b (0 or 1). For example, bit_rand(10, 1, 0.3) may return 0101000001.
bit_posite (varbit a, int b, boolean c) returns int[]	Returns a subscript array. The subscript array indicates the positions of the bits whose values are b (0 or 1). Subscripts start at 0. If the value of c is true, subscripts are returned in a positive sequence. If the value of c is false, subscripts are returned in a negative sequence. For example, bit_posite ('11110010011', 1, true) returns [0,1,2,3,6,9,10], and bit_posite ('11110010011', 1, false) returns [10,9,6,3,2,1,0].

Function	Description
bit_posite (varbit a, int b, int c, boolean d) returns int[]	Returns a subscript array. The subscript array indicates the positions of the bits whose values are b (0 or 1). The number of subscripts in the subscript array is c. Subscripts start at 0. If the value of d is true, subscripts are returned in a positive sequence. If the value of d is false, subscripts are returned in a negative sequence. For example, bit_posite ('11110010011', 1, 3, true) returns [0,1,2], and bit_posite ('11110010011', 1, 3, false) returns [10,9,6].
get_bit_array (varbit a, int b, int c, int d) returns int[]	Obtains a specified number c of bits that start at position b, identifies the bits whose values are d (0 or 1) among the obtained bits, and returns a subscript array. The subscript array indicates the positions of the identified bits. For example, get_bit_array('111110000011', 3, 5, 1) returns [3,4] for the specified 11000 bit string.
get_bit_array (varbit a, int b, int[] c) returns int[]	Obtains the bits that are specified by a subscript array c, identifies the bits whose values are b (0 or 1) among the obtained bits, and returns a subscript array. The subscript array indicates the positions of the identified bits. The bits that are not included in the subscript array c are not counted. For example, get_bit_array('111110000011', 1, array[1,5,6,7,10,11]) returns [1,10,11].
set_bit_array (varbit a, int b, int c, int[] d, int e) returns varbit	Changes the values of the bits that are specified by a subscript array d to values b (0 or 1), and fills the bits that are beyond the original length with values c (0 or 1). The number of bits that are returned is e. For example, set_bit_array('111100001111', 1, 0, array[4,5,6,7], 2) returns 111111001111.
set_bit_array_record (varbit a, int b, int c, int[] d) returns (varbit,int[])	Changes the values of the bits that are specified by a subscript array d to values b (0 or 1), and fills the bits that are beyond the original length with values c (0 or 1). This function not only returns a bit string but also a subscript array. The subscript array indicates the positions of the bits whose values are changed. For example, set_bit_array_record('111100001111', 0, 1, array[1,15]) returns 1011000011111110 and [1,15].

Function	Description
set_bit_array_record (varbit a, int b, int c, int[] d, int e) returns (varbit,int[])	Changes the values of the bits that are specified by a subscript array d to values b (0 or 1), and fills the bits that are beyond the original length with values c (0 or 1). This function not only returns a bit string but also a subscript array. The subscript array indicates the positions of the bits whose values are changed. This function returns results immediately after it changes values for a total of e bits. For example, set_bit_array_record('111100001111', 1, 0, array[1,4,5,6,7], 2) returns 111111001111 and [4,5].
bit_count_array (varbit a, int b, int[] c) returns int	Counts the number of values b (0 or 1) among the bits that are specified by a subscript array c. For example, bit_count_array('1111000011110000', 1, array[1,2,7,8]) returns 3.

Basic usage

- Create the varbitx plug-in.

  ```
  CREATE EXTENSION varbitx;
  ```

- Delete the varbitx plug-in.

  ```
  DROP EXTENSION varbitx;
  ```

- Call the functions that are supported by the varbitx plug-in.

  You can execute the SELECT <function> statement to call the functions.

  - Execute the following statement to call the bit_count function:

    ```
    select bit_count('1111000011110000', 1, 5, 4);
    ```

    The following result is returned:

    ```
     bit_count
    -----------
             1
    (1 row)
    ```

- Execute the following statement to call the set_bit_array_record function:

```
select set_bit_array_record('111100001111', 1, 0, array[1,4,5,6,7], 2);
```

The following result is returned:

```
 set_bit_array_record
-----------------------
(111111001111,"{4,5}")
(1 row)
```

For more information about the functions and their descriptions, see Functions supported.

5.6. Use the RDKit plug-in

This topic describes how to use the RDKit plug-in of ApsaraDB RDS for PostgreSQL to implement functions such as molecular computing and search.

Prerequisites

Your RDS instance runs PostgreSQL 12.

Context

RDKit supports two data types: the mol data type that is used to describe molecular types, and the fp data type that is used to describe molecular fingerprints. It allows for comparison computing, similarity computing based on the Tanimoto and Dice coefficients, and GiST indexing.

For more information about the SQL statements that are supported by RDKit, visit RDKit SQL.

Precautions

- Input and output functions based on the mol data type comply with the simplified molecular input line entry specification (SMILES).
- Input and output functions based on the fp data type comply with the bytea format that is used to store binary data.

Create the RDKit plug-in

```
postgres=# create extension rdkit ;
CREATE EXTENSION
```

Default parameter settings

```
postgres=# show rdkit.tanimoto_threshold ;
 rdkit.tanimoto_threshold
--------------------------
 0.5
(1 row)
postgres=# show rdkit.dice_threshold;
 rdkit.dice_threshold
----------------------
 0.5
(1 row)
```

Indexes supported

- B-tree and hash indexes are supported for comparison computing operations that are based on the mol and fp data types. Examples:

```
CREATE INDEX molidx ON pgmol (mol);
CREATE INDEX molidx ON pgmol (fp);
```

- GiST indexes are supported for the following operations that are based on the mol and fp data types: "mol % mol", "mol # mol", "mol @> mol", "mol <@ mol", "fp % fp", and "fp # fp." Example:

```
CREATE INDEX molidx ON pgmol USING gist (mol);
```

Sample functions

- The tanimoto_sml function calculates the degree of similarity based on the Tanimoto coefficient.

```
postgres=# \df tanimoto_sml
                          List of functions
 Schema |    Name     | Result data type | Argument data types | Type
--------+-------------+------------------+---------------------+------
 public | tanimoto_sml | double precision | bfp, bfp          | func
 public | tanimoto_sml | double precision | sfp, sfp          | func
(2 rows)
```

- The dice_sml function calculates the degree of similarity based on the Dice coefficient.

```
postgres=# \df dice_sml
                          List of functions
 Schema |   Name    | Result data type | Argument data types | Type
--------+-----------+------------------+---------------------+------
 public | dice_sml | double precision | bfp, bfp          | func
 public | dice_sml | double precision | sfp, sfp          | func
(2 rows)
```

- If the second argument is a substructure of the first argument, the substruct function returns the TRUE value.

```
postgres=# \df substruct
                          List of functions
 Schema |  Name   | Result data type | Argument data types | Type
--------+-----------+------------------+---------------------+------
 public | substruct | boolean          | mol, mol            | func
 public | substruct | boolean          | mol, qmol           | func
 public | substruct | boolean          | reaction, reaction  | func
(3 rows)
```

Basic operations

- `mol % mol` and `fp % fp`

  If the degree of similarity that is calculated based on the Tanimoto coefficient is less than the value of the rdkit.tanimoto_threshold GUC variable, the TRUE value is returned.

- `mol # mol` and `fp # fp`

  If the degree of similarity that is calculated based on the Dice coefficient is less than the value of the rdkit.dice_threshold GUC variable, the TRUE value is returned.

- `mol @> mol`

  If the left operand contains the right operand, the TRUE value is returned.

- `mol <@ mol`

  If the right operand contains the left operand, the TRUE value is returned.

6.Run cross-database queries

6.1. Read and write external data files by using oss_fdw

Alibaba Cloud allows you to use the oss_fdw plug-in to load data from an OSS bucket to a database on an ApsaraDB RDS for PostgreSQL or PPAS instance and write data from the database to the OSS bucket.

Prerequisites

The RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 9.4
- PostgreSQL 10

Example

```
# Create an oss_fdw plug-in for the RDS for PostgreSQL database.
create extension oss_fdw;  --- For an RDS for PPAS database, execute the select rds_manage_extension('crea
te','oss_fdw'); statement.
# Create a server.
CREATE SERVER ossserver FOREIGN DATA WRAPPER oss_fdw OPTIONS
    (host 'oss-cn-hangzhou.aliyuncs.com' , id 'xxx', key 'xxx',bucket 'mybucket');
# Create a foreign table for OSS.
CREATE FOREIGN TABLE ossexample
   (date text, time text, open float,
    high float, low float, volume int)
    SERVER ossserver
    OPTIONS ( filepath 'osstest/example.csv', delimiter ',' ,
       format 'csv', encoding 'utf8', PARSE_ERRORS '100');
# Create a table to store loaded data.
create table example
      (date text, time text, open float,
       high float, low float, volume int);
# Load data from the ossexample table to the example table.
insert into example select * from ossexample;
# oss_fdw estimates the data size in the OSS bucket and configures a query plan.
explain insert into example select * from ossexample;
                      QUERY PLAN
------------------------------------------------------------------------
 Insert on example  (cost=0.00..1.60 rows=6 width=92)
   ->  Foreign Scan on ossexample  (cost=0.00..1.60 rows=6 width=92)
       Foreign OssFile: osstest/example.csv.0
       Foreign OssFile Size: 728
(4 rows)
# Write data from the example table to the ossexample table.
insert into ossexample select * from example;
explain insert into ossexample select * from example;
                    QUERY PLAN
-------------------------------------------------------------------
 Insert on ossexample  (cost=0.00..16.60 rows=660 width=92)
   ->  Seq Scan on example  (cost=0.00..16.60 rows=660 width=92)
(2 rows)
```

For information about the parameters, see the following sections.

oss_fdw parameters

The oss_fdw plug-in uses a method similar to other Foreign Data Wrappers (FDWs) to encapsulate external data stored in OSS buckets. You can use oss_fdw to read data from OSS buckets. This process is similar to reading data from tables. oss_fdw provides unique parameters to connect to OSS buckets and parse file data.

> **Note**
> - oss_fdw can read and write the following types of files in OSS buckets: TEXT and CSV files, including TEXT and CSV files compressed by using gzip.
> - The value of each parameter must be enclosed in double quotation marks (") and cannot contain any unnecessary spaces.

CREATE SERVER parameters

Parameter	Description
ossendpoint	The internal OSS endpoint, which is used as the host address.
id oss	The account ID.
key oss	The account key.
bucket	The OSS bucket. You must create an OSS account before you configure this parameter.

The following fault tolerance parameters can be used for data import and export. If the network connectivity is poor, you can adjust these parameters to ensure successful import and export.

Parameter	Description
oss_connect_timeout	The timeout period of connection. Unit: seconds. Default value: 10.
oss_dns_cache_timeout	The timeout period of DNS cache. Unit: seconds. Default value: 60.
oss_speed_limit	The minimum transmission rate. Unit: bit/s. Default value: 1024 (1 Kbit/s).
oss_speed_time	The maximum time period to tolerate the minimum transmission rate. Unit: seconds. Default value: 15.

> **Note** If the default values of oss_speed_limit and oss_speed_time are used, a timeout occurs when the transmission rate is slower than 1 Kbit/s for 15 consecutive seconds.

CREATE FOREIGN TABLE parameters

Parameter	Description

Parameter	Description
filepath	The file name that contains the path in the OSS bucket. • The file name contains the path and does not contain the bucket name. • This parameter matches multiple files in the specified path in the OSS bucket, so you can load multiple files to the database at a time. • You can import files named in the format of filepath or filepath.x to the database. The values of x must be consecutive numbers starting from 1. For example, among the files named filepath, filepath.1, filepath.2, filepath.3, and filepath.5, the first four files are matched and imported. The filepath.5 file is not imported.
dir	The virtual file directory in the OSS bucket. • The value must end with a forward slash (/). • All files (excluding subfolders and files in the subfolders) in the specified virtual file directory are matched and imported to the database.
prefix	The path name prefix of the data file. The prefix does not support regular expressions. You can only configure one of the prefix, filepath, and dir parameters.
format	The file format. Set the value to csv.
encoding	The data encoding format. Common encoding formats in PostgreSQL are supported, such as UTF-8.
parse_errors	The fault-tolerant parsing mode. If an error occurs in the parsing process, the entire row of data is ignored.
delimiter	The column delimiter.
quote	The quote character for the file.
escape	The escape character for the file.
null	This parameter sets the value of the column matching a specified string to null. For example, null 'test' sets the value of the test column to null.
force_not_null	This parameter sets the value of the specified column to a non-null value. For example, if the value of the id column is empty, force_not_null 'id' sets the value of the id column to an empty string, instead of null.
compressiontype	The format of files to be read and written in the OSS bucket. • none: the uncompressed text files. This is the default value. • gzip: the gzip-compressed files.
compressionlevel	The compression level of files written to the OSS bucket. Valid values: 1 to 9. Default value: 6.

Note

- The filepath and dir parameters are configured in the OPTIONS field.
- You must specify either the filepath or dir parameter.
- The export only supports the dir parameter and does not support the filepath parameter.

CREATE FOREIGN TABLE parameters in the export

- oss_flush_block_size: the buffer size for data written to the OSS bucket at a time. Default value: 32 MB. Valid values: 1 MB to 128 MB.
- oss_file_max_size: the maximum file size for data written to the OSS bucket. Extra data is written to another file. Default value: 1024 MB. Valid values: 8 MB to 4000 MB.
- num_parallel_worker: the number of parallel compression threads in which data is written to the OSS bucket. Valid values: 1 to 8. Default value: 3.

Auxiliary functions

FUNCTION oss_fdw_list_file (relname text, schema text DEFAULT 'public')

- This function obtains the names and sizes of files that match a foreign table in the OSS bucket.
- The unit of the file size is bytes.

```
select * from oss_fdw_list_file('t_oss');
        name         |  size
---------------------------------+-----------
 oss_test/test.gz.1 | 739698350
 oss_test/test.gz.2 | 739413041
 oss_test/test.gz.3 | 739562048
(3 rows)
```

Auxiliary features

oss_fdw.rds_read_one_file: specifies the file that matches the foreign table when data is read. The foreign table only matches the specified file during data import.

Example: set oss_fdw.rds_read_one_file = 'oss_test/example16.csv.1';

```
set oss_fdw.rds_read_one_file = 'oss_test/test.gz.2';
select * from oss_fdw_list_file('t_oss');
        name         |  size
---------------------------------+-----------
 oss_test/test.gz.2 | 739413041
(1 rows)
```

oss_fdw usage notes

- oss_fdw is a foreign table plug-in developed based on the PostgreSQL FOREIGN TABLE framework.

- Data import efficiency varies based on the OSS configuration and PostgreSQL cluster resources, such as CPU, I/O, and memory.
- Make sure that the RDS for PostgreSQL instance resides in the same region as the OSS bucket. This ensures data import efficiency. For more information, see OSS domain names.
- If the error `ERROR: oss endpoint userendpoint not in aliyun white list` is reported when SQL statements are read from the foreign table, you can use the public OSS endpoint of the required region. For more information, see Regions and endpoints. If the problem persists, submit a ticket.

Error information

When an error occurs during the import or export, the following error information is recorded in logs:

- code: the HTTP status code of the failed 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. If you require assistance in solving a problem, you can submit a ticket containing req_id of the failed request to OSS developers.

For more information about the errors, see the following references. Timeout errors can be handled based on oss_ext parameters.

- Object Storage Service document center
- CREATE FOREIGN TABLE in PostgreSQL
- OSS error handling
- OSS error response

ID and key encryption

If the id and key parameters in CREATE SERVER are not encrypted, other users can obtain your ID and key in plaintext by executing the `select * from pg_foreign_server` statement. You can use symmetric encryption to hide the ID and key and use different keys for different instances to protect your data. However, to avoid incompatibility with instances of earlier versions, do not add data types as you do in Greenplum.

The encrypted ID and key are displayed as follows:

```
postgres=# select * from pg_foreign_server ;
 srvname  | srvowner | srvfdw | srvtype | srvversion | srvacl |                                                                                                srvoptions
----------+----------+--------+---------+------------+--------+-----------------------------------------------------------------------------------------------------------------
-------------------------------------------------------
---------------------------------
 ossserver |       10 |  16390 |         |            |        | {host=oss-cn-hangzhou-zmf.aliyuncs.com,id=MD5xxxxxxx,key=MD
5xxxxxxx,bucket=067862}
```

The encrypted string starts with MD5. The total length divided by 8 gets a remainder of 3. Encryption is not performed again when the exported data is imported. You cannot create a key and ID that starts with MD5.

6.2. Use mysql_fdw to read and write data to a MySQL database

This topic describes how to use the mysql_fdw plug-in of ApsaraDB RDS for PostgreSQL to read and write data to a database on an ApsaraDB RDS for MySQL instance or to a user-created MySQL database.

Prerequisites

- Your ApsaraDB for RDS instance runs PostgreSQL 10 or 12 based on standard or enhanced SSDs.
- Communication between your ApsaraDB RDS for PostgreSQL instance and the target MySQL database is normal. You can configure whitelists and firewalls to ensure proper communication. For more information, see Configure a whitelist for an ApsaraDB RDS for PostgreSQL instance and What do I do if I cannot connect an ECS instance to an ApsaraDB for RDS instance?

Context

PostgreSQL 9.6 and later support parallel computing. PostgreSQL 11 can complete queries by using joins among up to 1 billion data records in seconds. A number of users prefer to use PostgreSQL to build small-sized data warehouses and process highly concurrent access requests. PostgreSQL 13 is under development. It will support columnar storage engines that further improve analysis capabilities.

The mysql_fdw plug-in establishes a connection to synchronize data from a MySQL database to your ApsaraDB RDS for PostgreSQL instance.

Procedure

1. Create the mysql_fdw plug-in.

```
postgres=> create extension mysql_fdw;
CREATE EXTENSION
```

2. Define a MySQL server.

```
postgres=> CREATE SERVER <The name of the MySQL server>
postgres->     FOREIGN DATA WRAPPER mysql_fdw
postgres->     OPTIONS (host '<The endpoint used to connect to the MySQL server>', port '<The port us
ed to connect the MySQL server>');
CREATE SERVER
```

   Example:

```
postgres=> CREATE SERVER mysql_server
postgres->     FOREIGN DATA WRAPPER mysql_fdw
postgres->     OPTIONS (host 'rm-xxx.mysql.rds.aliyuncs.com', port '3306');
CREATE SERVER
```

3. Map the MySQL server to an account created on your ApsaraDB RDS for PostgreSQL instance. That account is used to read and write data to the target MySQL database on the MySQL server.

```
postgres=> CREATE USER MAPPING FOR <The username of the account to which the MySQL server is mapped>
SERVER <The name of the MySQL server>
OPTIONS (username '<The username used to log on to the target MySQL database>', password '<The password used to log on to the target MySQL database>');
CREATE USER MAPPING
```

Example:

```
postgres=> CREATE USER MAPPING FOR pgtest
SERVER mysql_server
OPTIONS (username 'mysqltest', password 'Test1234!') ;
CREATE USER MAPPING
```

4. Create a foreign MySQL table by using the account that you mapped to the MySQL server in the previous step.

> **Note** The field names in the foreign MySQL table must be the same as those in the target table of the target MySQL database. You can choose to create only the fields you want to query. For example, if the target table in the target MySQL database contains three fields, ID, NAME, and AGE, you only need to create two fields, ID and NAME, in the foreign MySQL table.

```
postgres=> CREATE FOREIGN TABLE <The name of the foreign MySQL table> (<The name of Field 1> <The data type of Field 1>,<The name of Field 2> <The data type of Field 2>...) server <The name of the MySQL server> options (dbname '<The name of the target MySQL database>', table_name '<The name of the target table in the target MySQL database>'); CREATE FOREIGN TABLE
```

Example:

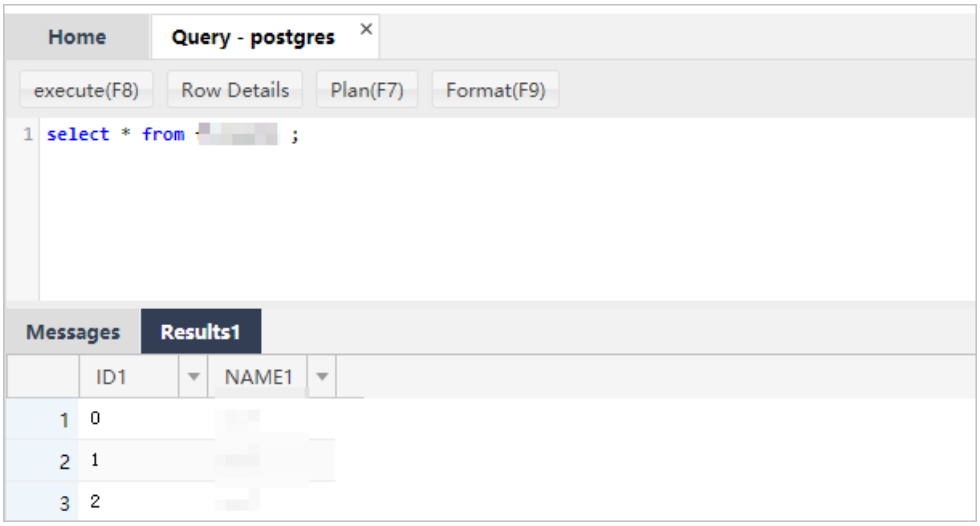
```
postgres=> CREATE FOREIGN TABLE ft_test (id1 int, name1 text) server mysql_server options (dbname 'test123', table_name 'test');
CREATE FOREIGN TABLE
```

What to do next

You can use the foreign MySQL table to test the performance of reading and writing data to the target MySQL database.

> **Note** Data can be written to the target table in the target MySQL database only when the target table is assigned a primary key. If the target table is not assigned a primary key, the following error is reported:
>
> ```
> ERROR: first column of remote table must be unique for INSERT/UPDATE/DELETE operation.
> ```

```
postgres=> select * from ft_test ;
postgres=> insert into ft_test values (2,'abc');
INSERT 0 1
postgres=> insert into ft_test select generate_series(3,100),'abc';
INSERT 0 98
postgres=> select count(*) from ft_test ;
 count
-------
  99
(1 row)
```



Check query plans to find out how the requests sent from your ApsaraDB RDS for PostgreSQL instance are executed to query data from the target MySQL database.

```
postgres=> explain verbose select count(*) from ft_test ;
                        QUERY PLAN
-------------------------------------------------------------------------------
 Aggregate  (cost=1027.50..1027.51 rows=1 width=8)
   Output: count(*)
   -> Foreign Scan on public.ft_test  (cost=25.00..1025.00 rows=1000 width=0)
      Output: id, info
      Remote server startup cost: 25
      Remote query: SELECT NULL FROM `test123`.`test`
(6 rows)
postgres=> explain verbose select id from ft_test where id=2;
                        QUERY PLAN
-------------------------------------------------------------------------
 Foreign Scan on public.ft_test  (cost=25.00..1025.00 rows=1000 width=4)
   Output: id
   Remote server startup cost: 25
   Remote query: SELECT `id` FROM `test123`.`test` WHERE ((`id` = 2))
(4 rows)
```

6.3. Use the log_fdw plug-in

This topic describes how to use the log_fdw plug-in to query the database logs of an RDS PostgreSQL instance.

Prerequisites

The RDS instance runs PostgreSQL 11.

Context

The log_fdw plug-in provides the following two functions:

- list_postgres_log_files(): lists all .csv log files.
- create_foreign_table_for_log_file(IN table_name text, IN log_server text, IN log_file text): creates a foreign table associated with a specific .csv log file.

Procedure

1. Create the log_fdw plug-in.

   ```
   postgres=> create extension log_fdw;
   CREATE EXTENSION
   ```

2. Create a definition for the log server.

   ```
   postgres=> create server <The name of the log server> foreign data wrapper log_fdw;
   ```

Example:

```
postgres=> create server log_server foreign data wrapper log_fdw;
CREATE SERVER
```

3. Invoke the list_postgres_log_files() function to list all .csv log files.

```
postgres=> select * from list_postgres_log_files() order by 1;
         file_name          | file_size_bytes
------------------------------------+-----------------
 postgresql-2020-01-10_095546.csv |        3794
 postgresql-2020-01-10_100336.csv |      318318
 postgresql-2020-01-11_000000.csv |      198437
 postgresql-2020-01-11_083546.csv |        4775
 postgresql-2020-01-13_030618.csv |        3347
```

4. Invoke the create_foreign_table_for_log_file(IN table_name text, IN log_server text, IN log_file text) function to create a foreign table associated with a specific .csv log file.

```
postgres=> select create_foreign_table_for_log_file('<The name to use for the foreign table>', '<The name of the log server>', '<The name of the .csv log file associated with the foreign table>');
```

Example:

```
postgres=> select create_foreign_table_for_log_file('ft1', 'log_server', 'postgresql-2020-01-13_030618.csv');
 create_foreign_table_for_log_file
-----------------------------------
 t
(1 row)
```

5. Query the foreign table to obtain the data of the .csv log file associated with it.

```
postgres=> select log_time, message from <The name of the foreign table to query> order by log_time desc limit 2;
```

Example:

```
postgres=> select log_time, message from ft1 order by log_time desc limit 2;
       log_time       |                    message
----------------------------+-----------------------------------------------------------------------
 2020-01-13 03:35:00.003+00 | cron job 1 completed: INSERT 0 1 1
 2020-01-13 03:35:00+00     | cron job 1 starting: INSERT INTO cron_test VALUES ('Hello World')
(2 rows)
```

Schema of a foreign table

```
postgres=> \d+ ft1
                                Foreign table "public.ft1"
     Column      |            Type             | Collation | Nullable | Default | FDW options | Storage  | Stats target | Description
-----------------+-----------------------------+-----------+----------+---------+-------------+----------+--------------+-------------
 log_time        | timestamp(3) with time zone |           |          |         |             | plain    |              |
 user_name       | text                        |           |          |         |             | extended |              |
 database_name   | text                        |           |          |         |             | extended |              |
 process_id      | integer                     |           |          |         |             | plain    |              |
 connection_from | text                        |           |          |         |             | extended |              |
 session_id      | text                        |           |          |         |             | extended |              |
 session_line_num| bigint                      |           |          |         |             | plain    |              |
 command_tag     | text                        |           |          |         |             | extended |              |
 session_start_time | timestamp with time zone |           |          |         |             | plain    |              |
 virtual_transaction_id | text                |           |          |         |             | extended |              |
 transaction_id  | bigint                      |           |          |         |             | plain    |              |
 error_severity  | text                        |           |          |         |             | extended |              |
 sql_state_code  | text                        |           |          |         |             | extended |              |
 message         | text                        |           |          |         |             | extended |              |
 detail          | text                        |           |          |         |             | extended |              |
 hint            | text                        |           |          |         |             | extended |              |
 internal_query  | text                        |           |          |         |             | extended |              |
 internal_query_pos | integer                  |           |          |         |             | plain    |              |
 context         | text                        |           |          |         |             | extended |              |
 query           | text                        |           |          |         |             | extended |              |
 query_pos       | integer                     |           |          |         |             | plain    |              |
 location        | text                        |           |          |         |             | extended |              |
 application_name| text                        |           |          |         |             | extended |              |
Server: log_server
FDW options: (filename 'postgresql-2020-01-13_030618.csv')
```

6.4. Use the tds_fdw plug-in

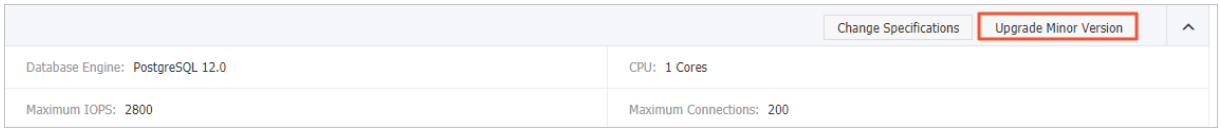
This topic describes the tds_fdw plug-in that is used to query data in other types of databases.

Prerequisites

The instance runs one of the following PostgreSQL versions:

- PostgreSQL 12 (kernel version 20200421 and later)
- PostgreSQL 11 (kernel version 20200402 and later)

Note To view the kernel version, perform the following steps: Log on to the ApsaraDB for RDS console, find the target RDS instance, and navigate to the **Basic Information** page. Then, in the **Configuration Information** section, check whether the **Upgrade Minor Version** button exists. If the button exists, click it to view the kernel version. If the button does not exist, it indicates that you are already using the latest kernel version. For more information, see Upgrade the kernel version of an ApsaraDB RDS for PostgreSQL instance.



Context

tds_fdw is a PostgreSQL foreign data wrapper that you can use to connect to databases. These databases use the Tabular Data Stream (TDS) protocol. Databases include Sybase and Microsoft SQL Server.

For more information, visit postgres_fdw.

Create a tds_fdw plug-in

After you have connected to an instance, execute the following statement to create a tds_fdw plug-in:

```
create extension tds_fdw;
```

Use tds_fdw

1. Execute the following statement to create a server:

   ```
   CREATE SERVER mssql_svr
    FOREIGN DATA WRAPPER tds_fdw
    OPTIONS (servername '127.0.0.1', port '1433', database 'tds_fdw_test', tds_version '7.1');
   ```

2. Create a foreign table. You can use one of the following methods to create a foreign table:
   - Execute the following statement to create a foreign table. You must specify the table_name parameter.

     ```
     CREATE FOREIGN TABLE mssql_table (
      id integer,
      data varchar)
      SERVER mssql_svr
      OPTIONS (table_name 'dbo.mytable', row_estimate_method 'showplan_all');
     ```

   - Execute the following statement to create a foreign table. You must specify the schema_name and table_name parameters.

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 data varchar)
 SERVER mssql_svr
 OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

- Execute the following statement to create a foreign table. You must specify the query parameter.

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 data varchar)
 SERVER mssql_svr
 OPTIONS (query 'SELECT * FROM dbo.mytable', row_estimate_method 'showplan_all');
```

- Execute the following statement to create a foreign table. You must specify a foreign column name.

```
CREATE FOREIGN TABLE mssql_table (
 id integer,
 col2 varchar OPTIONS (column_name 'data'))
 SERVER mssql_svr
 OPTIONS (schema_name 'dbo', table_name 'mytable', row_estimate_method 'showplan_all');
```

3. Execute the following statement to create a user mapping:

```
CREATE USER MAPPING FOR postgres
 SERVER mssql_svr
 OPTIONS (username 'sa', password '123456');
```

4. Execute the following statement to import a schema from a foreign table:

```
IMPORT FOREIGN SCHEMA dbo
 EXCEPT (mssql_table)
 FROM SERVER mssql_svr
 INTO public
 OPTIONS (import_default 'true');
```

6.5. Use the oracle_fdw plug-in

This topic describes how to use the oracle_fdw plug-in to connect to an Oracle database. It also provides details about how to create a PostgreSQL table and synchronize data to an Oracle table.

Prerequisites

- Your RDS instance runs PostgreSQL 12 with the kernel version of 20200421 or later.

> **Note** You can execute the `show rds_supported_extensions;` statement to check whether the current kernel version supports the oracle_fdw plug-in. If the current kernel version does not support the oracle_fdw plug-in, you must first upgrade the kernel version.

- The Oracle client version is 11.2 or later.
- The Oracle server version is based on the Oracle client version. For more information, see Oracle documentation.

Context

oracle_fdw is a PostgreSQL plug-in that provides a Foreign Data Wrapper (FDW). It provides easy access to Oracle databases and allows you to synchronize data between PostgreSQL and Oracle.

For more information, see oracle_fdw.

Precautions

- If you want to execute the UPDATE or DELETE statements, you must set the key parameter to true for primary key columns when you create a foreign table. For more information, see Create a foreign table.
- The data types of columns in the foreign table must be identifiable and convertible for oracle_fdw. For more information about the conversion rules supported by oracle_fdw, see Data types.
- oracle_fdw can push the WHERE and ORDER BY clauses down to Oracle databases.
- oracle_fdw can push down join operations to Oracle databases. Pushdown has the following limits:
  - Both tables for a join must be defined in the same database mapping.
  - Joins between three or more tables cannot be pushed down.
  - Joins must be included in a SELECT statement.
  - Cross joins without join conditions cannot be pushed down.
  - If a join is pushed down, ORDER BY clauses are not pushed down.
- oracle_fdw supports PostGIS. After PostGIS is installed, oracle_fdw further supports the following spatial data types:
  - POINT
  - LINE
  - POLYGON
  - MULTIPOINT
  - MULTILINE
  - MULTIPOLYGON

Procedure

1. Create an oracle_fdw plug-in. The statement is as follows:

   ```
   CREATE EXTENSION oracle_fdw;
   ```

2. Create an Oracle database mapping. One of the following two statements can be used:

```
CREATE SERVER <Server name>
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (dbserver '//<Endpoint>:<Port>/<Database name>');
```

Example:

```
CREATE SERVER <Server name>
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (dbserver '//127.0.0.1:5432/oradbname');
```

```
CREATE SERVER oradb
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (host '<Endpoint>', port '<Port>', dbname '<Database name>');
```

Example:

```
CREATE SERVER oradb
FOREIGN DATA WRAPPER oracle_fdw
OPTIONS (host '127.0.0.1', port '5432', dbname 'oradbname');
```

3. Create a user mapping. The statement is as follows:

```
CREATE USER MAPPING
FOR <PostgreSQL username> SERVER <Mapping name>
OPTIONS (user '<Oracle database username>', password '<User password>');
```

> **Note** If you do not store the Oracle user credentials in your PostgreSQL database, set the user parameter to an empty string and provide external authorization credentials.

Example:

```
CREATE USER MAPPING
FOR pguser SERVER oradb
OPTIONS (user 'orauser', password 'orapwd');
```

4. Create a foreign table. Example:

```
CREATE FOREIGN TABLE oratab (
    id      integer OPTIONS (key 'true')  NOT NULL,
    text    character varying(30),
    floating  double precision  NOT NULL
) SERVER oradb OPTIONS (table 'ORATAB',
                schema 'ORAUSER',
                max_long '32767',
                readonly 'false',
                sample_percent, '100',
                prefetch, '200');
```

Note The structure of the foreign table must be consistent with that of the mapped Oracle table.

The following table describes the parameters in OPTIONS.

Parameter	Description
key	Specifies whether to set a column as a primary key column. Valid values: true and false. Default value: false. If you want to execute the UPDATE and DELETE statements, you must set the value to true for all primary key columns.
table	The name of the Oracle table. The value must be in uppercase, and this parameter must be specified. You can also use an Oracle SQL statement to define the value of the table parameter. Example: `OPTIONS (table '(SELECT col FROM tab WHERE val = ''string'')')`. In this case, do not use the schema parameter.
schema	The Oracle username for accessing a table that does not belong to the currently connected user. The value must be in uppercase.
max_long	The maximum length of columns that have the LONG, LONG RAW, or XMLTYPE data types in the Oracle table. Valid values: 1 to 1073741823. Default value: 32767.
readonly	Specifies whether the Oracle table is read-only. If the value is true, you cannot execute the INSERT, UPDATE, and DELETE statements.
sample_percent	The percentage of Oracle table blocks that are randomly selected to calculate PostgreSQL table statistics. Valid values: 0.000001 to 100. Default value: 100.
prefetch	The number of rows that are fetched for a single round-trip transmission between PostgreSQL and Oracle during a foreign table scan. Valid values: 0 to 1024. Default value: 200. The value 0 indicates that the prefetch function is disabled.

After you create the foreign table, you can use it to perform operations on the Oracle table. Basic SQL statements such as DELETE, INSERT, UPDATE, and SELECT are supported. Foreign table definitions can be imported. The statement is as follows:

```
IMPORT FOREIGN SCHEMA <ora_schema_name>
FROM SERVER <server_name>
INTO <schema_name>
OPTIONS (case 'lower');
```

> **Note** case has the following values:
>
> - keep: uses the same object names as those in Oracle. In most cases, the names are in uppercase.
> - lower: converts all object names to lowercase.
> - smart: converts only the object names that are in all uppercase to lowercase.

Delete oracle_fdw

Execute the following SQL statement to delete the oracle_fdw plug-in:

```
DROP EXTENSION oracle_fdw;
```

7.Use the dblink and postgre_fdw plug-ins for cross-database operations

This topic describes how to use the dblink and postgre_fdw plug-ins provided with PostgreSQL to manage tables across databases.

Context

ApsaraDB for RDS instances that run PostgreSQL based on standard or enhanced SSDs support the dblink and postgres_fdw plug-ins. You can use these plug-ins to manage tables across databases on instances that reside in the same VPC. These instances include user-created PostgreSQL instances. If you want to access an RDS for PostgreSQL instance that resides in a different VPC, you can use an ECS instance in your VPC to redirect access requests between the database instances.

To purchase an RDS instance that runs PostgreSQL 11 based on standard or enhanced SSDs.

Precautions

Before you perform cross-database operations, consider the following items:

- If a user-created ECS-based PostgreSQL instance resides in the same VPC as your RDS for PostgreSQL instance, you can directly manage tables across these database instances.
- An ECS instance in your VPC can be used to redirect access requests between database instances. This applies if you want to manage tables across your RDS for PostgreSQL instance and a user-created ECS-based PostgreSQL instance that resides in a different VPC.
- You can use the oracle_fdw or mysql_fdw plug-in to connect a user-created PostgreSQL instance and an Oracle or MySQL instance that reside in different VPCs.
- If you manage tables across databases on the same RDS for PostgreSQL instance, you must set the host parameter to localhost and the port parameter to the local port that is obtained by running the `show port` command.

Use dblink

1. Create the dblink plug-in.

   ```
   create extension dblink;
   ```

2. Create a dblink connection to a remote RDS for PostgreSQL instance that resides in the same VPC as your source database.

```
postgres=> select dblink_connect('<The name of the connection>', 'host=<The internal endpoint used to connect to the remote RDS instance> port=<The internal port used to connect to the remote RDS instance> user=<The username used to log on to the target database on the remote RDS instance> password=<The password used to log on to the target database on the remote RDS instance> dbname=<The name of the target database on the remote RDS instance>');
postgres=> SELECT * FROM dblink('<The name of the connection>', '<The SQL command to run>') as <The name of the table to manage>(<The name of the column to manage> <The type of the column to manage>);
```

Example:

```
postgres=> select dblink_connect('a', 'host=pgm-bpxxxxx.pg.rds.aliyuncs.com port=3433 user=testuser2 password=passwd1234 dbname=postgres');
postgres=> select * from dblink('a','select * from products') as T(id int,name text,price numeric); //Query a table on the remote RDS instance.
```

For more information, see dblink.

Use postgres_fdw

1. Create a database.

   ```
   postgres=> create database <The name of the database>; //Create a database.
   postgres=> \c <The name of the created database> //Switch to the database that you created.
   ```

   Example:

   ```
   postgres=> create database db1;
   CREATE DATABASE
   postgres=> \c db1
   ```

2. Create the postgres_fdw plug-in.

   ```
   db1=> create extension postgres_fdw;
   ```

3. Create a remote database server that can connect to a remote RDS for PostgreSQL instance that resides in the same VPC as your source database.

```
db1=> CREATE SERVER <The name of the remote database server>
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host '<The internal endpoint used to connect to the remote RDS instance>,port '<The internal port used to connect to the remote RDS instance>', dbname '<The name of the target database on the remote RDS instance>');
db1=> CREATE USER MAPPING FOR <The username used to log on to your source database>
    SERVER <The name of the created remote database server>
    OPTIONS (user '<The username used to log on to the target database on the remote RDS instance>', password '<The password used to log on to the target database on the remote RDS instance>');
```

Example:

```
db1=> CREATE SERVER foreign_server1
    FOREIGN DATA WRAPPER postgres_fdw
    OPTIONS (host 'pgm-bpxxxxx.pg.rds.aliyuncs.com', port '3433', dbname 'postgres');
CREATE SERVER
db1=> CREATE USER MAPPING FOR testuser
    SERVER foreign_server1
    OPTIONS (user 'testuser2', password 'passwd1234');
CREATE USER MAPPING
```

4. Import an external table.

```
db1=> import foreign schema public from server foreign_server1 into <The name of the schema used by the external table>;  //Import an external table.
db1=> select * from <The name of the schema used by the external table>. <The name of the external table>    //Query a remote table.
```

Example:

```
db1=> import foreign schema public from server foreign_server1 into ft;
IMPORT FOREIGN SCHEMA
db1=> select * from ft.products;
```

For more information, see postgres_fdw.

8.Use the hll plug-in

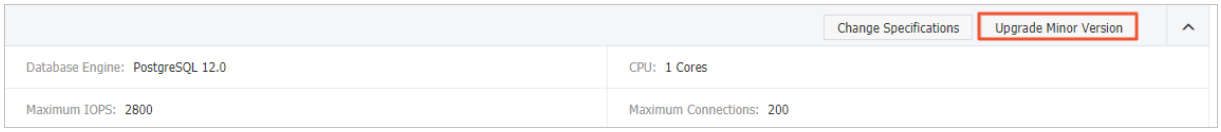
This topic describes the use of the HyperLogLog data type supported by the hll plug-in to estimate page views (PV) and unique visitors (UV).

Prerequisites

The instance runs one of the following PostgreSQL versions:

- PostgreSQL 12 (kernel version 20200421 and later)
- PostgreSQL 11 (kernel version 20200402 and later)

> **Note** To view the kernel version, perform the following steps: Log on to the ApsaraDB for RDS console, find the target RDS instance, and navigate to the **Basic Information** page. Then, in the **Configuration Information** section, check whether the **Upgrade Minor Version** button exists. If the button exists, click it to view the kernel version. If the button does not exist, it indicates that you are already using the latest kernel version. For more information, see Upgrade the kernel version of an ApsaraDB RDS for PostgreSQL instance.



Context

The hll plug-in supports an extendable, set-resembled data type HyperLogLog (hll) to estimate DISTINCT elements under a specified accuracy. For example, you can use 1,280 bytes of hll data to accurately estimate billions of DISTINCT elements. The hll plug-in is suitable for industries that need estimation analysis, such as Internet advertisement analysis to estimate PVs and UVs.

For more information about how to use the hll plug-in, visit postgresql-hll.

For more information about the detailed algorithm, visit HyperLogLog: the analysis of a near-optimal cardinality estimation algorithm.

Create an hll plug-in

After you connect to an instance, execute the following statement to create an hll plug-in:

```
CREATE EXTENSION hll;
```

Basic operations

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

  ```
  create table agg (id int primary key,userids hll);
  ```

- Execute the following statement to convert INT data to hll_hashval data:

  ```
  select 1::hll_hashval;
  ```

Basic operators

- The hll data type supports the following operators:
  - =
  - ! =
  - <>
  - ||
  - #

  Examples:

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

- The hll_hashval data type supports the following operators:
  - =
  - ! =
  - <>

  Examples:

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

Basic functions

- The hll plug-in supports hash functions such as hll_hash_boolean, hll_hash_smallint, and hll_hash_bigint. Examples:

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

- The hll plug-in supports the hll_add_agg function to convert the data type from INT to hll. Example:

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

- The hll plus-in supports the hll_union function to perform UNION operations on hll data. Example:

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

- The hll plus-in supports the hll_set_defaults function to set the accuracy. Example:

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

- The hll plug-in supports the hll_print function to display debug information. Example:

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

Example

```
create table access_date (acc_date date unique, userids hll);
insert into access_date select current_date, hll_add_agg(hll_hash_integer(user_id)) from generate_series(1
,10000) t(user_id);
insert into access_date select current_date-1, hll_add_agg(hll_hash_integer(user_id)) from generate_series
(5000,20000) t(user_id);
insert into access_date select current_date-2, hll_add_agg(hll_hash_integer(user_id)) from generate_series
(9000,40000) t(user_id);
postgres=# select #userids from access_date where acc_date=current_date;
   ? column?
------------------
 9725.85273370708
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-1;
   ? column?
------------------
 14968.6596883279
(1 row)
postgres=# select #userids from access_date where acc_date=current_date-2;
   ? column?
------------------
 29361.5209149911
(1 row)
```

9.Use the pg_cron plug-in

This topic describes how to use the pg_cron plug-in provided by RDS PostgreSQL to configure a scheduled task.

Prerequisites

Your RDS instance runs PostgreSQL 11.

> **Note** The pg_cron plug-in is only available to new RDS instances. If you want to use it in an existing RDS instance, you must submit a ticket.

Context

pg_cron is a CRON-based job scheduling plug-in. It uses the same syntax as standard CRON expressions, but can initiate PostgreSQL commands from databases.

Each scheduled task consists of the following two parts:

- Schedule

  The schedule to run the pg_cron plug-in. For example, the schedule specifies to run the pg_cron plug-in once every minute.

- Task

  The jobs to execute. Example: `select * from some_table` .

Syntax

The pg_cron plug-in follows the syntax used by standard CRON expressions. In this syntax, the wildcard * specifies to **run the pg_cron plug-in at any time** and a specific number specifies to only **run the pg_cron plug-in during the period specified by this number**.

```
┌──────────────────────────── Minute (0 to 59)
│ ┌──────────────────────────── Hour (0 to 23)
│ │ ┌──────────────────────────── Date (1 to 31)
│ │ │ ┌──────────────────────────── Month (1 to 12)
│ │ │ │ ┌──────────────────────────── Day within a week (0 to 6) (The value 0 indicates Sunday.)
│ │ │ │ │
│ │ │ │ │
│ │ │ │ │
* * * * *
```

Examples

- Create the pg_cron plug-in.

  ```
  CREATE EXTENSION pg_cron;
  ```

- Add jobs to the scheduled task.

```
-- Delete expired data at 3:30 am (GMT) every Saturday.
SELECT cron.schedule('30 3 * * 6', $$DELETE FROM events WHERE event_time < now() - interval '1 week'$$)
;
----------
-- Clear disks at 10:00 am (GMT) every day.
SELECT cron.schedule('0 10 * * *', 'VACUUM');
----------
-- Execute the specified script once every minute.
SELECT cron.schedule('* * * * *', 'select 1;');
----------
-- Execute the specified script at the 23th minute of every hour.
SELECT cron.schedule('23 * * * *', 'select 1;');
----------
-- Execute the specified script on the 4th day of every month.
SELECT cron.schedule('* * 4 * *', 'select 1;');
```

- View the current scheduled task.

```
SELECT * FROM cron.job;
 jobid | schedule  | command | nodename | nodeport | database | username | active
-------+-----------+----------+----------+----------+----------+----------+--------
    43 | 0 10 * * * | VACUUM; | localhost |    5433 | postgres | test    | t
```

- Delete a job from the scheduled task.

```
SELECT cron.unschedule(43);
```

> **Note** The number 43 indicates the ID of the job you want to delete.

10.Use the PL/Proxy plug-in for horizontal sharding

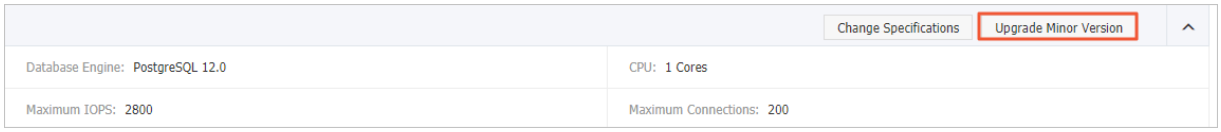
The PL/Proxy plug-in allows you to access your ApsaraDB RDS instance in CLUSTER or CONNECT mode.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 12 (with a minor engine version of 20200421 or later)
- PostgreSQL 11 (with a minor engine version of 20200402 or later)

> **Note** If you want to view the minor engine version, perform the following steps: Log on to the ApsaraDB RDS console, find your RDS instance and navigate to the **Basic Information** page. In the **Configuration Information** section of the page, check whether the **Upgrade Minor Version** button exists. If the button exists, you can click it to view and update the minor engine version. If the button does not exist, you are using the latest minor engine version. For more information, see Upgrade the kernel version of an ApsaraDB RDS for PostgreSQL instance.



Context

The PL/Proxy plug-in supports the following modes:

- CLUSTER

  This mode supports horizontal sharding and SQL replication.

- CONNECT

  This mode allows ApsaraDB RDS to route SQL requests to specified databases.

For more information about how to use the PL/Proxy plug-in, visit PL/Proxy.

Precautions

- You can directly manage tables across RDS instances that reside in the same virtual private cloud (VPC).
- An Elastic Compute Service (ECS) instance that resides in the same VPC as your RDS instance can serve as a proxy to redirect access requests for your RDS instance. This allows you to manage tables across RDS instances that reside in different VPCs.
- The number of data nodes that are served by the proxy node must be 2 to the power of n.

Test environment

Select an RDS instance as the proxy node and another two RDS instances as the data nodes. The following table provides details about the three RDS instances.

IP address	Node type	Instance name	Username
100.xx.xx.136	Proxy node	postgres	postgres
100.xx.xx.72	Data node	pl_db0	postgres
11.xx.xx.9	Data node	pl_db1	postgres

Create a PL/Proxy plug-in

Execute the following statement to create a PL/Proxy plug-in:

```
create extension plproxy
```

Create a PL/Proxy cluster

Note If you use the CONNECT mode, you can skip the operations that are described in this section.

1. Create a PL/Proxy cluster and specify the names, IP addresses, and ports of the RDS instances that you want to connect as data nodes in the cluster. Example:

```
postgres=# CREATE SERVER cluster_srv1 FOREIGN DATA WRAPPER plproxy
postgres-# OPTIONS (
postgres(#      connection_lifetime '1800',
postgres(#      disable_binary '1',
postgres(#      p0 'dbname=pl_db0 host=100.xxx.xxx.72 port=5678',
postgres(#      p1 'dbname=pl_db1 host=11.xxx.xxx.9 port=5678'
postgres(#      );
CREATE SERVER
```

2. Grant the permissions on the created PL/Proxy cluster to the postgres user. Example:

```
postgres=# grant usage on FOREIGN server cluster_srv1 to postgres;
GRANT
```

3. Create a user mapping. Example:

```
postgres=> create user mapping for postgres server cluster_srv1 options (user 'postgres');
CREATE USER MAPPING
```

Create a test table

Create a test table named users on each data node. Example:

```
create table users(userid int, name text);
```

Test the CLUSTER mode

To test horizontal sharding, perform the following steps:

1. Create a function that is used to insert data on each data node. Example:

```
pl_db0=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
pl_db0-> RETURNS integer AS $$
pl_db0$>     INSERT INTO users (userid, name) VALUES ($1,$2);
pl_db0$>     SELECT 1;
pl_db0$> $$ LANGUAGE SQL;
CREATE FUNCTION
pl_db1=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
pl_db1-> RETURNS integer AS $$
pl_db1$>     INSERT INTO users (userid, name) VALUES ($1,$2);
pl_db1$>     SELECT 1;
pl_db1$> $$ LANGUAGE SQL;
CREATE FUNCTION
```

2. Create a function that is used to insert data on the proxy node. This function has the same name as the function that is used to insert data on each data node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION insert_user(i_id int, i_name text)
postgres-> RETURNS integer AS $$
postgres$>    CLUSTER 'cluster_srv1';
postgres$>    RUN ON ANY;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

3. Create a function that is used to read data on the proxy node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>    CLUSTER 'cluster_srv1';
postgres$>    RUN ON ALL ;
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

4. Insert 10 test records on the proxy node. Example:

```
SELECT insert_user(1001, 'Sven');
SELECT insert_user(1002, 'Marko');
SELECT insert_user(1003, 'Steve');
SELECT insert_user(1004, 'lottu');
SELECT insert_user(1005, 'rax');
SELECT insert_user(1006, 'ak');
SELECT insert_user(1007, 'jack');
SELECT insert_user(1008, 'molica');
SELECT insert_user(1009, 'pg');
SELECT insert_user(1010, 'oracle');
```

5. View the data on each data node. The function that is used to insert data contains the RUN ON ANY statement. This statement randomly inserts data into either data node. Therefore, you may find the following data on the two data nodes:

```
pl_db0=> select * from users;
 userid | name
--------+--------
   1001 | Sven
   1003 | Steve
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
   1008 | molica
   1009 | pg
(8 rows)
pl_db1=> select * from users;
 userid | name
--------+--------
   1002 | Marko
   1010 | oracle
(2 rows)
```

> **Note** The query results indicate that the 10 data records are unevenly distributed between the two data nodes.

6. Invoke the function that is used to read data on the proxy node. This function contains the RUN ON ALL statement that reads data from both data nodes. Example:

```
postgres=> SELECT USERID,NAME FROM GET_USER_NAME();
 userid |  name
--------+--------
   1001 | Sven
   1003 | Steve
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
   1008 | molica
   1009 | pg
   1002 | Marko
   1010 | oracle
(10 rows)
```

To test SQL replication, perform the following steps:

1. Create a function that is used to truncate the users table on each node. Example:

```
pl_db0=> CREATE OR REPLACE FUNCTION trunc_user()
pl_db0-> RETURNS integer AS $$
pl_db0$>      truncate table users;
pl_db0$>      SELECT 1;
pl_db0$> $$ LANGUAGE SQL;
CREATE FUNCTION
pl_db1=> CREATE OR REPLACE FUNCTION trunc_user()
pl_db1-> RETURNS integer AS $$
pl_db1$>      truncate table users;
pl_db1$>      SELECT 1;
pl_db1$> $$ LANGUAGE SQL;
CREATE FUNCTION
postgres=> CREATE OR REPLACE FUNCTION trunc_user()
postgres-> RETURNS SETOF integer AS $$
postgres$>    CLUSTER 'cluster_srv1';
postgres$>    RUN ON ALL;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

2. Invoke the function that is used to truncate data on the proxy node. Example:

```
postgres=> SELECT TRUNC_USER();
 trunc_user
------------
          1
          1
(2 rows)
```

3. Create a function that is used to insert data on the proxy node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION insert_user_2(i_id int, i_name text)
postgres-> RETURNS SETOF integer AS $$
postgres$>    CLUSTER 'cluster_srv1';
postgres$>    RUN ON ALL;
postgres$> TARGET insert_user;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
```

4. Insert four test records into the proxy node. Example:

```
SELECT insert_user_2(1004, 'lottu');
SELECT insert_user_2(1005, 'rax');
SELECT insert_user_2(1006, 'ak');
SELECT insert_user_2(1007, 'jack');
```

5. View the data on each data node. Example:

```
pl_db0=> select * from users;
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
pl_db1=> select * from users;
 userid | name
--------+-------
   1004 | lottu
   1005 | rax
   1006 | ak
   1007 | jack
(4 rows)
```

> **Note** The data is the same on each data node. This indicates that SQL replication is successful.

6. Query data on the proxy node. You need only to execute the RUN ON ANY statement that randomly reads data from either data node. Example:

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name_2()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>    CLUSTER 'cluster_srv1';
postgres$>    RUN ON ANY ;
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
postgres=> SELECT USERID,NAME FROM GET_USER_NAME_2();
 userid | name
--------+-------
  1004 | lottu
  1005 | rax
  1006 | ak
  1007 | jack
(4 rows)
```

Test the CONNECT mode

When you use the CONNECT mode, you can access other RDS instances from the proxy node. Examples:

```
postgres=> CREATE OR REPLACE FUNCTION get_user_name_3()
postgres-> RETURNS TABLE(userid int, name text) AS $$
postgres$>    CONNECT 'dbname=pl_db0 host=100.81.137.72 port=56789';
postgres$> SELECT userid,name FROM users;
postgres$> $$ LANGUAGE plproxy;
CREATE FUNCTION
postgres=> SELECT USERID,NAME FROM GET_USER_NAME_3();
 userid | name
--------+-------
  1004 | lottu
  1005 | rax
  1006 | ak
  1007 | jack
(4 rows)
```

11.Fuzzy query (PG_ bigm)

The pg_bigm plug-in that is provided by ApsaraDB RDS for PostgreSQL supports full-text search. It allows you to create a 2-gram Generalized Inverted Index (GIN) index that is used to expedite full-text search queries.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

Differences between pg_bigm and pg_trgm

The pg_trgm plug-in is also provided by ApsaraDB RDS for PostgreSQL. However, it uses a 3-gram model to implement full-text search. The pg_bigm plug-in is developed based on the pg_trgm plug-in. The following table describes the differences between the two plug-ins.

Functionality	pg_trgm	pg_bigm
Phrase matching model	3-gram	2-gram
Index types	GIN and GiST	GIN
Operators	`LIKE` | `ILIKE` | `~` | `~*`	`LIKE`
Non-alphabet full-text search	Not supported	Supported
Full-text search with keywords that contain 1 to 2 characters	Slow	Fast
Similarity search	Supported	Supported
Maximum indexed column length	238,609,291 bytes (approximately equal to 228 MB)	107,374,180 bytes (approximately equal to 102 MB)

Precautions

- The length of the column on which you create a GIN index cannot exceed 107,374,180 bytes (approximately equal to 102 MB).
- If the data in your RDS instance is not encoded by using ASCII, we recommend that you change the encoding format to UTF8.

> **Note** To query the encoding format of your RDS instance, run the `select pg_encoding_to_char(encoding) from pg_database where datname = current_database();` command.

Basic operations

- Create the pg_bigm plug-in.

```
postgres=> create extension pg_bigm;
CREATE EXTENSION
```

- Create a GIN index.

```
postgres=> CREATE TABLE pg_tools (tool text, description text);
CREATE TABLE
postgres=> INSERT INTO pg_tools VALUES ('pg_hint_plan', 'Tool that allows a user to specify an optimizer HINT to PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_dbms_stats', 'Tool that allows a user to stabilize planner statistics in PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_bigm', 'Tool that provides 2-gram full text search capability in PostgreSQL');
INSERT 0 1
postgres=> INSERT INTO pg_tools VALUES ('pg_trgm', 'Tool that provides 3-gram full text search capability in PostgreSQL');
INSERT 0 1
postgres=> CREATE INDEX pg_tools_idx ON pg_tools USING gin (description gin_bigm_ops);
CREATE INDEX
postgres=> CREATE INDEX pg_tools_multi_idx ON pg_tools USING gin (tool gin_bigm_ops, description gin_bigm_ops) WITH (FASTUPDATE = off);
CREATE INDEX
```

- Run a full-text search query.

```
postgres=> SELECT * FROM pg_tools WHERE description LIKE '%search%';
 tool  |                  description
---------+-----------------------------------------------------------------
 pg_bigm | Tool that provides 2-gram full text search capability in PostgreSQL
 pg_trgm | Tool that provides 3-gram full text search capability in PostgreSQL
(2 rows)
```

- Run a similarity search query by using the `=%` operator.

```
postgres=> SET pg_bigm.similarity_limit TO 0.2;
SET
postgres=> SELECT tool FROM pg_tools WHERE tool =% 'bigm';
 tool
---------
 pg_bigm
 pg_trgm
(2 rows)
```

- Delete the pg_bigm plug-in.

```
postgres=> drop extension pg_bigm;
DROP EXTENSION
```

Basic functions

- likequery
  - Purpose: This function is used to generate a string that can be identified based on the LIKE keyword.
  - Request parameters: This function contains one request parameter. The data type for this parameter is STRING.
  - Return value: This function returns a string that can be identified based on the LIKE keyword.
  - Implementation:
    - Add a percent sign (`%`) preceding and following the keyword.
    - Use a backward slash (`\`) to escape the percent sign (`%`).
  - Example:

```
postgres=> SELECT likequery('pg_bigm has improved the full text search performance by 200%');
                    likequery
-----------------------------------------------------------------
 %pg\_bigm has improved the full text search performance by 200\%%
(1 row)
postgres=> SELECT * FROM pg_tools WHERE description LIKE likequery('search');
 tool  |                   description
---------+-----------------------------------------------------------------
 pg_bigm | Tool that provides 2-gram full text search capability in PostgreSQL
 pg_trgm | Tool that provides 3-gram full text search capability in PostgreSQL
(2 rows)
```

- show_bigm
  - Purpose: This function is used to obtain all of the 2-gram elements that comprise a string.
  - Request parameters: This function contains one request parameter. The data type for this parameter is STRING.

- Return value: This parameter returns an array that consists of all the 2-gram elements of a string.
- Implementation:
  - Add a space preceding and following the string.
  - Identify all of the 2-gram elements in the string.
- Example:

```
postgres=> SELECT show_bigm('full text search');
                    show_bigm
--------------------------------------------------------------
{" f"," s"," t",ar,ch,ea,ex,fu,"h ","l ",ll,rc,se,"t ",te,ul,xt}
(1 row)
```

- bigm_similarity
  - Purpose: This function is used to obtain the similarity between two strings.
  - Request parameters: This function contains two request parameters. The data types for these parameters are STRING.
  - Return value: This function returns a floating-point number. The number indicates the similarity between the two strings.
  - Implementation:
    - Identify the 2-gram elements that are included in both the two strings.
    - The return value is within the range from 0 to 1. The value 0 indicates that the two strings are completely different. The value 1 indicates that the two strings are identical.

> **Note**
> - This function adds a space preceding and following each string. Therefore, the similarity between the `ABC` string and the `B` string is 0, and the similarity between the `ABC` string and the `A` string is 0.25.
> - This function supports case sensitivity. For example, it determines that the similarity between the `ABC` string and the `abc` string is 0.

- Example:

```
postgres=> SELECT bigm_similarity('full text search', 'text similarity search');
 bigm_similarity
-----------------
       0.5714286
(1 row)
postgres=> SELECT bigm_similarity('ABC', 'A');
 bigm_similarity
-----------------
            0.25
(1 row)
postgres=> SELECT bigm_similarity('ABC', 'B');
 bigm_similarity
-----------------
               0
(1 row)
postgres=> SELECT bigm_similarity('ABC', 'abc');
 bigm_similarity
-----------------
               0
(1 row)
```

- pg_gin_pending_stats
  - Purpose: This function is used to obtain the number of pages and the number of tuples in the pending list of a GIN index.
  - Request parameters: This function contains one parameter. This parameter specifies the name or OID of the GIN index.
  - Return value: This function returns two values: the number of pages in the pending list of the GIN index and the number of tuples in the pending list of the GIN index.

    > **Note** If you set the FASTUPDATE parameter to False for a GIN index, the GIN index does not have a pending list. In this case, this function returns two values 0.

  - Example:

```
postgres=> SELECT * FROM pg_gin_pending_stats('pg_tools_idx');
 pages | tuples
-------+--------
     0 |      0
(1 row)
```

Behavior control

- pg_bigm.last_update

  This parameter indicates the last date when the pg_bigm plug-in was updated. This parameter is read-only. You cannot reconfigure this parameter.

  Example:

  ```
  SHOW pg_bigm.last_update;
  ```

- pg_bigm.enable_recheck

  This parameter specifies whether to perform a recheck.

  > **Note** We recommend that you retain the default value ON. This allows you to obtain accurate query results.

  Example:

```
postgres=> CREATE TABLE tbl (doc text);
CREATE TABLE
postgres=> INSERT INTO tbl VALUES('He is awaiting trial');
INSERT 0 1
postgres=> INSERT INTO tbl VALUES('It was a trivial mistake');
INSERT 0 1
postgres=> CREATE INDEX tbl_idx ON tbl USING gin (doc gin_bigm_ops);
CREATE INDEX
postgres=> SET enable_seqscan TO off;
SET
postgres=> EXPLAIN ANALYZE SELECT * FROM tbl WHERE doc LIKE likequery('trial');
                                QUERY PLAN
-------------------------------------------------------------------------------------------------------------------
 Bitmap Heap Scan on tbl  (cost=20.00..24.01 rows=1 width=32) (actual time=0.020..0.021 rows=1 loops=1)
  Recheck Cond: (doc ~~ '%trial%'::text)
  Rows Removed by Index Recheck: 1
  Heap Blocks: exact=1
  ->  Bitmap Index Scan on tbl_idx  (cost=0.00..20.00 rows=1 width=0) (actual time=0.013..0.013 rows=2 loo
ps=1)
        Index Cond: (doc ~~ '%trial%'::text)
 Planning Time: 0.117 ms
 Execution Time: 0.043 ms
(8 rows)
postgres=>
postgres=> SELECT * FROM tbl WHERE doc LIKE likequery('trial');
       doc
----------------------
 He is awaiting trial
(1 row)
postgres=> SET pg_bigm.enable_recheck = off;
SET
postgres=> SELECT * FROM tbl WHERE doc LIKE likequery('trial');
        doc
--------------------------
 He is awaiting trial
 It was a trivial mistake
(2 rows)
```

- pg_bigm.gin_key_limit

  This parameter specifies the maximum number of 2-gram elements that can be used for a full-text search query. The default value is 0, which indicates that all 2-gram elements are used.

> **Note** If the use of all 2-gram elements triggers a performance decrease, you can decrease the value of this parameter.

- pg_bigm.similarity_limit

  This parameter specifies the threshold for similarity. The tuples whose similarity exceeds the specified threshold are returned as similarity search results.

12.Use the wal2json plug-in

This topic describes how to use the wal2json plug-in provided by RDS PostgreSQL to export logical log records as a file in JSON format.

Prerequisites

- Your RDS instance runs PostgreSQL 11/12.
- The wal_level parameter is set to logical. For more information, see Reconfigure parameters for an RDS PostgreSQL instance.

Context

wal2json is a logical decoding plug-in. It has access to tuples generated by INSERT and UPDATE statements and can parse log records produced by write-ahead logging (WAL).

The wal2json plug-in produces a JSON object for each transaction. All new and old tuples are available in the JSON object. In addition, there are options to include properties such as transaction timestamp, schema-qualified, data type, and transaction ID. You can obtain JSON objects by executing SQL statements. For more information, see Execute SQL statements to obtain JSON objects.

Execute SQL statements to obtain JSON objects

1. Use DMS to log on to an ApsaraDB RDS for PostgreSQL instance.
2. Execute the following statements to create a table and initialize the wal2json plug-in:

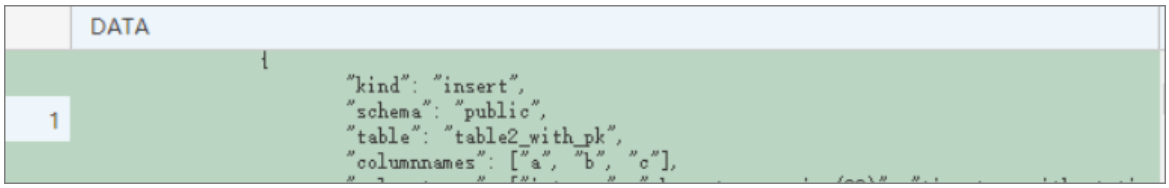
```
CREATE TABLE table2_with_pk (a SERIAL, b VARCHAR(30), c TIMESTAMP NOT NULL, PRIMARY KEY(a, c));
CREATE TABLE table2_without_pk (a SERIAL, b NUMERIC(5,2), c TEXT);
SELECT 'init' FROM pg_create_logical_replication_slot('test_slot', 'wal2json');
```

3. Execute the following statements to change data:

```
BEGIN;
INSERT INTO table2_with_pk (b, c) VALUES('Backup and Restore', now());
INSERT INTO table2_with_pk (b, c) VALUES('Tuning', now());
INSERT INTO table2_with_pk (b, c) VALUES('Replication', now());
DELETE FROM table2_with_pk WHERE a < 3;
INSERT INTO table2_without_pk (b, c) VALUES(2.34, 'Tapir');
UPDATE table2_without_pk SET c = 'Anta' WHERE c = 'Tapir';
COMMIT;
```

4. Execute the following statement to produce logical log records in JSON format:

```
SELECT data FROM pg_logical_slot_get_changes('test_slot', NULL, NULL, 'pretty-print', '1');
```



Note If you want to stop producing logical log records and release the resources used, execute the following statement:

```
SELECT 'stop' FROM pg_drop_replication_slot('test_slot');
```

13.Use the ZomboDB plug-in to integrate with Elasticsearch

ZomboDB is a PostgreSQL extension plug-in. It supports the access methods that are provided by native PostgreSQL. It also provides powerful text search and analytics features by using Elasticsearch.

Prerequisites

Your ApsaraDB for RDS instance runs PostgreSQL 11.

Context

ZomboDB provides a full set of query languages to query relational data. You can also create ZomboDB indexes. In this case, ZomboDB takes over remote Elasticsearch indexes and ensures transactionally correct query results from text search.

ZomboDB allows you to use Elasticsearch without the need to handle synchronization or communication issues.

Create and delete the ZomboDB plug-in

- Create the plug-in

```
CREATE EXTENSION zombodb;
```

- Delete the plug-in

```
DROP EXTENSION zombodb;
```

Examples

1. Create a table.

```
CREATE TABLE products (
  id SERIAL8 NOT NULL PRIMARY KEY,
  name text NOT NULL,
  keywords varchar(64)[],
  short_summary text,
  long_description zdb.fulltext,
  price bigint,
  inventory_count integer,
  discontinued boolean default false,
  availability_date date
);
```

2. Create a ZomboDB index for the table.

```
CREATE INDEX idxproducts
      ON products
   USING zombodb ((products. *))
    WITH (url='localhost:9200/');
```

Note The WITH clause is followed by an Elasticsearch endpoint, which points to a running Elasticsearch cluster.

3. Query data by using the ZomboDB index.

```
SELECT *
 FROM products
 WHERE products ==> '(keywords:(sports OR box) OR long_description:"wooden away"~5) AND price:[10
00 TO 20000]';
```

Note For more information about the query syntax, see ZomboDB Documentation.

14.Use the sql_firewall plug-in

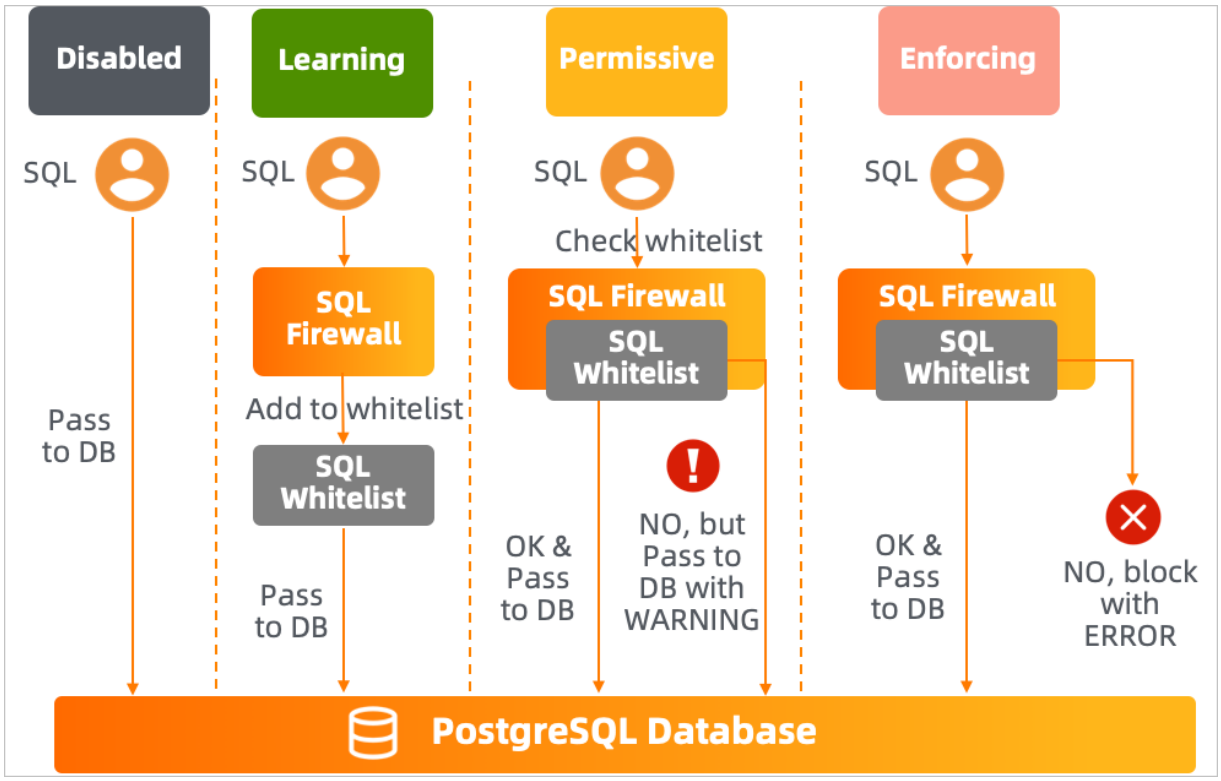
The sql_firewall plug-in is a database-level firewall that prevents against SQL injection. It learns defined SQL rules and stores the rules in your ApsaraDB RDS for PostgreSQL instance as a whitelist. User operations that do not comply with the rules are forbidden.

Prerequisites

Your RDS instance runs one of the following RDS editions:

- PostgreSQL 12
- PostgreSQL 11
- PostgreSQL 10

Learning, permissive, and enforcing modes



The sql_firewall plug-in supports the following modes:

- Learning: The plug-in records common SQL statements that are executed and adds them to a whitelist.
- Permissive: The plug-in checks SQL statements that will be executed. If the SQL statements are not in the whitelist, the plug-in executes the SQL statements but generates alerts.
- Enforcing: The plug-in checks SQL statements that will be executed. If the SQL statements are not in the whitelist, the plug-in does not execute the SQL statements and returns errors.

Procedure

1. Enable the learning mode of the sql_firewall plug-in. Wait for a specific period of time to ensure that the plug-in learns more SQL statements.
2. Switch the sql_firewall plug-in to the permissive mode. In this mode, the plug-in generates alerts

for SQL statements that are not in the whitelist. You can check whether these SQL statements are high-risky statements based on your business requirements. If these statements are not high-risky statements, switch to the learning mode and add these SQL statements to the whitelist.

3. Switch the sql_firewall plug-in to the enforcing mode. In this mode, the plug-in does not execute SQL statements that are not in the whitelist.

Operations

- Create the plug-in

```
create extension sql_firewall;
```

- Delete the plug-in

```
drop extension sql_firewall;
```

- Switch the mode

  In the ApsaraDB for RDS console, find the **sql_firewall.firewall** parameter. Modify the parameter value and restart your RDS instance. For more information, see Reconfigure parameters for an RDS PostgreSQL instance.

  Valid values for the **sql_firewall.firewall** parameter:

  - disable: disables the sql_firewall plug-in.
  - learning: enables the learning mode.
  - permissive: enables the permissive mode.
  - enforcing: enables the enforcing mode.

- Functionality functions
  - sql_firewall_reset()

    This function clears the whitelist. You can call this function only if you are authorized with the rds_superuser role and the enforcing mode is disabled.

  - sql_firewall_stat_reset()

    This function deletes statistics. You can call this function only if you are authorized with the rds_superuser role and the enforcing mode is disabled.

- View functions
  - sql_firewall.sql_firewall_statements

    This function returns all SQL statements in the whitelist of your RDS instance. This function also returns the number of times that each SQL statement is executed.

```
postgres=# select * from sql_firewall.sql_firewall_statements;
  userid | queryid  |          query          | calls
--------+------------+---------------------------------+-------
    10 | 3294787656 | select * from k1 where uid = ? ; |    4
(1 row)
```

- sql_firewall.sql_firewall_stat

  This function returns the number of alerts that are generated in permissive mode and the number of errors that are generated in enforcing mode. The first number is measured by sql_warning, and the second number is measured by sql_error.

```
postgres=# select * from sql_firewall.sql_firewall_stat;
 sql_warning | sql_error
-------------+-----------
           2 |         1
(1 row)
```

Examples

```
-- Permissive mode
  postgres=# select * from sql_firewall.sql_firewall_statements;
  WARNING:  Prohibited SQL statement
   userid |  queryid  |          query          | calls
  --------+------------+---------------------------------+-------
       10 | 3294787656 | select * from k1 where uid = 1; |     1
  (1 row)
  postgres=# select * from k1 where uid = 1;
   uid |   uname
  -----+-------------
     1 | Park Gyu-ri
  (1 row)
  postgres=# select * from k1 where uid = 3;
   uid |   uname
  -----+-----------
     3 | Goo Ha-ra
  (1 row)
  postgres=# select * from k1 where uid = 3 or 1 = 1;
  WARNING:  Prohibited SQL statement
   uid |    uname
  -----+---------------
     1 | Park Gyu-ri
     2 | Nicole Jung
     3 | Goo Ha-ra
     4 | Han Seung-yeon
     5 | Kang Ji-young
  (5 rows)
-- Enforcing mode
  postgres=# select * from k1 where uid = 3;
   uid |   uname
  -----+-----------
     3 | Goo Ha-ra
  (1 row)
  postgres=# select * from k1 where uid = 3 or 1 = 1;
  ERROR:  Prohibited SQL statement
  postgres=#
```

15.Use the failover slot feature for logical subscriptions

This topic describes the failover slot feature. This feature synchronizes all of the logical replication slots in your ApsaraDB RDS database system from the primary RDS instance to the secondary RDS instance.

Prerequisites

The primary RDS instance runs PostgreSQL 11.

Context

If you do not enable the failover slot feature, the logical replication slots in your database system cannot be automatically switched over to the new primary RDS instance in the event of a primary/secondary switchover. In this case, the logical subscriptions are disconnected. You must manually re-create the logical replication slots. The failover slot feature can synchronize the logical replication slots from the primary RDS instance to the secondary RDS instance. This ensures that the logical subscriptions remain connected.

> **Note** The failover slot feature supports only logical subscriptions. It does not support physical subscriptions.

You can enable the failover slot feature by setting the **rds_failover_slot_mode** parameter. After the setting is complete, the value of this parameter immediately takes effect and you do not need to restart the primary RDS instance. Valid values:

- sync: enables the failover slot feature in synchronous mode.
- async: enables the failover slot feature in asynchronous mode.
- off: disables the failover slot feature.

For more information about how to set this parameter, see Reconfigure parameters for an RDS PostgreSQL instance.

The following sections provide more information about the two modes.

Synchronous mode

In most cases, the synchronous mode ensures that no data from logical subscriptions is lost after a primary/secondary switchover. However, data may still be lost in some unexpected cases.

For example, if the secondary RDS instance remains disconnected for a long period of time or is being re-created, the latency between the primary and secondary RDS instances is high. If you perform a primary/secondary switchover at this time, data from the logical subscriptions may be lost. The lost data includes the data that is lost during the primary/secondary switchover. The lost data also includes the data that is generated on the new primary RDS instance.

To avoid data losses, you can set the **rds_priority_replication_force_wait** parameter to **on**. The default value of this parameter is **off**. After the setting is complete, the new value of this parameter immediately takes effect and you do not need to restart the primary RDS instance. After you set this parameter to on, the primary RDS instance does not send data to the subscriber until the secondary RDS instance is reconnected or re-created. However, this reduces the availability of logical subscriptions. Therefore, we recommend that you do not set this parameter to on.

Asynchronous mode

The asynchronous mode ensures that no data from logical subscriptions is lost after a primary/secondary switchover. However, duplicate data may be sent to the subscriber. This may cause inconsistent data.

If the inconsistent data affects your business, we recommend that you use the synchronous mode.

16.Use the pg_concurrency_control plug-in

ApsaraDB RDS for PostgreSQL provides a pg_concurrency_control plug-in to control concurrency of SQL statements.

Prerequisites

Your RDS instance runs PostgreSQL 10.

Parameters

Note The following parameters cannot be modified in the ApsaraDB RDS console. You can submit a ticket to modify them.

Parameter	Default value	Description
pg_concurrency_control.query_concurrency	0	Sets the maximum number of concurrent jobs in SELECT SQL statements. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for SELECT SQL statements.
pg_concurrency_control.bigquery_concurrency	0	Sets the maximum number of concurrent jobs in slow queries. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for slow queries. You can specify a statement as a slow query by using `hint "/*+bigsql*/"`. Example: `/*+bigsql*/select * from test;` The `select * from test;` statement is a slow query.
pg_concurrency_control.transaction_concurrency	0	Sets the maximum number of concurrent jobs for transaction blocks. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for transaction blocks.
pg_concurrency_control.autocommit_concurrency	0	Sets the maximum number of concurrent jobs in DML SQL statements. Valid values: 0 to 1024. Default value: 0. The default value indicates that concurrency control is disabled for DML SQL statements.
pg_concurrency_control.control_timeout	1s	Sets the maximum time to wait for a SELECT SQL statement, DML SQL statement, and transaction block. The minimum value is 30 ms, and the maximum value is 3s.

Parameter	Default value	Description
pg_concurrency_control.bigsql_control_timeout	1s	Sets the maximum time to wait for a slow query. The minimum value is 30 ms, and the maximum value is 3s.
pg_concurrency_control.timeout_action	TCC_break	Sets the action upon a timeout for a SELECT SQL statement, DML SQL statement, and transaction block. Valid values: • TCC_break: skips the statement in waiting and execute the statement that follows. • TCC_rollback: reports an error and rolls back the transaction. • TCC_wait: resets the timestamp after a timeout and continues to wait.
pg_concurrency_control.bigsql_timeout_action	TCC_wait	Sets the action upon a timeout for slow queries. Valid values: • TCC_break: skips the statement in waiting and execute the statement that follows. • TCC_rollback: reports an error and rolls back the transaction. • TCC_wait: resets the timestamp after a timeout and continues to wait.

Procedure

1. Run the following command to create the plug-in:

   ```
   create extension pg_concurrency_control;
   ```

2. Set the number of concurrent jobs to a value that is greater than 0 to enable concurrency control in the plug-in.

   For example, set the pg_concurrency_control.query_concurrency parameter to 10 to enable concurrency control for SELECT SQL statements. The methods to enable concurrency control for other types of statements are similar.

   > **Note** The parameters cannot be modified in the ApsaraDB RDS console. You can submit a ticket to modify them.

Example

Perform the following operations to enable concurrency control for custom SQL statements:

1. Run the following command to view information of the statement queue:

   ```
   select * from pg_concurrency_control_status();
   ```

   The system displays information similar to the following output:

```
 autocommit_count | bigquery_count | query_count | transaction_count
------------------+----------------+-------------+-------------------
                0 |              0 |           0 |                 0
(1 row)
```

2. Set the pg_concurrency_control.query_concurrency parameter to a value that is greater than 0, for example, 10.
3. Execute a slow query.

```
/*+ bigsql */ select pg_sleep(10);
```

4. Run the following command to view information of the statement queue again:

```
select * from pg_concurrency_control_status();
```

The system displays information similar to the following output:

```
 autocommit_count | bigquery_count | query_count | transaction_count
------------------+----------------+-------------+-------------------
                0 |              1 |           0 |                 0
(1 row)
```

> **Note** After the slow query is complete, the queue information is automatically cleared.

17.Use the RUM plug-in

This topic describes how to use the RUM plug-in of ApsaraDB RDS for PostgreSQL to run full-text searches.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 11
- PostgreSQL 10

Context

Generalized Inverted Index (GIN) allows you to run full-text searches by using the tsvector and tsquery data types. However, this may produce the following issues:

- Slow sorting

  ApsaraDB RDS for PostgreSQL can sort words only after it obtains the locations of the words. However, GIN does not store word locations. As a result, after ApsaraDB RDS for PostgreSQL runs a scan based on a GIN index, it must run another scan to retrieve the word locations.

- Slow queries for phrases

  GIN can search for phrases only after it obtains the locations of the phrases.

- Slow sorting of timestamps

  GIN does not store related information in indexes that contain morphemes. Therefore, an additional scan is required.

The RUM plug-in of ApsaraDB RDS for PostgreSQL is designed based on GIN. It allows you to store word or timestamp locations in RUM indexes.

However, the RUM plug-in requires more time than GIN to construct and insert indexes. This is because the RUM plug-in generates indexes based on write-ahead logging (WAL) logs and the generated RUM indexes contain more information than the keys that are used for encryption.

Universal operators

The RUM plug-in provides the following operators.

Operator	Data type	Description
tsvector <=> tsquery	float4	Returns the distances between the data objects of the tsvector type and those of the tsquery type.
timestamp <=> timestamp	float8	Returns the distance between two timestamps.
timestamp <=| timestamp	float8	Returns only the distance to the left-side timestamp.
timestamp |=> timestamp	float8	Returns only the distance to the right-side timestamp.

> **Note** The last three operators are also supported for the following data types: timestamptz, int2, int4, int8, float4, float8, money, and oid.

The following sections describe the functions that are provided by the RUM plug-in.

rum_tsvector_ops

- Supported data types: tsvector.
- Description: This function stores phrases of the tsvector data type along with the locations of the phrases. This function allows you to sort phrases by using the `<=>` operator. This function also allows you to search for phrases based on prefixes.
- Examples:
  i. Execute the following statements to create a table:

```
CREATE TABLE test_rum(t text, a tsvector);
CREATE TRIGGER tsvectorupdate
BEFORE UPDATE OR INSERT ON test_rum
FOR EACH ROW EXECUTE PROCEDURE tsvector_update_trigger('a', 'pg_catalog.english', 't');
INSERT INTO test_rum(t) VALUES ('The situation is most beautiful');
INSERT INTO test_rum(t) VALUES ('It is a beautiful');
INSERT INTO test_rum(t) VALUES ('It looks like a beautiful place');
```

  ii. Execute the following statement to create the RUM plug-in:

```
CREATE EXTENSION rum;
```

  iii. Execute the following statement to create an index:

```
CREATE INDEX rumidx ON test_rum USING rum (a rum_tsvector_ops);
```

  iv. Run the following two types of queries:

```
SELECT t, a <=> to_tsquery('english', 'beautiful | place') AS rank
  FROM test_rum
  WHERE a @@ to_tsquery('english', 'beautiful | place')
  ORDER BY a <=> to_tsquery('english', 'beautiful | place');
```

  The following result is returned:

```
           t           | rank
-------------------------------+---------
 It looks like a beautiful place | 8.22467
 The situation is most beautiful | 16.4493
 It is a beautiful             | 16.4493
(3 rows)
```

```
SELECT t, a <=> to_tsquery('english', 'place | situation') AS rank
  FROM test_rum
  WHERE a @@ to_tsquery('english', 'place | situation')
  ORDER BY a <=> to_tsquery('english', 'place | situation');
```

The following result is returned:

```
          t          | rank
---------------------------------+---------
 The situation is most beautiful | 16.4493
 It looks like a beautiful place | 16.4493
(2 rows)
```

rum_tsvector_hash_ops

- Supported data types: tsvector.
- Description: This function stores phrases of the tsvector data type along with the hash values and locations of the phrases. This function allows you to sort phrases by using the `<=>` operator. However, this function does not allow you to search for phrases based on prefixes.

rum_TYPE_ops

- Supported data types:
  - The `<` , `<=` , `=` , `>=` , `>` , and `<=>` operators support the following data types: int2, int4, int8, float4, float8, money, oid, time, timetz, date, interval, macaddr, inet, cidr, text, varchar, char, bytea, bit, varbit, numeric, timestamp, and timestamptz.
  - The `<=|` and `|=>` operators support the following data types: int2, int4, int8, float4, float8, money, oid, timestamp, and timestamptz.
- Description: This function can be used with the rum_tsvector_addon_ops, rum_tsvector_hash_addon_ops, and rum_anyarray_addon_ops functions.

rum_tsvector_addon_ops

- Supported data types: tsvector.
- Description: This function stores the word segmentation method that is used by the tsvector data type. This method also stores all the other word segmentation methods that are supported by the fields of the RUM plug-in.
- Examples:
  i. Execute the following statements to create a table with an index:

```
CREATE TABLE tsts (id int, t tsvector, d timestamp);
\copy tsts from 'rum/data/tsts.data'
CREATE INDEX tsts_idx ON tsts USING rum (t rum_tsvector_addon_ops, d)
  WITH (attach = 'd', to = 't');
```

  ii. Run the following query:

```
EXPLAIN (costs off)
  SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
```

The following result is returned:

```
                        QUERY PLAN
-----------------------------------------------------------------------------------
 Limit
  -> Index Scan using tsts_idx on tsts
       Index Cond: (t @@ '''wr'' & ''qh'''::tsquery)
       Order By: (d <=> 'Mon May 16 14:21:25 2016'::timestamp without time zone)
(4 rows)
```

iii. Run the following query:

```
SELECT id, d, d <=> '2016-05-16 14:21:25' FROM tsts WHERE t @@ 'wr&qh' ORDER BY d <=> '2016-05-16 14:21:25' LIMIT 5;
```

The following result is returned:

```
 id |            d            | ? column?
----+-------------------------------+---------------
 355 | Mon May 16 14:21:22.326724 2016 |    2.673276
 354 | Mon May 16 13:21:22.326724 2016 |  3602.673276
 371 | Tue May 17 06:21:22.326724 2016 | 57597.326724
 406 | Wed May 18 17:21:22.326724 2016 | 183597.326724
 415 | Thu May 19 02:21:22.326724 2016 | 215997.326724
(5 rows)
```

> **Note** When the RUM plug-in creates an index on a table whose data is sorted based on referenced additional information, errors may occur. This is because the RUM plug-in uses a suffix tree that has the following limits: Both edges and suffixes must be of a fixed length, and suffixes cannot have child nodes.

rum_tsvector_hash_addon_ops

- Supported data types: tsvector.
- Description: This function stores the hash values of the word libraries that are used by the tsvector data type. This function also stores the fields of the RUM plug-in. However, this function does not support prefix-based searches.

rum_tsquery_ops

- Supported data types: tsquery.
- Description: This function stores the branches of query trees.

- Examples:
  i. Execute the following statements to create a table with an index:

```
CREATE TABLE test_array (i int2[]);
INSERT INTO test_array VALUES ('{}'), ('{0}'), ('{1,2,3,4}'), ('{1,2,3}'), ('{1,2}'), ('{1}');
CREATE INDEX idx_array ON test_array USING rum (i rum_anyarray_ops);
```

  ii. Run the following query:

```
SET enable_seqscan TO off;
EXPLAIN (COSTS OFF) SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

  The following result is returned:

```
           QUERY PLAN
-----------------------------------------
 Index Scan using idx_array on test_array
   Index Cond: (i && '{1}'::smallint[])
   Order By: (i <=> '{1}'::smallint[])
(3 rows)
```

  iii. Run the following query:

```
SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

  The following result is returned:

```
   i
-----------
 {1}
 {1,2}
 {1,2,3}
 {1,2,3,4}
(4 rows)
```

rum_anyarray_ops

- Supported data types: anyarray.
- Description: This function stores anyarray elements at lengths that are similar to the lengths of arrays. This function supports the following operators: `&&`, `@>`, `<@`, `=`, and `%`. This function also allows you to sort data by using the `<=>` operator.
- Examples:
  i. Execute the following statements to create a table with an index:

```
CREATE TABLE test_array (i int2[]);
INSERT INTO test_array VALUES ('{}'), ('{0}'), ('{1,2,3,4}'), ('{1,2,3}'), ('{1,2}'), ('{1}');
CREATE INDEX idx_array ON test_array USING rum (i rum_anyarray_ops);
```

ii. Run the following query:

```
SET enable_seqscan TO off;
EXPLAIN (COSTS OFF) SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

The following result is returned:

```
           QUERY PLAN
-----------------------------------------
 Index Scan using idx_array on test_array
   Index Cond: (i && '{1}'::smallint[])
   Order By: (i <=> '{1}'::smallint[])
(3 rows)
```

iii. Run the following query:

```
SELECT * FROM test_array WHERE i && '{1}' ORDER BY i <=> '{1}' ASC;
```

The following result is returned:

```
   i
-----------
 {1}
 {1,2}
 {1,2,3}
 {1,2,3,4}
(4 rows)
```

rum_anyarray_addon_ops

- Supported data types: anyarray.
- Description: This function stores anyarray elements. This function also stores all the elements that are supported for the fields of the RUM plug-in.

18.Use the fuzzystrmatch plug-in to compute similarity between strings

ApsaraDB RDS for PostgreSQL provides the fuzzystrmatch plug-in. This plug-in supports the Soundex, Levenshtein, Metaphone, and Double Metaphone algorithms. You can use these algorithms to calculate the similarity and distance between strings.

Soundex

The Soundex algorithm converts similar-sounding words into the same code. However, this algorithm is unsuitable for non-English words.

The Soundex algorithm provides the following functions:

```
soundex(text) returns text
difference(text, text) returns int
```

- The soundex function converts a string into its Soundex code, such as A550.
- The difference function converts two strings into their Soundex codes. Then, the difference function reports the number of code matching positions between the two strings. A Soundex code consists of four characters. Therefore, the number of code matching positions ranges from 0 to 4. The value 0 indicates a zero match, and the value 4 indicates an exact match.

Examples:

```
SELECT soundex('hello world!') ;
SELECT soundex('Anne'), soundex('Andrew'), difference('Anne', 'Andrew');
SELECT soundex('Anne'), soundex('Margaret'), difference('Anne', 'Margaret');
CREATE TABLE s (nm text);
INSERT INTO s VALUES ('john');
INSERT INTO s VALUES ('joan');
INSERT INTO s VALUES ('wobbly');
INSERT INTO s VALUES ('jack');
SELECT * FROM s WHERE soundex(nm) = soundex('john');
SELECT * FROM s WHERE difference(s.nm, 'john') > 2;
```

Levenshtein

The Levenshtein algorithm calculates the Levenshtein distance between two strings.

The Levenshtein algorithm provides the following functions:

```
levenshtein(text source, text target, int ins_cost, int del_cost, int sub_cost) returns int
levenshtein(text source, text target) returns int
levenshtein_less_equal(text source, text target, int ins_cost, int del_cost, int sub_cost, int max_d) returns int
levenshtein_less_equal(text source, text target, int max_d) returns int
```

The following table describes the parameters that you must configure in the preceding functions.

Parameter	Description
source	The first string to compare. The string cannot be empty and can contain up to 255 characters in length.
target	The second string to compare. The string cannot be empty and can contain up to 255 characters in length.
ins_cost	The overhead that is required to insert characters.
del_cost	The overhead that is required to delete characters.
sub_cost	The overhead that is required to replace characters.
max_d	The maximum Levenshtein distance that is allowed between the two specified strings.

Note The levenshtein_less_equal function is an accelerated version of the levenshtein function. It is used only to calculate a short Levenshtein distance:

- If the actual distance is less than or equal to the value of the max_d parameter, the levenshtein_less_equal function returns the exact distance that is calculated.
- If the actual distance is greater than the value of the max_d parameter, the levenshtein_less_equal function returns a random distance that is greater than the value of the max_d parameter.
- If the value of the max_d parameter is negative, the levenshtein_less_equal and levenshtein functions return the same distance.

Examples:

```
SELECT levenshtein('GUMBO', 'GAMBOL');
SELECT levenshtein('GUMBO', 'GAMBOL', 2,1,1);
SELECT levenshtein_less_equal('extensive', 'exhaustive',2);
SELECT levenshtein_less_equal('extensive', 'exhaustive',4);
```

```
test=# SELECT levenshtein('GUMBO', 'GAMBOL');
 levenshtein
-------------
           2
(1 row)

test=# SELECT levenshtein('GUMBO', 'GAMBOL', 2,1,1);
 levenshtein
-------------
           3
(1 row)

test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',2);
 levenshtein_less_equal
------------------------
                      3
(1 row)

test=# SELECT levenshtein_less_equal('extensive', 'exhaustive',4);
 levenshtein_less_equal
------------------------
                      4
(1 row)
```

Metaphone

The Metaphone algorithm works in the same way as the Soundex algorithm. The Metaphone algorithm constructs a representative code for each specified string. If two strings have the same representative code, the Metaphone algorithm considers them to be similar.

The Metaphone algorithm provides the following functions:

```
metaphone(text source, int max_output_length) returns text
```

The following table describes the parameters that you must configure in the preceding functions.

Parameter	Description
source	A string that is not empty. The string can contain up to 255 characters in length.
max_output_length	The maximum length of the Metaphone code that can be returned. If the Metaphone code exceeds the maximum length, the Metaphone algorithm truncates the Metaphone code to the maximum length.

Example:

```
SELECT metaphone('GUMBO', 4);
```

Double Metaphone

The Double Metaphone algorithm obtains two similar-sounding codes for a specified string. These codes include a primary code and a secondary code. In most cases, the two codes are the same. They may be slightly different when you specify a non-English word. The difference varies based on the pronunciation.

The Double Metaphone algorithm provides the following functions:

```
dmetaphone(text source) returns text
dmetaphone_alt(text source) returns text
```

Examples:

```
select dmetaphone('gumbo');
select dmetaphone_alt('gumbo');
```

19.Use the pg_hint_plan plug-in to customize query plans

ApsaraDB RDS for PostgreSQL provides the pg_hint_plan plug-in. You can use the plug-in to add hints to SQL statements. This allows you to change the execution plans of SQL statements on an ApsaraDB RDS for PostgreSQL instance.

Prerequisites

Your RDS instance runs one of the following PostgreSQL versions:

- PostgreSQL 10
- PostgreSQL 9.4

Context

PostgreSQL uses a cost-based optimizer that works based on data statistics instead of static rules. The optimizer evaluates the cost of every possible execution plan for an SQL statement to your database system. This helps the optimizer select a final execution plan that has the lowest cost. However, the optimizer does not consider the possible internal relationships among data. Therefore, the final execution plan may not be perfect. You can use the pg_hint_plan plug-in to add hints to an SQL statement. These hints specify how you want to execute the SQL statement. This is another way to optimize the execution plans of SQL statements.

Basic usage

A hint starts with a forward slash, an asterisk, and a plus sign (`/*+`) and ends with an asterisk and a forward slash (`*/`). A hint consists of a hint name followed by parameters that are enclosed in a pair of parentheses. The parameters are separated by space characters. For readability purposes, you can separate each hint with a line break.

Example:

In this example, the HashJoin hint specifies that the SeqScan method is used to scan the pgbench_accounts table.

```
/*+
  HashJoin(a b)
  >SeqScan(a)
 */
EXPLAIN SELECT *
  FROM pgbench_branches b
  JOIN pgbench_accounts a ON b.bid = a.bid
 ORDER BY a.aid;
```

The following result is returned:

```
                        QUERY PLAN
-------------------------------------------------------------------------------------
 Sort  (cost=31465.84..31715.84 rows=100000 width=197)
   Sort Key: a.aid
   ->  Hash Join  (cost=1.02..4016.02 rows=100000 width=197)
         Hash Cond: (a.bid = b.bid)
         ->  Seq Scan on pgbench_accounts a  (cost=0.00..2640.00 rows=100000 width=97)
         ->  Hash  (cost=1.01..1.01 rows=1 width=100)
               ->  Seq Scan on pgbench_branches b  (cost=0.00..1.01 rows=1 width=100)
(7 rows)
```

Hint table

Hints can be used to optimize the execution plans of SQL statements. However, this is convenient only when SQL statements are editable. If SQL statements are not editable, you can place hints in a table named hint_plan.hints. The table consists of the following columns.

> **Note** By default, the user who creates the pg_hint_plan plug-in has the permissions on the hint table. The hints in the hint table take precedence over the hints that you add by using the pg_hint_plan plug-in.

Column	Description
id	The ID of the hint. The ID is unique and automatically generated.
norm_query_string	The pattern that matches the SQL statement to which you want to add the hint. The constants in the SQL statement must be replaced by wildcards (`?`). Space characters are crucial parts of the pattern.
application_name	The name of the application to which the hint is applied. If this parameter is left empty, the hint is applied to all applications.
hints	The comment that contains the hint. You do not need to include comment marks.

Example:

```
INSERT INTO hint_plan.hints(norm_query_string, application_name, hints)
  VALUES (
    'EXPLAIN (COSTS false) SELECT * FROM t1 WHERE t1.id = ? ;',
    '',
    'SeqScan(t1)'
  );
INSERT 0 1
postgres=# UPDATE hint_plan.hints
postgres-#    SET hints = 'IndexScan(t1)'
postgres-#  WHERE id = 1;
UPDATE 1
postgres=# DELETE FROM hint_plan.hints
postgres-#  WHERE id = 1;
DELETE 1
```

Hint types

Hints come in the following six types based on how they affect execution plans:

- Hints for scan methods

  This type of hint specifies the method that is used to scan the specified table. If the specified table has an alias, the pg_hint_plan plug-in identifies the table based on the alias. Supported scan methods include `SeqScan` , `IndexScan` , and `NoSeqScan` .

  The hints for scan methods are valid on ordinary tables, inherited tables, unlogged tables, temporary tables, and system tables. However, the hints for scan methods are invalid on external tables, table functions, statements in which the values of constants are specified, universal expressions, views, and subqueries.

  Example:

  ```
  /*+
    SeqScan(t1)
    IndexScan(t2 t2_pkey)
  */
  SELECT * FROM table1 t1 JOIN table table2 t2 ON (t1.key = t2.key);
  ```

- Hints for join methods

  This type of hint specifies the method that is used to join the specified tables.

  The hints for join methods are valid on ordinary tables, inherited tables, unlogged tables, temporary tables, external tables, system tables, table functions, statements in which the values of constants are specified, and universal expressions. However, the hints for join methods are invalid on views and subqueries.

- Hints for join order

This type of hint specifies the order in which two or more tables are joined. You can use one of the following methods to specify a hint for join order:

- Specify the order in which you want to join the specified tables, without restricting the direction at each join level.
- Specify the order in which you want to join the specified tables, as well as the direction at each join level.

Example:

```
/*+
  NestLoop(t1 t2)
  MergeJoin(t1 t2 t3)
  Leading(t1 t2 t3)
*/
SELECT * FROM table1 t1
  JOIN table table2 t2 ON (t1.key = t2.key)
  JOIN table table3 t3 ON (t2.key = t3.key);
```

- Hints for row number correction

  This type of hint corrects row number errors that are caused by the optimizer.

  Examples:

```
/*+ Rows(a b #10) */ SELECT... ;    //Sets the row number to 10.
/*+ Rows(a b +10) */ SELECT... ;    //Increases the row number by 10.
/*+ Rows(a b -10) */ SELECT... ;    //Decreases the row number by 10.
/*+ Rows(a b *10) */ SELECT... ;    //Increases the row number by 10 times.
```

- Hints for parallel execution

  This type of hint specifies the plan that is used to execute SQL statements in parallel.

  The hints for parallel execution are valid on ordinary tables, inherited tables, unlogged tables, and system tables. However, the hints for parallel execution are invalid on external tables, clauses in which the values of constants are specified, universal expressions, views, and subqueries. You can specify the internal tables of a view based on their real names or aliases.

  The following examples show how an SQL statement is executed in a different way on each table:

- Example 1:

```
explain /*+ Parallel(c1 3 hard) Parallel(c2 5 hard) */
    SELECT c2.a FROM c1 JOIN c2 ON (c1.a = c2.a);
```

The following result is returned:

```
                        QUERY PLAN
-----------------------------------------------------------------------------
 Hash Join  (cost=2.86..11406.38 rows=101 width=4)
  Hash Cond: (c1.a = c2.a)
  -> Gather  (cost=0.00..7652.13 rows=1000101 width=4)
     Workers Planned: 3
     -> Parallel Seq Scan on c1  (cost=0.00..7652.13 rows=322613 width=4)
  -> Hash  (cost=1.59..1.59 rows=101 width=4)
     -> Gather  (cost=0.00..1.59 rows=101 width=4)
        Workers Planned: 5
        -> Parallel Seq Scan on c2  (cost=0.00..1.59 rows=59 width=4)
```

- Example 2:

```
EXPLAIN /*+ Parallel(tl 5 hard) */ SELECT sum(a) FROM tl;
```

The following result is returned:

```
                        QUERY PLAN
-----------------------------------------------------------------------------
 Finalize Aggregate  (cost=693.02..693.03 rows=1 width=8)
  -> Gather  (cost=693.00..693.01 rows=5 width=8)
     Workers Planned: 5
     -> Partial Aggregate  (cost=693.00..693.01 rows=1 width=8)
        -> Parallel Seq Scan on tl  (cost=0.00..643.00 rows=20000 width=4)
```

- Hints for GUC parameter setting

  This type of hint temporarily changes the value of a GUC parameter. The values of GUC parameters in the execution plan help you achieve the expected effect. However, this does not apply if the specified hint conflicts with the execution plans of other SQL statements. If you set a GUC parameter more than once, the latest value takes effect.

  Example:

```
/*+ Set(random_page_cost 2.0) */
SELECT * FROM table1 t1 WHERE key = 'value';
```

The following table describes all of the supported hints.

Type	Hint	Description
Hints for scan methods	SeqScan(table)	Specifies a sequence scan.
	TidScan(table)	Specifies a TID scan.
	IndexScan(table[index...])	Specifies an index scan. You can specify an index.
	IndexOnlyScan(table[index...])	Specifies an index-only scan. You can specify an index.
	BitmapScan(table[index...])	Specifies a bitmap scan.
	NoSeqScan(table)	Prohibits a sequence scan.
	NoTidScan(table)	Prohibits a TID scan.
	NoIndexScan(table)	Prohibits an index scan.
	NoIndexOnlyScan(table)	Prohibits an index scan. Only tables are scanned.
	NoBitmapScan(table)	Prohibits a bitmap scan.
Hints for join methods	NestLoop(table table[table...])	Specifies a nested loop join.
	HashJoin(table table[table...])	Specifies a hash join.
	MergeJoin(table table[table...])	Specifies a merge join.
	NoNestLoop(table table[table...])	Prohibits a nested loop join.
	NoHashJoin(table table[table...])	Prohibits a hash join.
	NoMergeJoin(table table[table...])	Prohibits a merge join.
Hints for join order	Leading(table table[table...])	Specifies the join order.
	Leading(<join pair>)	Specifies the join order and direction.
Hints for row number correction	Rows(table table[table...] correction)	Corrects the row number of the join result that is obtained from the specified tables. The following operators are supported: `#<n>`, `+<n>`, `-<n>`, and `*<n>`. The `<n>` operator is supported by the strtod function.

Type	Hint	Description
Hints for parallel execution	Parallel(table <# of workers> [soft|hard])	Specifies or prohibits the parallel execution of the specified tables. The `<worker#>` parameter specifies the number of working programs that are required. The value 0 specifies to prohibit parallel execution. If the third parameter is set to soft, only the value of the max_parallel_workers_per_gather parameter is changed and the other parameters are specified by the optimizer. If the third parameter is set to hard, the values of all related parameters are changed. The third parameter is set to soft by default.
Hints for GUC parameter setting	Set(GUC-param value)	Specifies the value of a GUC parameter when the optimizer runs.

For more information, visit the official PostgreSQL website.