TencentDB for PostgreSQL

Best Practice

Product Documentation

Contents

Best Practice
postgres_fdw Extension for Cross-database Access

Last updated：2024-03-20 10:59:40

TencentDB for PostgreSQL provides a series of extensions for accessing external data sources, including data in other libraries of this instance or data from other instances. Cross-database access extensions include homogenous extensions such as dblink, postgresql_fdw, and heterogeneous extensions like mysql_fdw, cos_fdw. The steps to use cross-database access are as follows:

1. Use the "CREATE EXTENSION" statement to install the extensions.
2. Create an external server object and a link map for each remote database that needs to be connected.
3. Use the corresponding command to access the external table to access data.

As the cross-database access extensions can access directly across instances or perform cross-database access within the same instance, TencentDB for PostgreSQL has optimized permission control when creating external server objects. It categorizes them based on the environment of the target instance. In addition to the features provided by the open-source version, extra auxiliary parameters have been added to verify the user's identity and adjust network policies. For more details, please refer to Auxiliary Parameters below.

Note:

Please note that the dblink extension is currently only supported by TencentDB for PostgreSQL kernels with major version 10.

Extension Auxiliary Parameters

host

This is a compulsory parameter when accessing across instances. It refers to the IP address of the target instance.

port

This is a compulsory parameter when accessing across instances. It refers to the port of the target instance.

instanceid

Instance ID

It is used when accessing across instances in TencentDB for PostgreSQL. This parameter is mandatory when accessing across instances. The format is similar to postgres-xxxxxx, pgro-xxxxxx, and can be viewed on the console. If the target instance is on Tencent Cloud CVM, then it is the instance ID of the CVM, the format is similar to ins-xxxxx.

dbname

Refers to the name of the database in the remote PostgreSQL service to be accessed. For cross-database access in the same instance, you only need to configure this parameter and can leave other parameters empty.

access_type

Optional. Refers to the type of the target instance:

The target instance is a TencentDB instance, including TencentDB for PostgreSQL, TencentDB for MySQL, etc. If the type is not specified, this is the default.

The target instance is on a Tencent Cloud CVM.

The target instance is a public network-based self-built instance in Tencent Cloud.

The target instance is a cloud VPN-based instance.

The target instance is a self-built VPN-based instance.

The target instance is a Direct Connect-based instance.

uin

Optional. Refers to the account ID to which the instance belongs. This information is used to identify user permissions, and you can refer to View uin.

own_uin

Optional. Refers to the root account ID to which the instance belongs. This information is also needed to identify user permissions.

vpcid

Optional. Refers to the Virtual Private Cloud ID. If the target instance is in the Tencent Cloud CVM's VPC network, this parameter is required and can be found in the VPC Console.

subnetid

Optional. Refers to the Virtual Private Cloud Subnet ID. If the target instance is in the Tencent Cloud CVM's VPC network, this parameter is required and can be found in the Subnets section of the VPC Console.

dcgid

Optional. Refers to the Direct Connect ID. If the target instance needs to connect via leased line network, you need to provide this parameter value.

vpngwid

Optional. Refers to the VPN Gateway ID. If the target instance needs to connect through VPN, this parameter value needs to be provided.

region

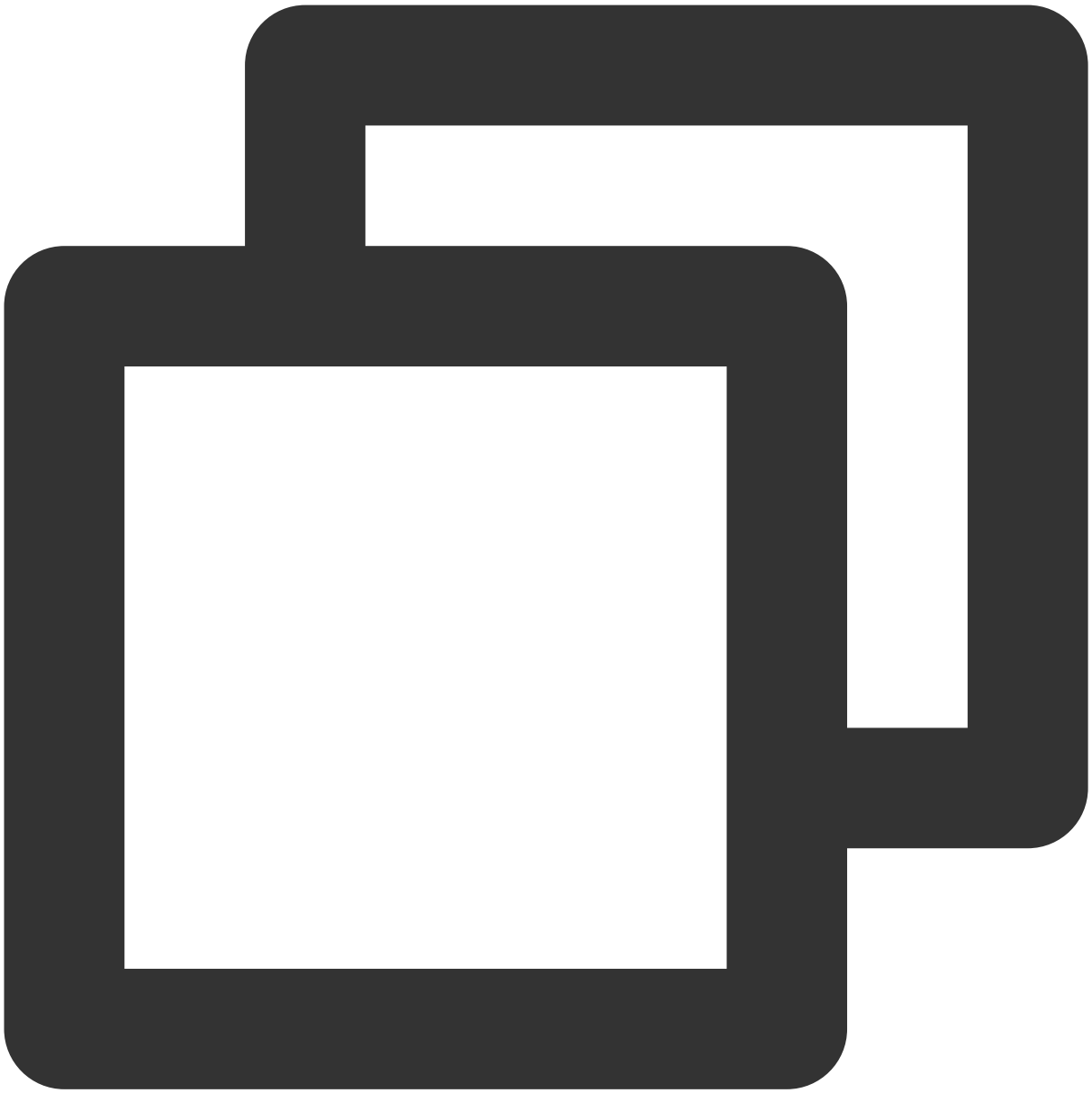
Optional. Refers to the region where the target instance is located. For example, "ap-guangzhou" represents Guangzhou. If you need to access data across regions, this parameter value needs to be provided.

Examples of How to Use postgres_fdw

Using the postgres_fdw extension, you can access data from other databases or other Postgres instances in this instance.

Step 1: Prerequisites

1. Create test data in the instance.

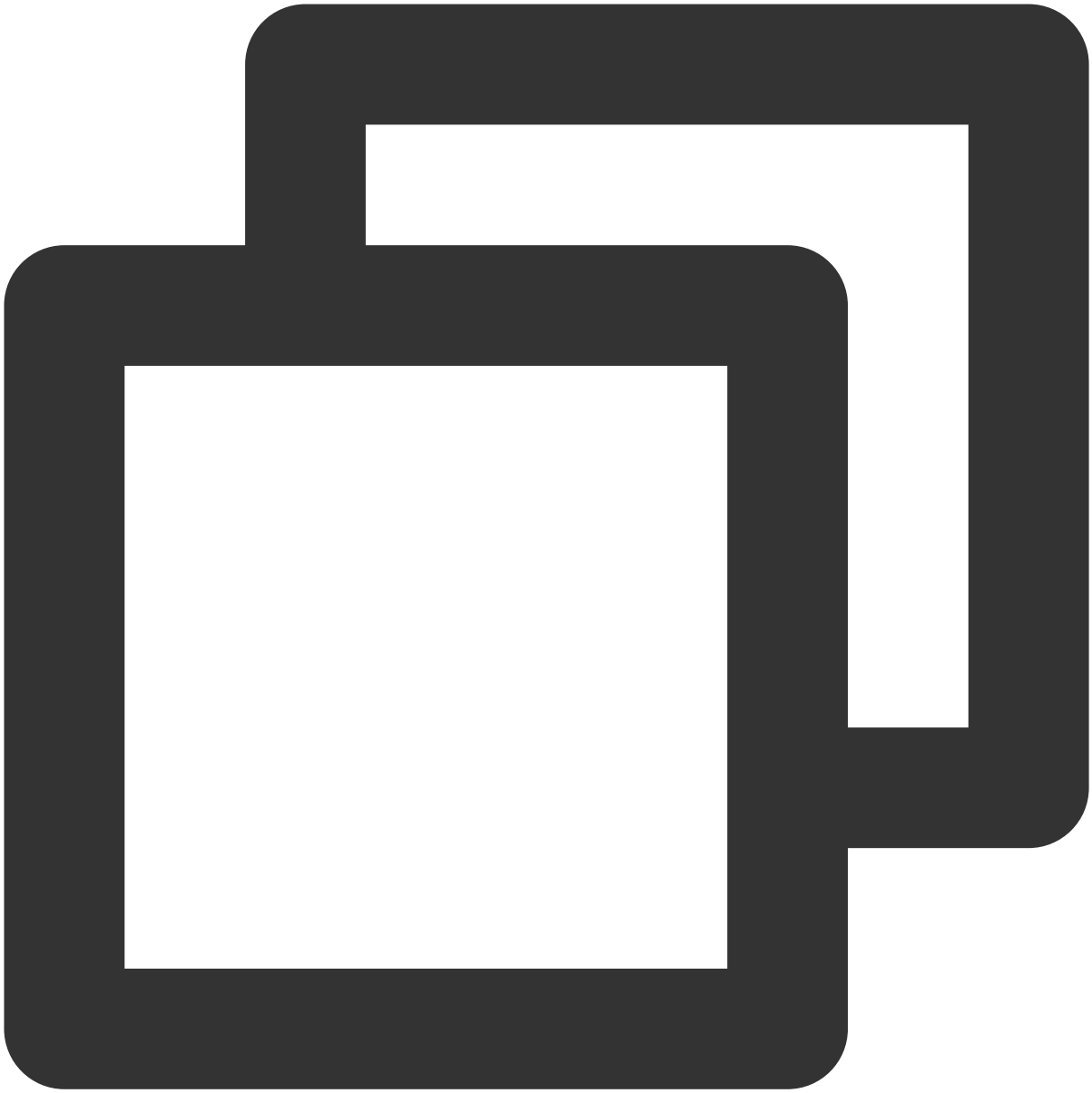


```
postgres=>create role user1 with LOGIN  CREATEDB PASSWORD 'password1';
postgres=>create database testdb1;
CREATE DATABASE
```

Note:

If an error occurs when creating an extension, please submit a ticket to contact Tencent Cloud after-sales for assistance.

2. Create test data in the target instance.



```
postgres=>create role user2 with LOGIN  CREATEDB PASSWORD 'password2';
postgres=> create database testdb2;
CREATE DATABASE
postgres=> \\c testdb2 user2
You are now connected to database "testdb2" as user "user2".
testdb2=> create table test_table2(id integer);
CREATE TABLE
testdb2=> insert into test_table2 values (1);
INSERT 0 1
```

Step 2. Create the postgres_fdw extension

Note:

If you encounter an issue stating 'extension does not exist' or 'insufficient privileges' while creating the extension, please submit a ticket for assistance.

```
#Create
postgres=> \\c testdb1
You are now connected to database "testdb1" as user "user1".
testdb1=> create extension postgres_fdw;
CREATE EXTENSION
```

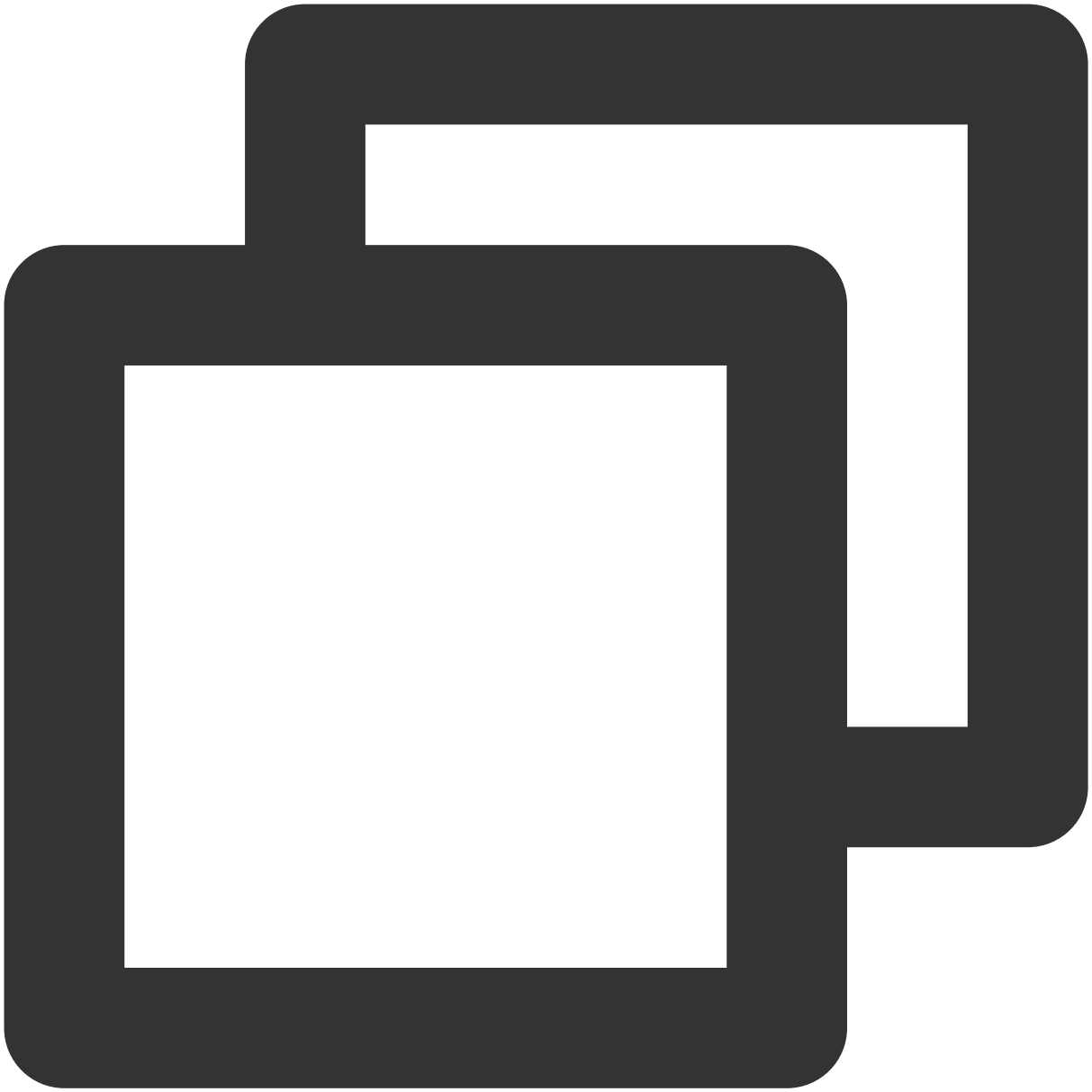
```
#View
testdb1=> \\dx
                              List of installed extensions
      Name      | Version |   Schema   |                     Description
--------------+---------+------------+---------------------------------------------
  plpgsql      | 1.0     | pg_catalog | PL/pgSQL procedural language
  postgres_fdw | 1.0     | public     | foreign-data wrapper for remote PostgreSQL
(2 rows)
```

Step 3. Create a Server

Note:

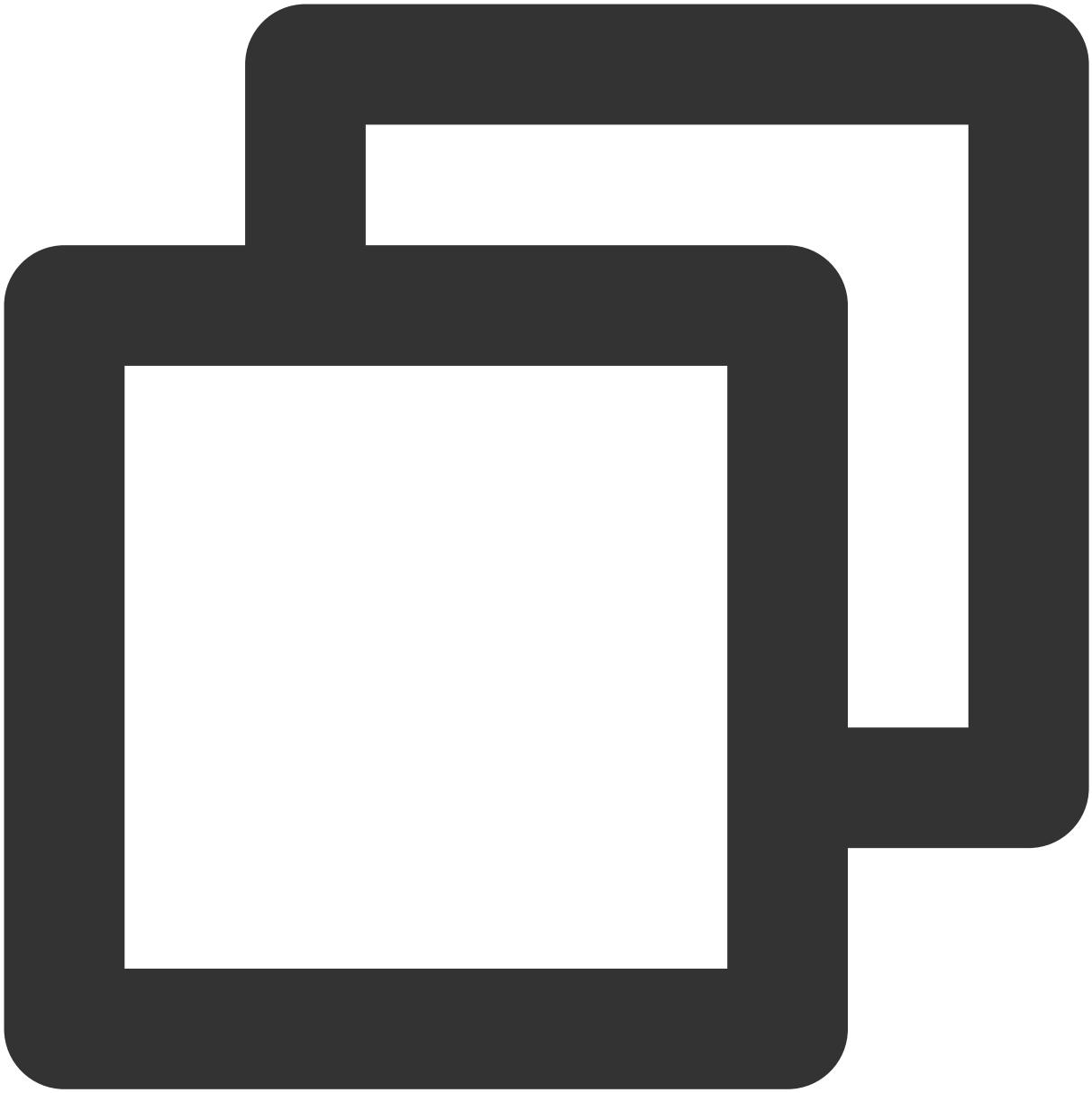
Cross-instance access is supported only for kernel version v10.17_r1.2, v11.12_r1.2, v12.7_r1.2, v13.3_r1.2, v14.2_r1.0, and later.

Cross-instance access.



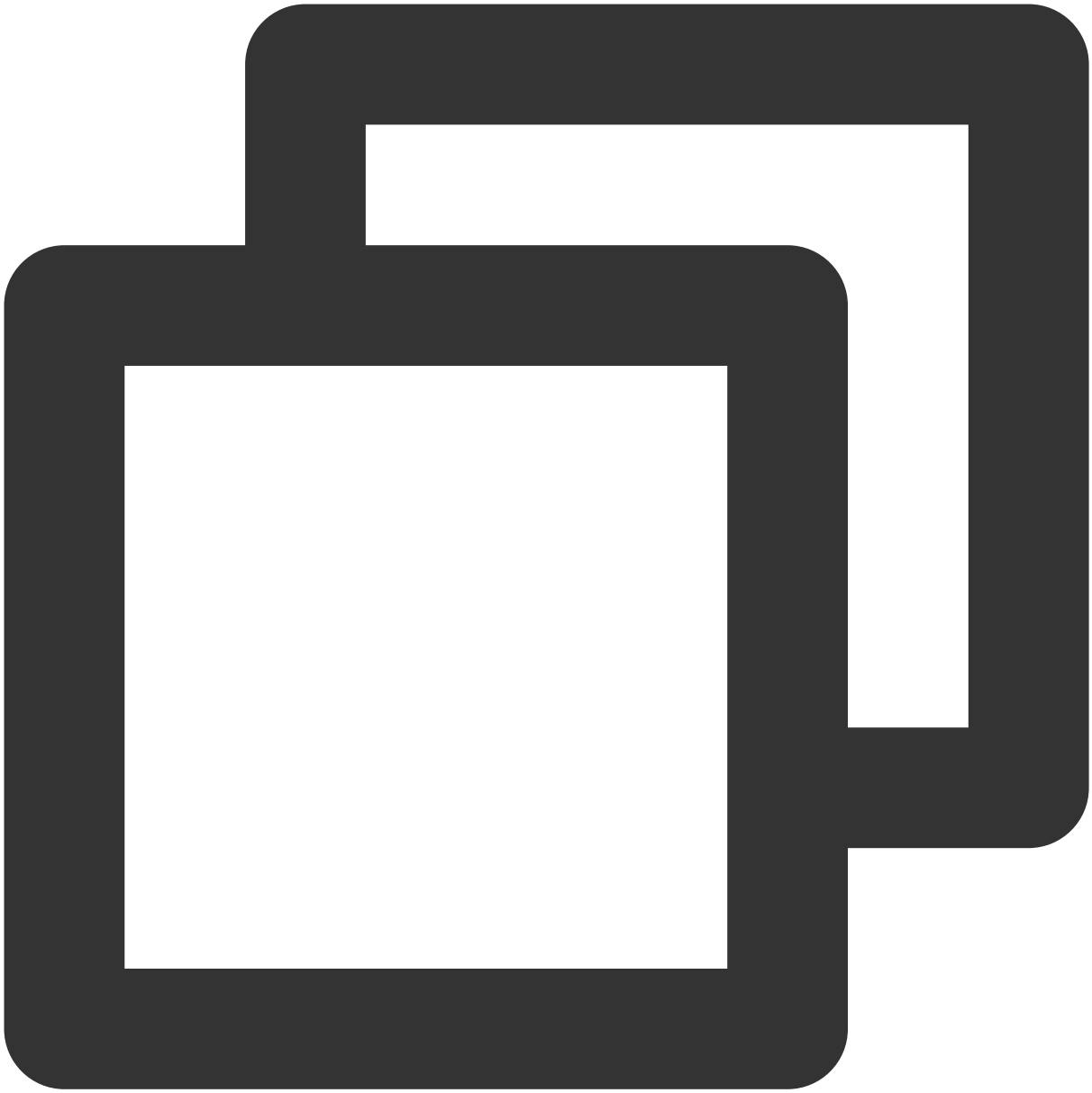
```
#Access data from the target instance `testdb2` from `testdb1` in this instance.
testdb1=>create server srv_test1 foreign data wrapper postgres_fdw options (host 'x
CREATE SERVER
```

If not crossing instances, and only accessing across databases, you only need to specify the dbname parameter.



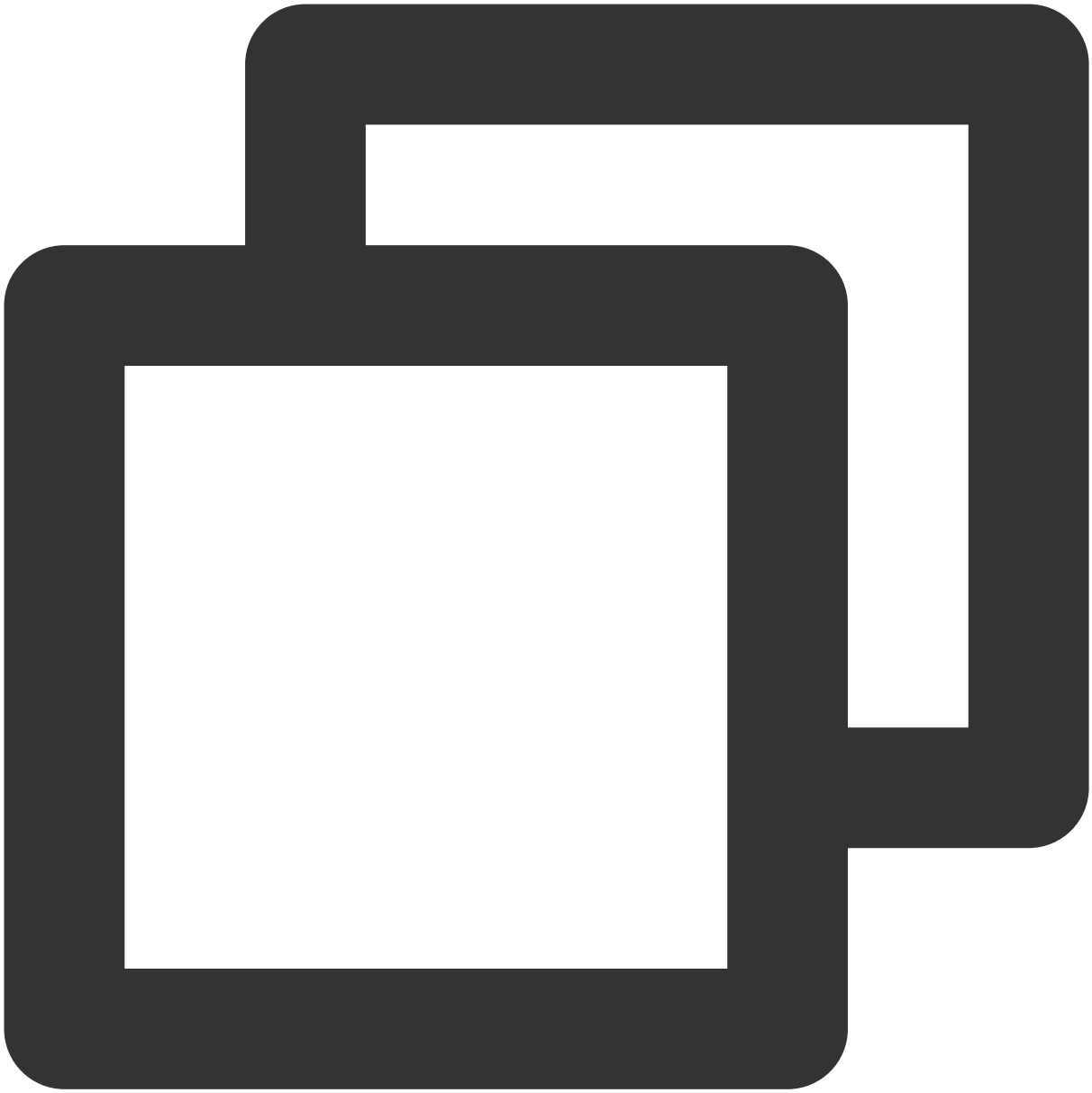
```
#Access the data of `testdb2` from `testdb1` in this instance
create server srv_test1 foreign data wrapper postgres_fdw options (dbname 'testdb2'
```

The target instance is on a Tencent Cloud CVM, and the network type is classic network.



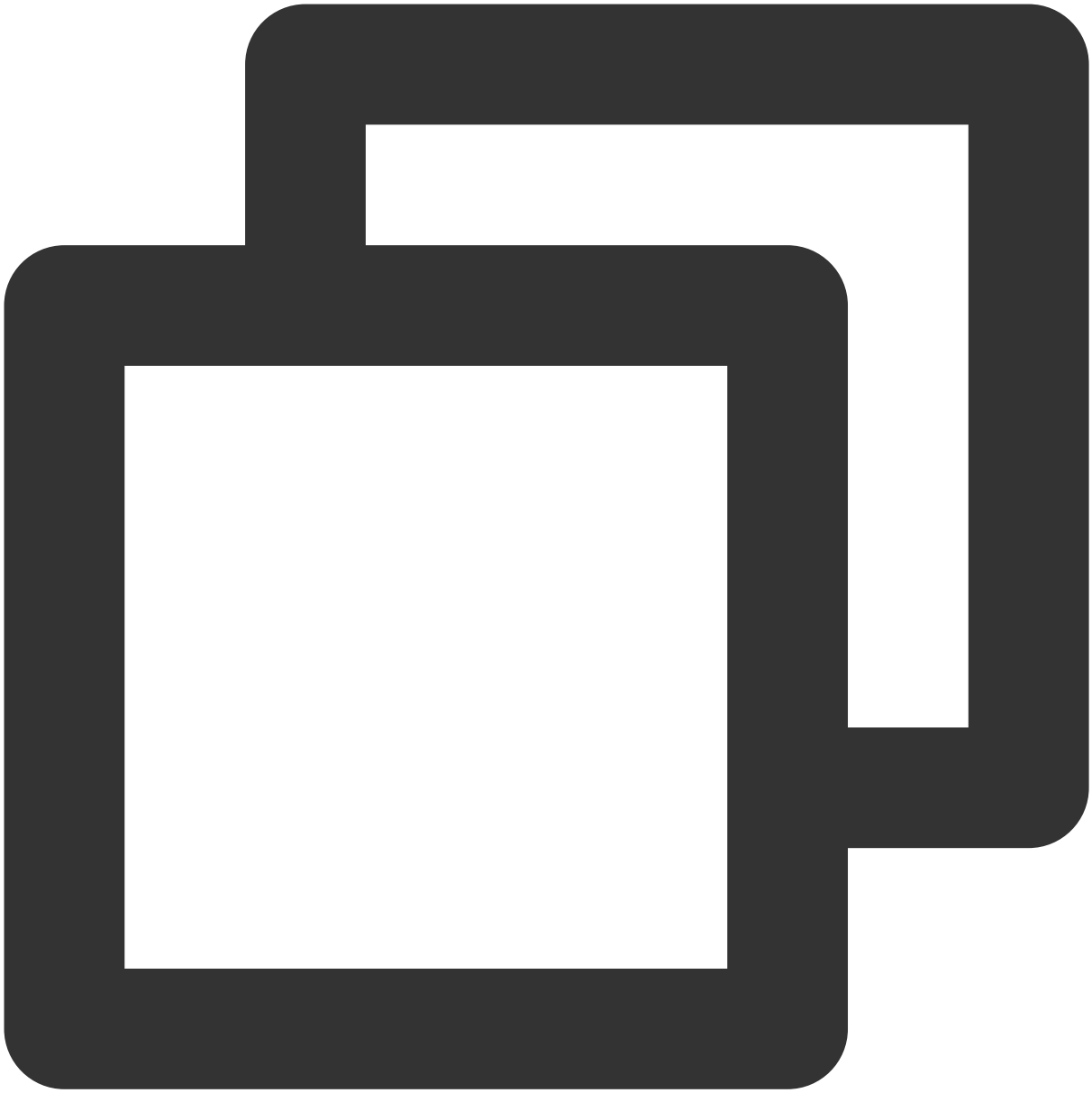
```
testdb1=>create server srv_test foreign data wrapper postgres_fdw options (host '
CREATE SERVER
```

The target instance is on a Tencent Cloud CVM, and the network type is VPC.



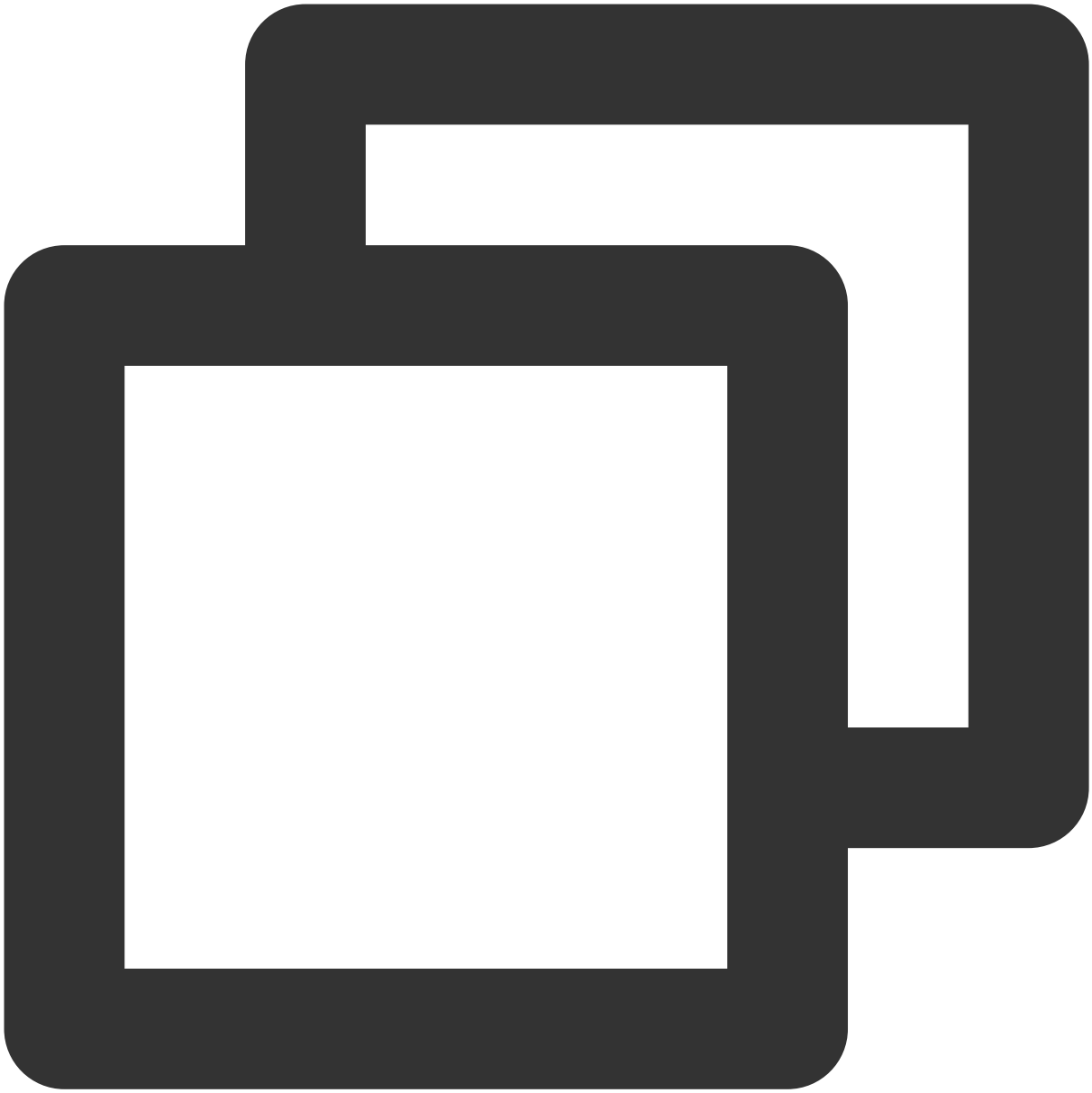
```
testdb1=>create server srv_test1 foreign data wrapper postgres_fdw options (host
CREATE SERVER
```

The target instance is a public network-based self-built instance in Tencent Cloud.



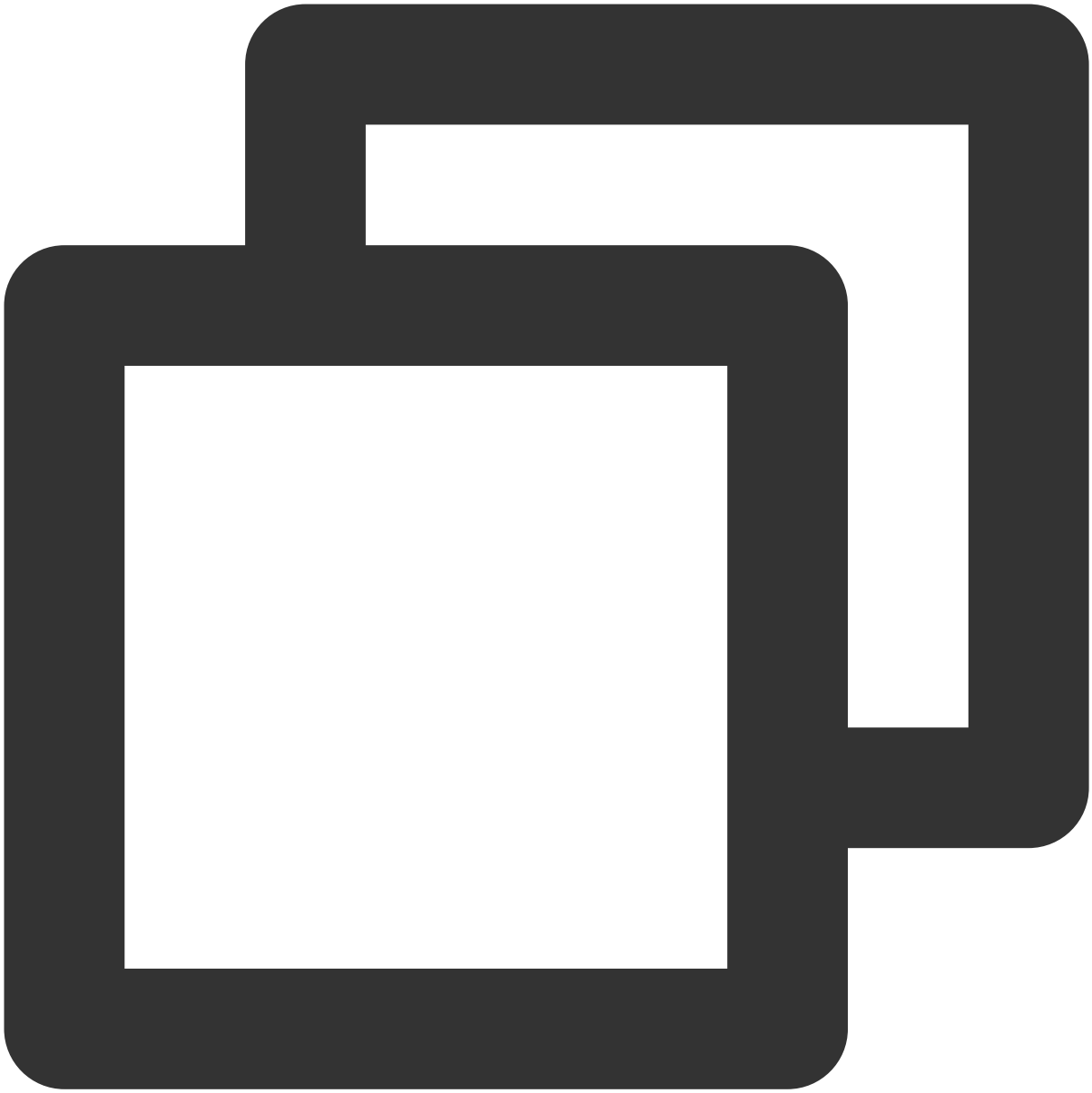
```
testdb1=>create server srv_test1 foreign data wrapper postgres_fdw options (host
CREATE SERVER
```

The target instance is a Tencent Cloud VPN-based instance.



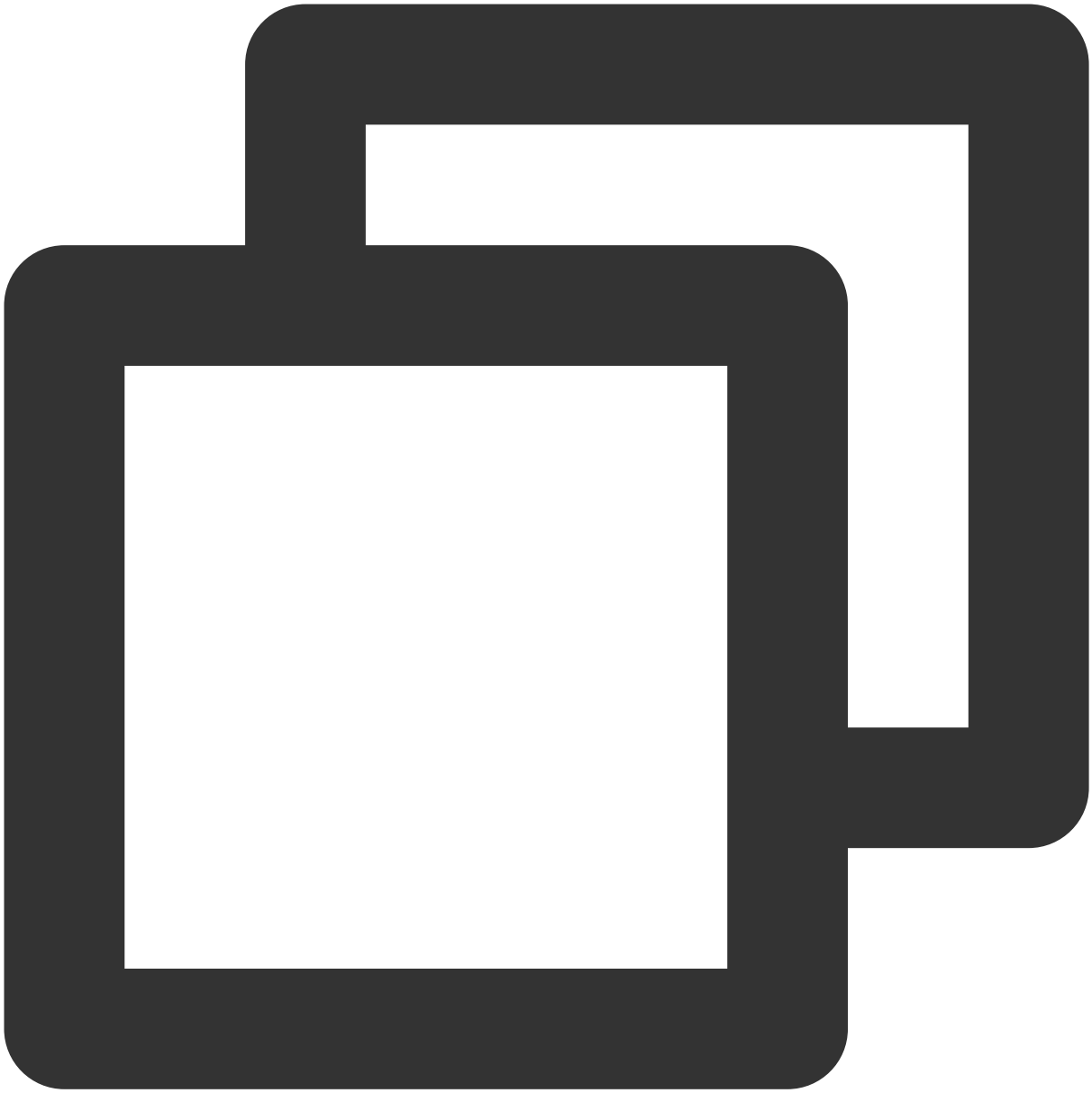
```
testdb1=>create server srv_test1 foreign data wrapper postgres_fdw options (host
```

The target instance is a self-built VPN-based instance.



```
testdb1=>create server srv_test1 foreign data wrapper postgres_fdw options (host
```

The target instance is a Direct Connect-based instance.

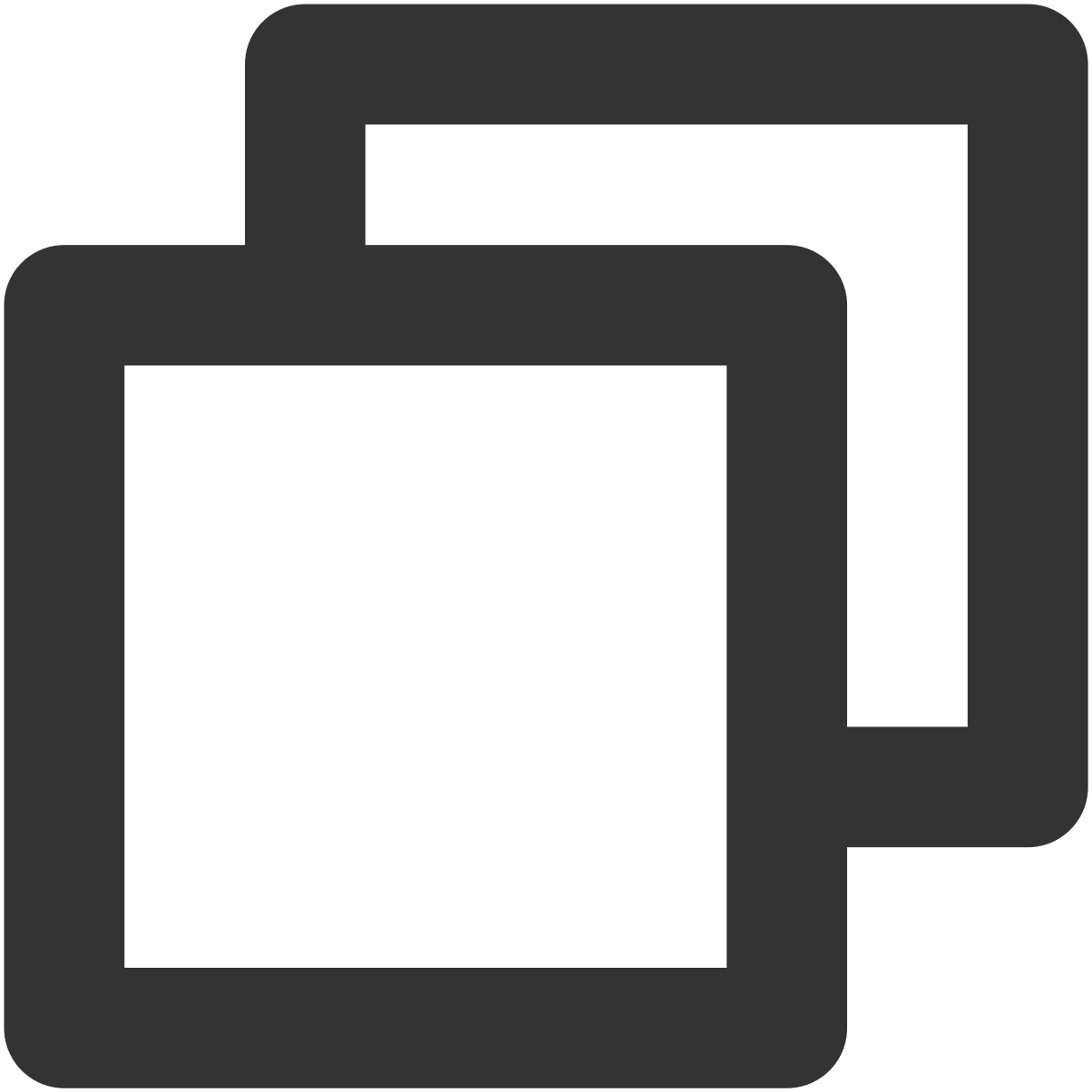


```
testdb1=>create server srv_test1 foreign data wrapper postgres_fdw options (host
CREATE SERVER
```

Step 4. Create a User Mapping

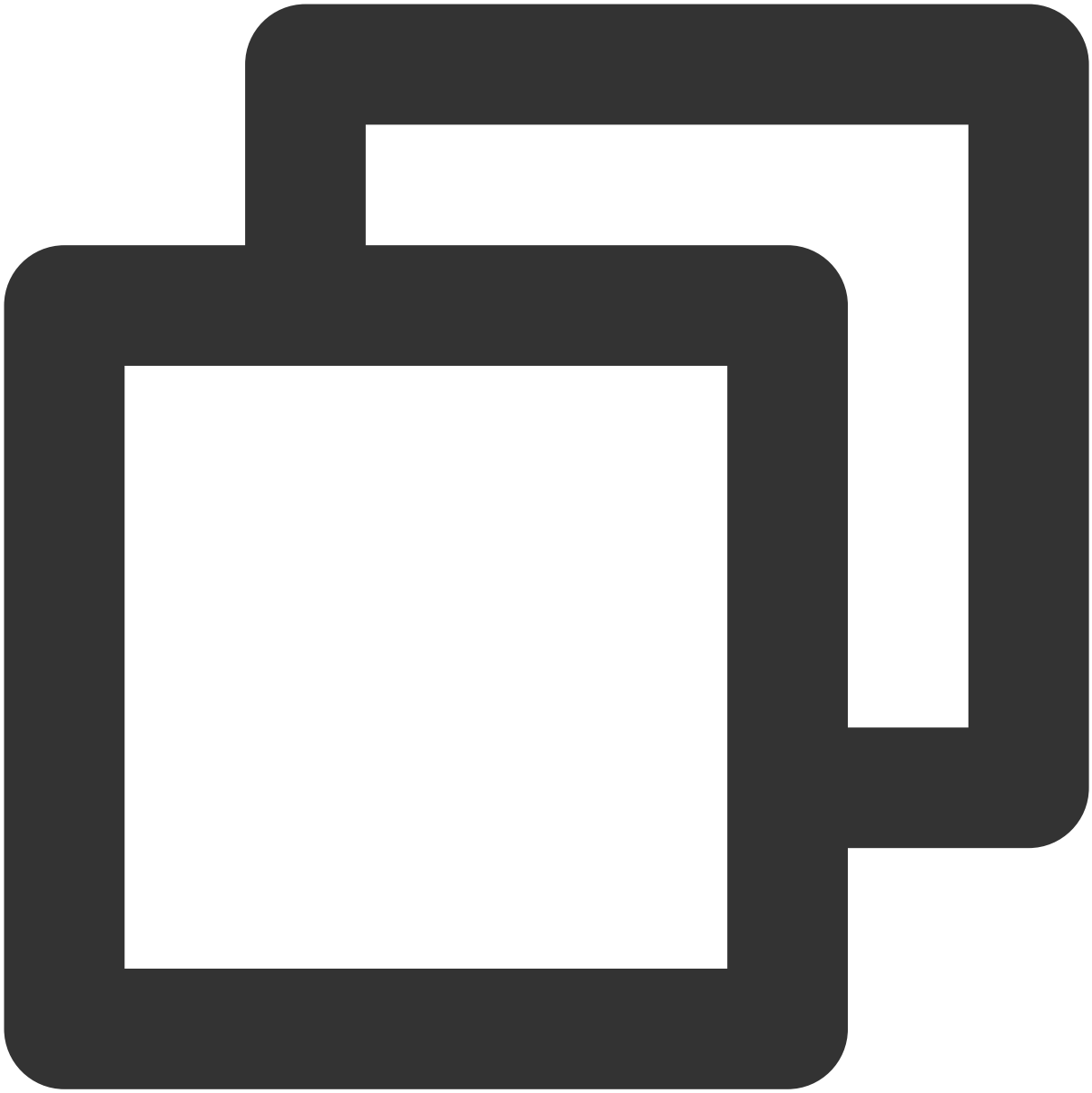
Note:

You can skip this step for cross-database access in the same instance.



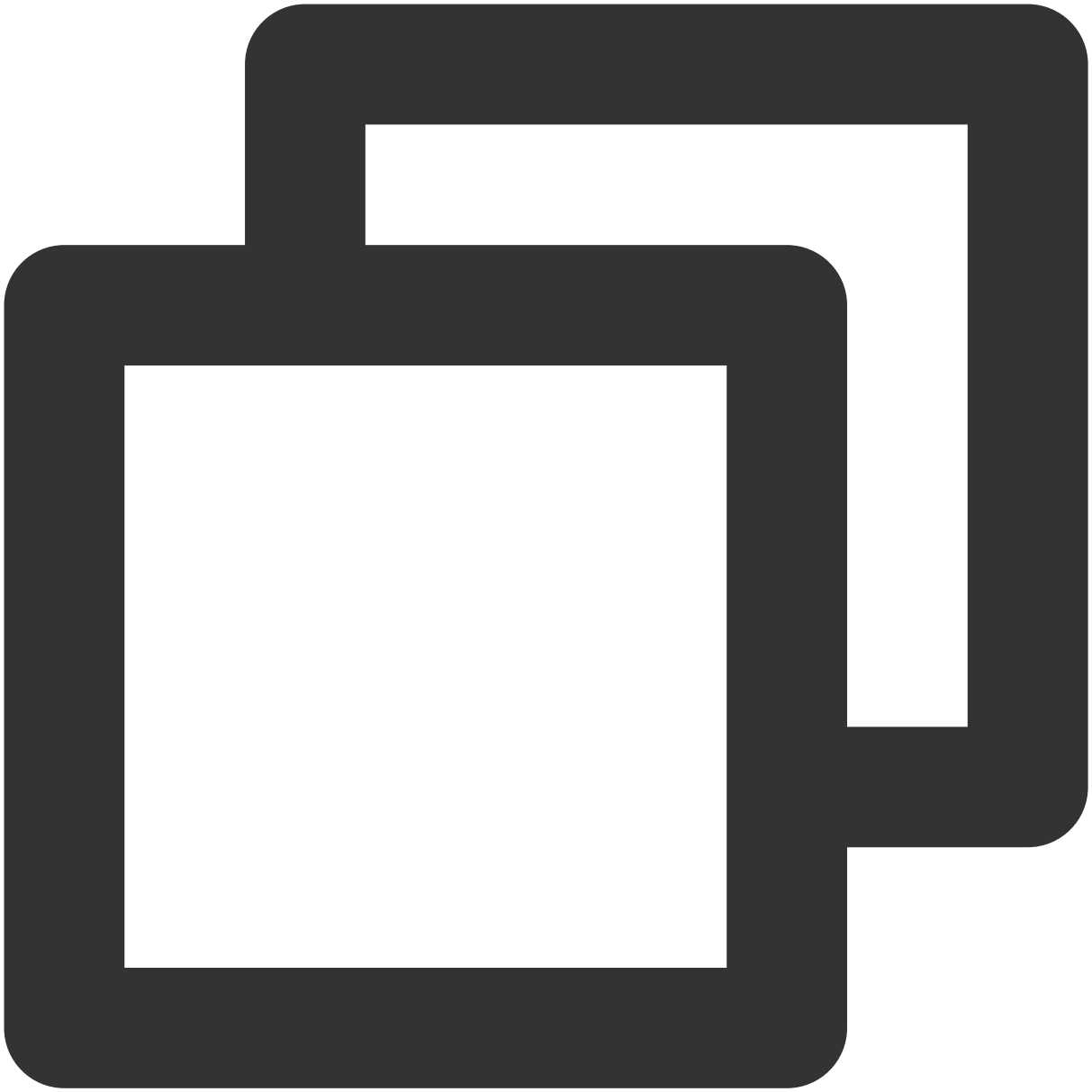
```
testdb1=> create user mapping for user1 server srv_test1 options (user 'user2',pass
CREATE USER MAPPING
```

Step 5. Creating a Foreign Table



```
testdb1=> create foreign table foreign_table1(id integer) server srv_test1 options(
CREATE FOREIGN TABLE
```

Step 6. Access Data from the Foreign Table



```
testdb1=> select * from foreign_table1;
  id
----
   1
(1 row)
```

Reference Link

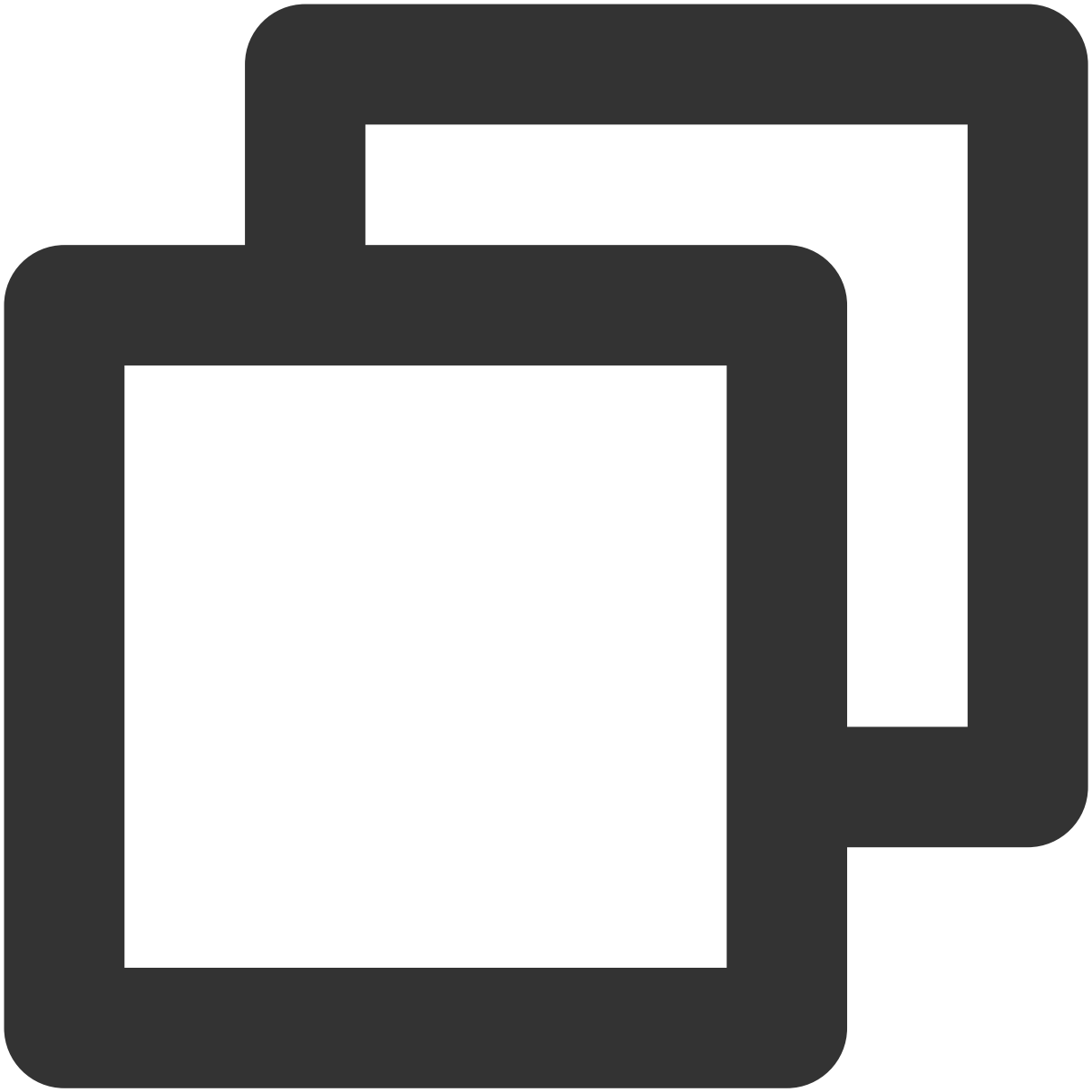
postgres_fdw

PostgreSQL 9.3 > CREATE SERVER

PostgreSQL 9.5 > CREATE SERVER
PostgreSQL 10 > CREATE SERVER
PostgreSQL 11 > CREATE SERVER
PostgreSQL 12 > CREATE SERVER
PostgreSQL 13 > CREATE SERVER
PostgreSQL 14 > CREATE SERVER

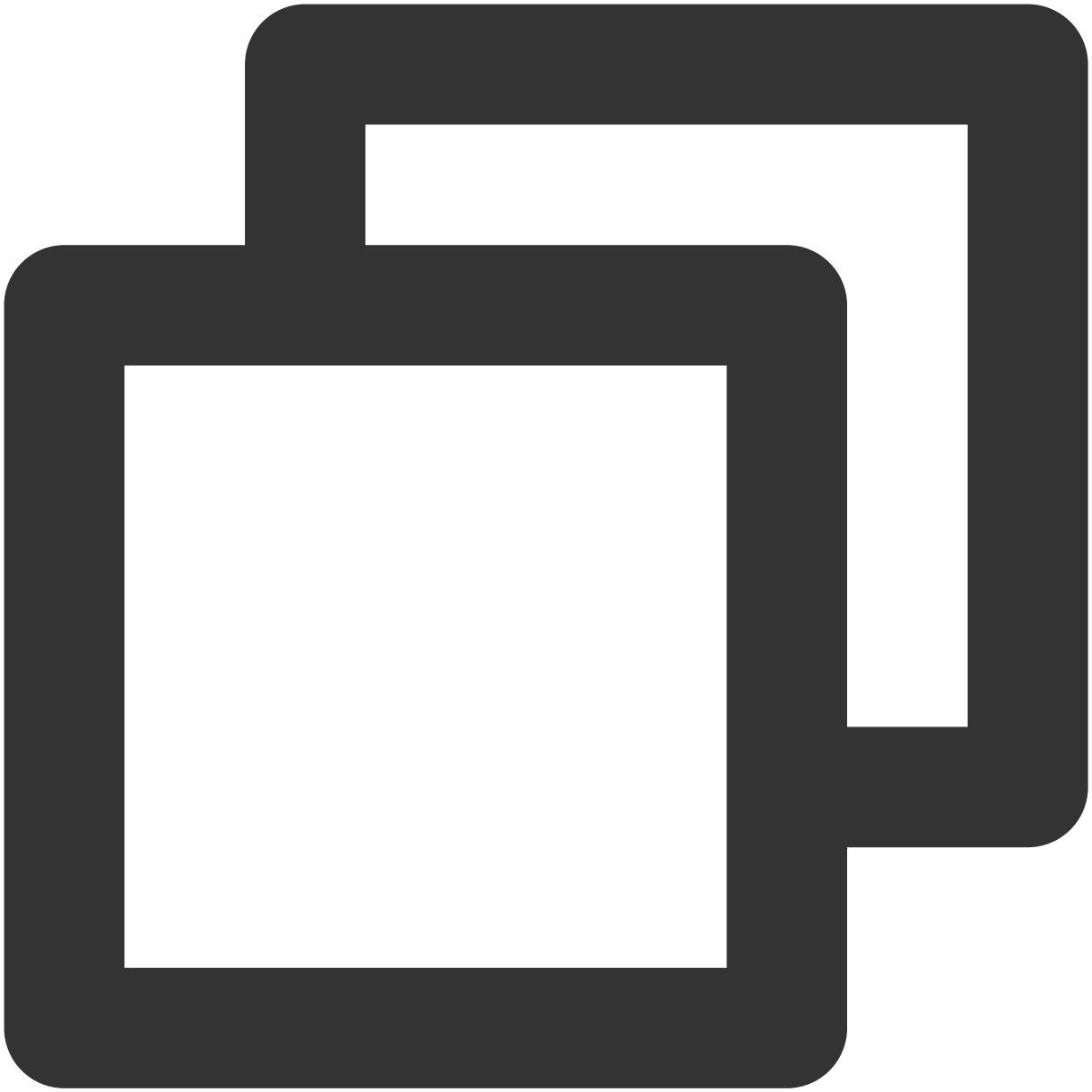
Example of How to Use dblink

Step 1: Create a dblink Extension



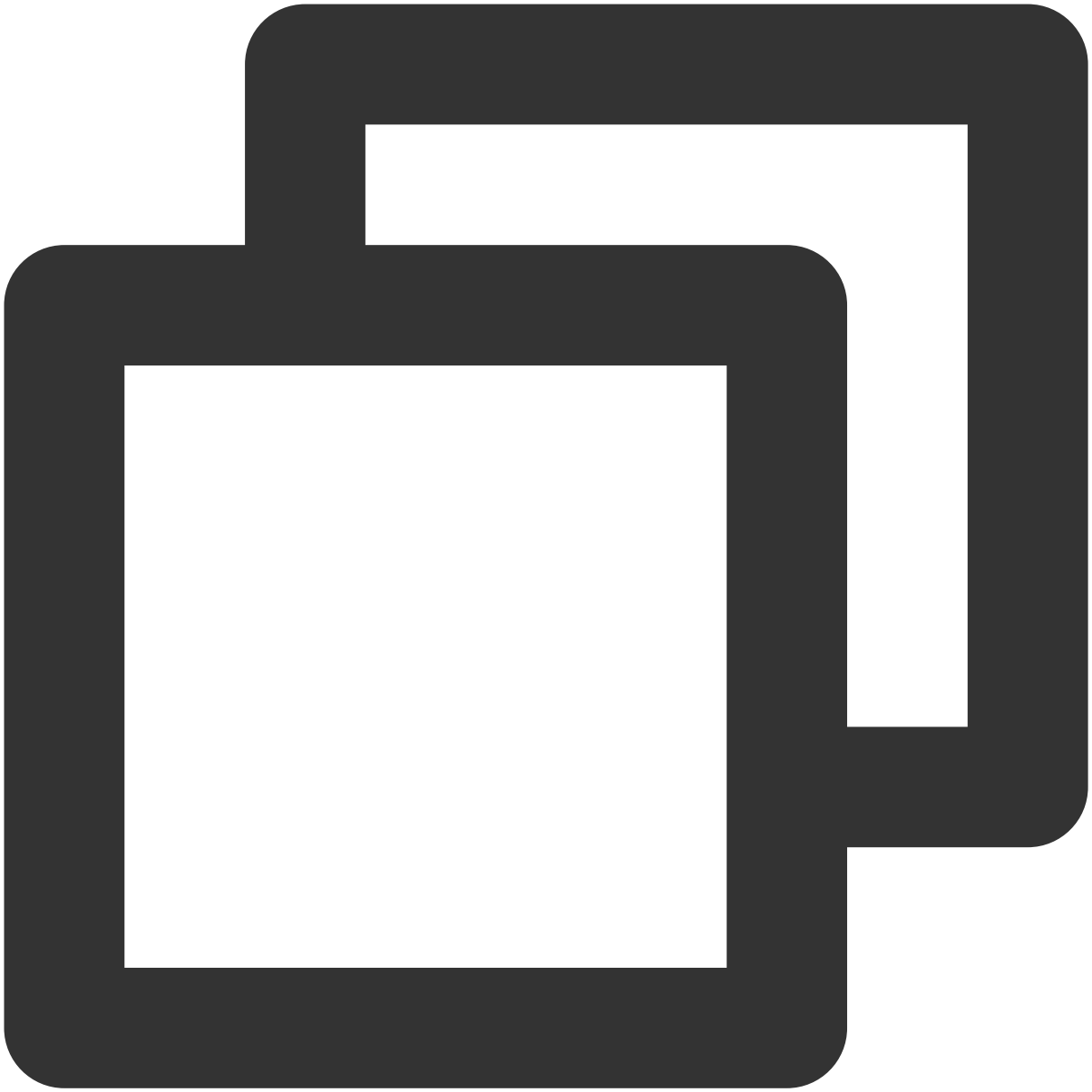
```
postgres=> create extension dblink;
postgres=> \\dx
                                        List of installed extensions
        Name         | Version |   Schema   |                              Description
---------------------+---------+------------+-------------------------------------
 dblink              | 1.2     | public     | connect to other PostgreSQL databases
 pg_stat_log         | 1.0     | public     | track runtime execution statistics of
 pg_stat_statements  | 1.6     | public     | track execution statistics of all SQL
 plpgsql             | 1.0     | pg_catalog | PL/pgSQL procedural language
(4 rows)
```

Step 2: Establish a dblink Link



```
select dblink_connect('yunpg1','host=10.10.10.11 port=5432 instanceid=postgres-2123
 dblink_connect
----------------
 OK
(1 row)
```

Step 3: Access External Data



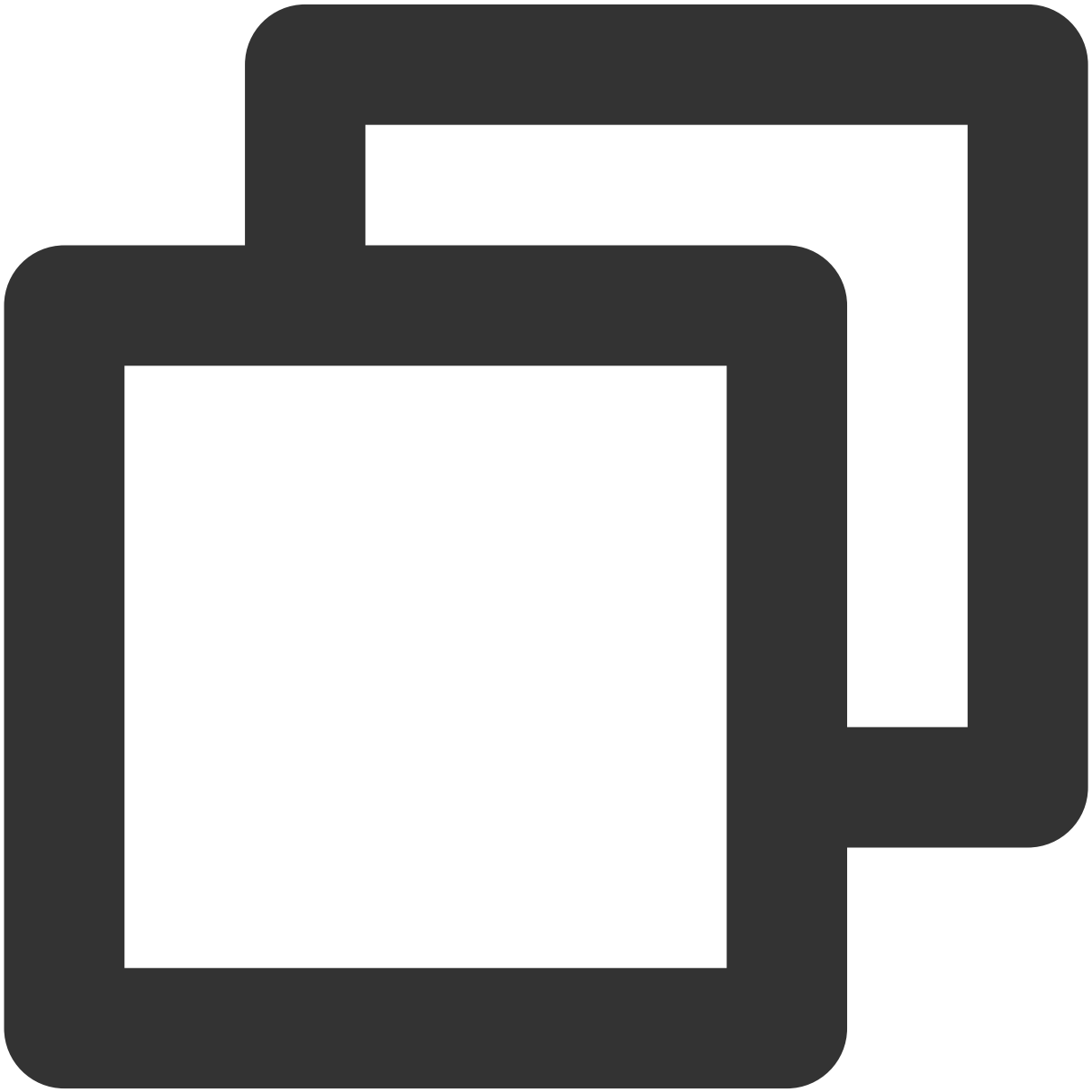
```
postgres=> select * from dblink('yunpg1','select catalog_name,schema_name,schema_ow
    a     |         b          |    c
----------+--------------------+---------
 postgres | pg_toast           | user_00
 postgres | pg_temp_1          | user_00
 postgres | pg_toast_temp_1    | user_00
 postgres | pg_catalog         | user_00
 postgres | public             | user_00
 postgres | information_schema | user_00
(6 rows)
```

Reference Link

dblink

Example of How to Use mysql_fdw

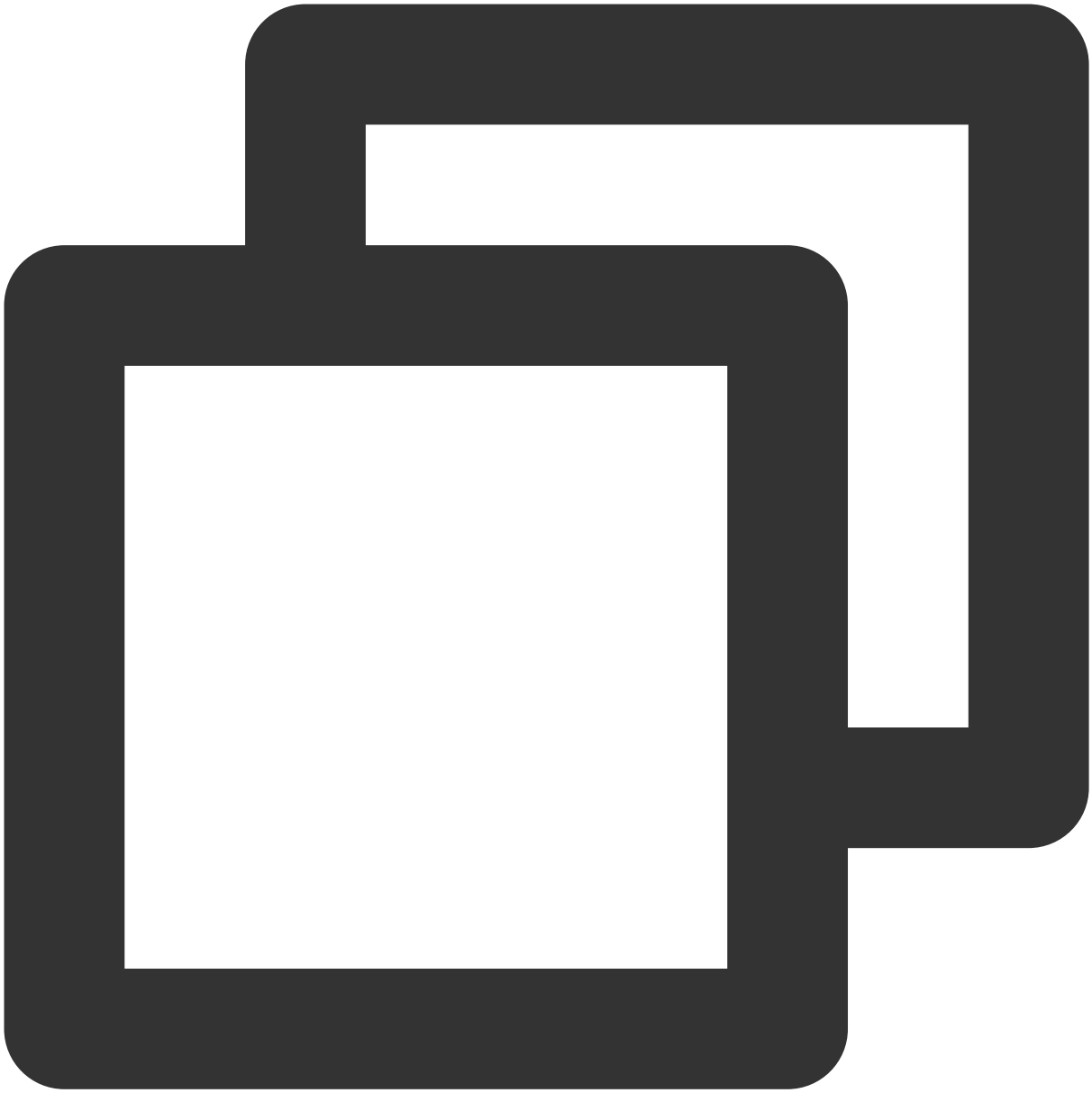
Step 1: Create mysql_fdw Extension



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

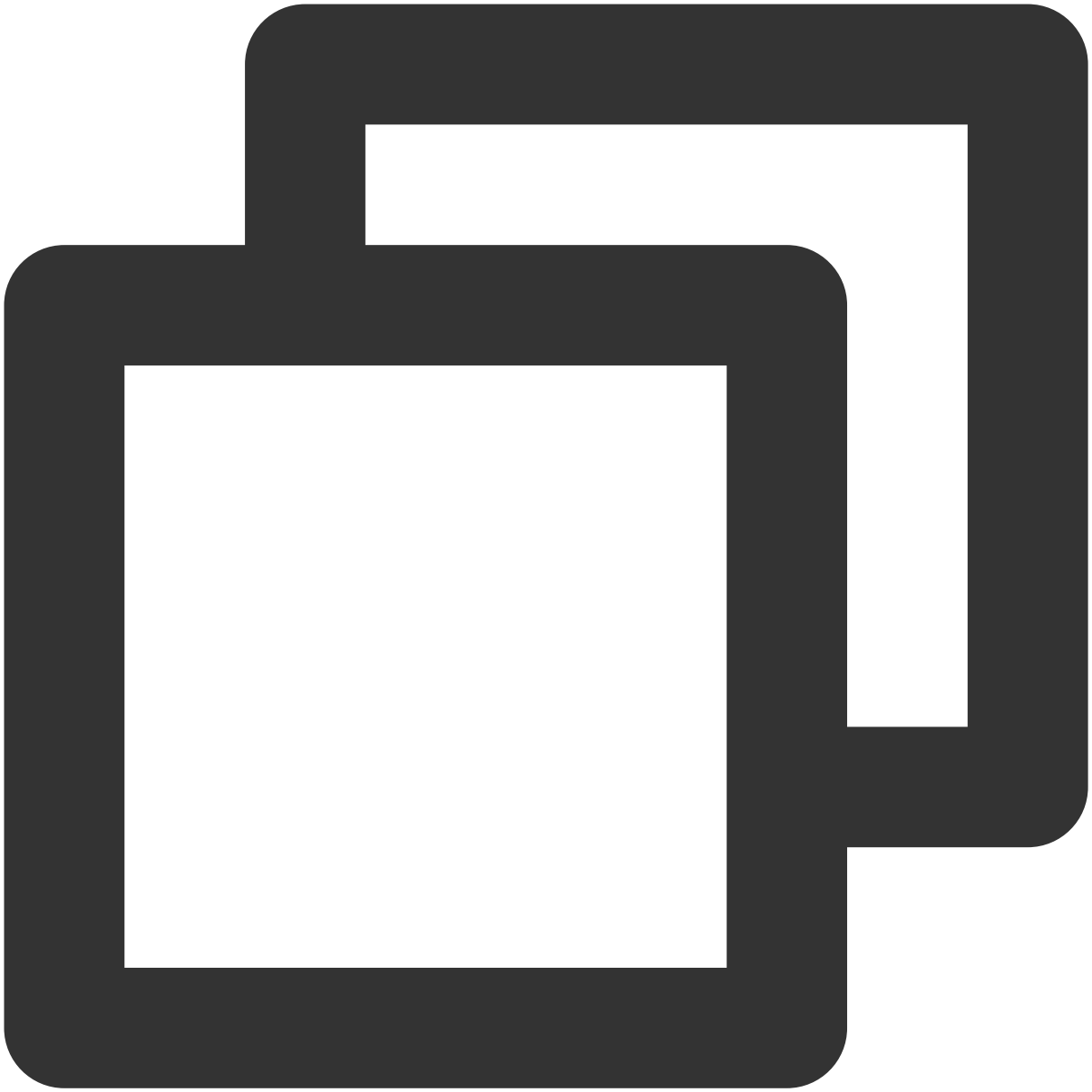
```
postgres=> \\dx;
                                                List of installed extensions
        Name         | Version |   Schema   |                              Descripti
---------------------+---------+------------+-------------------------------------
 dblink              | 1.2     | public     | connect to other PostgreSQL databases
 mysql_fdw           | 1.1     | public     | Foreign data wrapper for querying a My
 pg_stat_log         | 1.0     | public     | track runtime execution statistics of
 pg_stat_statements  | 1.9     | public     | track planning and execution statistic
 plpgsql             | 1.0     | pg_catalog | PL/pgSQL procedural language
(5 rows)
```

Step 2: Create a SERVER



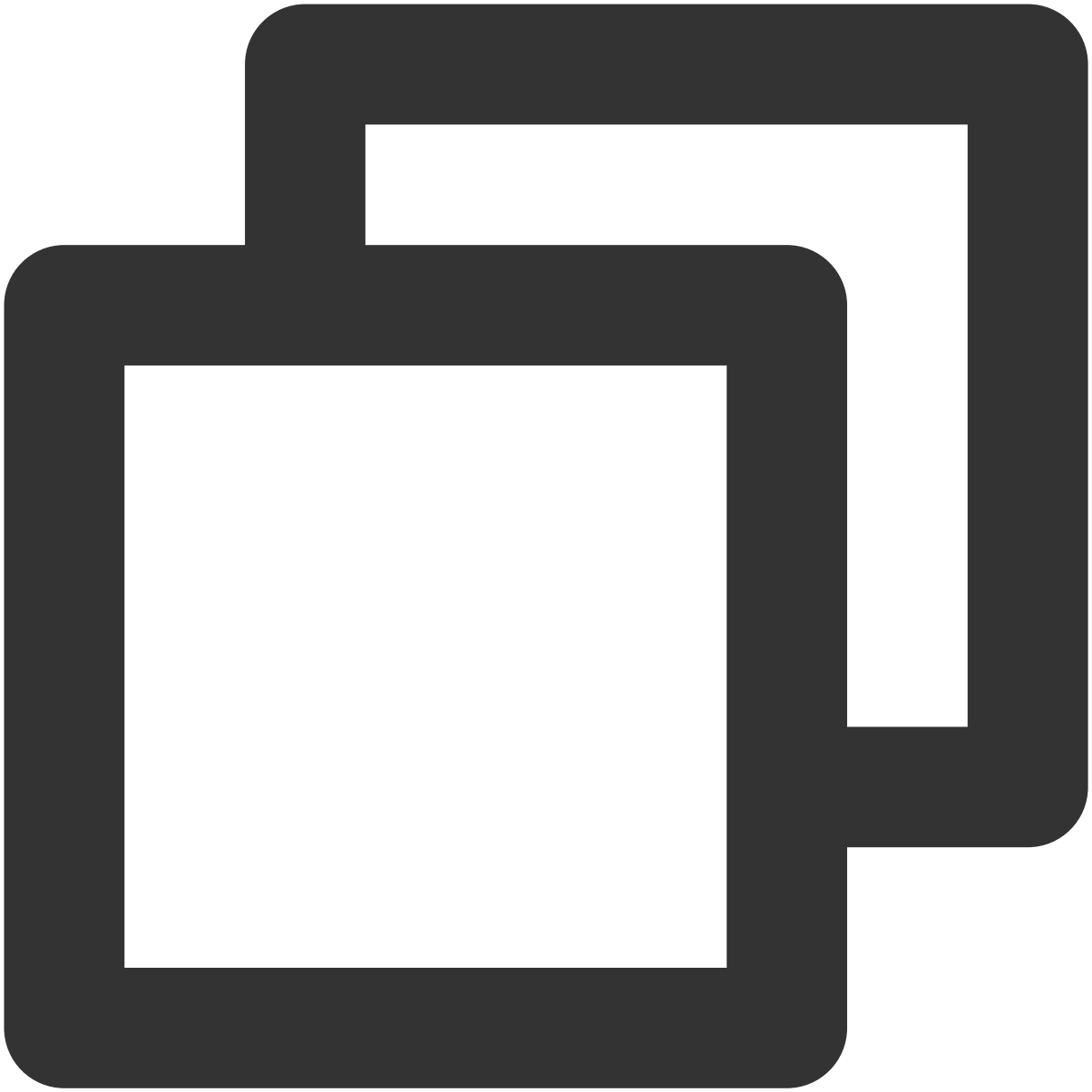
```
postgres=> CREATE SERVER mysql_svr  FOREIGN DATA WRAPPER mysql_fdw OPTIONS (host '1
CREATE SERVER
```

Step 3: Create External User Map



```
postgres=> CREATE USER MAPPING FOR PUBLIC SERVER mysql_svr OPTIONS (username 'fdw_u
CREATE USER MAPPING
```

Step 4: Access External Data



```
postgres=> IMPORT FOREIGN SCHEMA hrdb FROM SERVER mysql_svr INTO public;
```

Reference Link

mysql_fdw

Examples of How to Use cos_fdw

Please refer to the document Supporting Tiered Storage Based on cos_fdw Extension for cos_fdw usage examples.

Notes

Pay attention to the following for the target instance:

1. The HBA restrictions of PostgreSQL need to be loosen to allow mapped users created (such as user2) to access via MD5. For HBA modification, please refer to 20.1. The pg_hba.conf File.
2. If the target instance is not a TencentDB instance and has hot standby mode enabled, after a failover, you'll need to update the server connection address manually or recreate the server configuration.

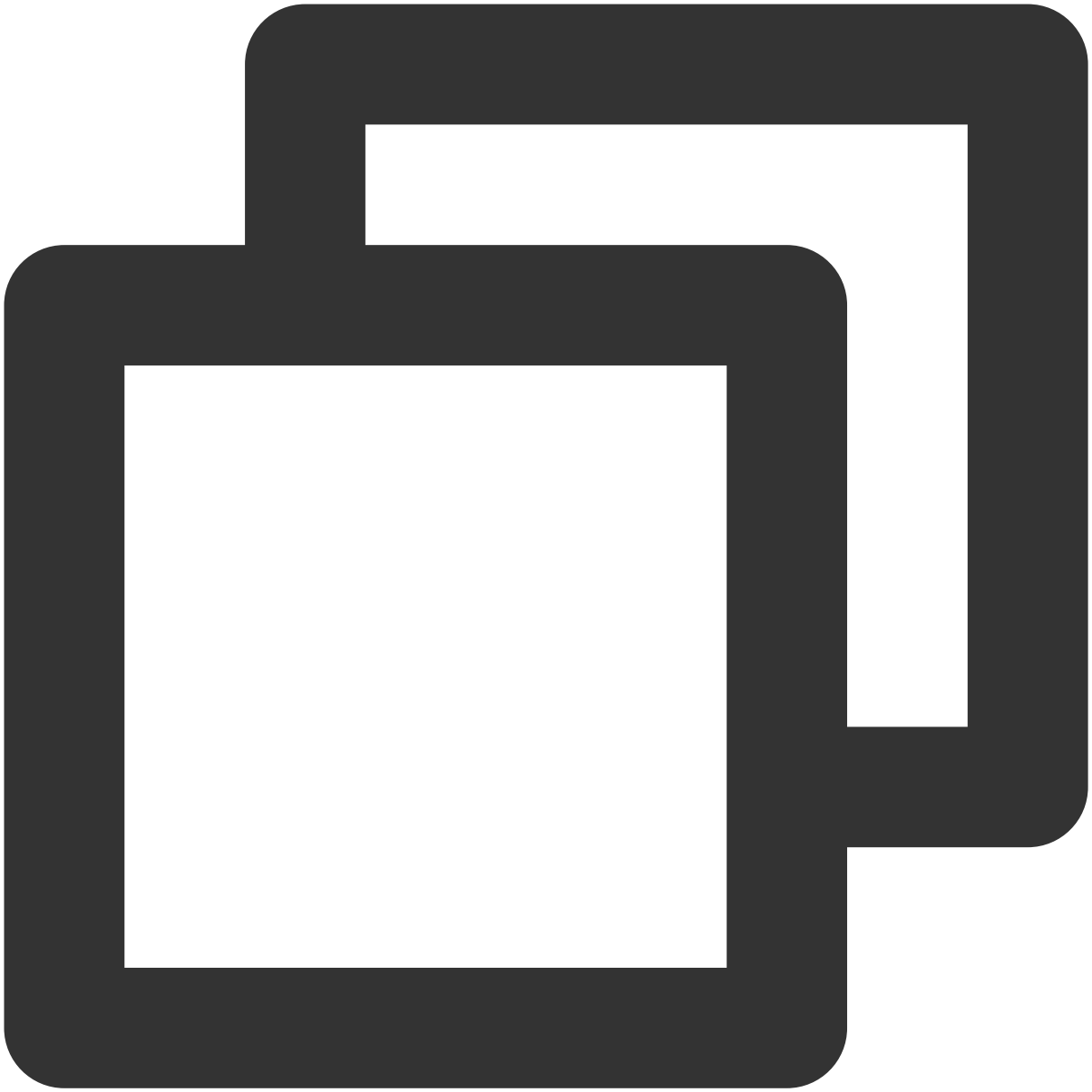
Automatically Creating Partition in PostgreSQL

Last updated：2024-01-24 11:20:59

In earlier versions of PostgreSQL, the table partitioning feature can be supported through inheritance; for example, a table partition can be created monthly by time, and data can be recorded in particular partitions. PostgreSQL 10 and later support declarative partitioning. This document describes how to create partitions in advance or in real time based on the written data.
The following are several common schemes for PostgreSQL to automatically create partitioned tables.

Use Cases

In practical use cases of partitioned tables, the time field is generally used as the partition key; for example, if the partition field type is timestamp, the partitioning method can be "list of values".
The table structure is as follows:



```
CREATE TABLE tab
(
    id   bigint GENERATED ALWAYS AS IDENTITY,
    ts   timestamp NOT NULL,
    data text
) PARTITION BY LIST ((ts::date));
CREATE TABLE tab_def PARTITION OF tab DEFAULT;
```

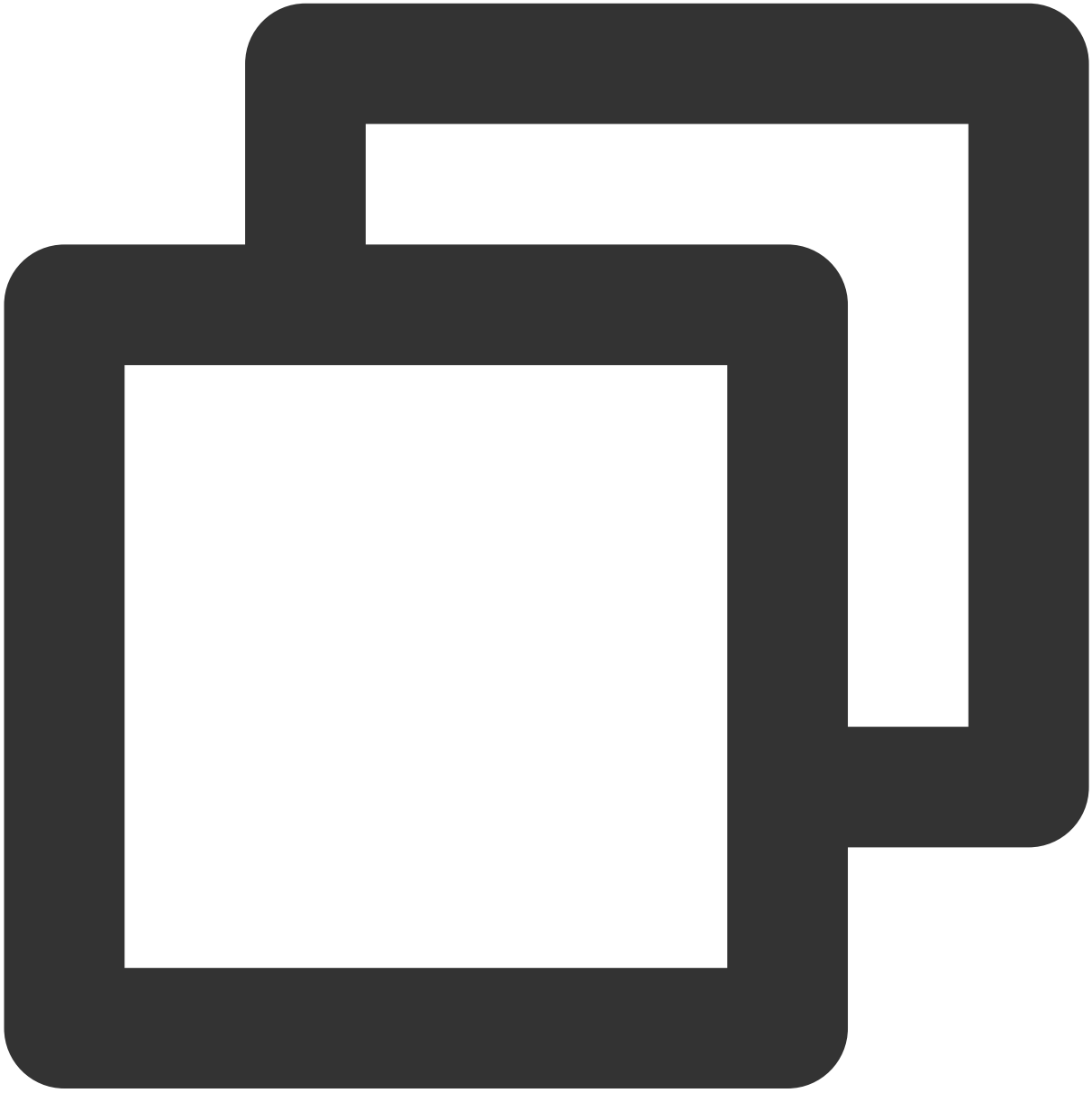
Partition creation is generally divided into the following two scenarios:

1. Scheduled partition creation

You can create partitions in advance with the help of a task scheduling tool. Common tools and partition creation methods are as follows:

Using system schedulers such as Crontab (Linux, Unix, etc.) and Task Scheduler (Windows)

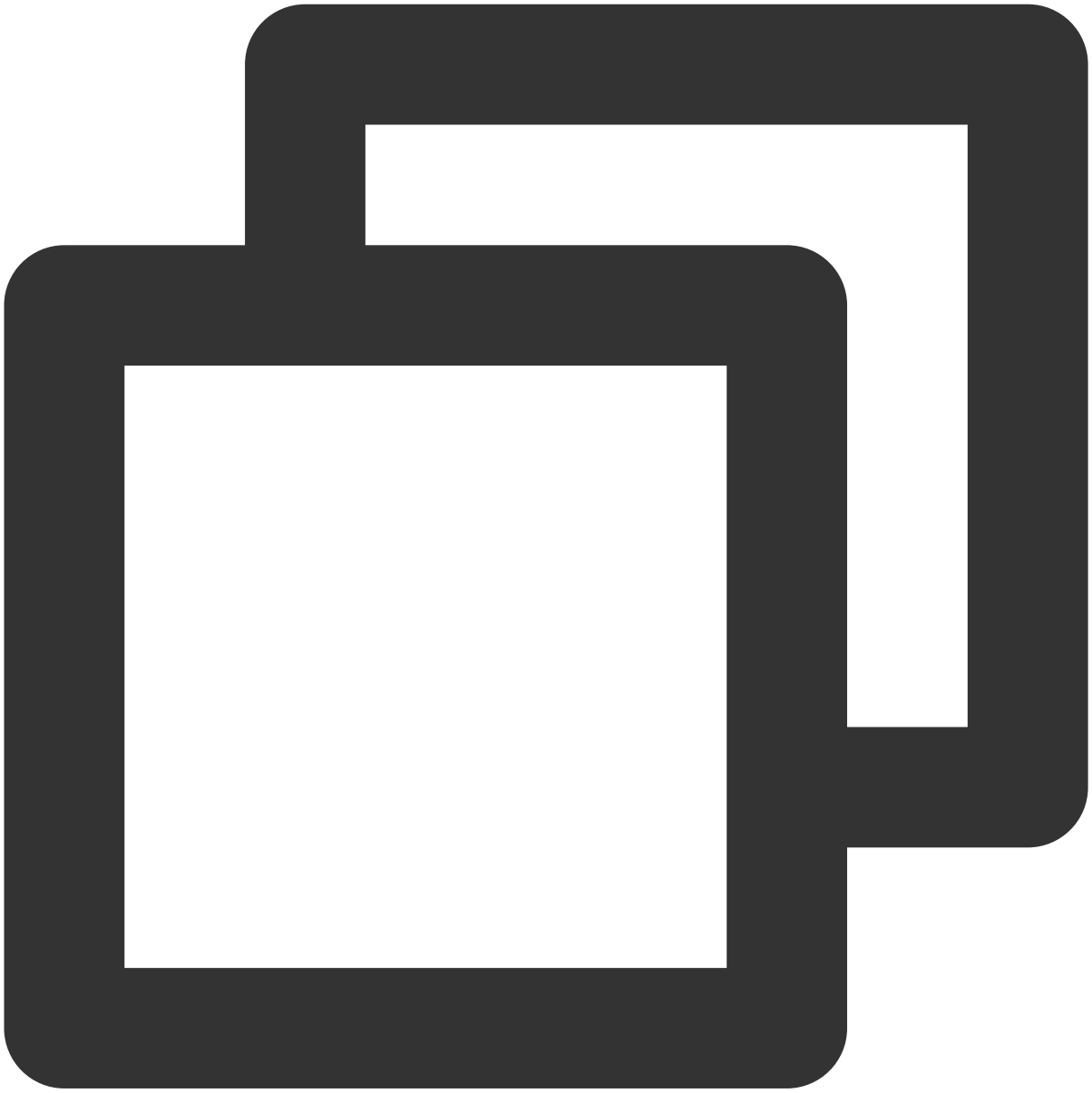
Taking Linux as an example, create a partitioned table at 14:00 every day for the next day:



```
cat > /tmp/create_part.sh <<EOF
dateStr=\\$(date -d '+1 days' +%Y%m%d);
psql -c "CREATE TABLE tab_\\$dateStr (LIKE tab INCLUDING INDEXES); ALTER TABLE tab
EOF
(crontab -l 2>/dev/null; echo "0 14 * * * bash /tmp/create_part.sh ") | crontab -
```

Using built-in schedulers such as pg_cron and pg_timetable

Taking pg_cron as an example, create a partitioned table at 14:00 every day for the next day:

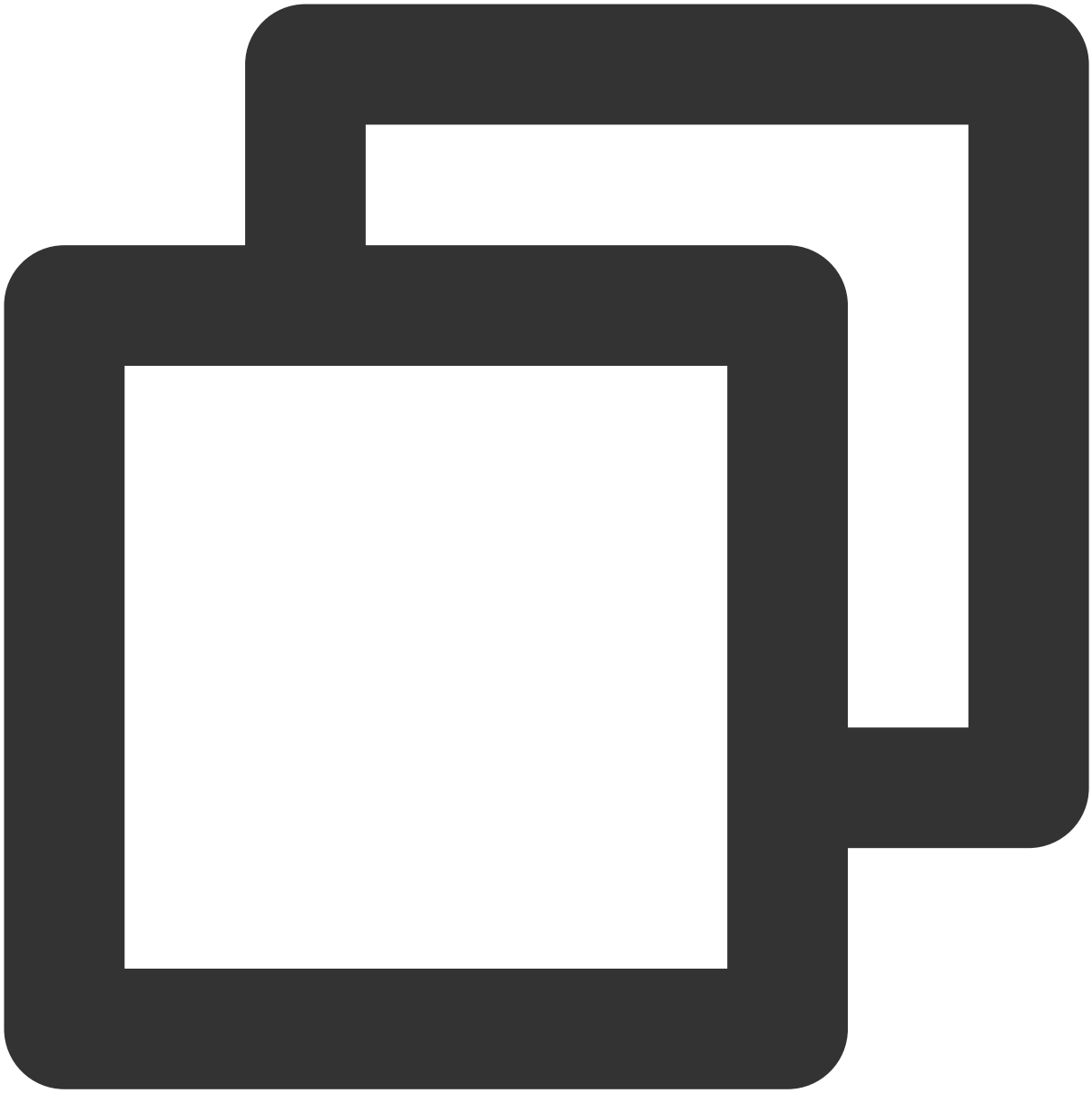


```
CREATE OR REPLACE FUNCTION create_tab_part() RETURNS integer
    LANGUAGE plpgsql AS
$$
DECLARE
    dateStr varchar;
BEGIN
```

```
    SELECT to_char(DATE 'tomorrow', 'YYYYMMDD') INTO dateStr;
    EXECUTE
        format('CREATE TABLE tab_%s (LIKE tab INCLUDING INDEXES)', dateStr);
    EXECUTE
        format('ALTER TABLE tab ATTACH PARTITION tab_%s FOR VALUES IN (%L)', dateSt
    RETURN 1;
END;
$$;

CREATE EXTENSION pg_cron;

SELECT cron.schedule('0 14 * * *', $$SELECT create_tab_part();$$);
```

Using dedicated partition management extensions such as pg_partman

Taking pg_partman as an example, create a partitioned table every day for the next day:



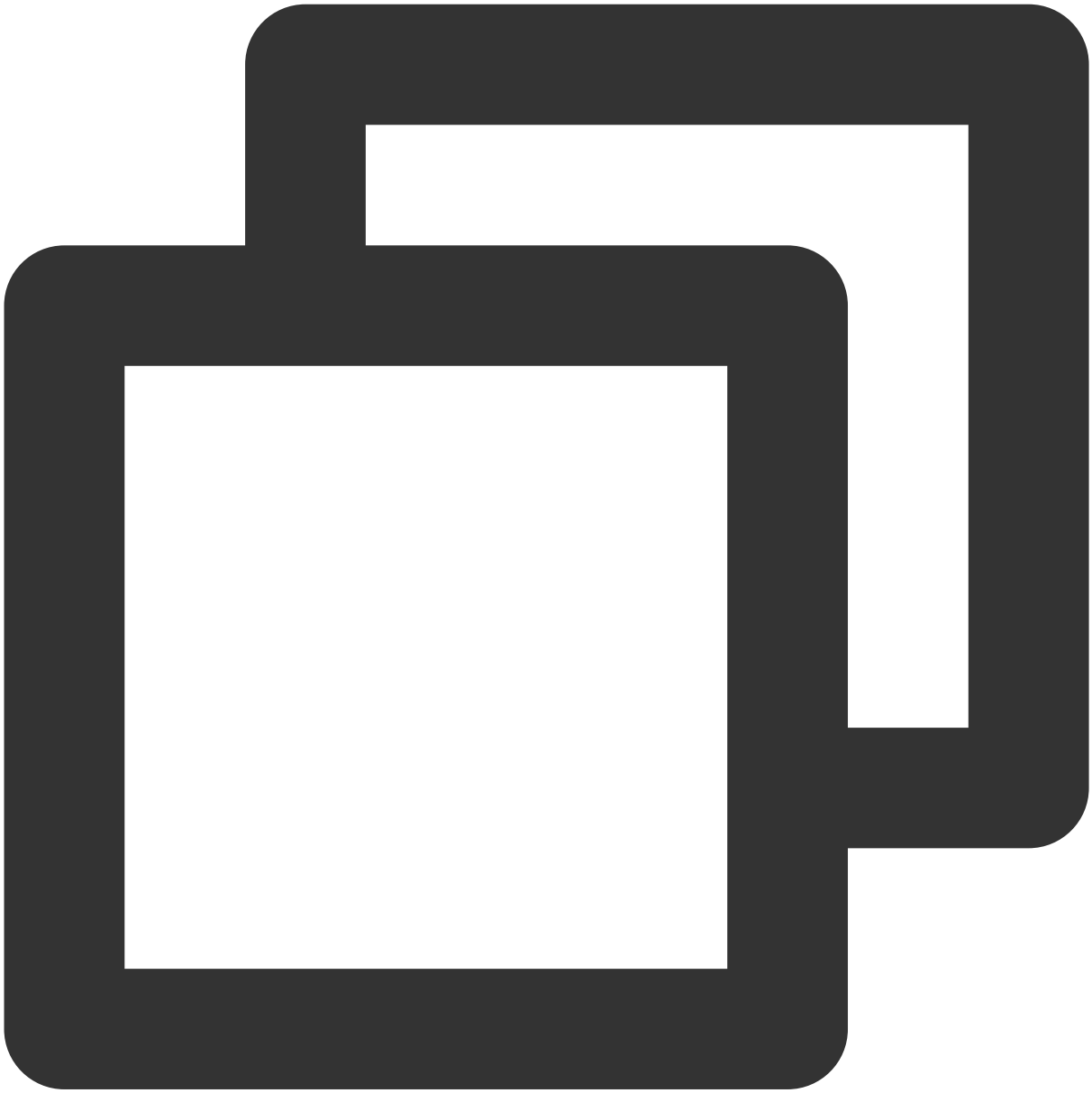
```
CREATE EXTENSION pg_partman;

SELECT partman.create_parent(p_parent_table => 'public.tab',
                             p_control => 'ts',
                             p_type => 'native',
                             p_interval=> 'daily',
                             p_premake => 1);
```

2. On-demand real-time partition creation

If you want to create partitions according to the need of data insertion, so you can determine whether there is data in a time range based on whether a partition exists, this generally can be implemented with triggers.

Note that there are two problems with this method:

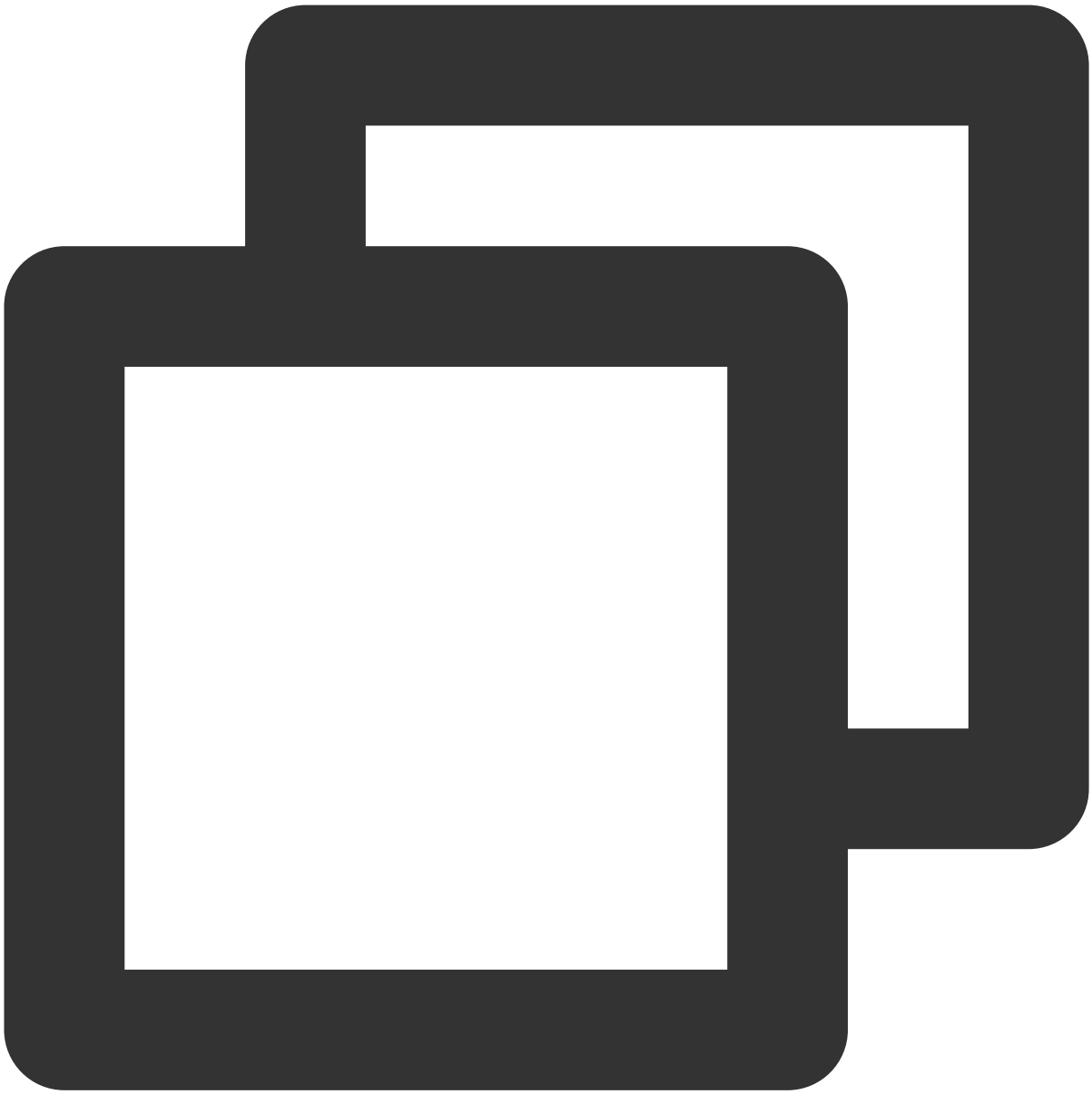
Only PostgreSQL 13 and later provide BEFORE/FOR EACH ROW triggers for partitioned tables.



```
ERROR:  "tab" is a partitioned table
DETAIL:  Partitioned tables cannot have BEFORE / FOR EACH ROW triggers.
```

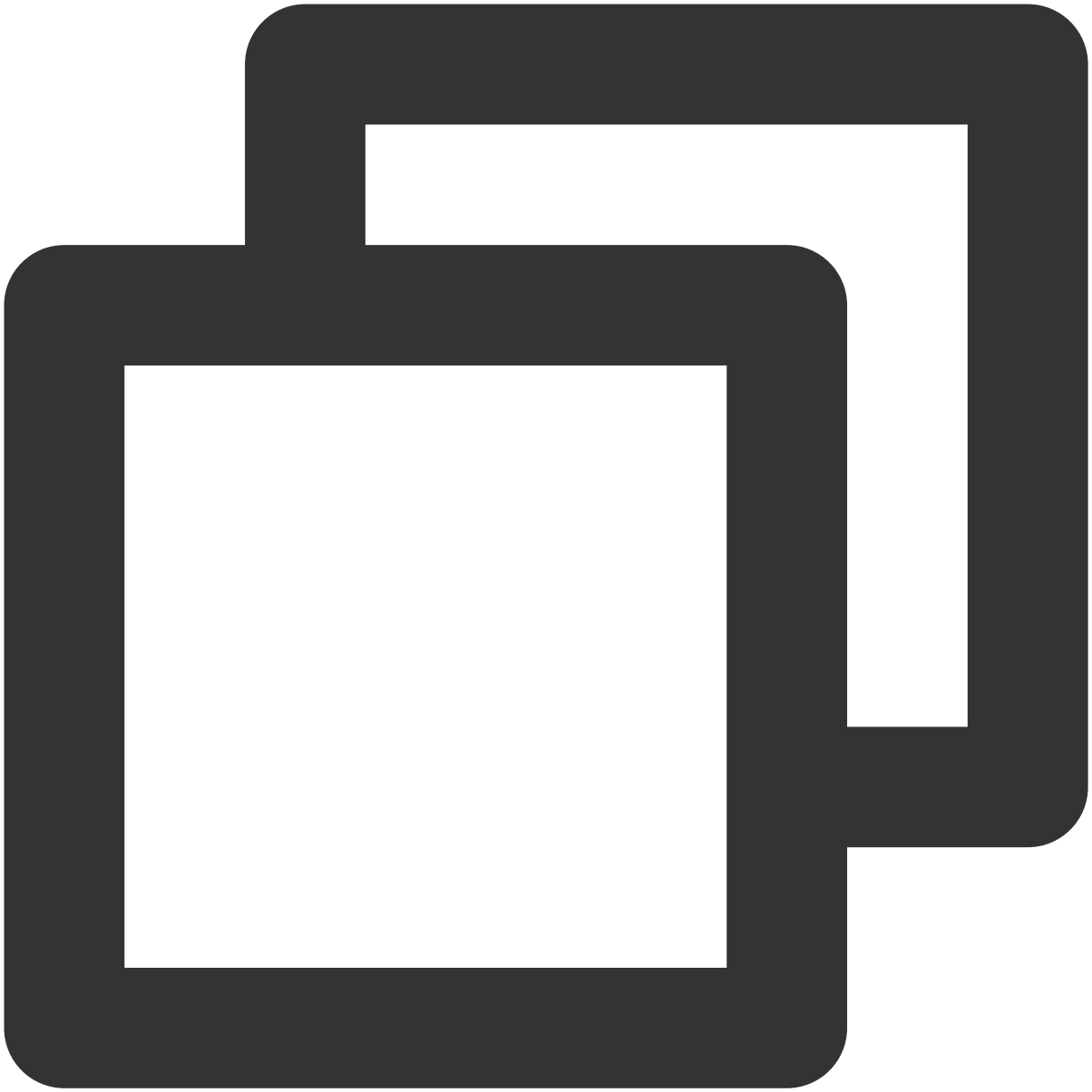
When data is inserted, the partitioned table definition cannot be modified due to the table lock; that is, child tables cannot be attached. Therefore, another connection must be used to perform the ATTACH operation. Here, the

LISTEN/NOTIFY mechanism can be used to ask another connection to modify the partition definition.



```
ERROR:  cannot CREATE TABLE .. PARTITION OF "tab"
      because it is being used by active queries in this session
Or
ERROR:  cannot ALTER TABLE "tab"
      because it is being used by active queries in this session
```

Trigger (implementing child table creation and NOTIFY)



```
CREATE FUNCTION part_trig() RETURNS trigger
    LANGUAGE plpgsql AS
$$
BEGIN
    BEGIN
        /* try to create a table for the new partition */
        EXECUTE
            format('CREATE TABLE %I (LIKE tab INCLUDING INDEXES)', 'tab_' || to_cha

        /*
         * tell listener to attach the partition
```

```
         * (only if a new table was created)
         */
        EXECUTE
            format('NOTIFY tab, %L', to_char(NEW.ts, 'YYYYMMDD'));
    EXCEPTION
        WHEN duplicate_table THEN
            NULL; -- ignore
    END;

    /* insert into the new partition */
    EXECUTE
        format('INSERT INTO %I VALUES ($1.*)', 'tab_' || to_char(NEW.ts, 'YYYYMMDD'
        USING NEW;

    /* skip insert into the partitioned table */
    RETURN NULL;
END;
$$;

CREATE TRIGGER part_trig
    BEFORE INSERT
    ON TAB
    FOR EACH ROW
    WHEN (pg_trigger_depth() < 1)
EXECUTE FUNCTION part_trig();
Code (implementing LISTEN and ATTACH for child tables)
#!/usr/bin/env python3.9
# encoding:utf8
import asyncio

import psycopg2
from psycopg2.extensions import ISOLATION_LEVEL_AUTOCOMMIT

conn = psycopg2.connect('application_name=listener')
conn.set_isolation_level(ISOLATION_LEVEL_AUTOCOMMIT)
cursor = conn.cursor()
cursor.execute(f'LISTEN tab;')


def attach_partition(table, date):
    with conn.cursor() as cs:
        cs.execute('ALTER TABLE "%s" ATTACH PARTITION "%s_%s" FOR VALUES IN (\\'%s\


def handle_notify():
    conn.poll()
    for notify in conn.notifies:
```

```
        print(notify.payload)
        attach_partition(notify.channel, notify.payload)
    conn.notifies.clear()

loop = asyncio.get_event_loop()
loop.add_reader(conn, handle_notify)
loop.run_forever()
```

Summary

This document describes two schemes for automatic partition creation as summarized below:
The solutions in the **scheduled partition creation** scenario are simple and easy to understand, but they depend on the schedule management mechanism of the system or extension and incur additional management costs during Ops and migration.
In the **on-demand real-time partition creation** scenario, the number of unnecessary partitions can be reduced according to the actual data pattern, but a later version (≥13) and an additional connection are required, making the scheme more complicated.
You can choose an appropriate automatic partition creation method based on your business conditions.

Scenario	Version	Implementation	Need of System Scheduler or Extension Required	Need of Additional Connection Mechanism Required	Cost
Scheduled partition creation	PostgreSQL 10	Easy	Yes	No	High
On-demand real-time partition creation	PostgreSQL 13 or later	Complicated	No	Yes	Low

Searching in High Numbers of Tags Based on pg_roaringbitmap

Last updated：2024-01-24 11:20:59

Many business use cases have the tag-based query feature. If the data volume and tag value quantity are high, a large amount of storage capacity will be used, and the performance will be poor. Therefore, how to filter target resources efficiently and quickly without taking up too much storage space has become a challenge for business management optimization.
This document describes how to easily search in an ultra high number of tags based on the pg_roaringbitmap extension.

pg_roaringbitmap Overview

pg_roaringbitmap is a compressed bitmap storage extension based on roaring bitmap. It supports roaring bitmap access as well as set, aggregation, and other operations.

Roaring Bitmap Usage

Roaring bitmap is often used to store user attribute tags in business use cases. You can create, read, update, and delete these attribute tags and filter specific users by tag union, intersection, etc. In this way, you can quickly find what you want from a massive amount of attribute data. This not only improves the performance, but also reduces the used storage space, making it very useful for big data analysis scenarios.
For example, in traditional mode, a music application has a user tag list as follows:

User ID	Username	Interest Tag
1	John	{Classical, jazz, R&B, country}
2	Jane	{Folk, instrumental}
3	Harry	{Hip hop, jazz, R&B, reggae}
...	...	...
1000000000	Text 2	{Rock}

To find all users who like instrumental music, the system needs to search for them against the interest tag column, find rows with "instrumental" in the tag, and return the data to the application.

Generally, the simplest way is to create a user interest table in the database according to the structure of the above table first, and then run an array query statement to find interest tags for inclusion search. However, if there are high volumes of data and tag values, more storage space will be used, and the performance will be very poor. Therefore, you need to find an alternative. You can split this table into three tables, use the interest tag as the primary key, and store tagged users as bitmaps as shown below:

User table:

User ID	Username
1	John
2	Jane
N	...

Tag table:

Tag ID	Username
1	Classical
2	Folk
N	...

User tag table:

Tag ID	Username
1	[1,3,7,123,423]
2	[5,31]
N	...

When you need to find users who like listening to classical and folk music at the same time, you can directly perform a bitmap query by user ID in the user tag table. This can greatly improve the performance and reduce the capacity usage.

Query Performance Comparison Between Traditional Method and Roaring Bitmap Method

Preparing test scenario

1. Create a function for random character generation.

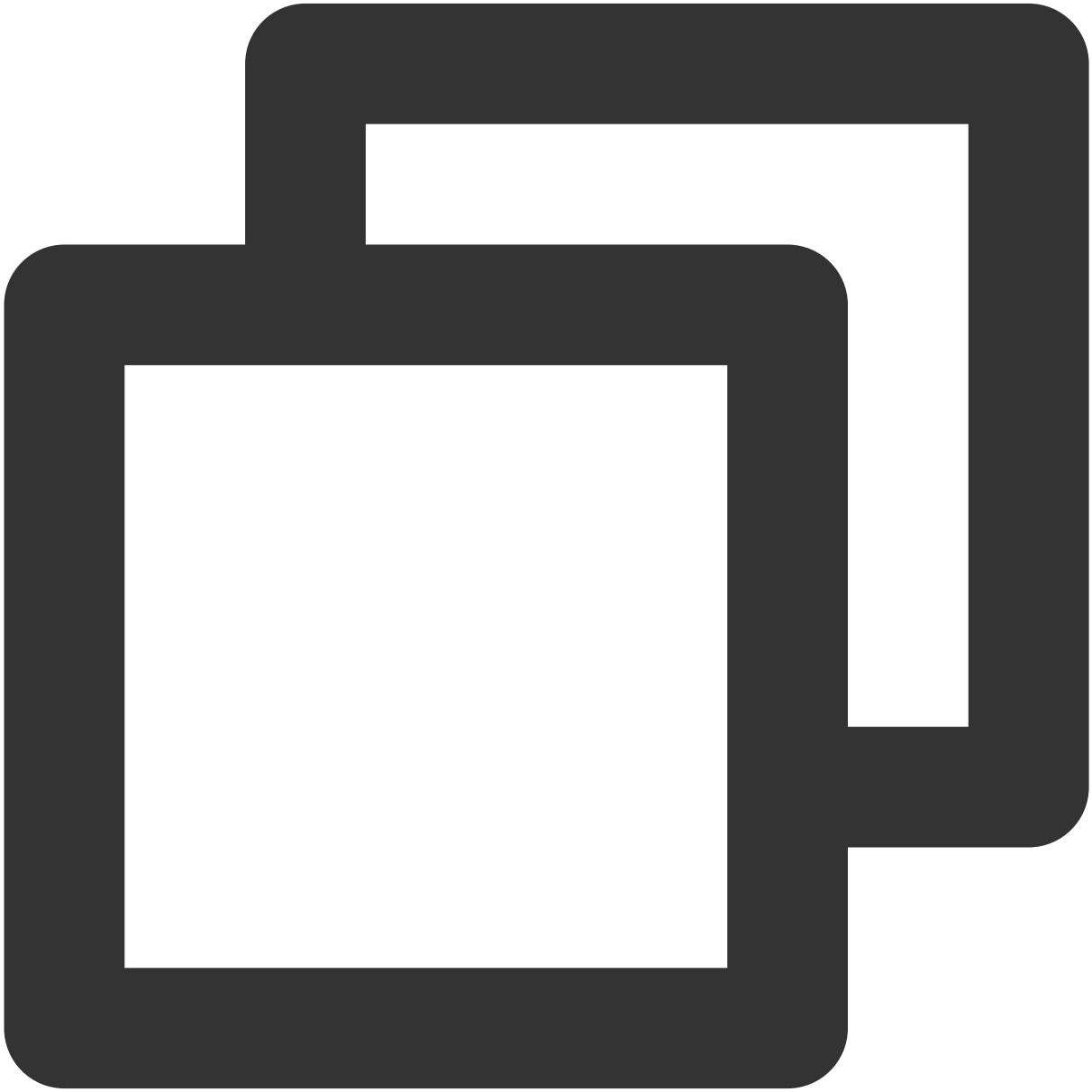
```
create or replace function random_string(length integer) returns text as
$$
declare
chars text[] := '{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W
result text := '';
i integer := 0;
length2 integer := (select trunc(random() * length + 1));
begin
```

```
if length2 < 0 then
raise exception 'Given length cannot be less than 0';
end if;
for i in 1..length2 loop
result := result || chars[1+random()*(array_length(chars, 1)-1)];
end loop;
return result;
end;
$$ language plpgsql;
```

2. Create a function that generates an array of random integers.

```
create or replace function random_int_array(int, int)
returns int[] language sql as
$$
select array_agg(round(random()* $1)::int)
from generate_series(1, $2)
$$;
```

3. Create a function that generates an array of random characters.



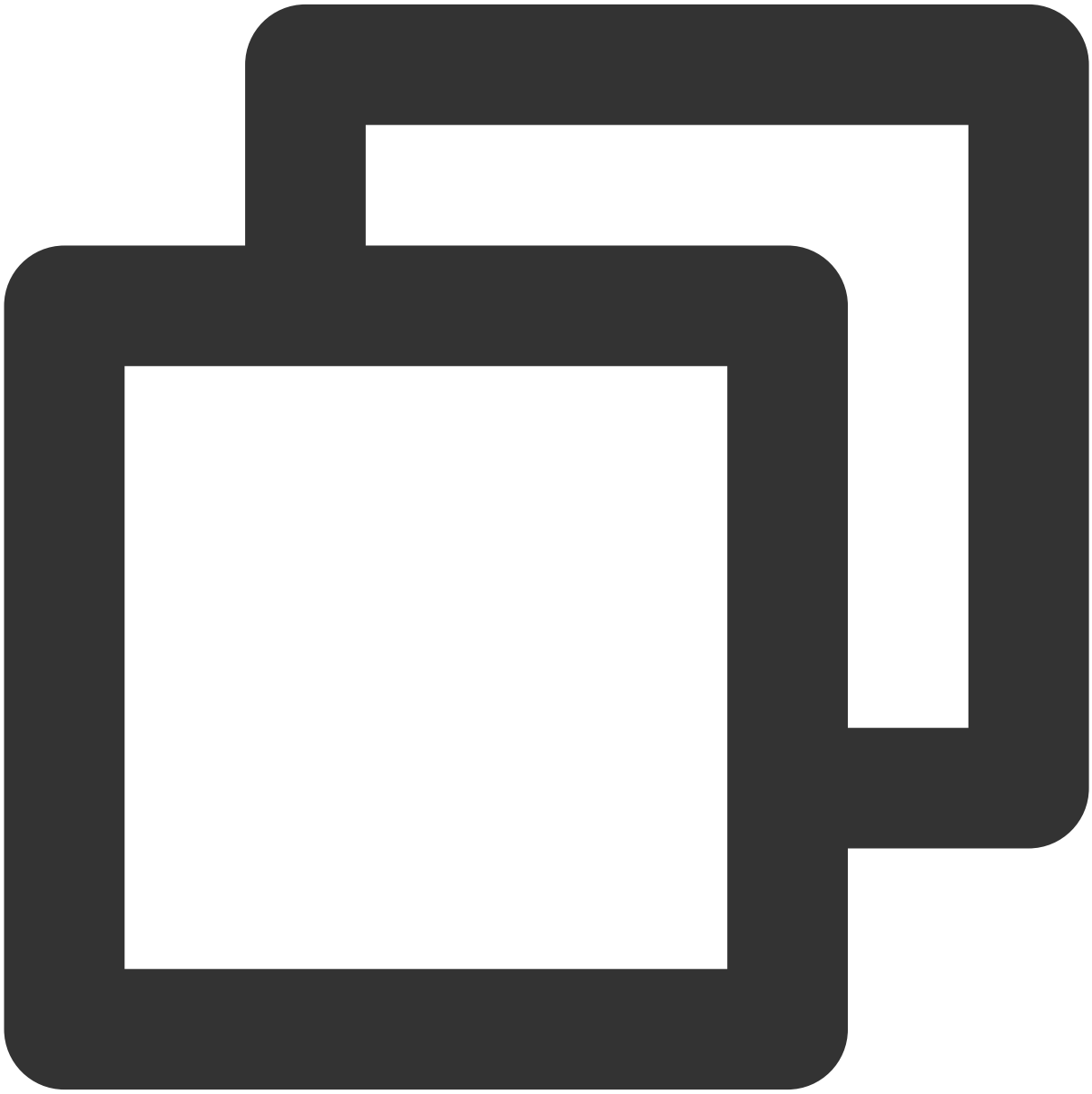
```
create or replace function random_string_array(int, int)
returns TEXT[] language sql as
```

```
$$
select array_agg(random_string($1)) from generate_series(1, $2);
$$;
```

Scheme 1: Traditional method

One table does it all.

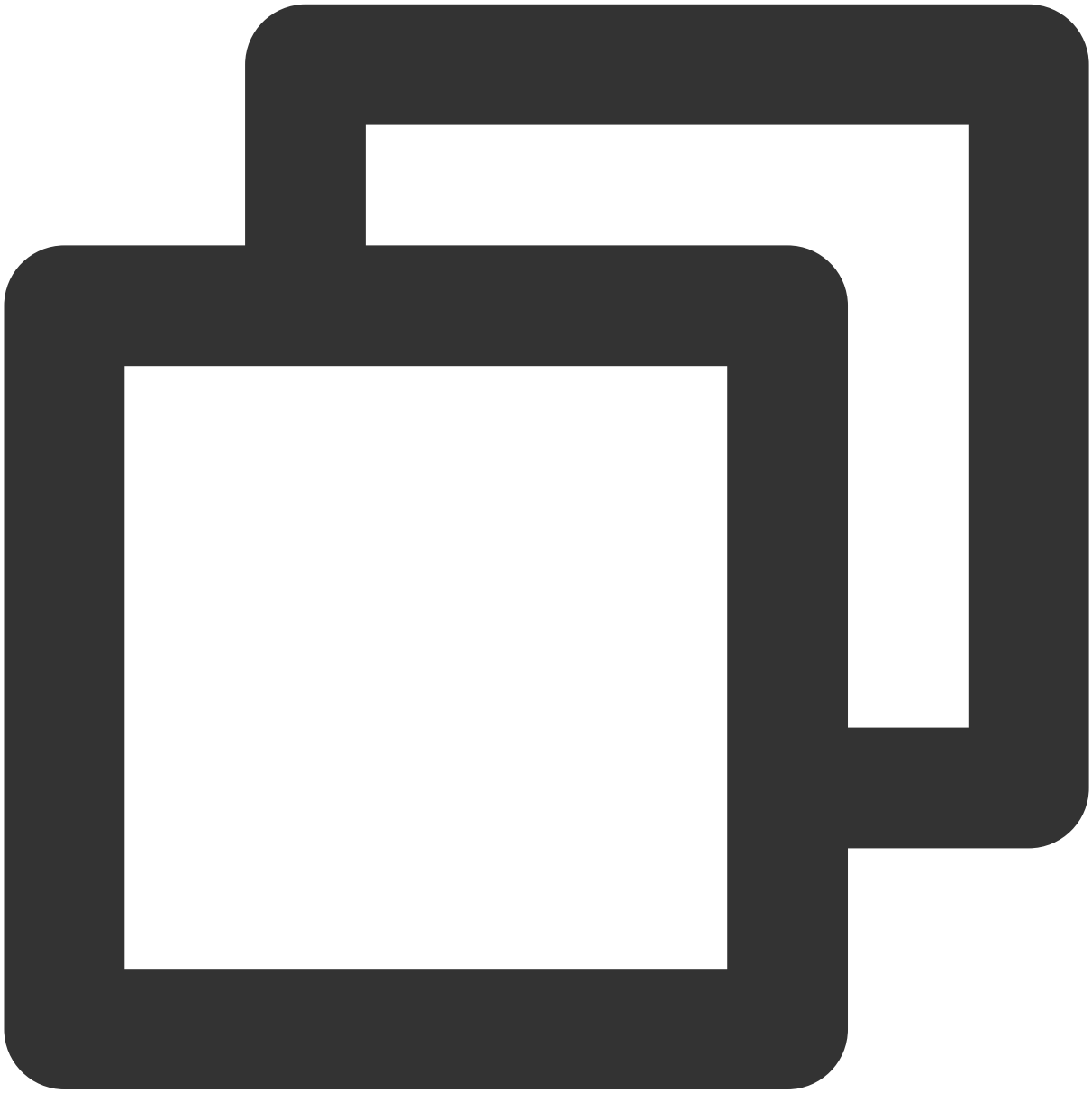
1. Create a table containing all the data.



```
create table account(
uin bigint primary KEY,
```

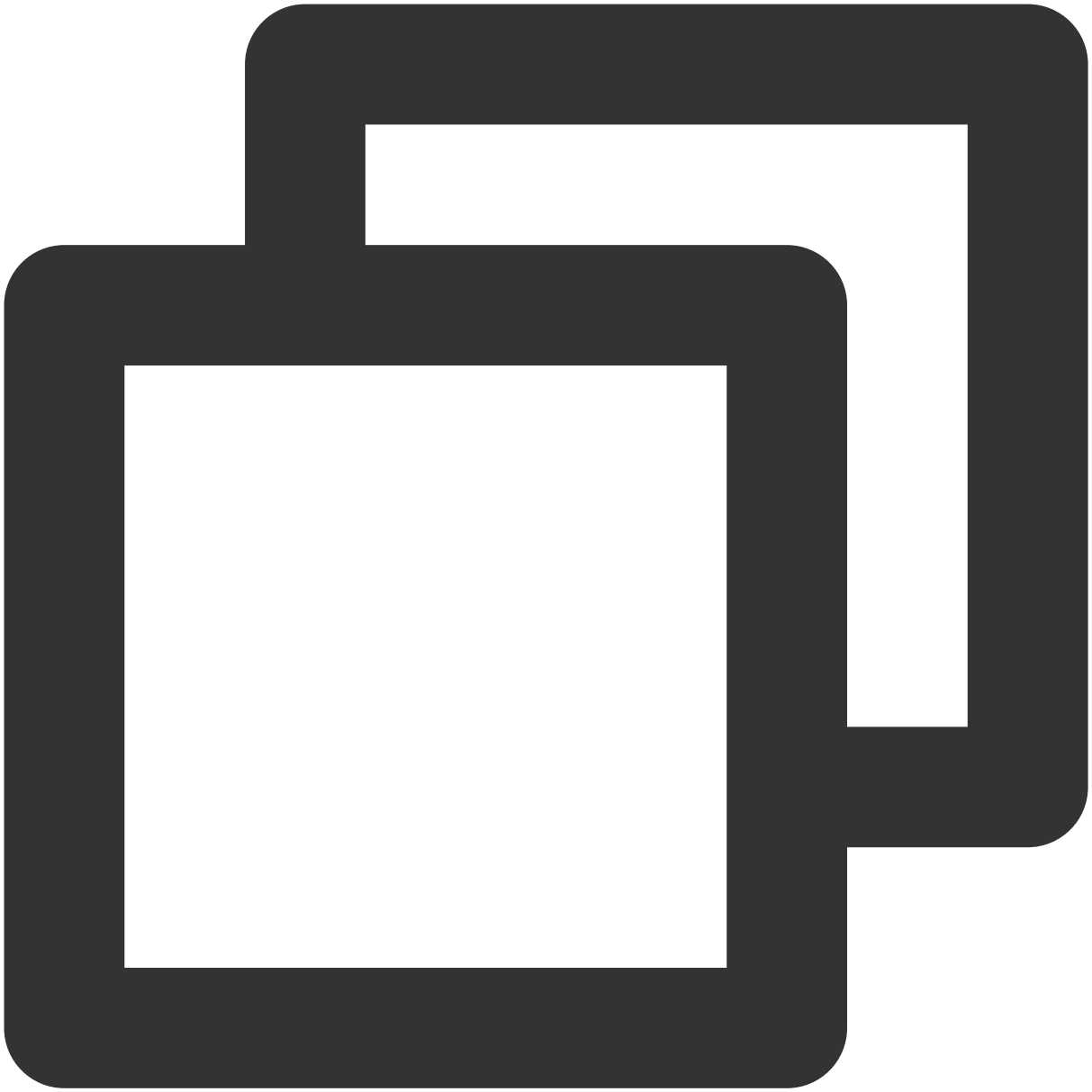
```
name varchar,
tag TEXT []
);
```

2. Insert the data of ten million simulated accounts with the function as described in the preparations, and then create a GIN index.



```
insert into account select generate_series(1,10000000), random_string(20),random_st
create index tag_inx on account USING GIN(tag);
```

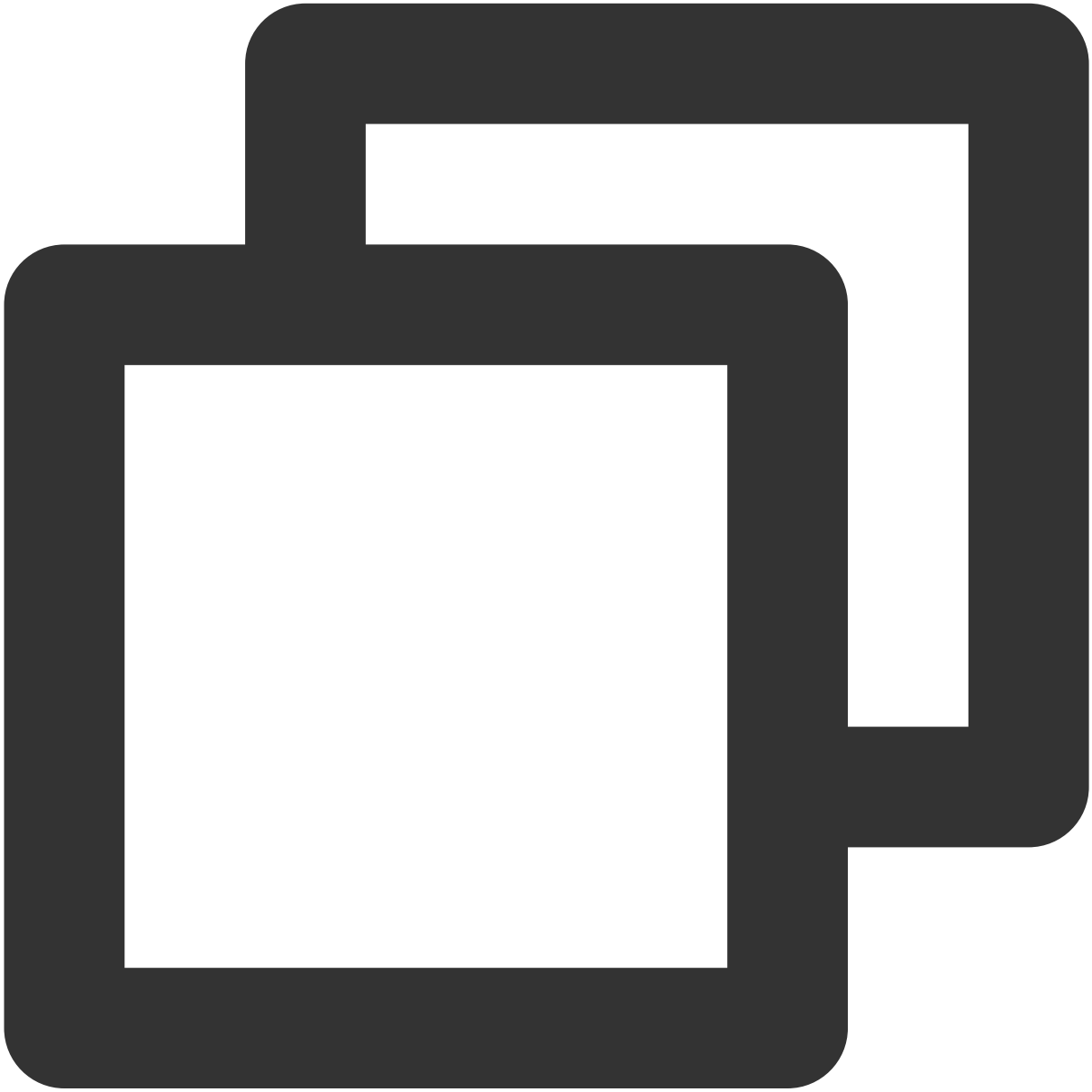
3. Run a query to list users with tags `GN` and `o` .



```
explain analyze select uin,name from account where tag @>ARRAY['GN','o'];

QUERY PLAN

-----------------------------------------------------------------------------------
-----------------
Bitmap Heap Scan on account (cost=52.81..466.86 rows=105 width=19) (actual time=4.2
184 loops=1)
Recheck Cond: (tag @> '{GN,o}'::text[])
Heap Blocks: exact=184
-> Bitmap Index Scan on tag_inx (cost=0.00..52.78 rows=105 width=0) (actual time=4.
```

```
ows=184 loops=1)
Index Cond: (tag @> '{GN,o}'::text[])
Planning Time: 0.108 ms
Execution Time: 4.528 ms
```

4. Run a query to list xx users with tags `lvXe` and `Zt` (the query will be slow when executed for the first time).



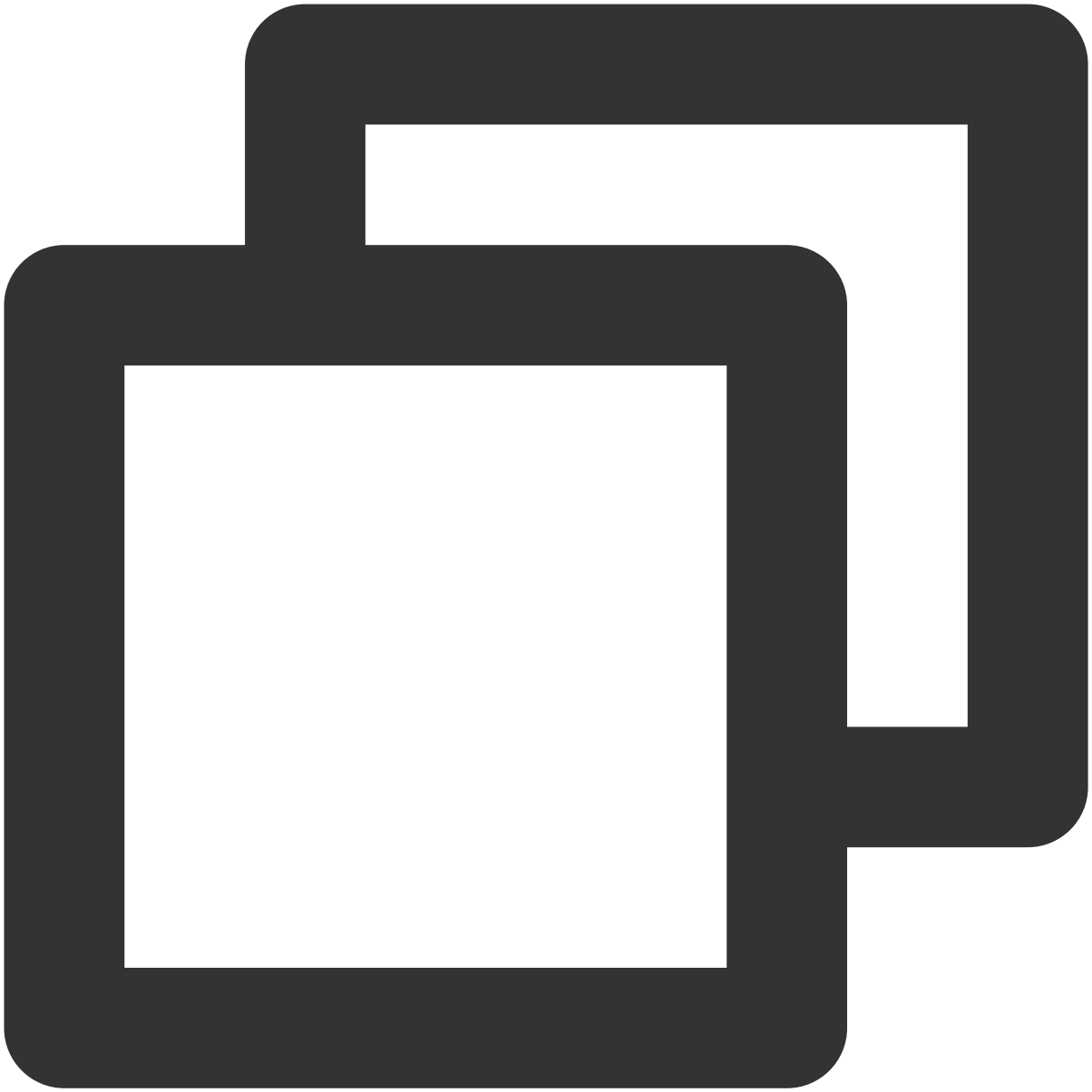
```
explain analyze select count(uin) from account where tag && ARRAY['lvXe','Zt'];

-----------------------------------------------------------------------------------
-------------------------
```

```
Aggregate (cost=21816.39..21816.40 rows=1 width=8) (actual time=8.236..8.238 rows=1
-> Bitmap Heap Scan on account (cost=109.08..21800.56 rows=6332 width=8) (actual ti
901 rows=5390 loops=1)
Recheck Cond: (tag && '{lvXe,Zt}'::text[])
Heap Blocks: exact=5327
-> Bitmap Index Scan on tag_inx (cost=0.00..107.49 rows=6332 width=0) (actual time=
.0.962 rows=5390 loops=1)
Index Cond: (tag && '{lvXe,Zt}'::text[])
Planning Time: 0.110 ms
Execution Time: 8.270 ms
```

Scheme 2: Optimized scheme

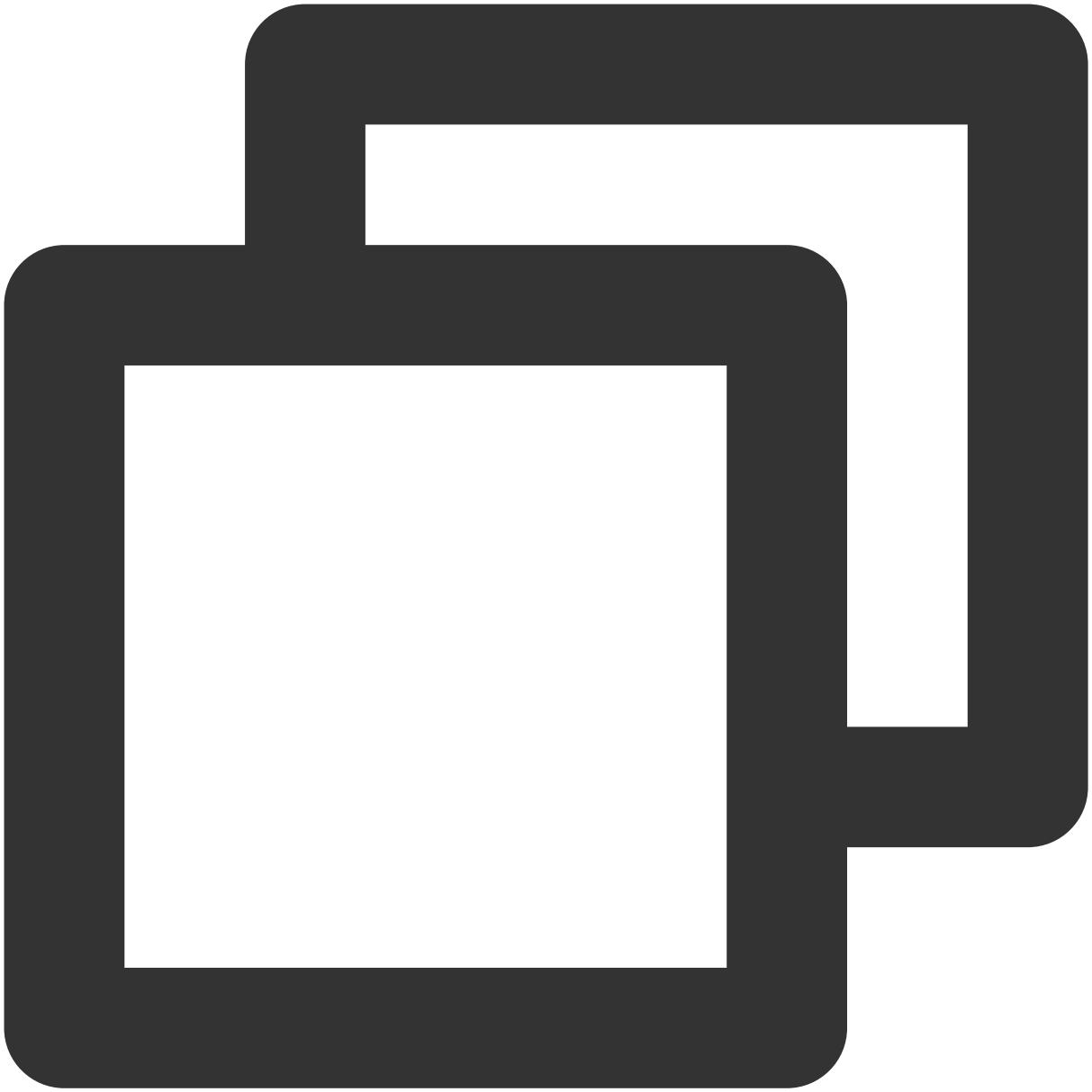
In order to reduce the performance loss caused by the types of tag fields in the query, change the actual `tag` in the above table to `tagid` .

1. Introduce a new tag dictionary table.



```
create table tag_dict (
tagid int primary key,
taginfo text
);
```

2. Suppose there are 100,000 dictionary types in total.

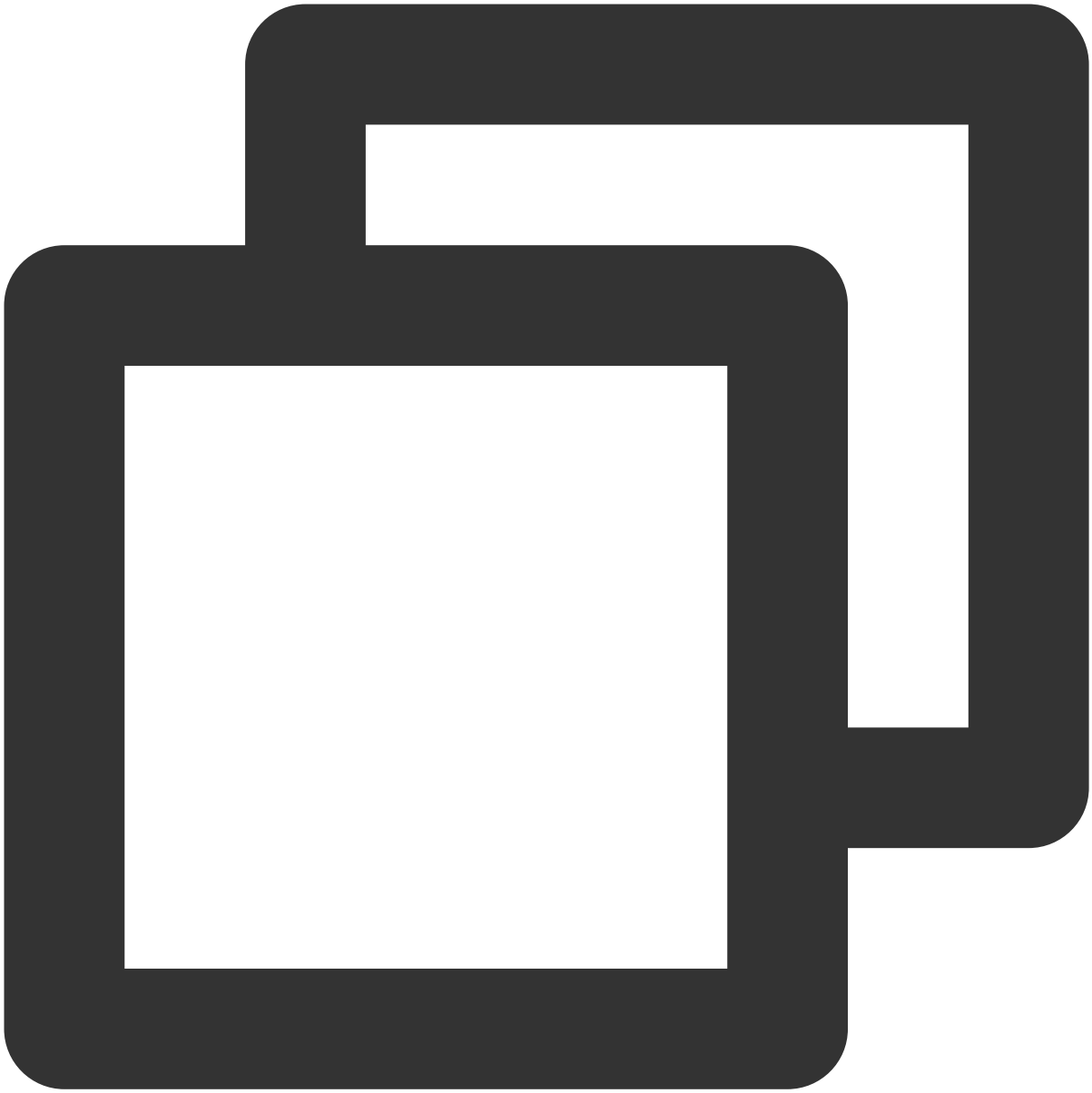


```
insert into tag_dict select generate_series(1,100000), md5(random()::text);
```

3. Create another table to store user and tag information.

```
create table account1(
uin bigint primary KEY,
name varchar,
tag INT []
);
```

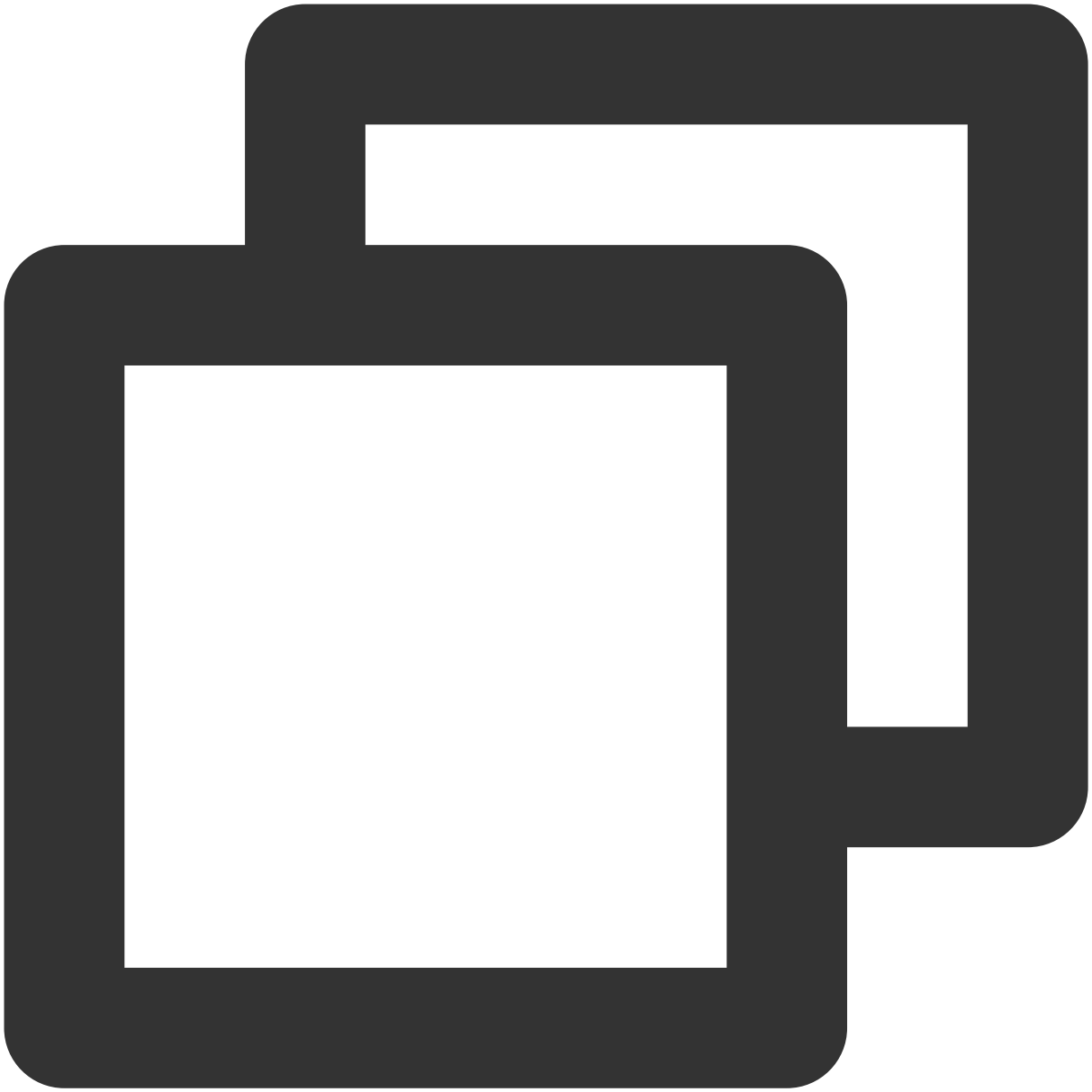
4. Insert the data of ten million accounts.



```
insert into account1 select generate_series(1,10000000), random_string(20),random_i
```

5. List users with both tag IDs 100 and 5711.

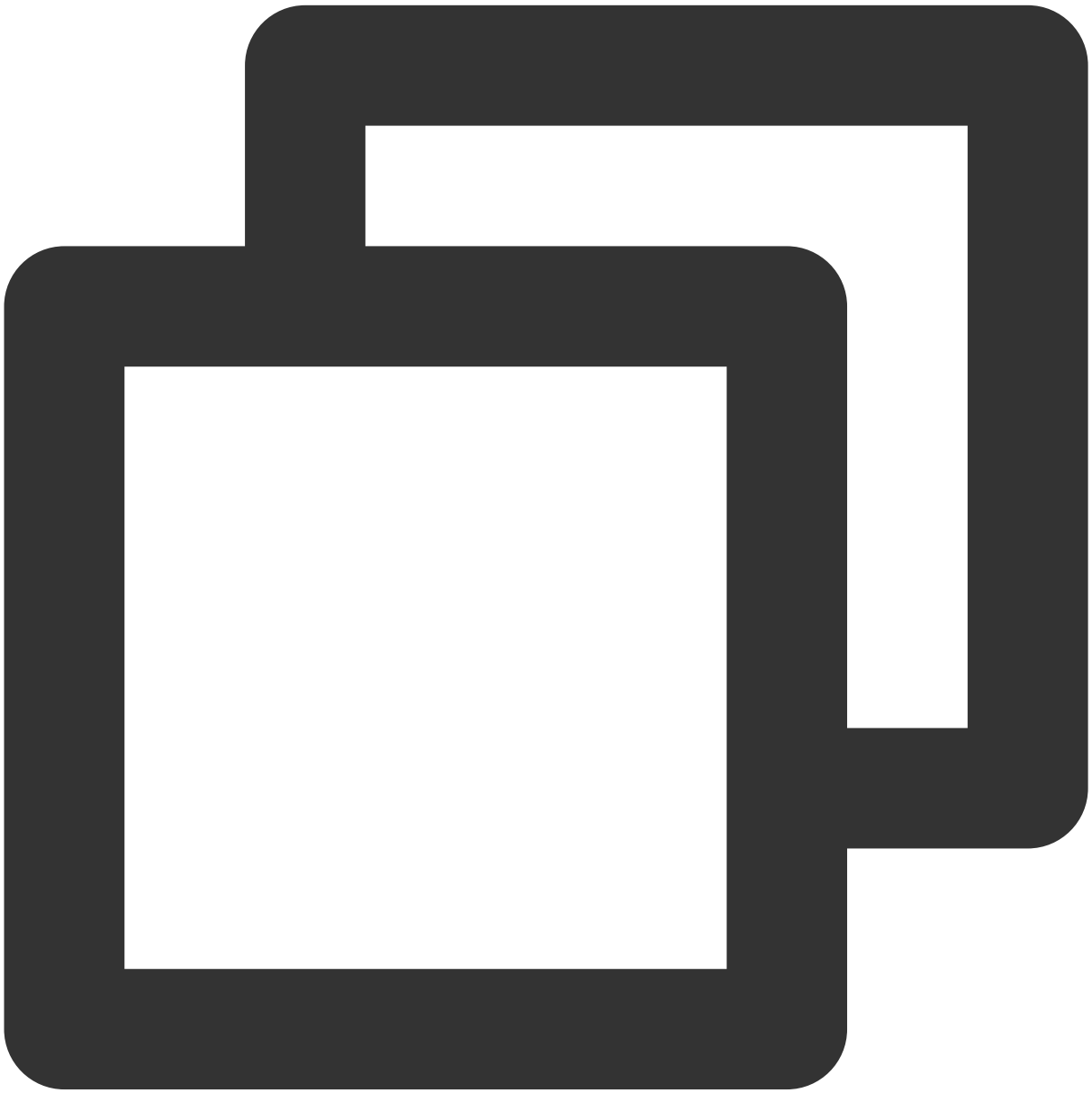
Before indexing:



```
test=> explain analyze select uin,name from account1 where tag @> ARRAY[100,5711];
QUERY PLAN
-----------------------------------------------------------------------------------
Gather (cost=1000.00..191007.68 rows=250 width=19) (actual time=982.585..1000.806 r
Workers Planned: 2
Workers Launched: 2
-> Parallel Seq Scan on account1 (cost=0.00..189982.68 rows=104 width=19) (actual t
Filter: (tag @> '{100,5711}'::integer[])
Rows Removed by Filter: 3333333
Planning Time: 0.205 ms
JIT:
```

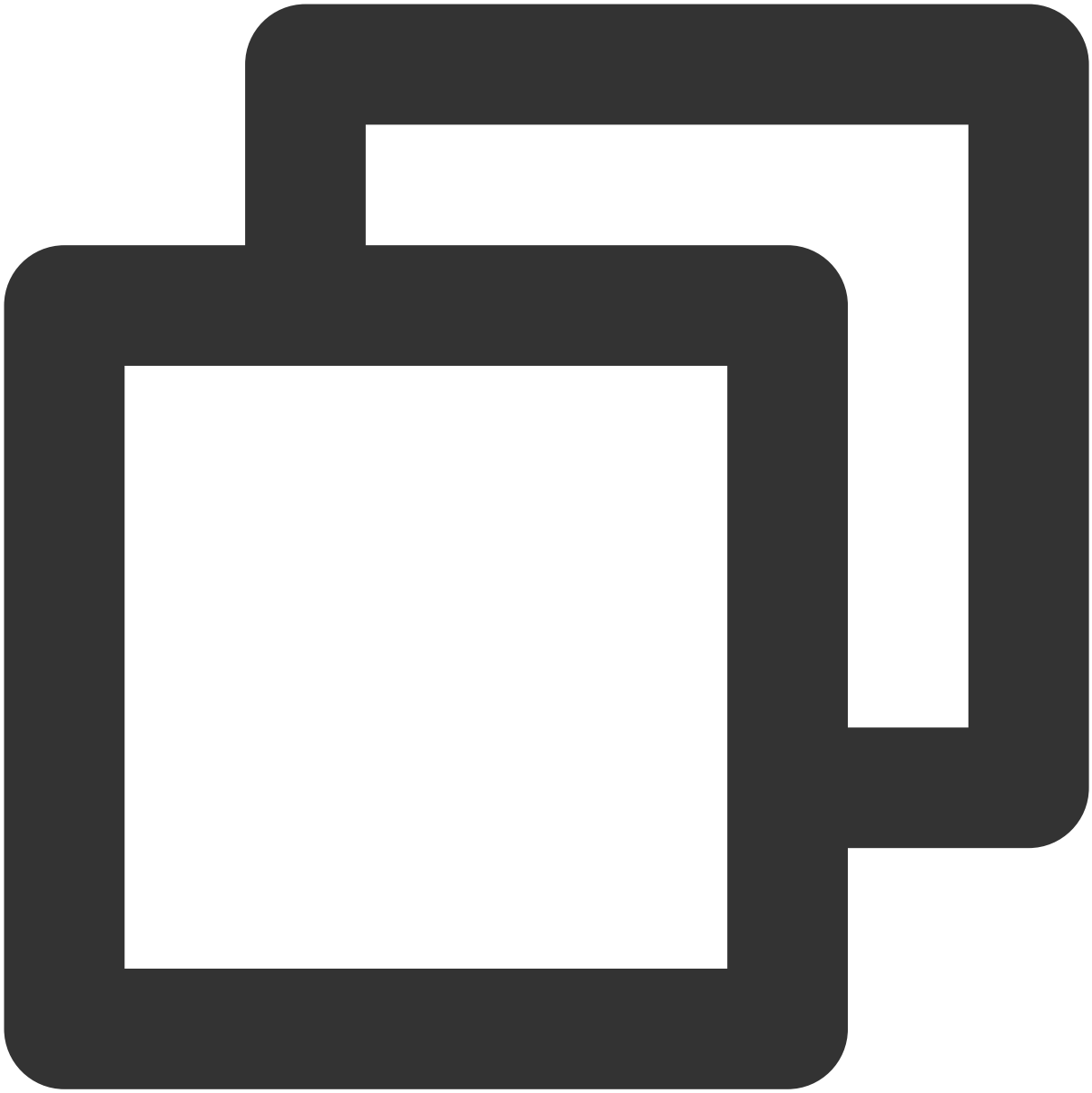
```
Functions: 12
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 2.280 ms, Inlining 0.000 ms, Optimization 1.176 ms, Emission 14.
Execution Time: 1001.574 ms
(12 rows)
```

Add an index:



```
create index tag_inx_2 on account1 USING GIN(tag);
```

After indexing:



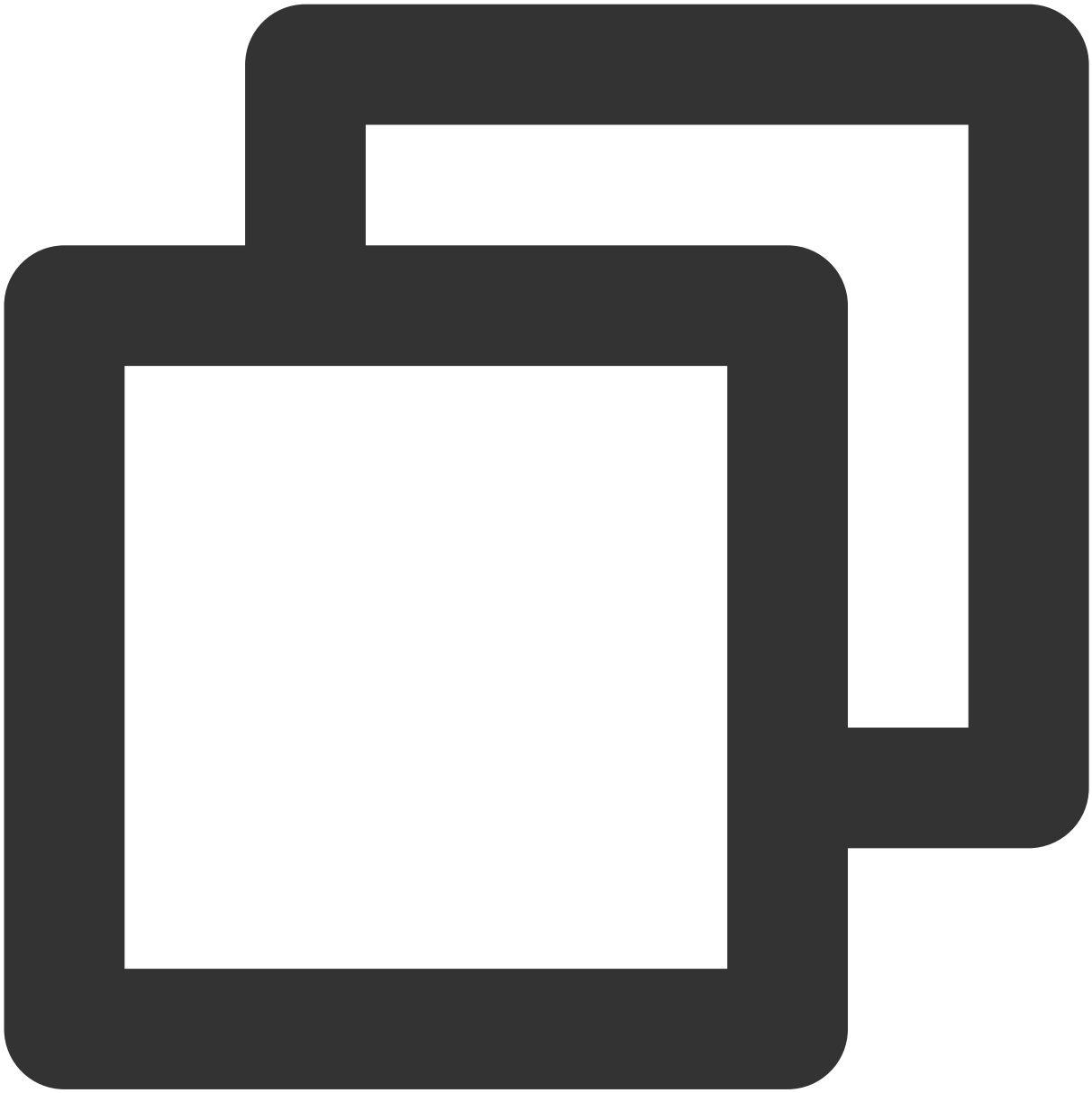
```
test=> explain analyze select uin,name from account1 where tag @> ARRAY[100,5711];
QUERY PLAN
-------------------------------------------------------------------------------
Bitmap Heap Scan on account1 (cost=49.94..1021.13 rows=250 width=19) (actual time=0
Recheck Cond: (tag @> '{100,5711}'::integer[])
-> Bitmap Index Scan on tag_inx_2 (cost=0.00..49.87 rows=250 width=0) (actual time=
Index Cond: (tag @> '{100,5711}'::integer[])
Planning Time: 0.410 ms
Execution Time: 0.171 ms
(6 rows)
```

6. List users with both tag IDs 61568 and 97350.

```
test=> explain analyze select uin,name from account1 where tag @> ARRAY[61568,97350
QUERY PLAN
-----------------------------------------------------------------------------------
Bitmap Heap Scan on account1 (cost=49.94..1021.13 rows=250 width=19) (actual time=0
Recheck Cond: (tag @> '{61568,97350}'::integer[])
Heap Blocks: exact=1
-> Bitmap Index Scan on tag_inx_2 (cost=0.00..49.87 rows=250 width=0) (actual time=
Index Cond: (tag @> '{61568,97350}'::integer[])
Planning Time: 0.071 ms
```

```
Execution Time: 0.151 ms
(7 rows)
```

7. List xx users who share interests with xx users (tag IDs 100 and 5711).



```
test=> explain analyze select count(uin) from account1 where tag && ARRAY[61568,973
QUERY PLAN
-------------------------------------------------------------------------------------
------
Gather (cost=1961.06..173801.15 rows=99750 width=19) (actual time=5.020..28.885 row
Workers Planned: 2
Workers Launched: 2
```

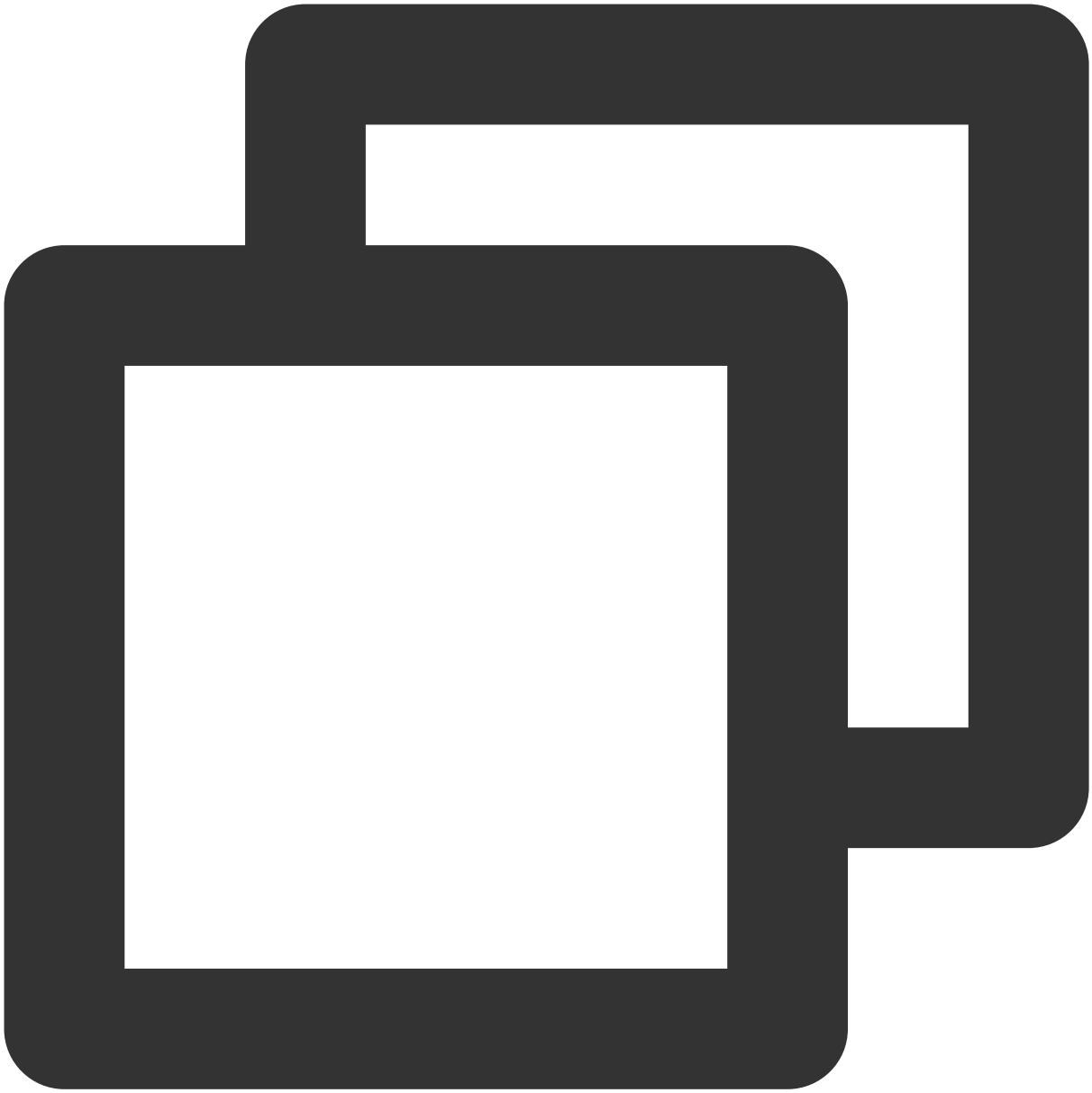
```
-> Parallel Bitmap Heap Scan on account1 (cost=961.06..162826.15 rows=41562 width=1
ps=3)
Recheck Cond: (tag && '{61568,97350}'::integer[])
Heap Blocks: exact=2053
-> Bitmap Index Scan on tag_inx_2 (cost=0.00..936.12 rows=99750 width=0) (actual ti
Index Cond: (tag && '{61568,97350}'::integer[])
Planning Time: 0.082 ms
JIT:
Functions: 12
Options: Inlining false, Optimization false, Expressions true, Deforming true
Timing: Generation 2.078 ms, Inlining 0.000 ms, Optimization 0.270 ms, Emission 3.4
Execution Time: 29.725 ms
(14 rows)
```

Scheme 3: Roaring bitmap

1. Create the extension first. It is integrated in TencentDB for PostgreSQL natively, so you don't need to care about compilation and other operations. You can directly create it in the database.

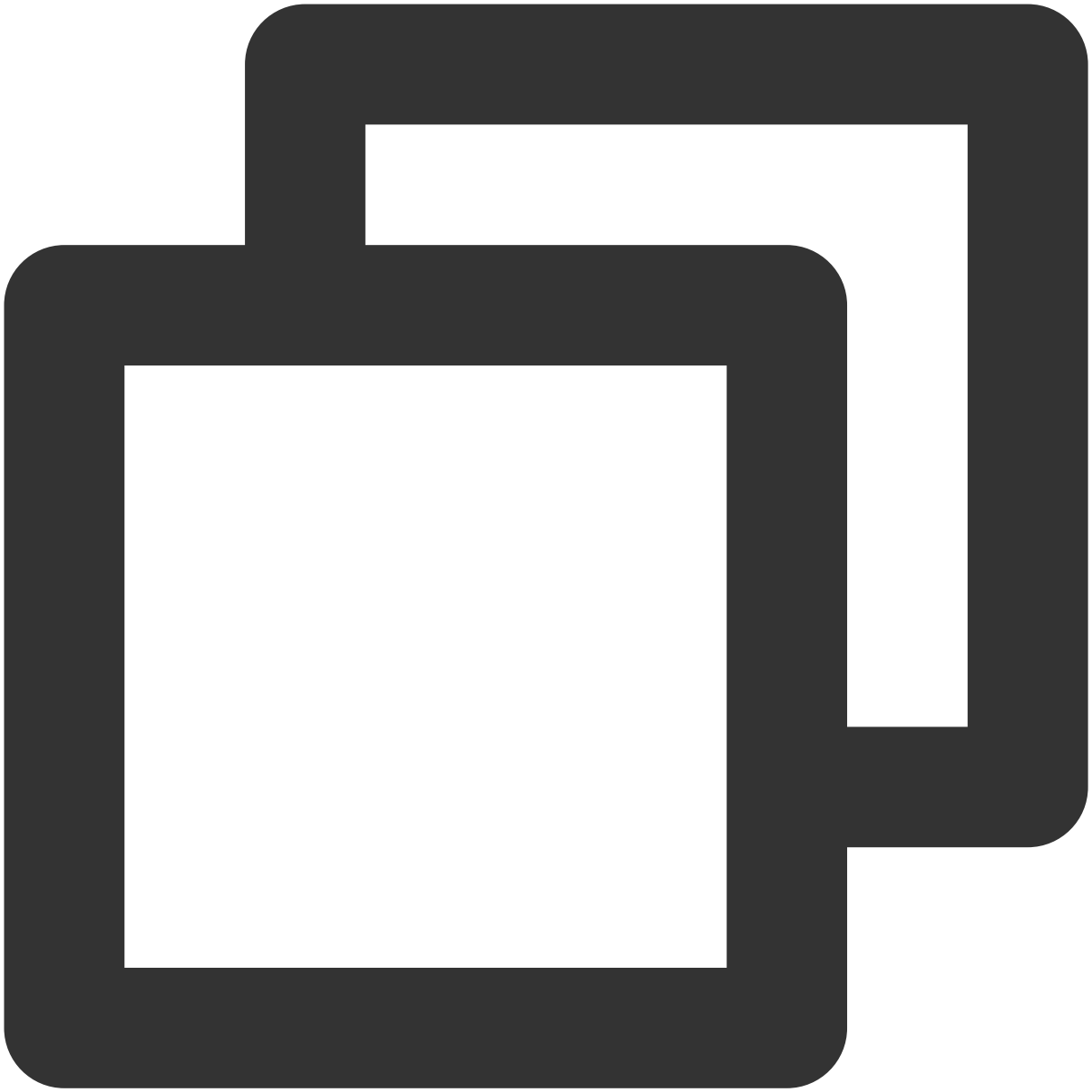
```
create extension roaringbitmap;
```

2. Create a tag-user mapping table.



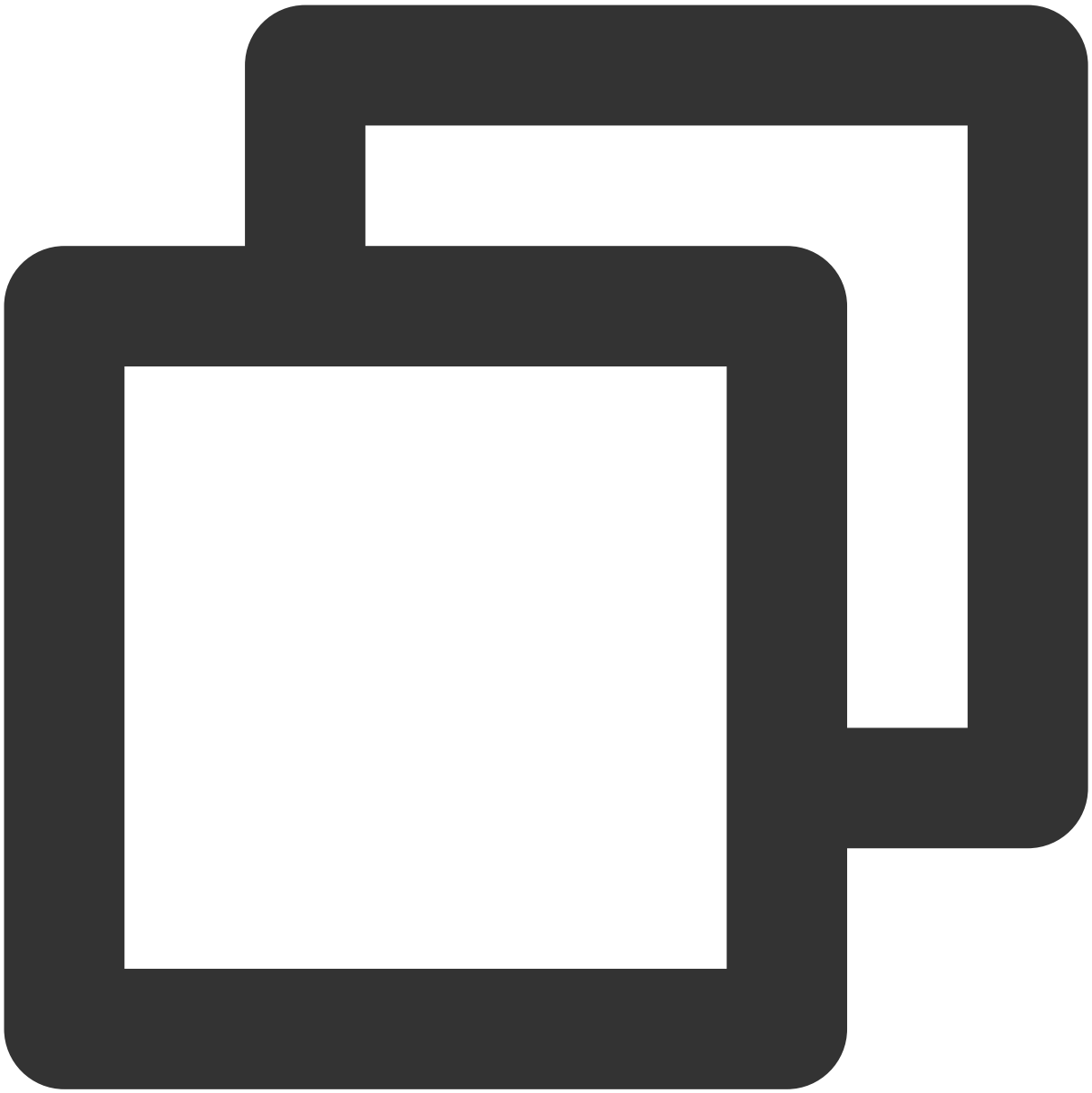
```
create table tag_uin_list(
tagid int primary key,
uin_offset int,
uinbits roaringbitmap
);
```

3. Insert 100,000 tags and corresponding user data according to the previously created tag table.



```
insert into tag_uin_list
select tagid, uin_offset, rb_build_agg(uin::int) as uinbits from
(
select
unnest(tag) as tagid,
(uin / (2^31)::int8) as uin_offset,
mod(uin, (2^31)::int8) as uin
from account1
) t
group by tagid, uin_offset;
```

4. Query the number of users with tags 1, 3, 10, and 200.



```
explain analyze select sum(ub) from
(
select uin_offset,rb_or_cardinality_agg(uinbits) as ub
from tag_uin_list
where tagid in (1,3,10,200)
group by uin_offset
) t;
QUERY PLAN
-----------------------------------------------------------------------------------
```

```
------------
Aggregate (cost=32.47..32.48 rows=1 width=32) (actual time=0.964..0.966 rows=1 loop
-> GroupAggregate (cost=32.42..32.46 rows=1 width=12) (actual time=0.955..0.956 row
Group Key: tag_uin_list.uin_offset
-> Sort (cost=32.42..32.43 rows=4 width=22) (actual time=0.107..0.109 rows=4 loops=
Sort Key: tag_uin_list.uin_offset
Sort Method: quicksort Memory: 25kB
-> Bitmap Heap Scan on tag_uin_list (cost=17.20..32.38 rows=4 width=22) (actual tim
)
Recheck Cond: (tagid = ANY ('{1,3,10,200}'::integer[]))
Heap Blocks: exact=4
-> Bitmap Index Scan on tag_uin_list_pkey (cost=0.00..17.20 rows=4 width=0) (actual
=4 loops=1)
Index Cond: (tagid = ANY ('{1,3,10,200}'::integer[]))
Planning Time: 0.289 ms
Execution Time: 1.083 ms
(13 rows)
```

5. View the list of users with tags 1, 3, 10, and 200.

```
explain analyze select uin_offset,rb_or_agg(uinbits) as ub
from tag_uin_list
where tagid in (1,3,10,200)
group by uin_offset;
QUERY PLAN
--------------------------------------------------------------------------------
------
GroupAggregate (cost=32.42..32.46 rows=1 width=36) (actual time=0.246..0.246 rows=1
Group Key: uin_offset
-> Sort (cost=32.42..32.43 rows=4 width=22) (actual time=0.043..0.045 rows=4 loops=
Sort Key: uin_offset
```

```
Sort Method: quicksort Memory: 25kB
-> Bitmap Heap Scan on tag_uin_list (cost=17.20..32.38 rows=4 width=22) (actual tim
Recheck Cond: (tagid = ANY ('{1,3,10,200}'::integer[]))
Heap Blocks: exact=4
-> Bitmap Index Scan on tag_uin_list_pkey (cost=0.00..17.20 rows=4 width=0) (actual
Index Cond: (tagid = ANY ('{1,3,10,200}'::integer[]))
Planning Time: 0.119 ms
Execution Time: 0.310 ms
(12 rows)
```

6. Viewing index size and table size

```
test=> select relname, pg_size_pretty(pg_relation_size(relid)) from pg_stat_user_ta
  relname     | pg_size_pretty
--------------+----------------
account       | 1545 MB
account1      | 1077 MB
t_user        | 651 MB
tag_dict      | 6672 kB
tag_uin_list | 5888 kB
(5 rows)
test=> select indexrelname, pg_size_pretty(pg_relation_size(relid)) from pg_stat_us
  indexrelname     | pg_size_pretty
-------------------+----------------
tag_inx            | 1545 MB
account_pkey       | 1545 MB
tag_inx_2          | 1077 MB
account1_pkey      | 1077 MB
t_user_pkey        | 651 MB
tag_dict_pkey      | 6672 kB
tag_uin_list_pkey | 5888 kB
(7 rows)
```

Conclusion

The query performance comparison of different schemes is as follows:

Query Item	Scheme 1	Scheme 2	Roaring Bitmap Scheme
Query the list of users with the specified tag	4.528 ms	0.151 ms	0.310 ms
Query the number of users with the same tag	8.27 ms	29.725 ms	1.083 ms
Storage capacity statistics	4,635 MB	3,244.344 MB	1,237.12 MB

As can be clearly seen from the above three schemes, the optimization effect is very obvious. The roaring bitmap scheme works very well in terms of query speed and storage capacity usage.

Querying People Nearby with One SQL Statement

Last updated：2024-01-24 11:20:59

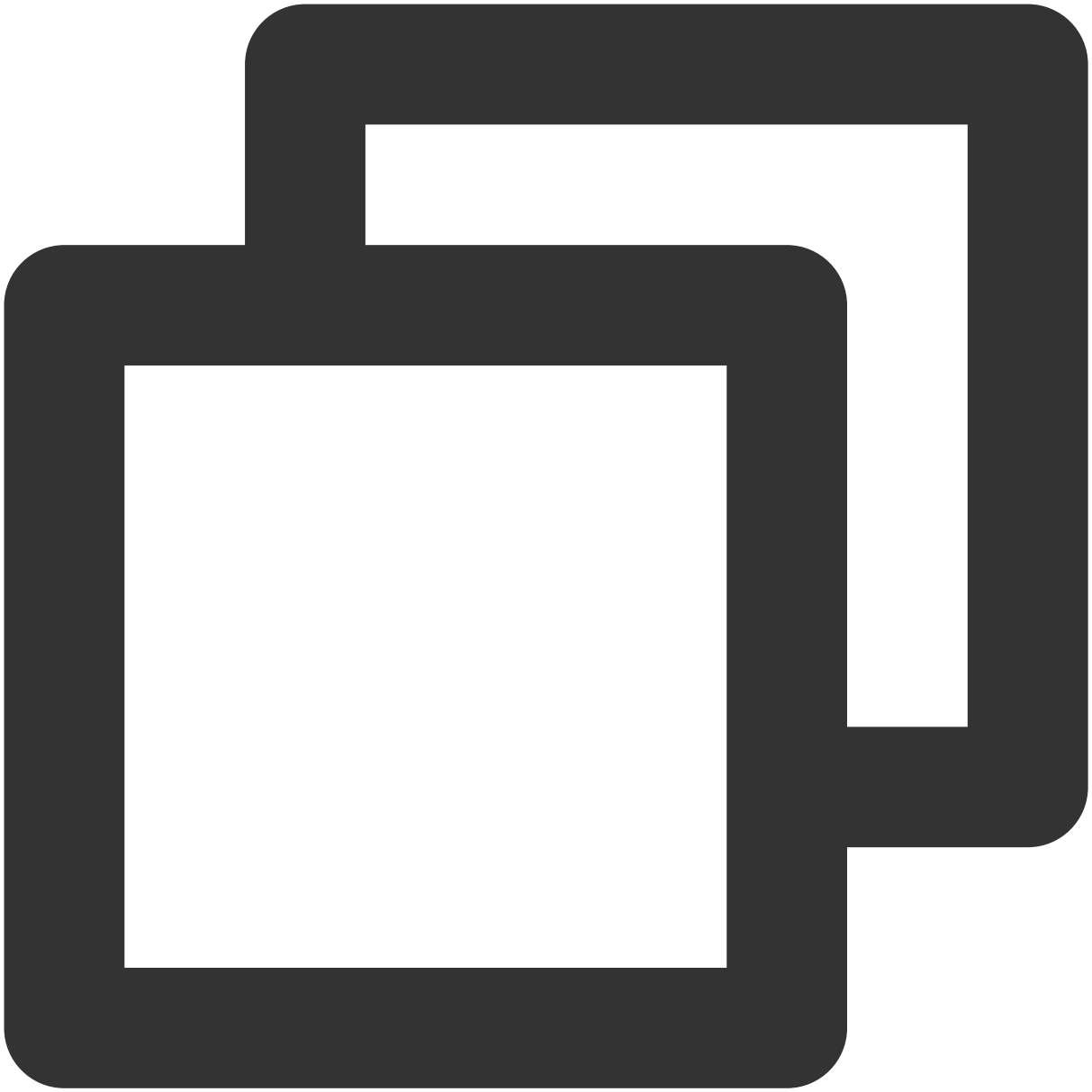
PostGIS is an extension of the PostgreSQL relational database. It follows the specifications of OpenGIS and provides the following spatial information service features: spatial objects, indexes, operation functions, and operators. PostGIS supports all spatial data types, including POINT, LINESTRING, POLYGON, MULTIPOINT, MULTILINESTRING, MULTIPOLYGON, and GEOMETRYCOLLECTION.
PostGIS is also the most comprehensive and powerful spatial and geographic database engine in the industry. Many business use cases nowadays require the "XXX nearby" feature. PostGIS and PostgreSQL can work together to implement this feature quickly.
This document describes how to implement the "objects nearby" feature with PostGIS.

Prerequisites

You have a PostgreSQL instance.
This instance supports the PostGIS extension.

Step 1. Create the extension

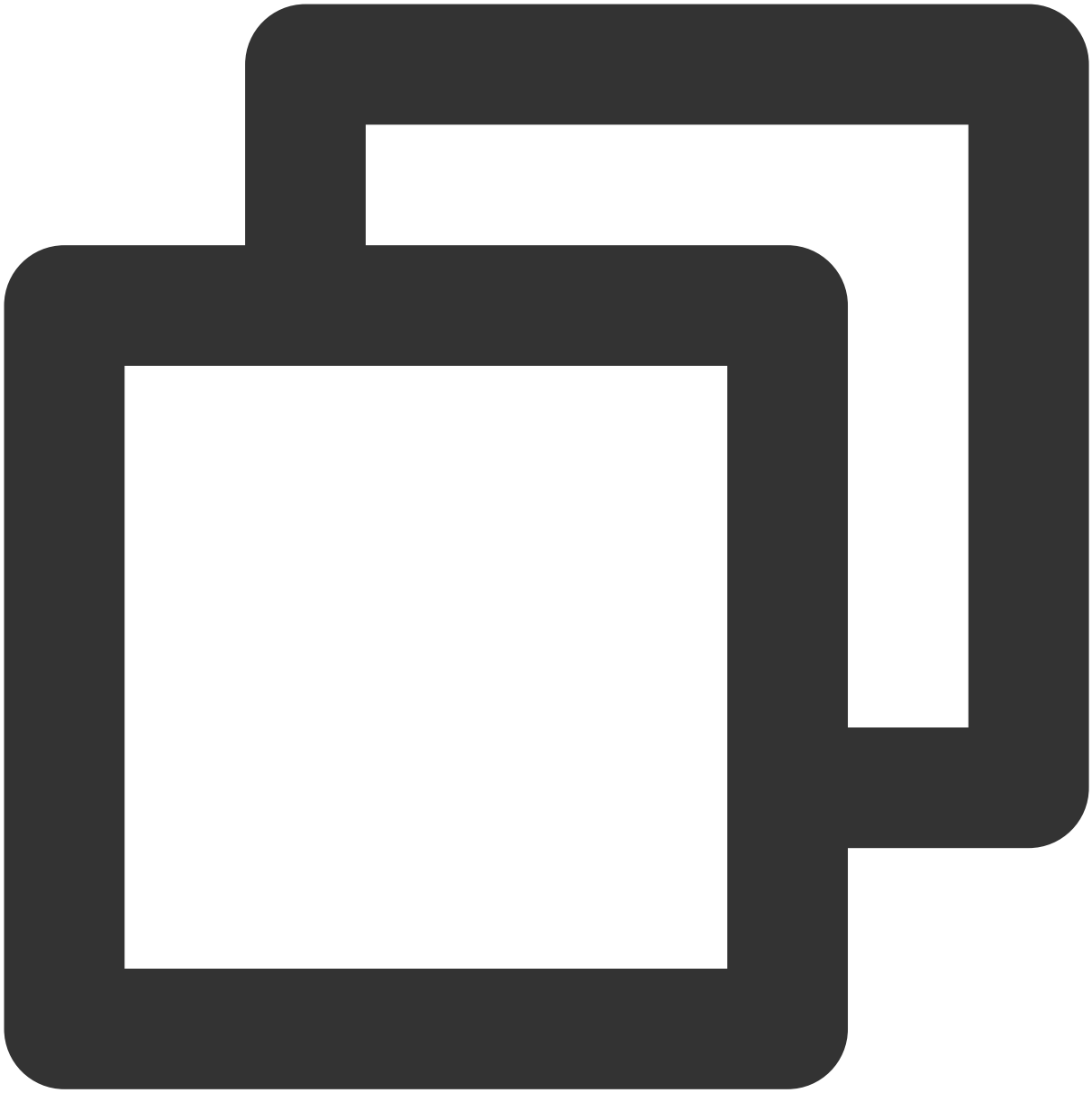
Log in to the business database instance and run the following commands. For the login methods, see Connecting to TencentDB for PostgreSQL Instance.



```
\\c test
CREATE EXTENSION postgis;
CREATE EXTENSION postgis_topology;
```

Step 2. Create a test table and an index

Run the following commands in the business database. You can customize the name after `TABLE` .



```
CREATE TABLE t_user(uid int PRIMARY KEY,name varchar(20),location geometry);
CREATE INDEX t_user_location on t_user USING GIST(location);
```

Step 3. Insert test data

```
## Create an automatic name generation function
create or replace function random_string(length integer) returns text as
$$
declare
chars text[] := '{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F,G,H,I,J,K,L,M,N,O,P,Q,R,S,T,U,V,W
result text := '';
i integer := 0;
length2 integer := (select trunc(random() * length + 1));
begin
if length2 < 0 then
raise exception 'Given length cannot be less than 0';
```

```
end if;
for i in 1..length2 loop
result := result || chars[1+random()*(array_length(chars, 1)-1)];
end loop;
return result;
end;
$$ language plpgsql;

## Insert ten million rows of test data
insert into t_user select generate_series(1,10000000), random_string(20),st_setsrid
```

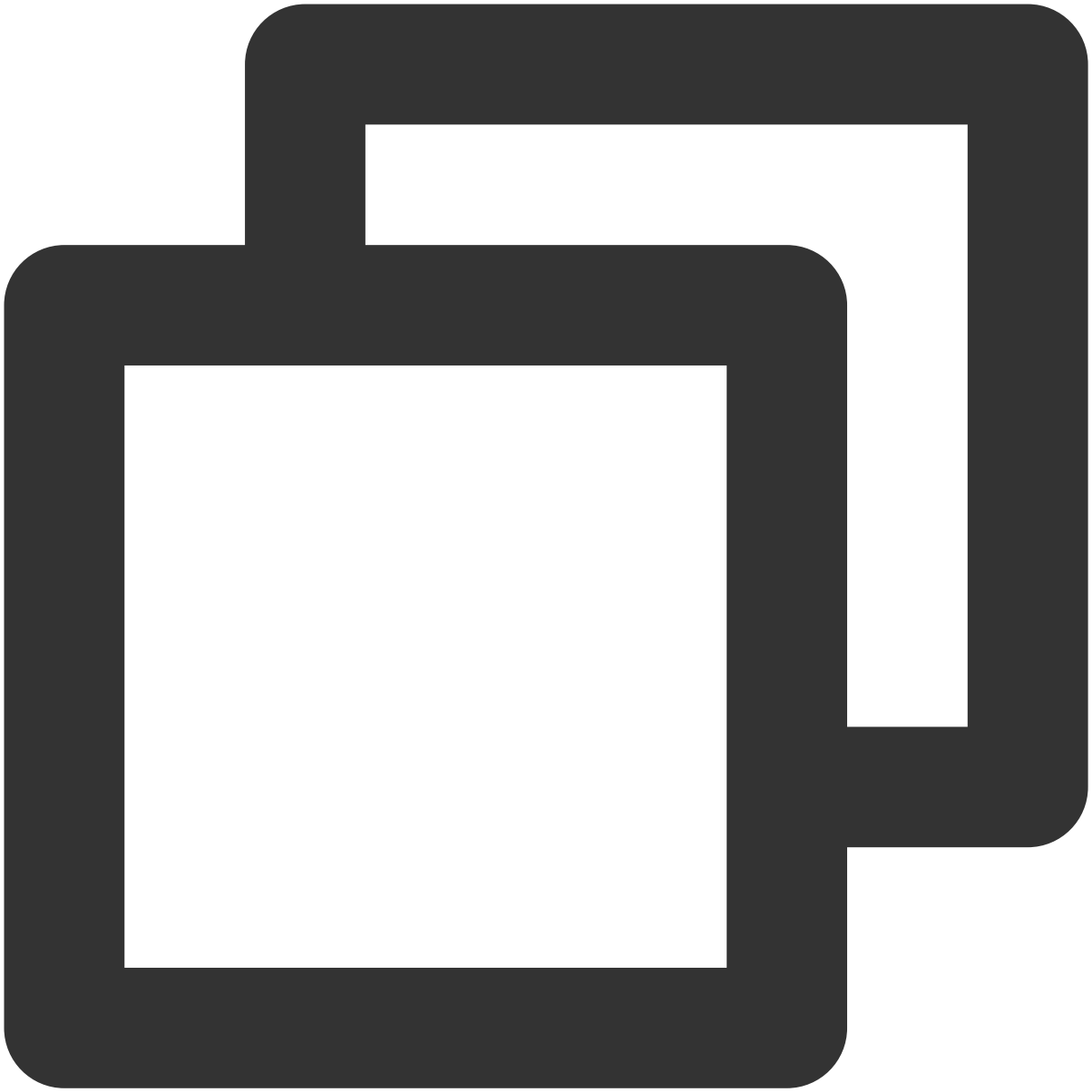
Step 4. Query people nearby

1. Select a random coordinate here. The coordinate of Tiananmen Square (116.404177,39.909652) is used as an example.
2. Use it as the coordinate for query to find the five objects closest to it in the database, and then output the distances of these objects from it (in 00' km).

Note:

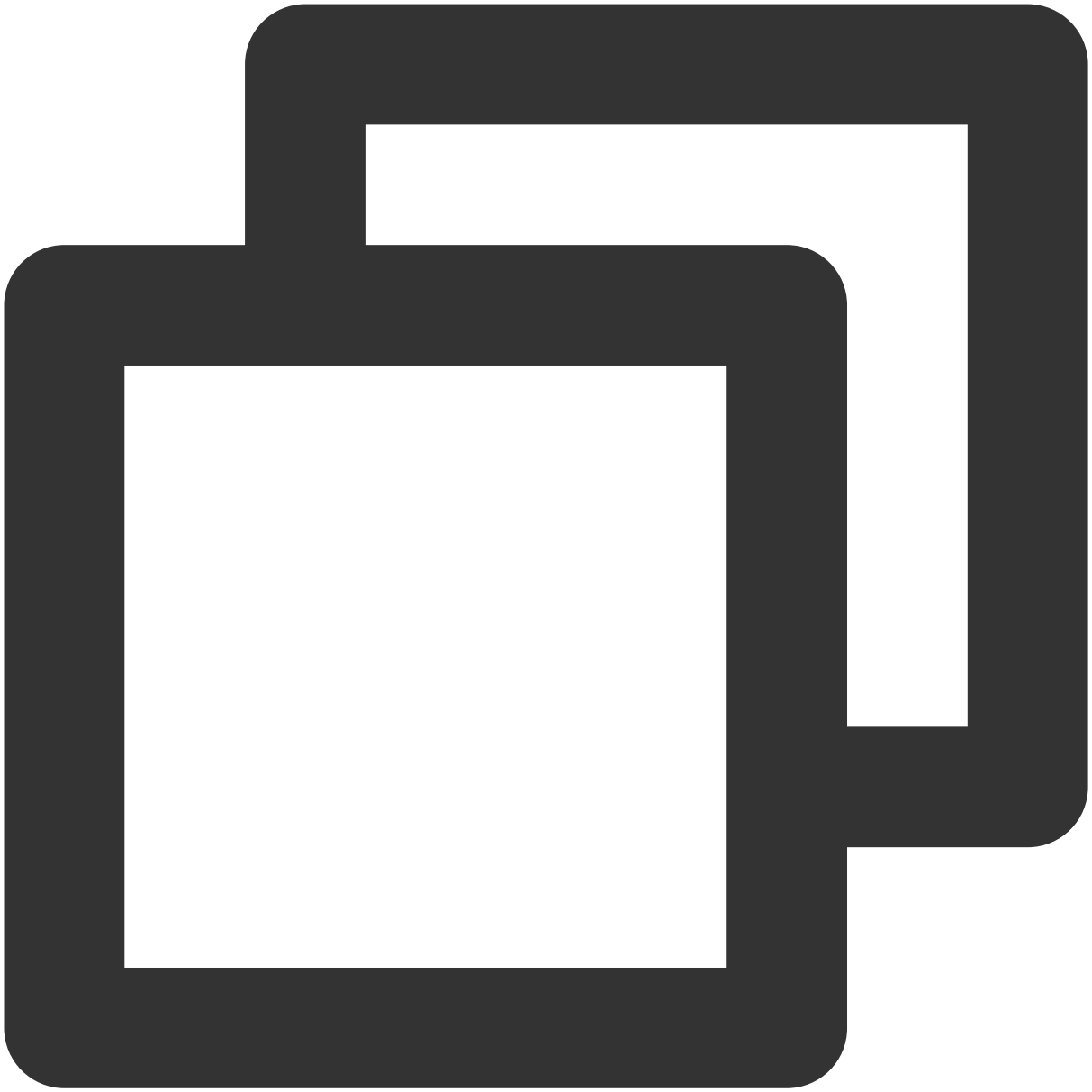
WGS 84 is the most popular geographic coordinate system. Internationally, each coordinate system is assigned an EPSG code, which is 4326 for WGS 84. GPS is based on WGS 84, so the coordinate data obtained is usually in WGS 84 and generally stored as WGS 84.

Run the following command:



```
select uid, name, ST_AsText(location), ST_Distance(ST_GeomFromText('POINT(116.40417
```

3. View all objects within 1000 meters of this coordinate object and their distances.



```
select uid, name, ST_AsText(location),ST_Distance(ST_GeomFromText('POINT(116.404177
```

Configuring TencentDB for PostgreSQL as GitLab's External Data Source

Last updated：2024-01-24 11:20:59

Background

GitLab is a GitHub-like self-hosting Git repository management system service for you to implement internal management of Git repositories. It helps you keep your code confidential and easily modify the deployed code, with the following strengths:
It provides GitLab Community Edition for you to locate code on servers.
It provides an unlimited number of private and public repositories free of charge.
It allows you to share a small amount of code in your project as needed instead of the entire project.
The latest GitLab version (12.1) currently supports only PostgreSQL rather than MySQL as the metadatabase for the following reasons:
MySQL doesn't support `WITH` until version 8.
To increase MySQL's limits on columns, the operations will be extremely complicated, which may cause MySQL to reject storing data.
MySQL doesn't allow you to restrict the length of fields of TEXT type.
MySQL doesn't support partition indexes.
In contrast, PostgreSQL supports all the above scenarios. Therefore, GitLab integrates PostgreSQL in its installation package. However, integrated database services may have certain security risks for some enterprises, and their database reliability and availability cannot be guaranteed. To ensure the reliability of the code hosting service, some businesses and enterprises choose to use stable external database services. However, GitLab supports Patroni-based high-availability databases only on GitLab HA Repmgr edition, and it incurs high costs to maintain the cluster on your own. In this case, you can use TencentDB for PostgreSQL to greatly simplify such maintenance operations.
This document describes how to replace the embedded database service in GitLab with TencentDB for PostgreSQL.

Step 1. Install GitLab

1. Prepare resources
CentOS Linux release 7.6.1810 (Core).
gitlab-ce 14.9.3.
One CVM instance with over 4 GB memory and over 50 GB disk. We recommend you use `/opt` to mount an independent data disk.

One TencentDB for PostgreSQL instance. Configure its specification based on your actual conditions. You can use an instance with a low specification initially and scale it up later as needed. Select the instance version based on the GitLab version.

2. Download GitLab

Click here and find the target GitLab installation package, download it, and upload it to the target server.

3. Install GitLab

Use the `root` account to run the following statements to install GitLab. If a message indicating that the dependency packages are not installed is displayed in the last step, you can directly use yum or other installation tools to install them.

```
curl https://packages.gitlab.com/install/repositories/gitlab/gitlab ce/script.rpm.s
sh gitlab-ee_install.sh
export EXTERNAL_URL=https://gitlab.example.com
yum install -y curl policycoreutils-python openssh-server cronie
rpm -ivh gitlab-ce-13.10.2-ce.0.el7.x86_64.rpm
```

Step 2. Initialize the PostgreSQL data source

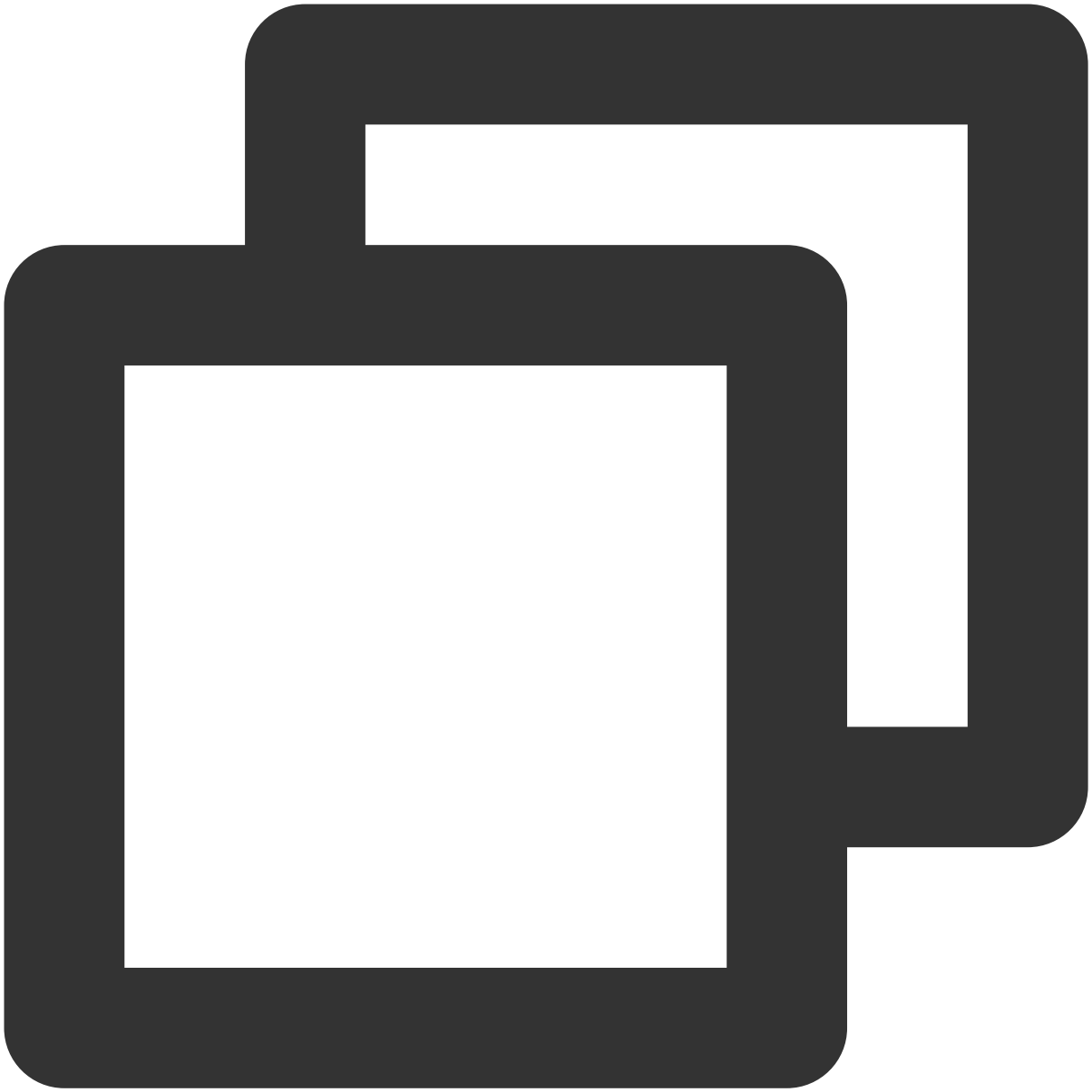
1. You can directly use a cloud database service, such as TencentDB for PostgreSQL. To create a TencentDB for PostgreSQL instance, see Creating TencentDB for PostgreSQL Instance.

Note:

The database must be created or installed with the same version as GitLab; otherwise, a version mismatch error will be reported during GitLab initialization, causing database creation to fail.

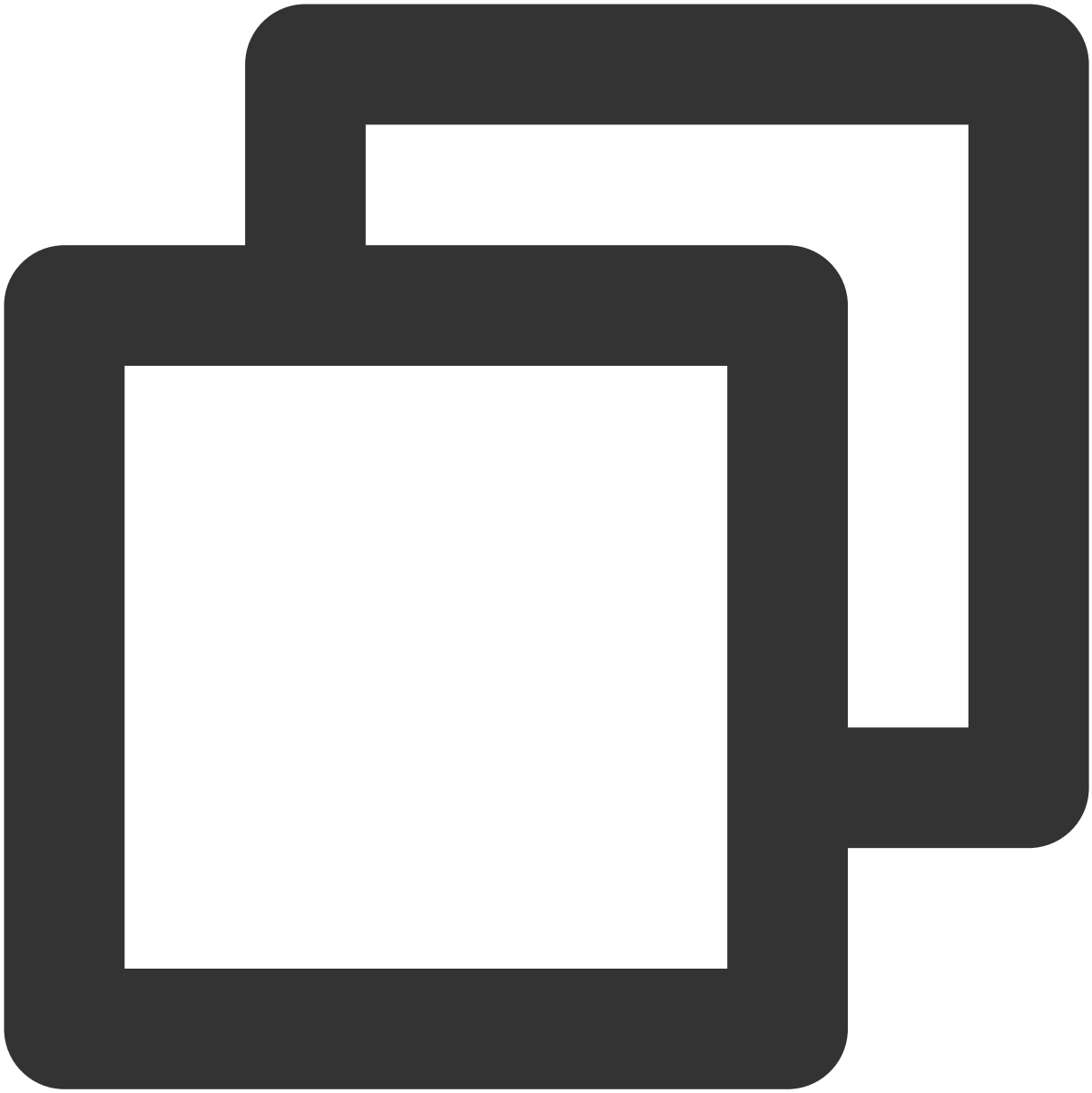
GitLab Version	Earliest Supported PostgreSQL Version
13.0	11
14.0	12

2. Use the client to log in to TencentDB for PostgreSQL. You can use psql to check whether the database can be directly accessed, and if not, check the network connection and security group configuration.



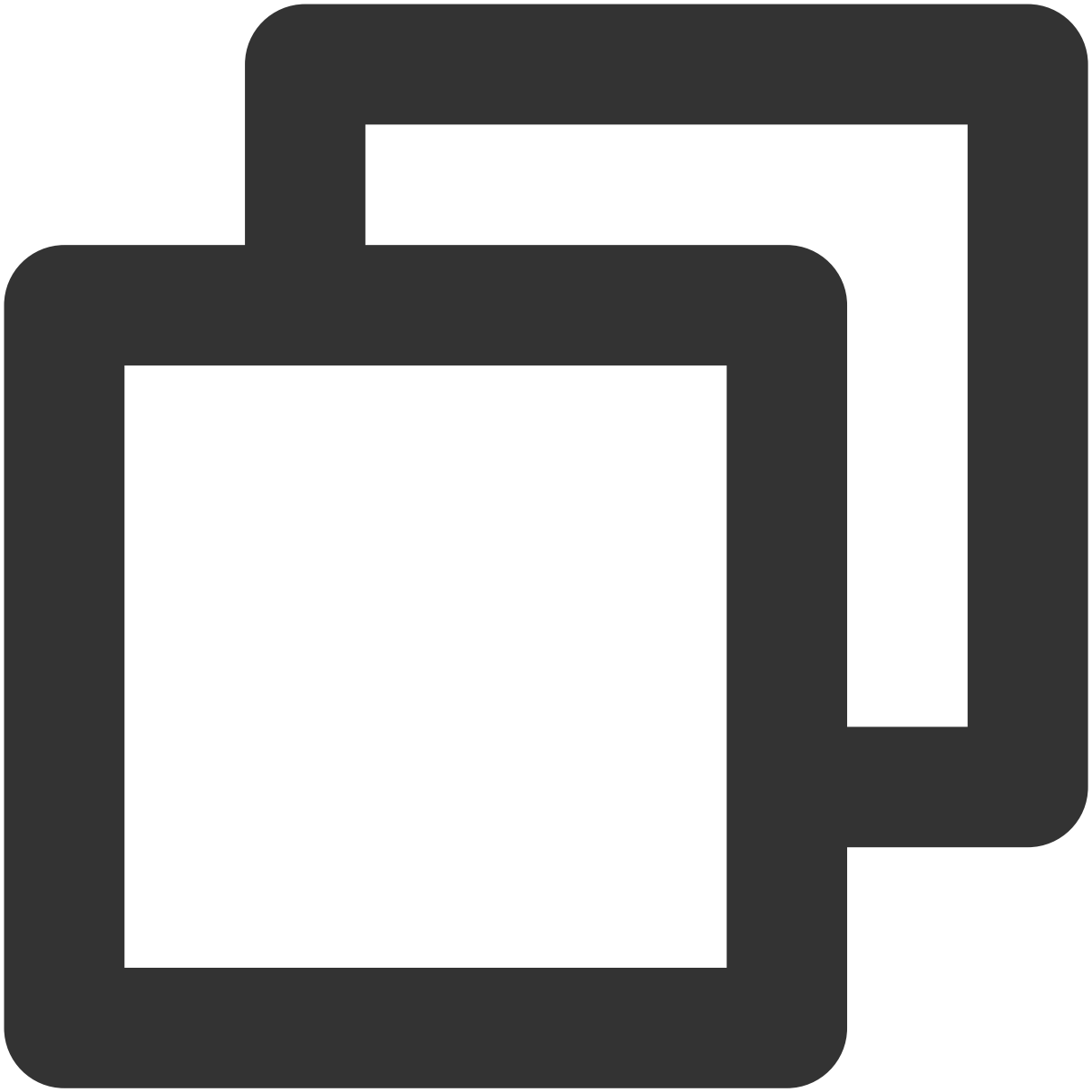
```
psql -U <database admin> -p <port> -d postgres -h <access address>
```

3. First, create an account to be used by GitLab in the database, such as `gitlab`. Note that the account must have the `superuser` privileges or admin privileges granted by TencentDB such as `pg_tencentdb_superuser`.



```
create user gitlab login password 'gitlab_****_password#123';
grant gitlab to <current admin account>; grant pg_tencentdb_superuser to gitlab;
```

4. Create a database to be managed and used by `gitlab` .



```
create database gitlab owner=gitlab ENCODING = 'UTF8';
```

Note:

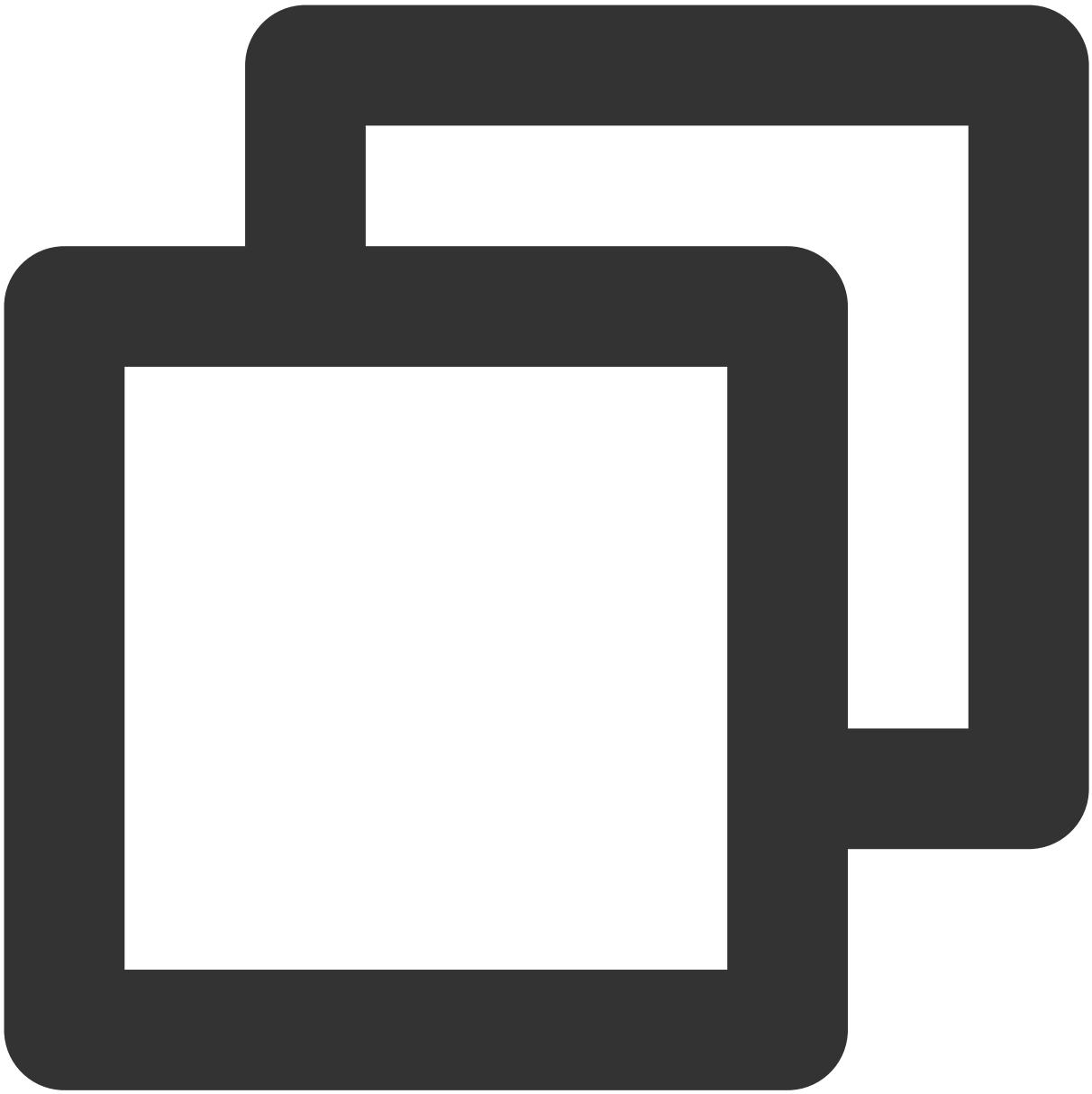
The GitLab database must support the `pg_trgm`, `btree_gist`, and `plpgsql` extensions, which don't need to be created in advance. They will be automatically created during GitLab initialization, but you should ensure that they can be created successfully.

Step 3. Modify the GitLab metadatabase to TencentDB for PostgreSQL

1. Log in to the server where GitLab is installed, find the GitLab configuration file, which is `/etc/gitlab/gitlab.rb` by default. The file has no configuration information by default. You can run the following command to view it:

```
# cat /etc/gitlab/gitlab.rb |grep -v ^# | grep -v ^$
external_url 'http://gitlab.example.com'
```

2. Add the following information at the end of the file to add the TencentDB for PostgreSQL data source to GitLab:



```
## postgresql connect
## Set this parameter to `false`, indicating to disable the embedded PostgreSQL dat
postgresql['enable'] = false
gitlab_rails['db_adapter'] = "postgresql"
gitlab_rails['db_encoding'] = "utf8"
## Database name
gitlab_rails['db_database'] = "gitlab"
gitlab_rails['db_pool'] = 100 ## Database user
gitlab_rails['db_username'] = "gitlab"
```

```
## Password, which can be changed as needed
gitlab_rails['db_password'] = "gitlab_Test_password#123" ## Access address
gitlab_rails['db_host'] = "gz-tdcpg-ep-6kvx6p19.sql.tencentcdb.com" ## Access port
gitlab_rails['db_port'] = "25870"
```

Note that if the access address is set to a domain name, the following message will be displayed during initialization: `ActiveRecord::ConnectionNotEstablished: could not translate host name "gz-tdcpg ep-6kvx6p19.sql.tencentcdb.com " to address: Name or service not known`

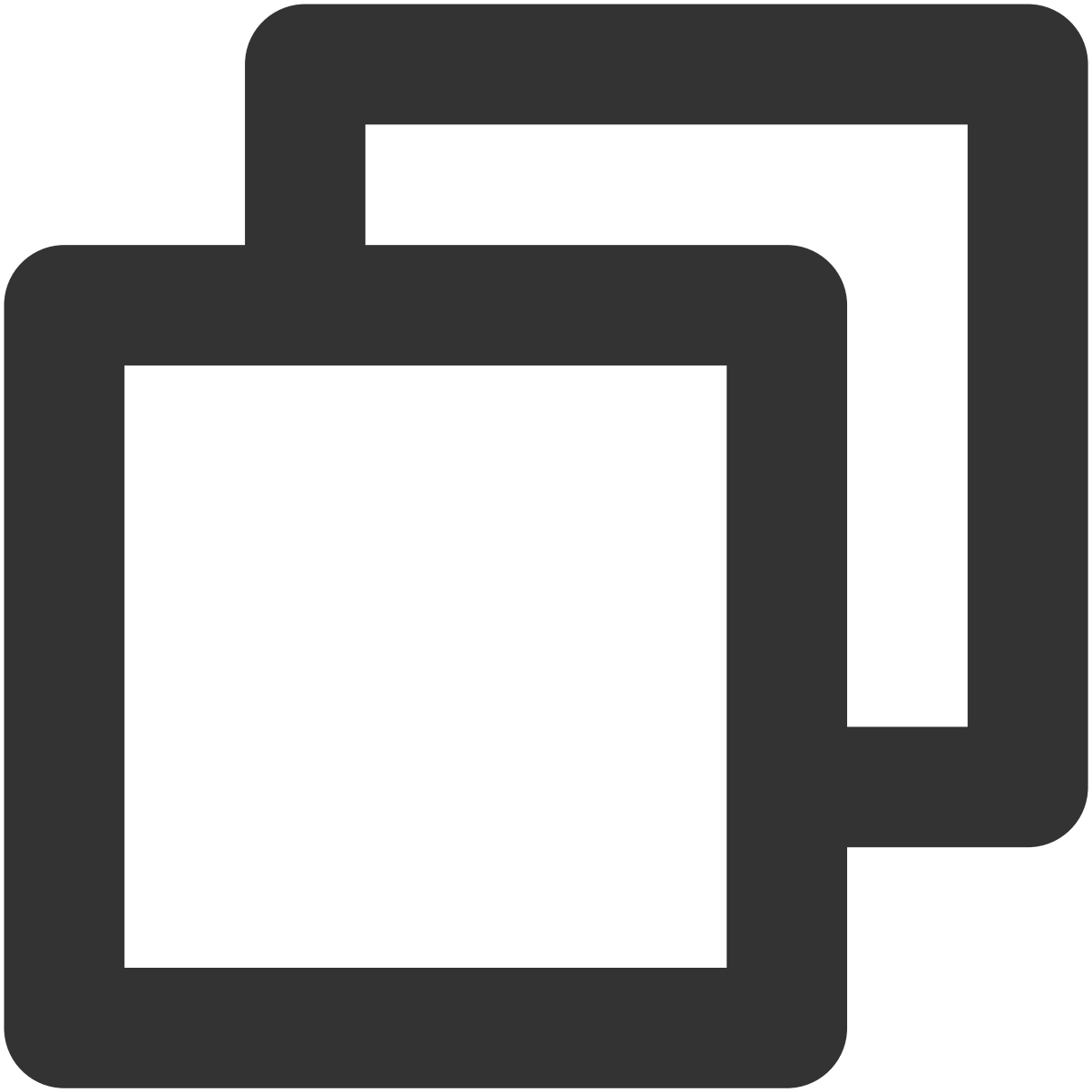
If the database access address is a domain name, run the `ping` command to find the IP address of the domain name or a DNS server that can resolve it. We recommend you not directly modify an access domain name to an IP address, as in scenarios where a domain name is used, the database backend is usually configured with load balancing or high availability. In this case, you can directly configure the DNS server or host on the server. If the database service changes, you can directly modify the DNS service or host to avoid modifying the GitLab service.

Step 4. Initialize, log in to, and use GitLab

1. Run the following command to use GitLab, which may take a while, so wait patiently. When `gitlab Reconfigured!` is displayed, GitLab has been initialized.

```
gitlab-ctl reconfigure
```

2. Run the following command to start GitLab:

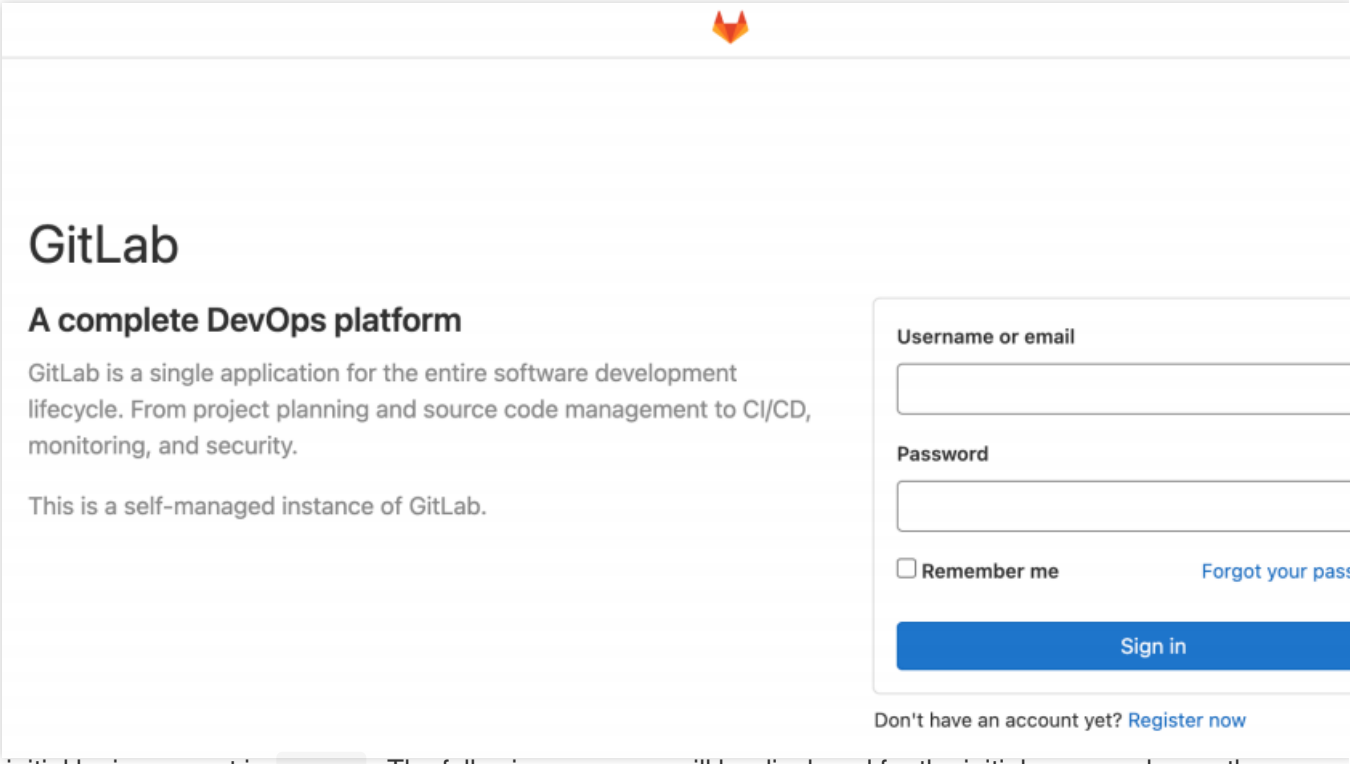


```
gitlab-ctl startok
```

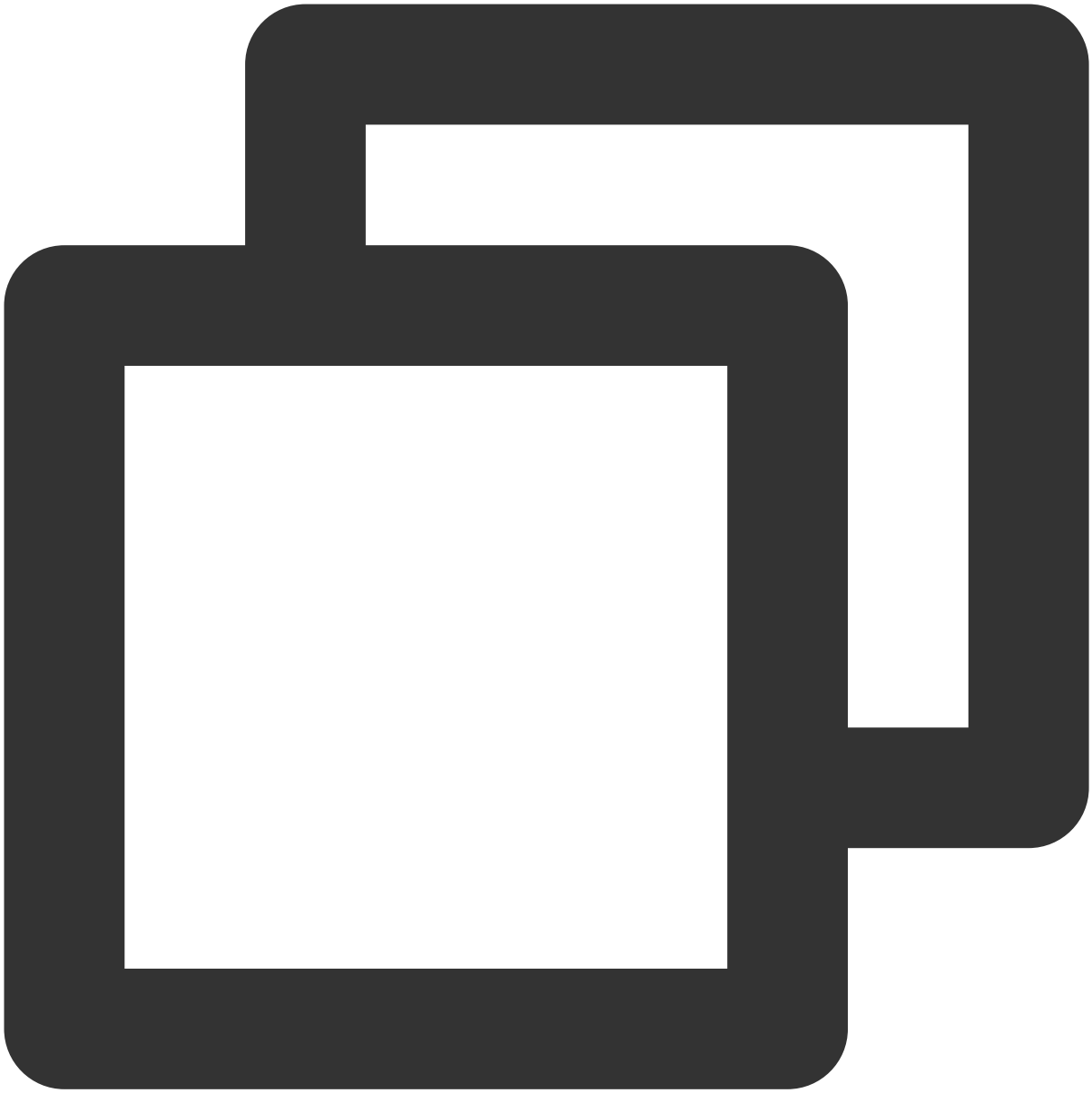
3. You can access GitLab at the following URL. If access fails, it may be caused by the server firewall.

Sample address: `http://{accessible server IP address}/users/sign_in`

The login page is as shown below:



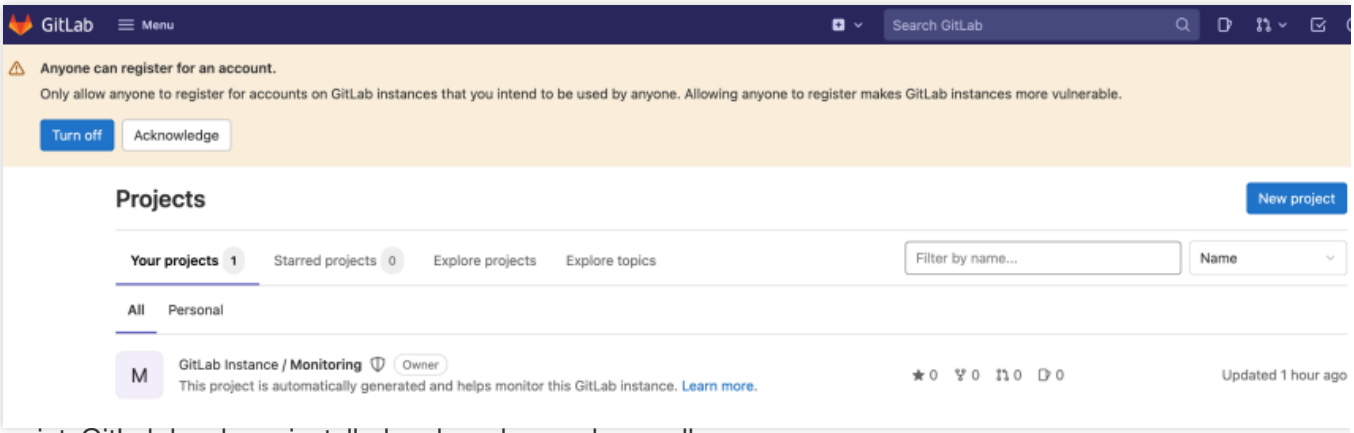
4. The initial login account is `root` . The following message will be displayed for the initial password upon the completion of initialization:



```
Password stored to /etc/gitlab/initial_root_password. This file will be
cleaned up in first reconfigure run after 24 hours.
```

Note:

You can find the initial password in this file on the server. After login, remember to change the password.



At this point, GitLab has been installed and can be used normally.

Supporting Tiered Storage Based on cos_fdw Extension

Last updated：2024-01-24 11:20:59

Background

As the core component for data storage and processing, a database will have an increasing data volume as the business develops. Some historical or archived data may exist over time or because of the business design logic. Such data is seldom accessed by the business but cannot be deleted, as it may be used in some scenarios. To improve the database's processing performance, you need to store such data in a cold storage class.
For databases, it is very important to store as much data as possible and provide better unified data processing APIs. For such user requirements, TencentDB for PostgreSQL offers a tiered data storage scheme. Its core principle is to offer storage media at different costs for your choice. For example, you can store cold data in a storage class with a lower performance but at lower costs and store hot data in high-performance SSDs at higher costs. This scheme guarantees smooth operations of your business and reduces the storage costs, making it extremely cost-effective.

Overview

COS is an object storage service provided by Tencent Cloud. Currently, tiered storage is mainly implemented by connecting to and parsing COS data through the cos_fdw extension.
You can use the cos_fdw extension to load COS data into TencentDB for PostgreSQL tables and access COS data just like a regular table, thereby implementing hot/cold data separation. You don't need to care about how different storage media are accessed. You only need to configure COS data files to TencentDB for PostgreSQL.

Solution Strengths

Unified engine: It provides multiple types of storage media with no need to modify the code at the business layer. You can implement unified access directly over the PostgreSQL protocol.
Lower costs: Compared with high-performance SSDs, its overall costs are 86.25% lower.
Ease of use: You only need to export the source data to a CSV file in COS and create a foreign table in TencentDB for PostgreSQL based on the extension. Then, you can use the foreign table just like the original table.
Unlimited storage: COS offers an unlimited storage capacity. You can dynamically store data as needed without worrying about the capacity.

Support for joined table queries: You can query joined tables in multiple storage media and join tables across partitions. As such operations require a unified data fusion node, they cannot be directly performed in other databases.

Supported Versions

Currently, tiered storage is supported for the following TencentDB for PostgreSQL versions:
PostgreSQL 10
PostgreSQL 11
PostgreSQL 12
PostgreSQL 13
PostgreSQL 14

Using cos_fdw

Use cos_fdw in the following steps:
1. Export the data.
2. Upload the data to COS.
3. Create the cos_fdw extension.
4. Create a foreign server.
5. Create a foreign table.
6. Query the foreign table.

Initializing Environment

First, you need to apply for a relay server, such as a CVM instance, with a low specification in the same region and AZ as the database and COS bucket.
Recommended OS: CentOS 7.
1. Install the PostgreSQL client as instructed in Linux downloads (Red Hat family).

```
sudo yum install -y
https://download.postgresql.org/pub/repos/yum/reporpms/EL-7-
x86_64/pgdg-redhat-repo-latest.noarch.rpm
sudo yum install -y postgresql13
```

2. After the installation is completed, run the `psql` command to access the database and check whether the client is installed successfully:

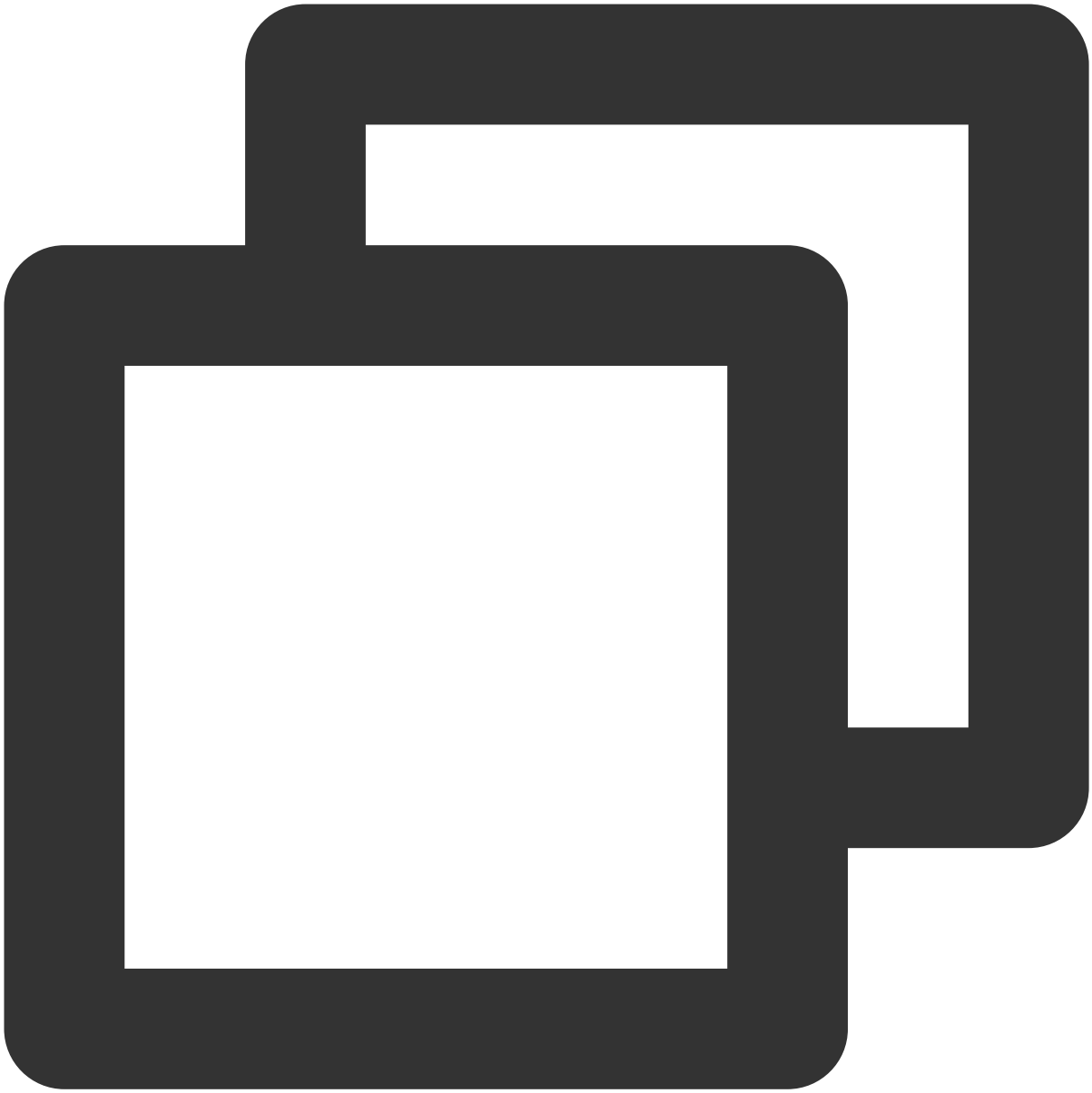
```
psql -Uroot -p 5432 -h 10.x.x.8 -d postgres
Password for user root:
psql (13.6, server 13.3)
Type "help" for help.

postgres=>
```

3. After the PostgreSQL client is installed, mount COS. You can use COSFS to mount COS to the server, which eliminates the need to use a larger CVM instance for data dumping and upload. For more information, see COSFS.
4. Run the following command to install dependency packages for your current environment:

```
sudo yum install libxml2-devel libcurl-devel -y
```

5. Download the COSFS installation package from GitHub.

6. After the download is completed, upload the package to the server and run the following command to install COSFS:

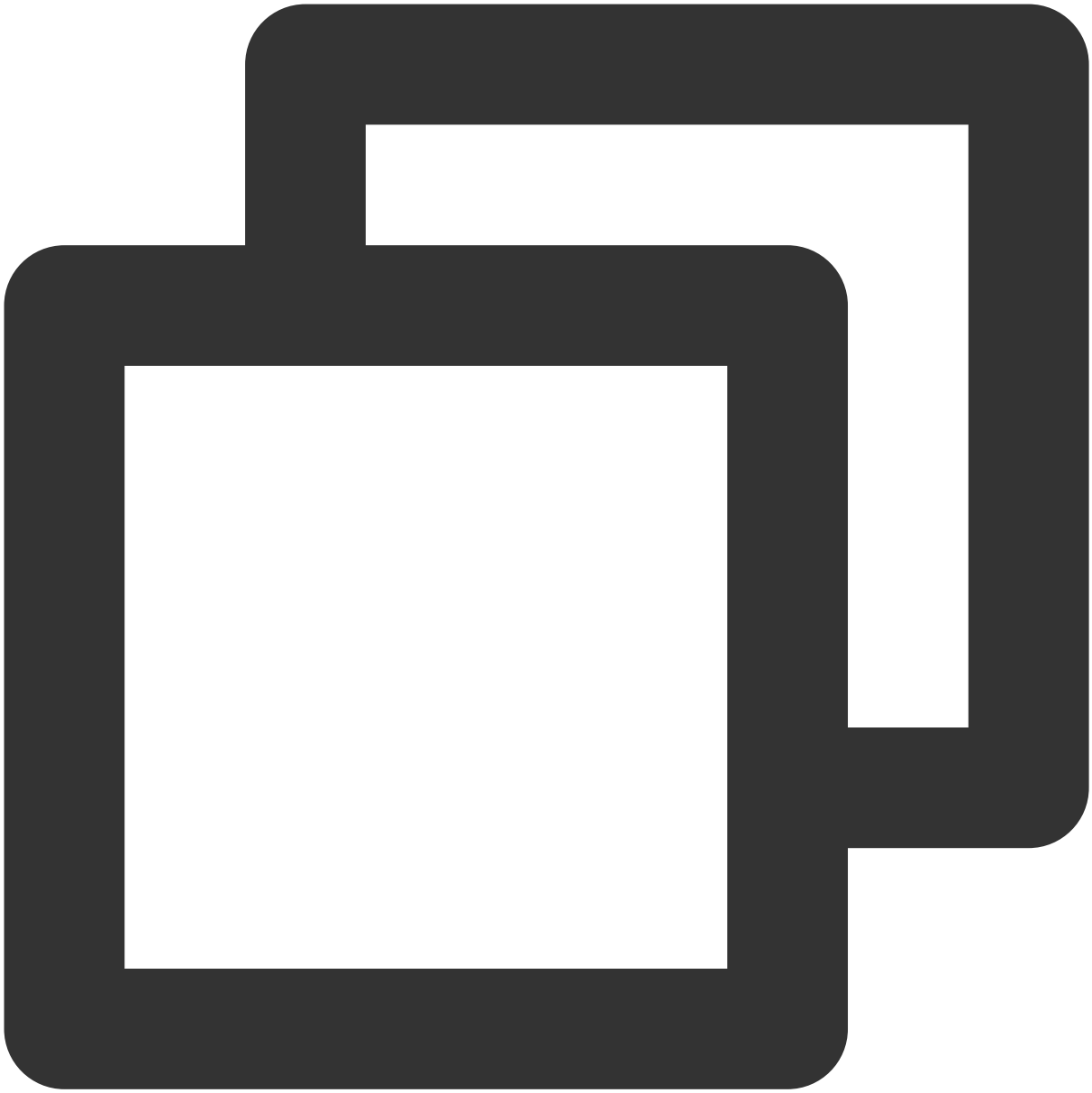


```
rpm -ivh cosfs-1.0.19-centos7.0.x86_64.rpm
```

Note:

If dependency packages are installed, but COSFS still cannot be installed successfully, add the `--force` parameter to the command to forcibly install it.

7. After installing COSFS, run the following command to mount the COS bucket to the relay server.

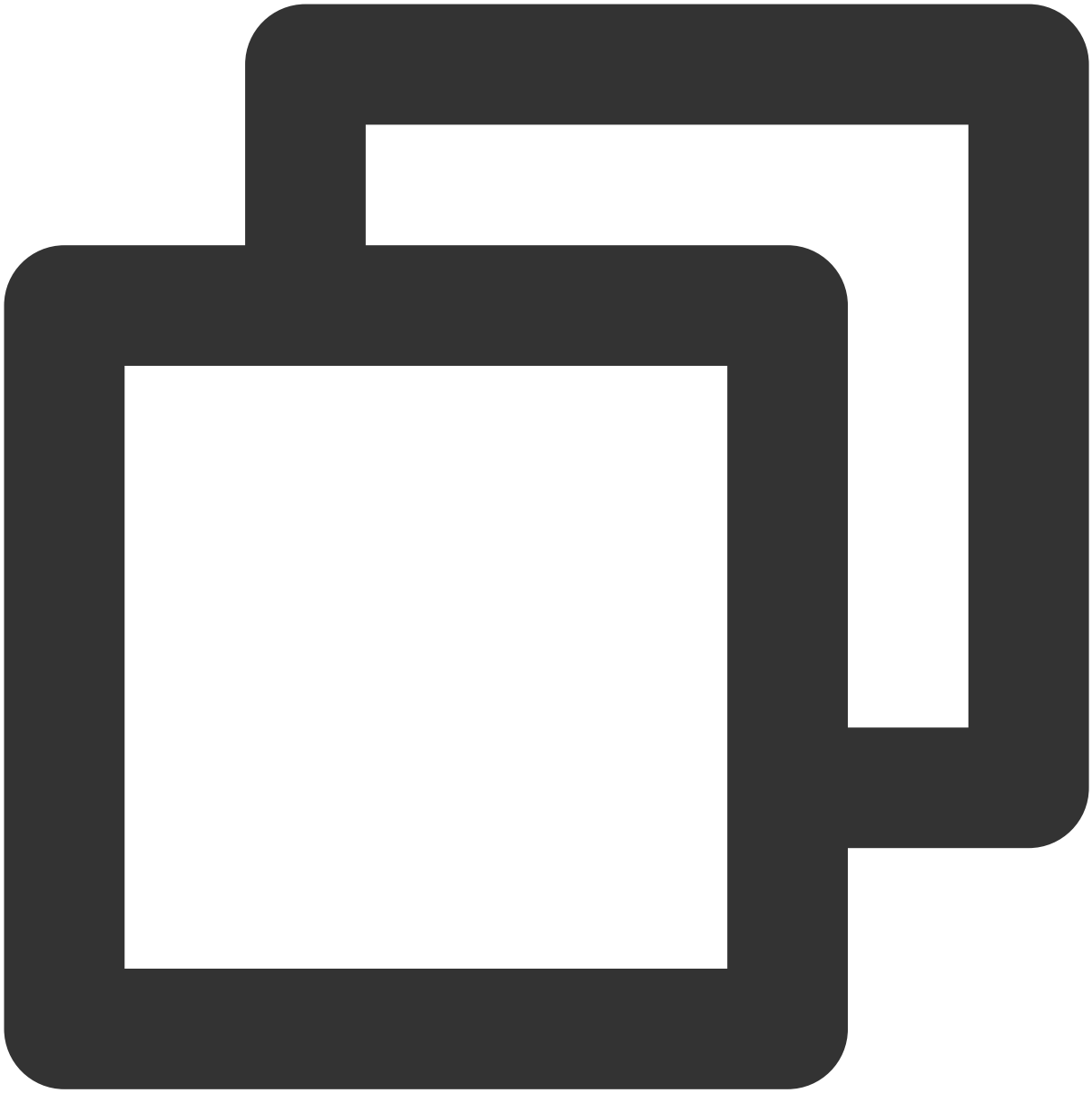


```
echo <BucketName-APPID>:<SecretId>:<SecretKey> > /etc/passwd-cosfs
chmod 640 /etc/passwd-cosfs
cosfs <BucketName-APPID> <MountPoint> -ourl=http://cos.<Region>.myqcloud.c
om -odbglevel=info -oallow_other
```

`BucketName-APPID` is the format of the bucket name.

`SecretId` and `SecretKey` are the key information.

8. After mounting, go to the mounted directory and copy a file to check whether the mounting succeeds. You can also run `df -h` to view the mounting status.



```
[root@VM-4-17-centos ~]# df -h
Filesystem Size Used Avail Use% Mounted on
devtmpfs 1.9G 0 1.9G 0% /dev
tmpfs 1.9G 0 1.9G 0% /dev/shm
tmpfs 1.9G 472K 1.9G 1% /run
tmpfs 1.9G 0 1.9G 0% /sys/fs/cgroup
/dev/vda1 50G 3.0G 44G 7% /
tmpfs 379M 0 379M 0% /run/user/0
cosfs 256T 0 256T 0% /mnt/pgstorage
```

Exporting Data

After mounting is completed, export the data.

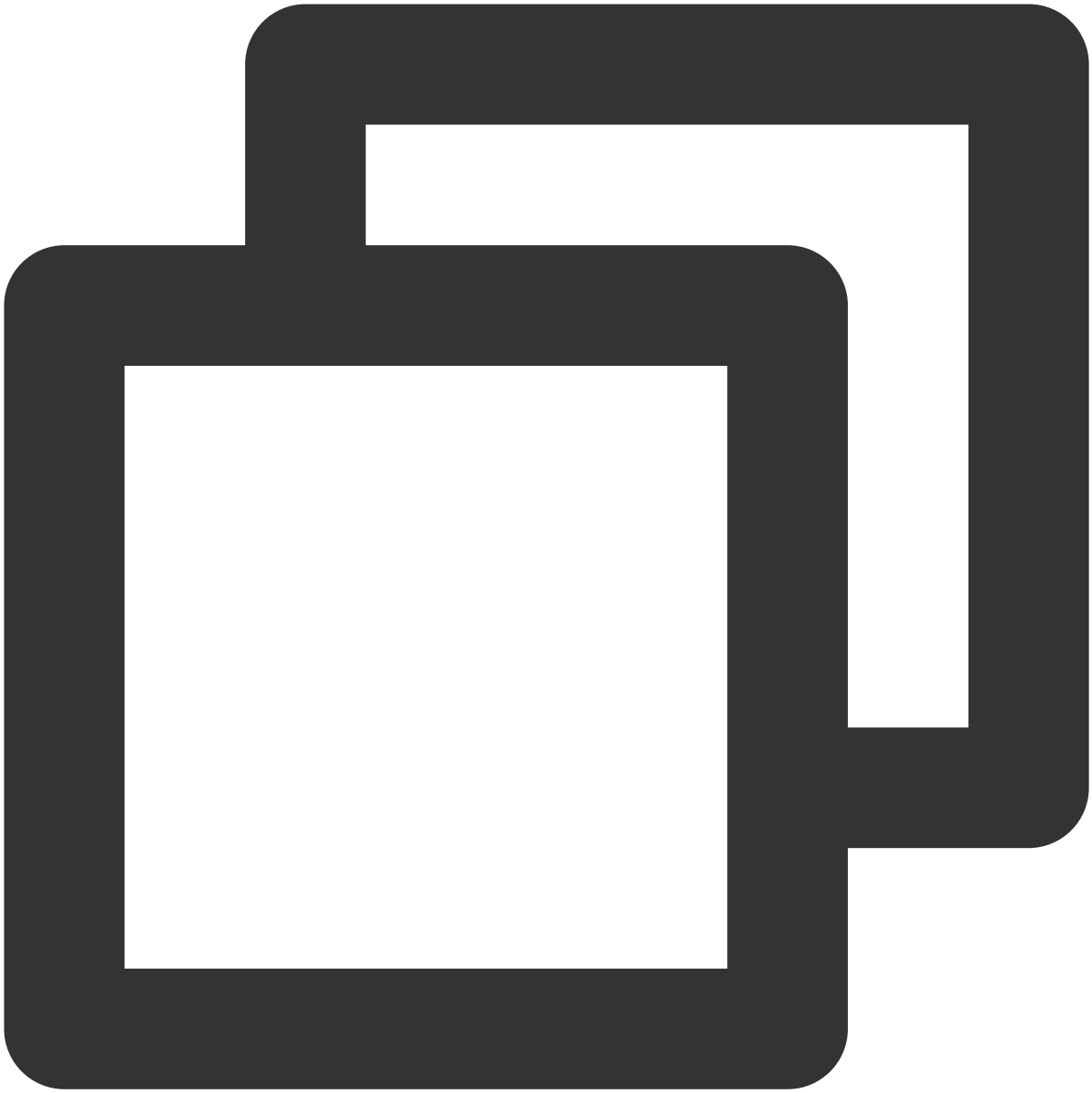
If the `sensor_log` table exists, it needs to be in the following structure:

```
CREATE TABLE sensor_log (
 sensor_log_id SERIAL PRIMARY KEY,
 location VARCHAR NOT NULL,
 reading BIGINT NOT NULL,
 reading_date TIMESTAMP NOT NULL
```

```
);
CREATE INDEX idx_sensor_log_location ON sensor_log (location);
CREATE INDEX idx_sensor_log_date ON sensor_log (reading_date);
insert into sensor_log(location,reading,reading_date) values('38c-
1401',293857,current_timestamp);
insert into sensor_log(location,reading,reading_date) values('38c-
1402',293858,current_timestamp);
insert into sensor_log(location,reading,reading_date) values('34c-
1401',293859,current_timestamp);
insert into sensor_log(location,reading,reading_date) values('18c-
1401',2938510,current_timestamp);
```

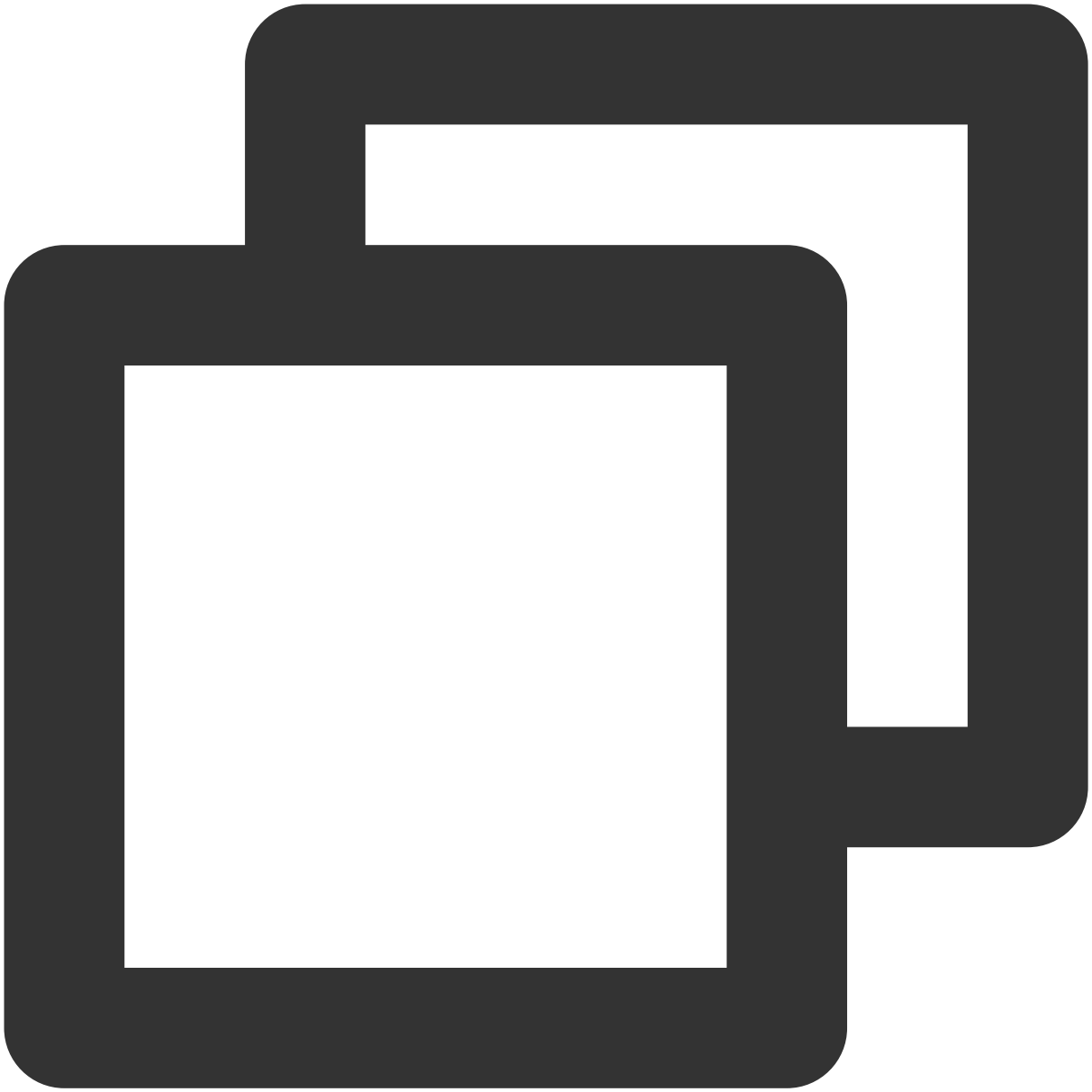
If you use psql to export the data, follow the steps below (do not carry the header during export):

Export the entire table:



```
psql -U root -p 5432 -h 10.0.4.8 -d hehe -c \\COPY sensor_log
(sensor_log_id,location, reading,reading_date) TO '/mnt/xxx/sensor_log.csv' WITH
csv;
```

Export specified data (for scenarios such as data filtering, multi-table join, and view):



```
psql -U root -p 5432 -h 10.0.4.8 -d hehe -c '\\COPY (select * from sensor_log
where location='18c-1401') TO '/mnt/pgstorage/sensor_log.csv' WITH csv;'
```

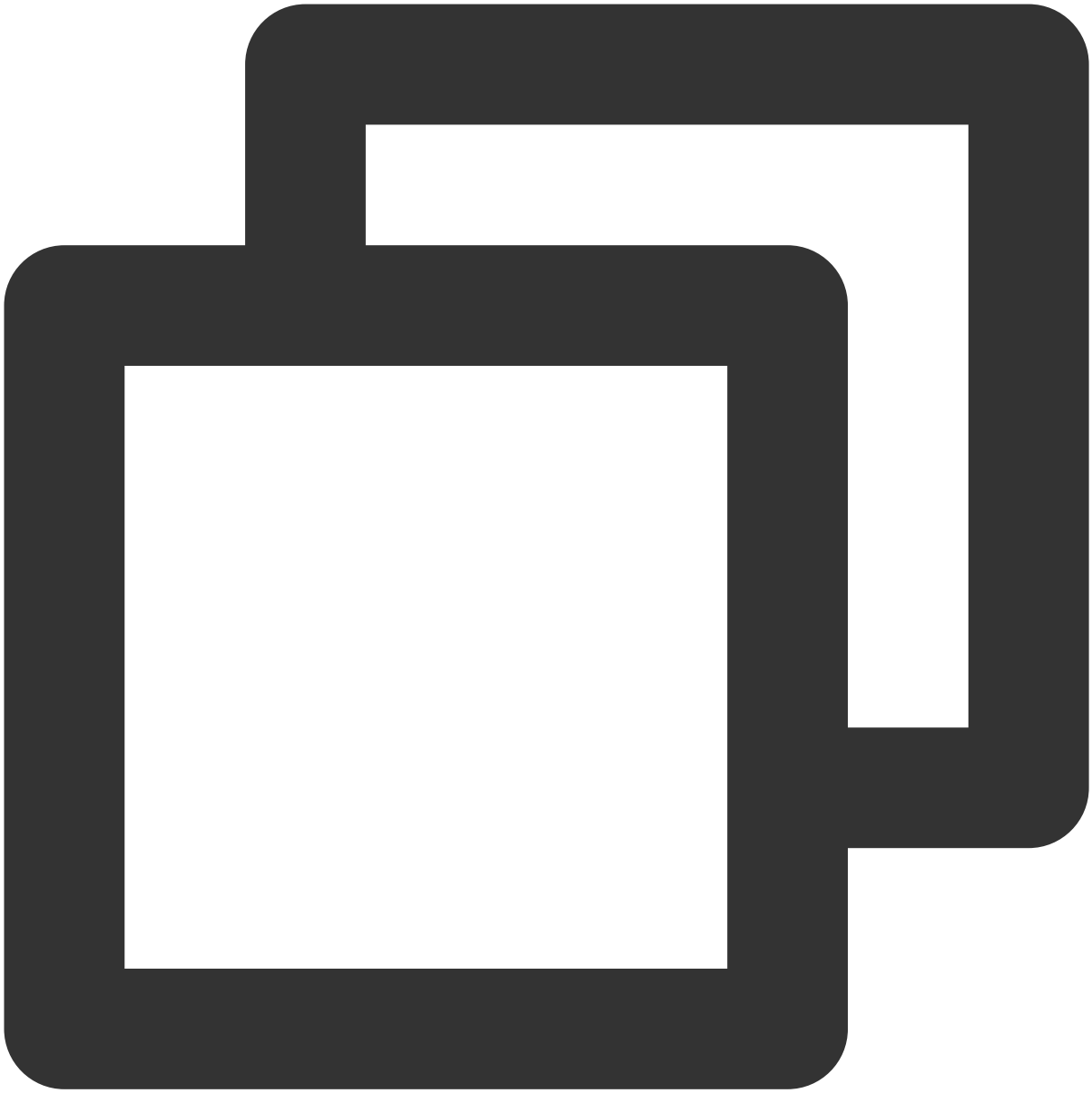
After the above statement is executed, you can find the exported file in the corresponding directory in the COS bucket. The CSV file exported to COS doesn't need to contain column names.

Creating Extension

The cos_fdw extension will encrypt the secret ID and secret key of COS. The encryption algorithm relies on the pgcrypto extension. Therefore, you need to install pgcrypto first.

```
CREATE EXTENSION pgcrypto;
CREATE EXTENSION cos_fdw;
```

Creating Foreign Server



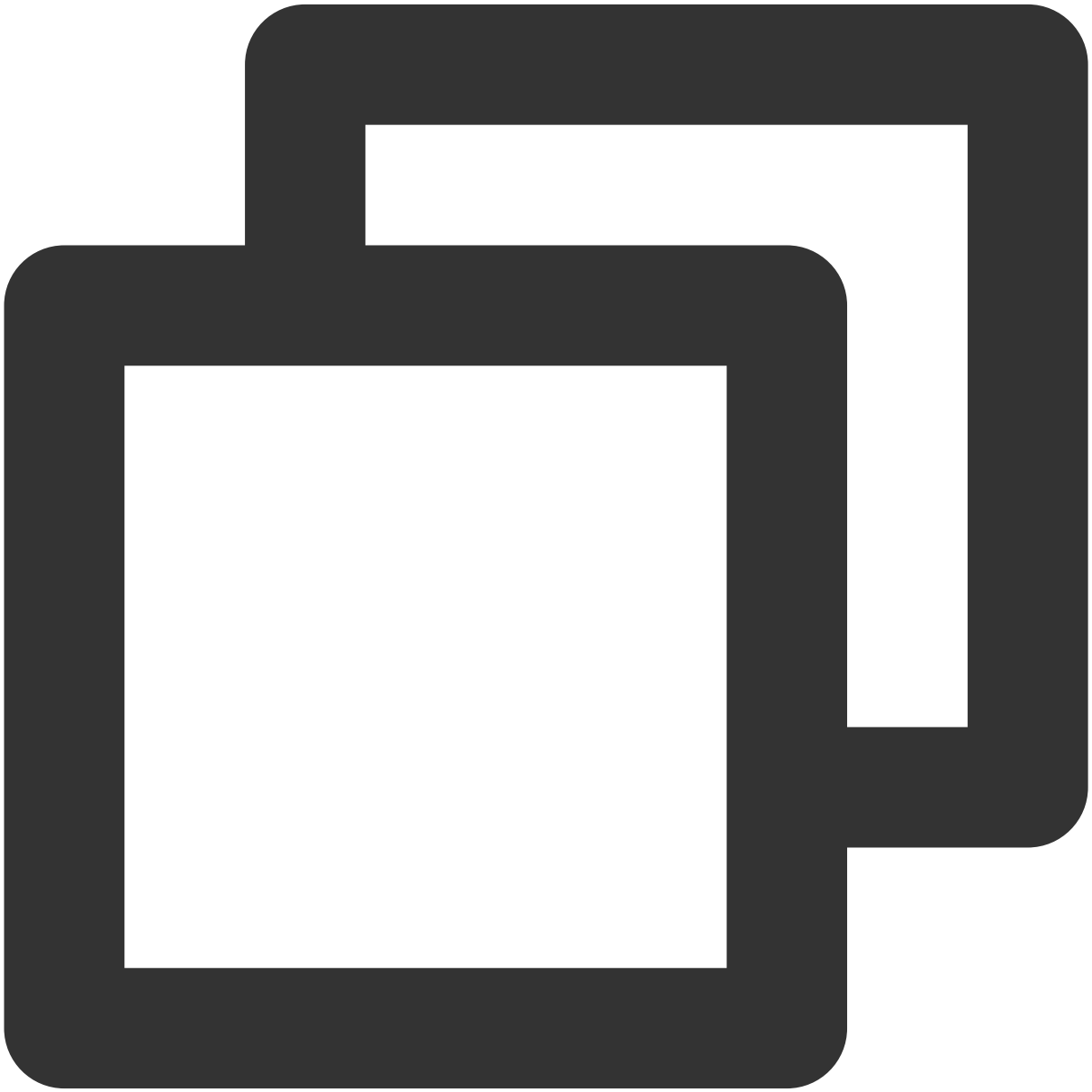
```
CREATE SERVER cos_server FOREIGN DATA WRAPPER cos_fdw OPTIONS(
 host 'xxxxxx.cos.ap-nanjing.myqcloud.com',
 bucket 'xxxxxxxx',
 id 'xxxxxxxx',
 key 'xxxxxxxxxx'
);
```

Note:

The domain name configured in `host` is the access address of the COS bucket. The address doesn't need to contain the `http` or `https` prefix as the protocol.

`id` and `key` of the foreign server are sensitive information, which will be encrypted and stored by cos_fdw. Different instances use different keys to maximize the user information protection. You can run `SELECT * FROM pg_foreign_server;` to view the information.

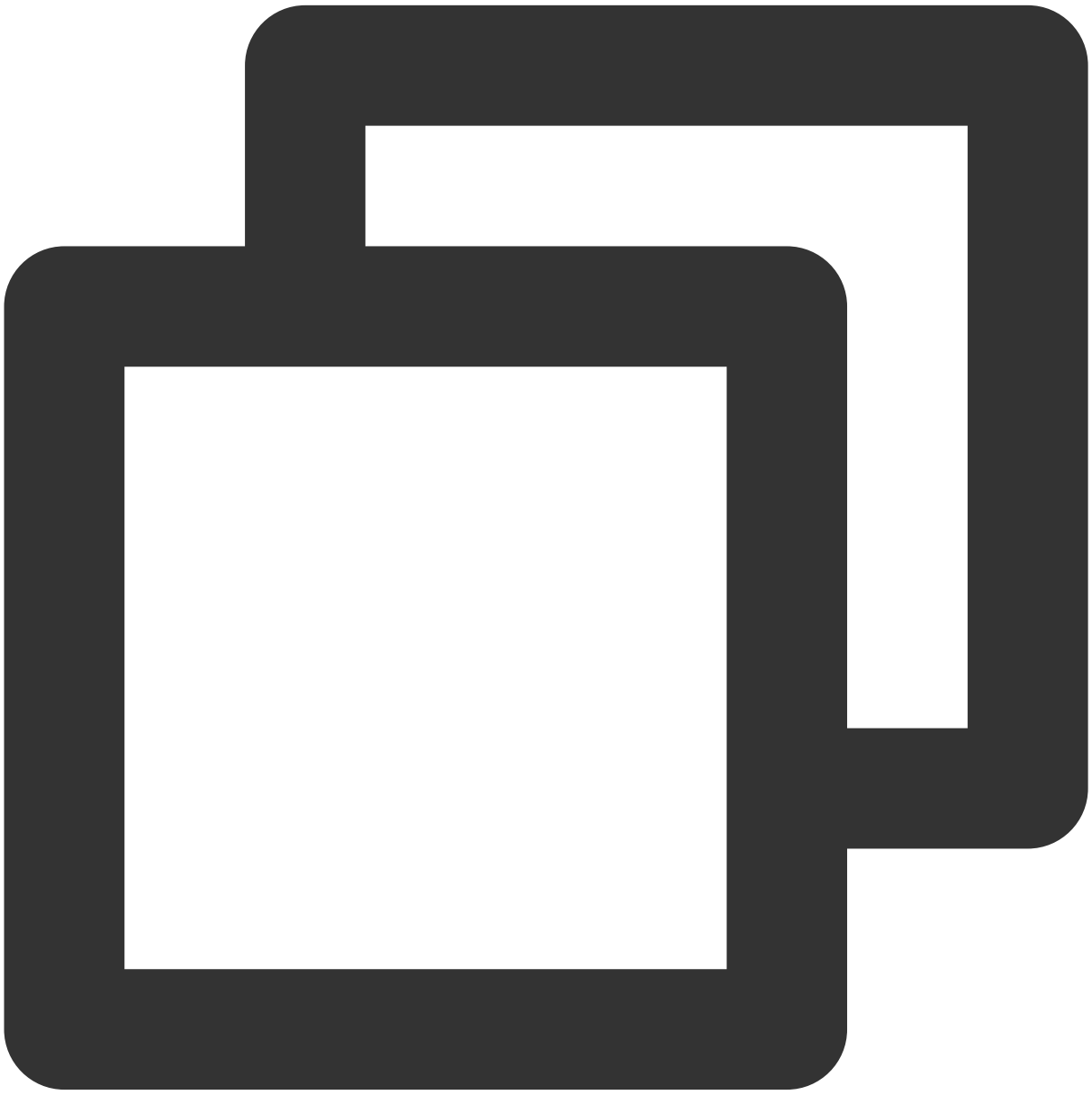
Creating COS Foreign Table



```
CREATE FOREIGN TABLE test_csv (
 word1 text OPTIONS (force_not_null 'true'),
```

```
 word2 text OPTIONS (force_not_null 'off') ) SERVER cos_server OPTIONS (
 filepath '/test.csv',
 format 'csv',
 null 'NULL'
);
```

`cos_fdw` allows you to map multiple COS files to the same foreign table. To do so, enter multiple filenames in the `filepath` parameter and separate them with commas (do not add spaces).



```
CREATE FOREIGN TABLE multi_csv (
 word1 text OPTIONS (force_not_null 'true'),
```

```
  word2 text OPTIONS (force_not_null 'off') ) SERVER cos_server OPTIONS (
  filepath '/a.csv,/b.csv,/c.csv.2',
  format 'csv',
  null 'NULL'
 );
```

Querying Foreign Table

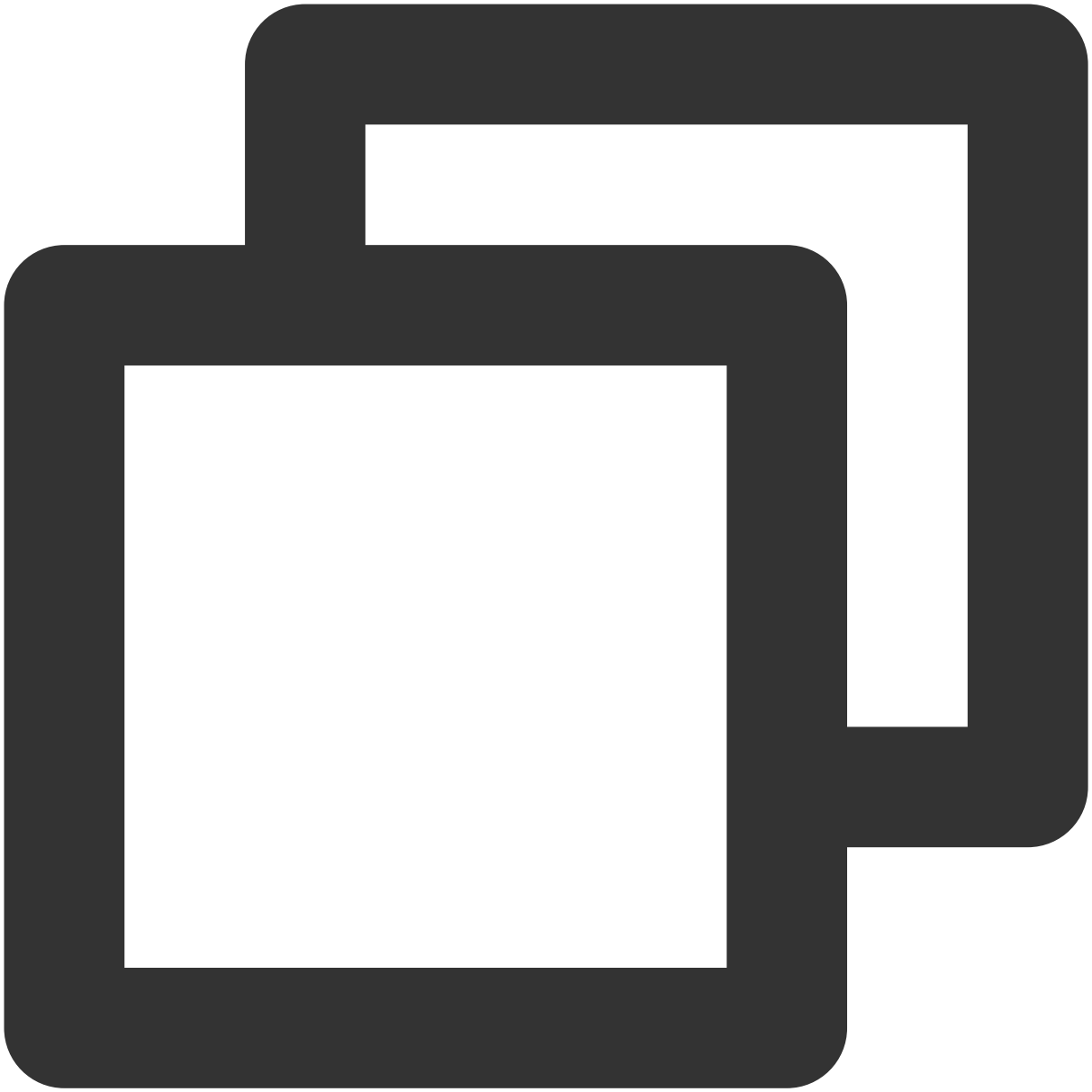
Scheduling query plan

cos_fdw can estimate the size of foreign files for scheduling the query plan. For a foreign table mapped to multiple COS files, cos_fdw can print out the size of each file and calculate the total size of all files.

```
-- Single file
postgres=# EXPLAIN SELECT * FROM test_csv;
 QUERY PLAN
-----------------------------------------------------------------
--------------
Foreign Scan on test_csv (cost=0.00..1.10 rows=1 width=128)
 Foreign COS Url: https://xxxxxxx.cos.ap-nanjing.myqcloud.com
 Foreign COS File Path: /test_csv.csv
 Foreign each COS File Size(Bytes): 86
 Foreign total COS File Size(Bytes): 86
(5 rows)
```

```
-- Multiple files
postgres=# EXPLAIN SELECT * FROM multi_csv;
 QUERY PLAN
-----------------------------------------------------------------
---------------
Foreign Scan on multi_csv (cost=0.00..1.20 rows=2 width=128)
 Foreign COS Url: https://xxxxxxxxxx.cos.ap-nanjing.myqcloud.com
 Foreign COS File Path: /a.csv,/b.csv,/c.csv.2
 Foreign each COS File Size(Bytes): 15,172,86
 Foreign total COS File Size(Bytes): 273
(5 rows)
```

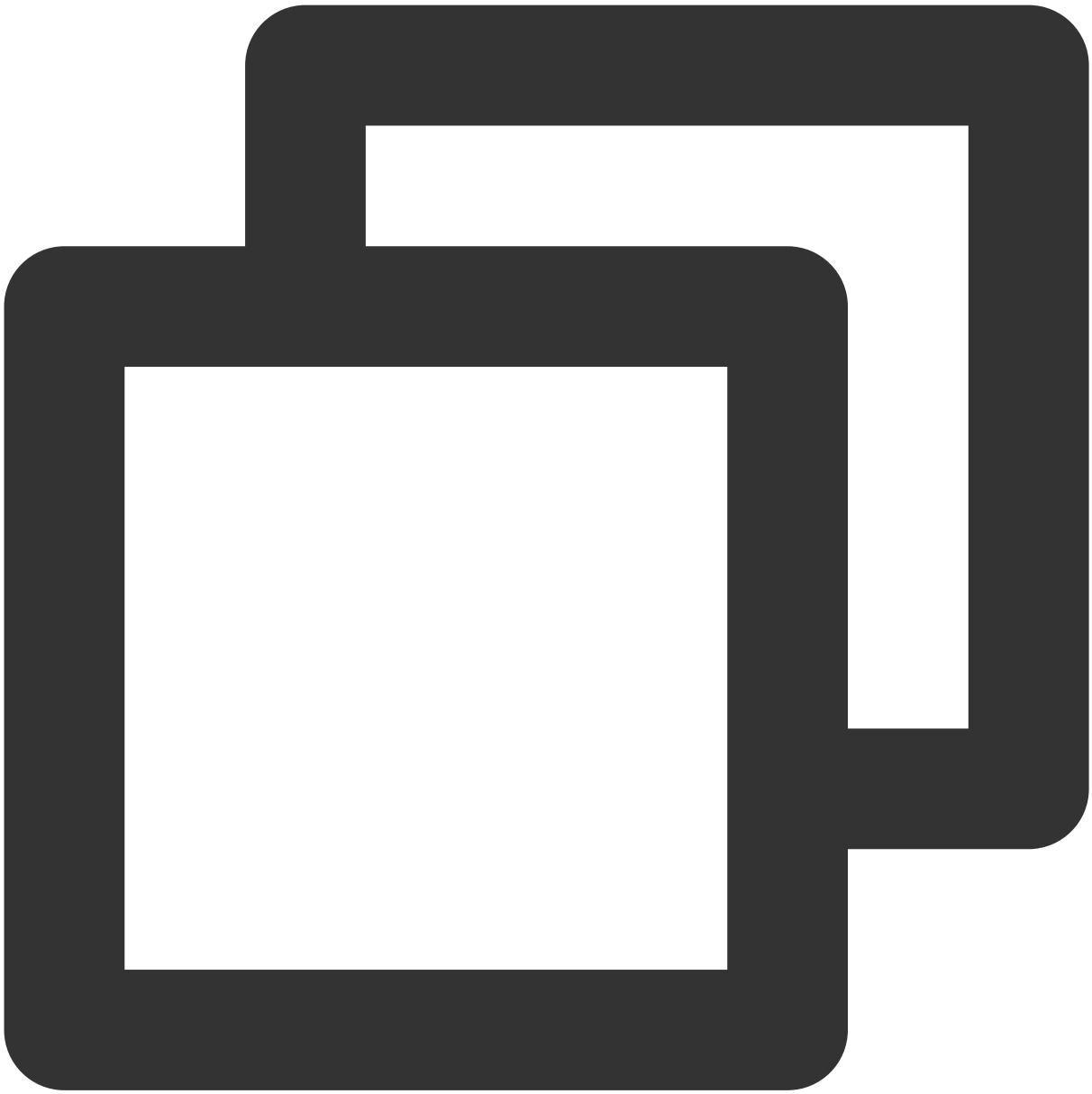
Querying data



```
postgres=# SELECT * FROM test_csv;
word1 | word2 | word3 | word4
-------+-------+-------+-------
AAA | aaa | 123 |
XYZ | xyz | | 321
NULL | | |
NULL | | |
ABC | abc | | (5 rows)
```

Importing data from foreign table to local table

You can use statements similar to `insert into ... select * from ...;` to import data from a foreign table to a local table.



```
postgres=# CREATE TABLE local_test_csv (
postgres(# a text,
postgres(# b text,
postgres(# c text,
postgres(# d text
postgres(# );
CREATE TABLE
postgres=# INSERT INTO local_test_csv SELECT * FROM test_csv;
```

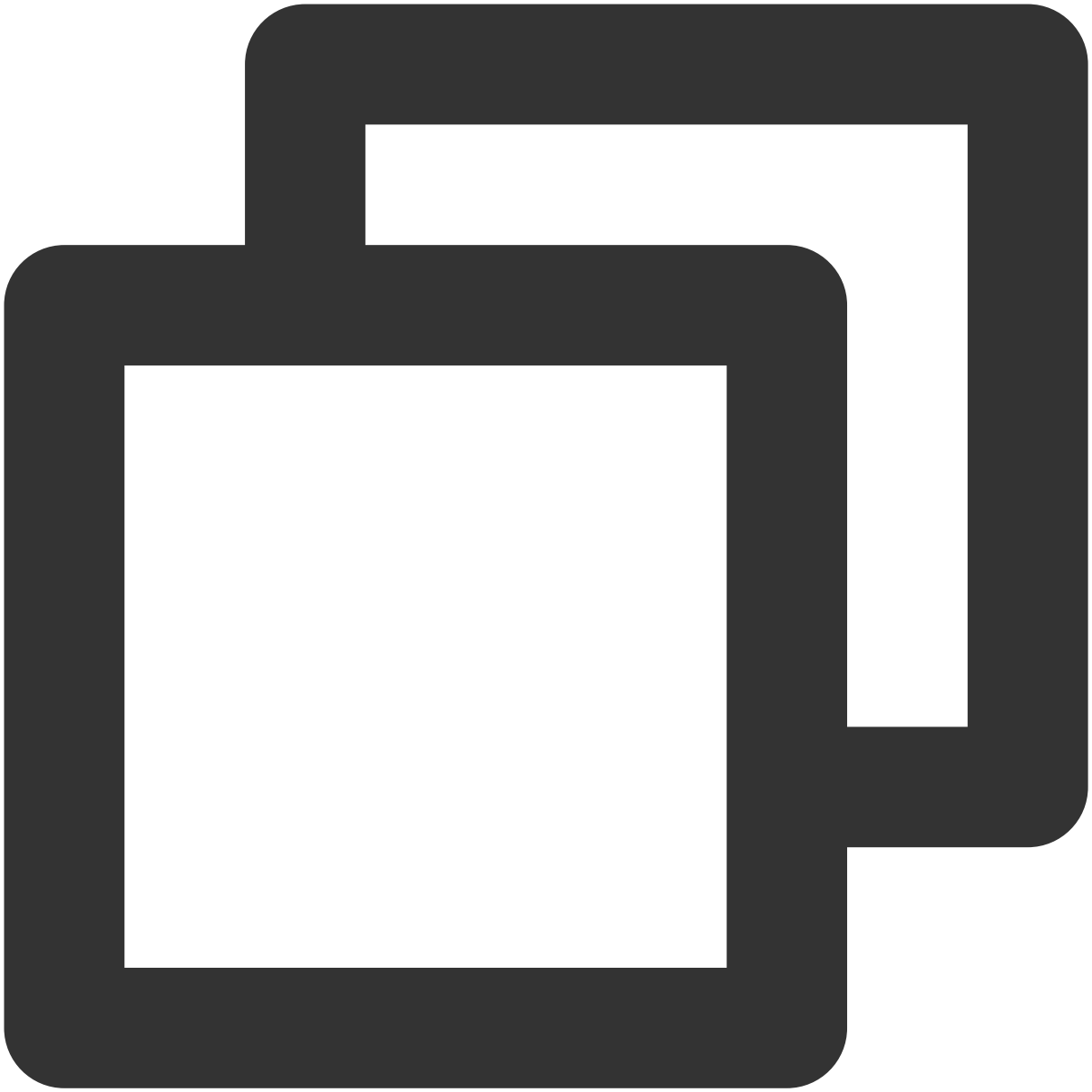
```
INSERT 0 5
postgres=# SELECT * FROM local_test_csv;
 a | b | c | d
------+-----+-----+-----
AAA | aaa | 123 |
XYZ | xyz | | 321
NULL | | |
NULL | | |
ABC | abc | | (5 rows)
```

Querying partitioned table

```
postgres=# CREATE TABLE pt (a int, b text) partition by list (a);
CREATE TABLE
postgres=# CREATE FOREIGN TABLE p1 partition of pt for values in (1) SERVER
cos_server
postgres-# OPTIONS (format 'csv', filepath '/list1.csv', delimiter ',');
CREATE FOREIGN TABLE
postgres=# CREATE TABLE p2 partition of pt for values in (2);
CREATE TABLE
-- Partitioned tables can be queried
postgres=# SELECT tableoid::regclass, * FROM pt;
tableoid | a | b
```

```
----------+---+-----
p1 | 1 | foo
p1 | 1 | bar
(2 rows)
postgres=# SELECT tableoid::regclass, * FROM p1;
tableoid | a | b
----------+---+-----
p1 | 1 | foo
p1 | 1 | bar
(2 rows)
postgres=# SELECT tableoid::regclass, * FROM p2;
tableoid | a | b
----------+---+---
(0 rows)
-- Currently, data cannot be written to foreign tables
postgres=# INSERT INTO pt VALUES (1, 'xyzzy'); -- ERROR
ERROR: cannot route inserted tuples to a foreign table
-- As local tables are not affected, data can be written to local partitioned table
postgres=# INSERT INTO pt VALUES (2, 'xyzzy');
INSERT 0 1
postgres=# SELECT tableoid::regclass, * FROM pt;
tableoid | a | b
----------+---+-------
p1 | 1 | foo
p1 | 1 | bar
p2 | 2 | xyzzy
(3 rows)
postgres=# SELECT tableoid::regclass, * FROM p1;
tableoid | a | b
----------+---+-----
p1 | 1 | foo
p1 | 1 | bar
(2 rows)
postgres=# SELECT tableoid::regclass, * FROM p2;
tableoid | a | b
----------+---+-------
p2 | 2 | xyzzy
(1 row)
```

Dropping Extension



```
DROP EXTENSION cos_fdw;
```

Parameters

`CERATE SERVER` parameters

Parameter	Description
host	

	Address for accessing COS over the private network. Note that `host` cannot contain the `http` or `https` prefix.
bucket	Bucket name in the format of `BucketName-APPID`
id	Account secret ID
key	Account secret key

`CREATE FOREIGN TABLE` parameters

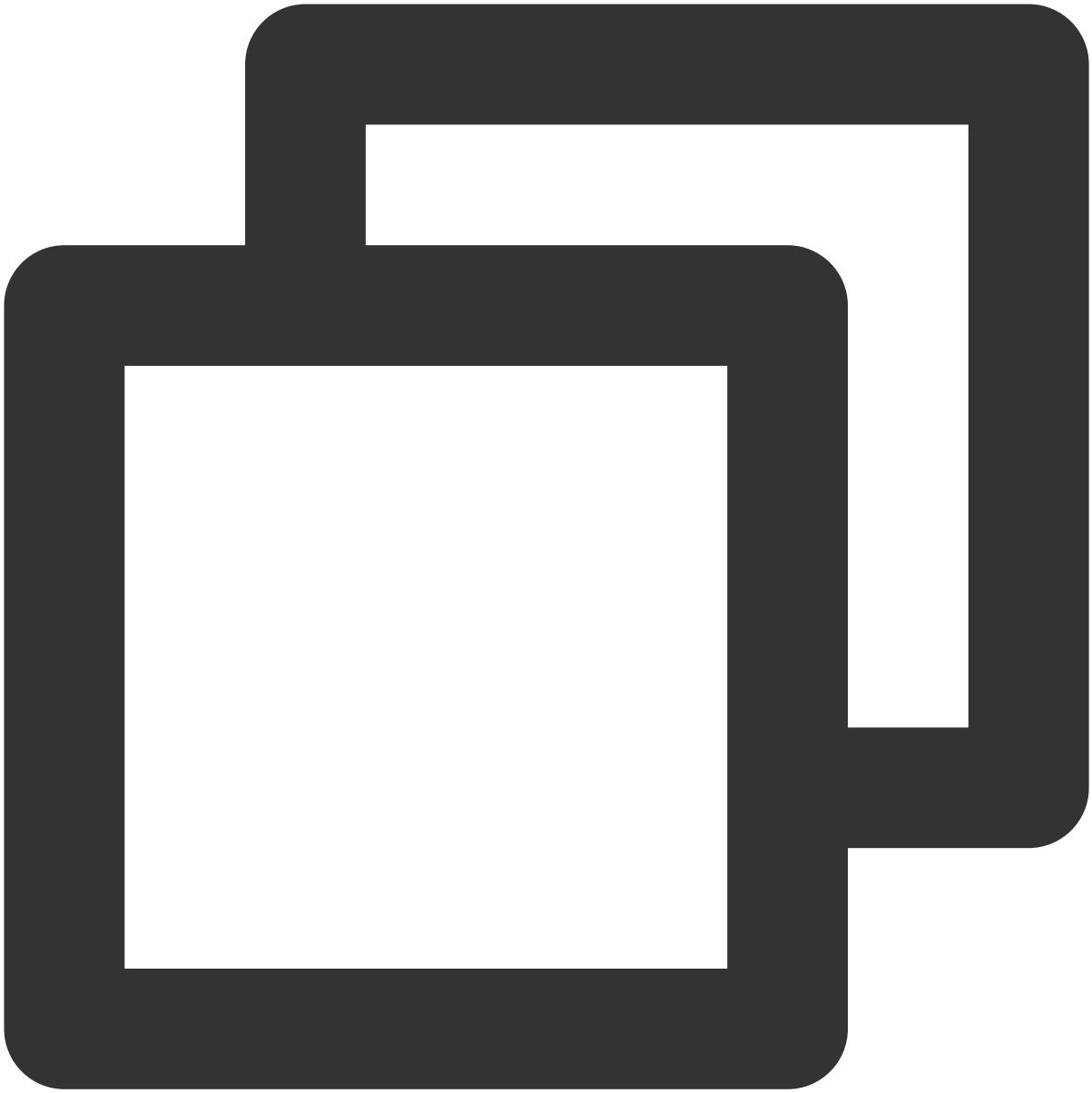
Parameter	Description
filepath	Sample
format	Data format, which currently can only be CSV.
delimiter	Data delimiter
quote	Data quote character
escape	Data escape character
encoding	Data encoding
null	Specifies that the column matching the corresponding string is `null`. For example, `null 'NULL'` indicates to specify the string of the column value 'NULL' to `null`
force_not_null	Specifies that the column's value should not match an empty string. For example, `force_not_null 'id'` indicates that if the value of the `id` column is empty, the value queried from the foreign table is an empty string but not `null`.
force_null	Specifies that the column's value matches an empty string. For example, `force_null 'id'` indicates that if the value of the `id` column is empty, the value queried from the foreign table is `null`.

Error Handling

When a data request sent by cos_fdw to COS times out, the following will be displayed:

code: HTTP status code of the abnormal request.

HTTP header: Error information. For more information on the format, see Common Response Headers. You can submit a ticket with the `x-cos-request-id` for assistance. If the field is empty, the request failed to be sent to COS.



```
• postgres=# SELECT * FROM test_csv; • ERROR: COS api return error. • DETAIL: COS a
• HTTP/1.1 403 Forbidden
• Content-Type: application/xml
• Content-Length: 0 • Connection: keep-alive
• Date: Thu, 07 Apr 2022 09:00:22 GMT
• Server: tencent-cos
• x-cos-request-id: NjI0ZWE4MjZfNDc1NGU0MDlfMjI3ZTJfMTI3YTJjMWM=
• x-cos-trace-id:
OGVmYzZiMmQzYjA2OWNhODk0NTRkMTBiOWVmMDAxODc0OWRkZjk0ZDM1NmI1M2E2MTRlY2MzZDhmNmI5MWI
```

Implement Read/Write Separation via pgpool

Last updated：2024-03-27 17:08:19

1. Install pgpool

Download pgpool and install it, download address.

```
$ ./configure
```

```
$ make
```

```
$ make install
```

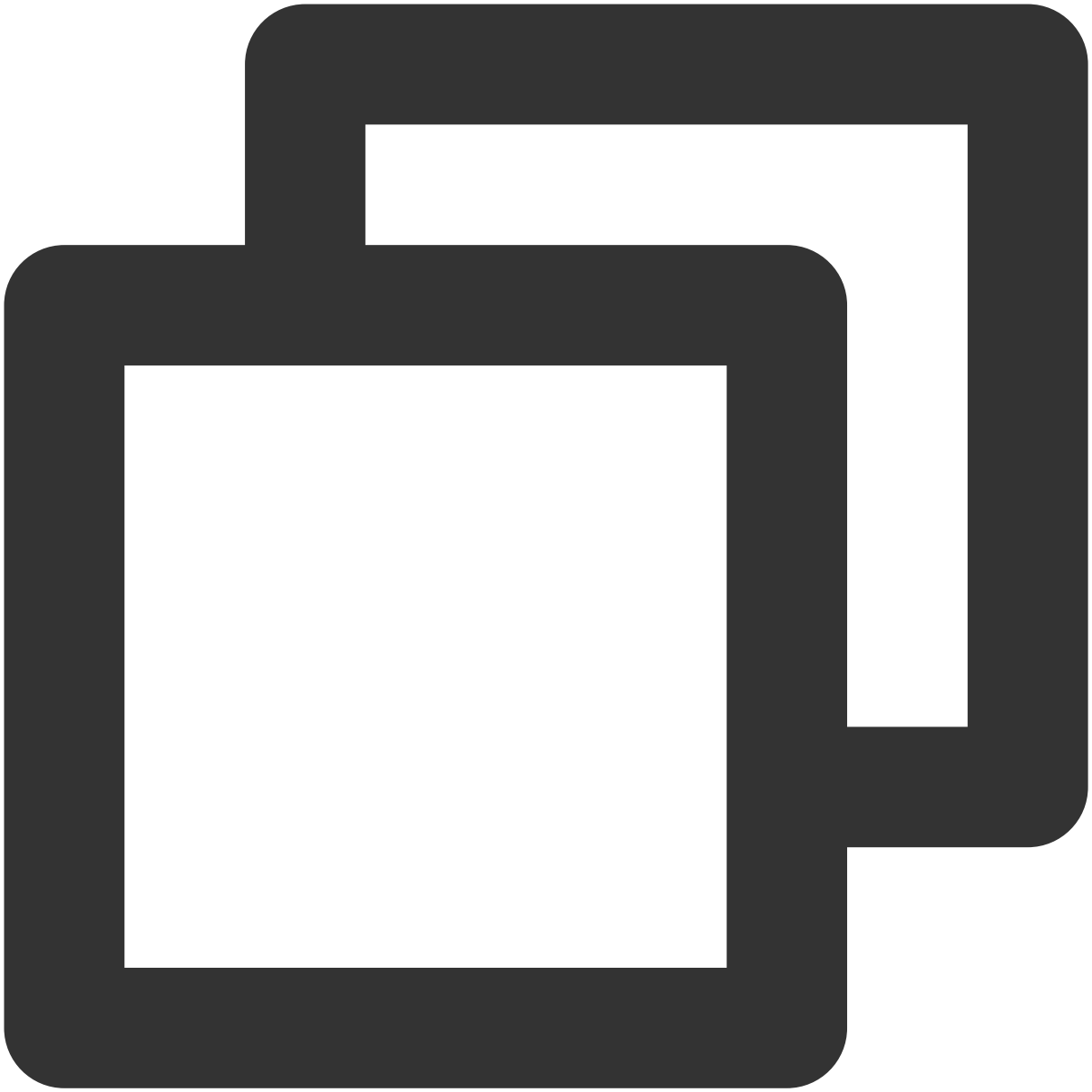
2. Configuration File

Note:

Use pgpool to implement the Cloud Load Balancer access. All authentication occurs between the client and pgpool, and the client still needs to continue the PostgreSQL's authentication process.

Configure the pool_passwd password file

The pool_passwd password file is required when connecting to the database through pgpool. You can generate the password file using the following command:



```
[root@VM-0-15-tencentos ~]# cd /usr/local/bin
[root@VM-0-15-tencentos bin]# pg_md5 --md5auth --username=dbadmin password
[root@VM-0-15-tencentos bin]# more /usr/local/etc/pool_passwd
dbadmin:md50b0cdb5c1d1f30fe83e5a72061749681
```

Configure the pgpool.conf file

After installing pgpool-II, pgpool.conf.sample is automatically created. We recommend copying or renaming it to pgpool.conf, then you can edit it freely.

```
$ cp /usr/local/etc/pgpool.conf.sample /usr/local/etc/pgpool.conf
```

pgpool-II only accepts local connections to port 9999 by default. If you want to accept connections from other hosts, please set up.

```
listen_addresses = 'localhost'
port = 9999
```

The important pgpool configurations are as follows, please refer to:

```
#------------------------------------------------------------------------------
# BACKEND CLUSTERING MODE
# Choose one of: 'streaming_replication', 'native_replication',
#      'logical_replication', 'slony', 'raw' or 'snapshot_isolation'
# (change requires restart)
#------------------------------------------------------------------------------
backend_clustering_mode = 'streaming_replication'
#------------------------------------------------------------------------------
# CONNECTIONS
#------------------------------------------------------------------------------
# - pgpool Connection Settings -
listen_addresses = '0.0.0.0'
                                    # what host name(s) or IP address(es) to listen
on;
                                    # comma-separated list of addresses;
                                    # defaults to 'localhost'; use '*' for all
                                    # (change requires restart)
port = 9989
                                    # Port number
                                    # (change requires restart)
unix_socket_directories = '/tmp'
                                    # Unix domain socket path(s)
                                    # The Debian package defaults to
                                    # /var/run/postgresql
                                    # (change requires restart)
#unix_socket_group = ''
                                    # The Owner group of Unix domain socket(s)
                                    # (change requires restart)
reserved_connections = 0
                                    # Number of reserved connections.
                                    # Pgpool-II does not accept connections if over
```

```
                                    # num_init_children - reserved_connections.
# - pgpool Communication Manager Connection Settings -
pcp_listen_addresses = ''
                                    # what host name(s) or IP address(es) for pcp
process to listen on;
                                    # comma-separated list of addresses;
                                    # defaults to 'localhost'; use '*' for all
                                    # (change requires restart)
pcp_port = 9898
                                    # Port number for pcp
                                    # (change requires restart)
pcp_socket_dir = '/tmp'
                                    # Unix domain socket path for pcp
                                    # The Debian package defaults to
                                    # /var/run/postgresql
                                    # (change requires restart)
listen_backlog_multiplier = 2
                                    # Set the backlog parameter of listen(2) to
                                    # num_init_children * listen_backlog_multiplier.
                                    # (change requires restart)
serialize_accept = off
                                    # whether to serialize accept() call to avoid
thundering herd problem
                                    # (change requires restart)
# - Backend Connection Settings -
backend_hostname0 = '172.16.0.3'
                                    # Host name or IP address to connect to for
backend 0
backend_port0 = 5432
                                    # Port number for backend 0
backend_weight0 = 1
                                    # Weight for backend 0 (only in load balancing
mode)
#backend_data_directory0 = '/data'
                                    # Data directory for backend 0
backend_flag0 = 'ALWAYS_PRIMARY'
```

```
                                    # Controls various backend behavior
                                    # ALLOW_TO_FAILOVER, DISALLOW_TO_FAILOVER
                                    # or ALWAYS_PRIMARY
backend_application_name0 = 'server0'
                                    # walsender's application_name, used for "show
pool_nodes" command
backend_hostname1 = '172.16.0.12'
backend_port1 = 5432
backend_weight1 = 1
#backend_data_directory1 = '/data1'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'
# - Authentication -
enable_pool_hba = on
                                    # Use pool_hba.conf for client authentication
pool_passwd = 'pool_passwd'
                                    # File name of pool_passwd for md5
authentication.
                                    # "" disables pool_passwd.
                                    # (change requires restart)
allow_clear_text_frontend_auth = off
                                    # Allow Pgpool-II to use clear text password
authentication
                                    # with clients, when pool_passwd does not
                                    # contain the user password
# - SSL Connections -
ssl =off
                                    # Enable SSL support
                                    # (change requires restart)
#------------------------------------------------------------------------------
# POOLS
#------------------------------------------------------------------------------
num_init_children = 32
                                    # Maximum Number of concurrent sessions allowed
                                    # (change requires restart)
max_pool = 4
```

```
                                    # Number of connection pool caches per
connection
                                    # (change requires restart)
# - Life time -
child_life_time = 5min
                                    # Pool exits after being idle for this many
seconds
child_max_connections = 0
                                    # Pool exits after receiving that many
connections
                                    # 0 means no exit
connection_life_time = 0
                                    # Connection to backend closes after being idle
for this many seconds
                                    # 0 means no close
client_idle_limit = 0
                                    # Client is disconnected after being idle for
that many seconds
                                    # (even inside an explicit transactions!)
                                    # 0 means no disconnection
#------------------------------------------------------------------------------
# FILE LOCATIONS
#------------------------------------------------------------------------------
pid_file_name = '/var/run/pgpool/pgpool.pid'
                                    # PID file name
                                    # Can be specified as relative to the"
                                    # location of pgpool.conf file or
                                    # as an absolute path
                                    # (change requires restart)
logdir = '/tmp'
                                    # Directory of pgPool status file
                                    # (change requires restart)
#------------------------------------------------------------------------------
# CONNECTION POOLING
#------------------------------------------------------------------------------
connection_cache = on
```

```
                                        # Activate connection pools
                                        # (change requires restart)
                                        # Semicolon separated list of queries
                                        # to be issued at the end of a session
                                        # The default is for 8.3 and later
reset_query_list = 'ABORT; DISCARD ALL'
                                        # The following one is for 8.2 and before
#------------------------------------------------------------------------------
# LOAD BALANCING MODE
#------------------------------------------------------------------------------
load_balance_mode = on
                                        # Activate load balancing mode
                                        # (change requires restart)
ignore_leading_white_space = on
                                        # Ignore leading white spaces of each query
write_function_list = ''
                                        # Comma separated list of function names
                                        # that write to database
                                        # Regexp are accepted
                                        # If both read_only_function_list and
write_function_list
                                        # is empty, function's volatile property is
checked.
                                        # If it's volatile, the function is regarded as
a
                                        # writing function.
allow_sql_comments = off
                                        # if on, ignore SQL comments when judging if
load balance or
                                        # query cache is possible.
                                        # If off, SQL comments effectively prevent the
judgment
                                        # (pre 3.4 behavior).
disable_load_balance_on_write = 'transaction'
                                        # Load balance behavior when write query is
issued
```

```
                                        # in an explicit transaction.
                                        #
                                        # Valid values:
                                        #
                                        # 'transaction' (default):
                                        #     if a write query is issued, subsequent
                                        #     read queries will not be load balanced
                                        #     until the transaction ends.
                                        #
                                        # 'trans_transaction':
                                        #     if a write query is issued, subsequent
                                        #     read queries in an explicit transaction
                                        #     will not be load balanced until the
session ends.
                                        #
                                        # 'dml_adaptive':
                                        #     Queries on the tables that have already
been
                                        #     modified within the current explicit
transaction will
                                        #     not be load balanced until the end of the
transaction.
                                        #
                                        # 'always':
                                        #     if a write query is issued, read queries
will
                                        #     not be load balanced until the session
ends.
                                        #
                                        # Note that any query not in an explicit
transaction
                                        # is not affected by the parameter except
'always'.
statement_level_load_balance = off
                                        # Enables statement level load balancing
#------------------------------------------------------------------------------
```

```
# HEALTH CHECK GLOBAL PARAMETERS
#------------------------------------------------------------------------------
health_check_period = 0
                                    # Health check period
                                    # Disabled (0) by default
health_check_timeout = 20
                                    # Health check timeout
                                    # 0 means no timeout
health_check_user = 'nobody'
                                    # Health check user
health_check_password = ''
                                    # Password for health check user
                                    # Leaving it empty will make Pgpool-II to first
look for the
                                    # Password in pool_passwd file before using the
empty password
health_check_database = ''
                                   # Database name for health check. If '', tries
'postgres' frist,
health_check_max_retries = 60
                                    # Maximum number of times to retry a failed
health check before giving up.
health_check_retry_delay = 1
                                    # Amount of time to wait (in seconds) between
retries.
connect_timeout = 10000
                                    # Timeout value in milliseconds before giving up
to connect to backend.
                                    # Default is 10000 ms (10 second). Flaky network
user may want to increase
                                    # the value. 0 means no timeout.
                                    # Note that this value is not only used for
health check,
                                    # but also for ordinary conection to backend.
```

3. Configure the PCP Command

pgpool-II has an interface for management purposes, used to access database node information, shut down pgpool-II, etc. To use the PCP command, user authentication is required. This kind of authentication is different from PostgreSQL user authentication. It requires defining a username and password in the pcp.conf file. In this file, a username and password appear in pairs on each line, separated by a colon (:). Passwords are in MD5-hashed format.

4. Configure Database Nodes

```
# - Backend Connection Settings -
backend_hostname0 = '172.16.0.30'
                                    # Host name or IP address to connect to for
backend 0
backend_port0 = 5432
                                    # Port number for backend 0
backend_weight0 = 1
                                    # Weight for backend 0 (only in load balancing
mode)
#backend_data_directory0 = '/data'
                                    # Data directory for backend 0
backend_flag0 = 'ALWAYS_PRIMARY'
                                    # Controls various backend behavior
                                    # ALLOW_TO_FAILOVER, DISALLOW_TO_FAILOVER
                                    # or ALWAYS_PRIMARY
backend_application_name0 = 'server0'
                                    # walsender's application_name, used for "show
pool_nodes" command
backend_hostname1 = '172.16.0.16'
backend_port1 = 5432
backend_weight1 = 1
#backend_data_directory1 = '/data1'
backend_flag1 = 'DISALLOW_TO_FAILOVER'
backend_application_name1 = 'server1'
```

When `load_balance_mode` is set to true, pgpool-II will distribute SELECT queries among database nodes.

```
load_balance_mode = on
```

```
                                    # Activate load balancing mode
                                    # (change requires restart)
ignore_leading_white_space = on
                                    # Ignore leading white spaces of each query
write_function_list = ''
                                    # Comma separated list of function names
                                    # that write to database
                                    # Regexp are accepted
                                    # If both read_only_function_list and
write_function_list
                                    # is empty, function's volatile property is
checked.
                                    # If it's volatile, the function is regarded as
a
                                    # writing function.
allow_sql_comments = off
                                    # if on, ignore SQL comments when judging if
load balance or
                                    # query cache is possible.
                                    # If off, SQL comments effectively prevent the
judgment
                                    # (pre 3.4 behavior).
disable_load_balance_on_write = 'transaction'
                                    # Load balance behavior when write query is
issued
                                    # in an explicit transaction.
                                    #
                                    # Valid values:
                                    #
                                    # 'transaction' (default):
                                    #     if a write query is issued, subsequent
                                    #     read queries will not be load balanced
                                    #     until the transaction ends.
                                    #
                                    # 'trans_transaction':
                                    #     if a write query is issued, subsequent
```

```
                                        #     read queries in an explicit transaction
                                        #     will not be load balanced until the
session ends.
                                        #
                                        # 'dml_adaptive':
                                        #     Queries on the tables that have already
been
                                        #     modified within the current explicit
transaction will
                                        #     not be load balanced until the end of the
transaction.
                                        #
                                        # 'always':
                                        #     if a write query is issued, read queries
will
                                        #     not be load balanced until the session
ends.
                                        #
                                        # Note that any query not in an explicit
transaction
                                        # is not affected by the parameter except
'always'.
statement_level_load_balance = off
                                        # Enables statement level load balancing
```

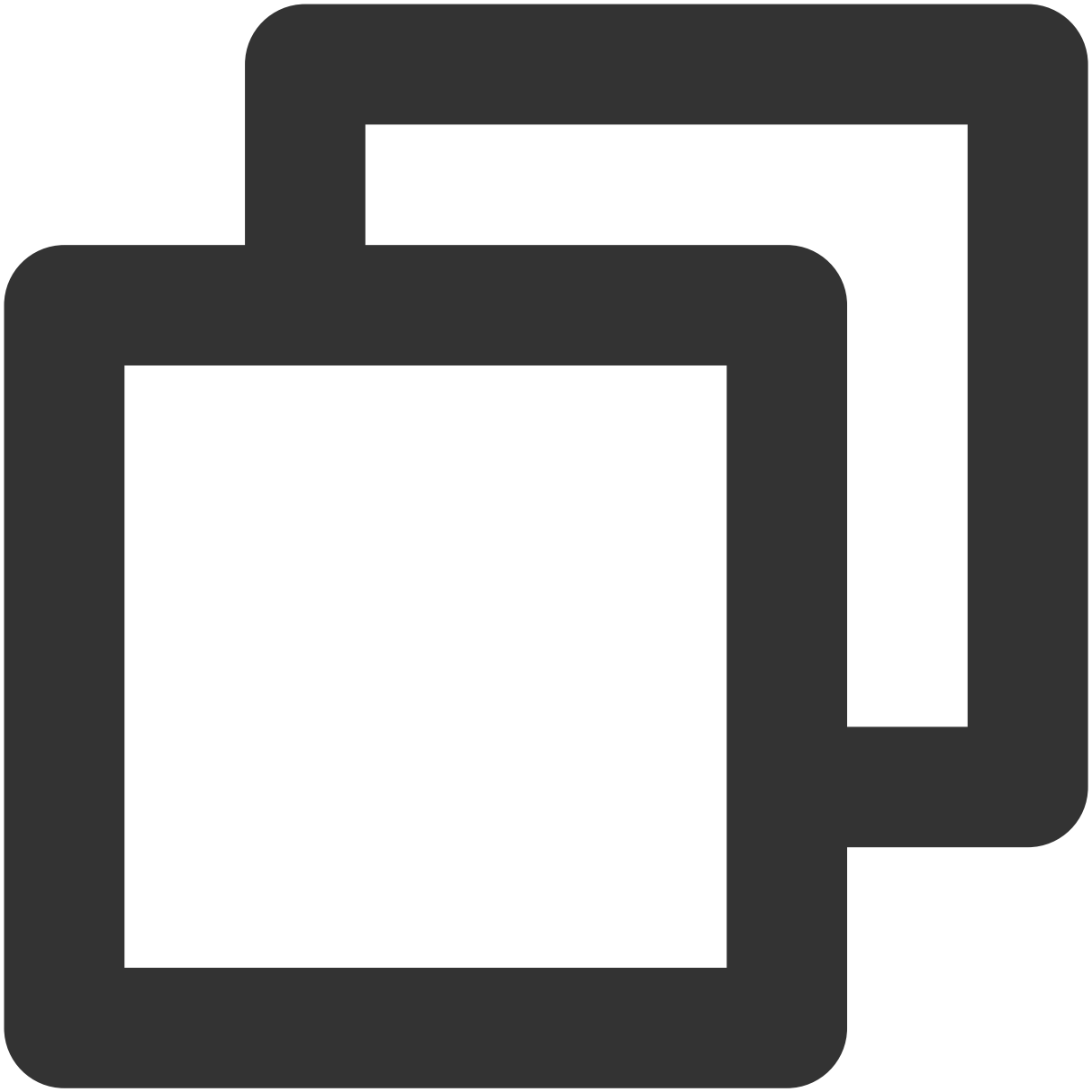
5. Start pgpool-II and Verify Read-Write Separation

```
$ pgpool -n -d > /tmp/pgpool.log 2>&1 &
```

Note:

Connect and query pg_is_in_recovery(), then disconnect and reconnect to query pg_is_in_recovery() again. If alternating responses of false and true are returned, it indicates that requests are being alternately sent to the master and slave servers, thereby indicating successful read-write separation.

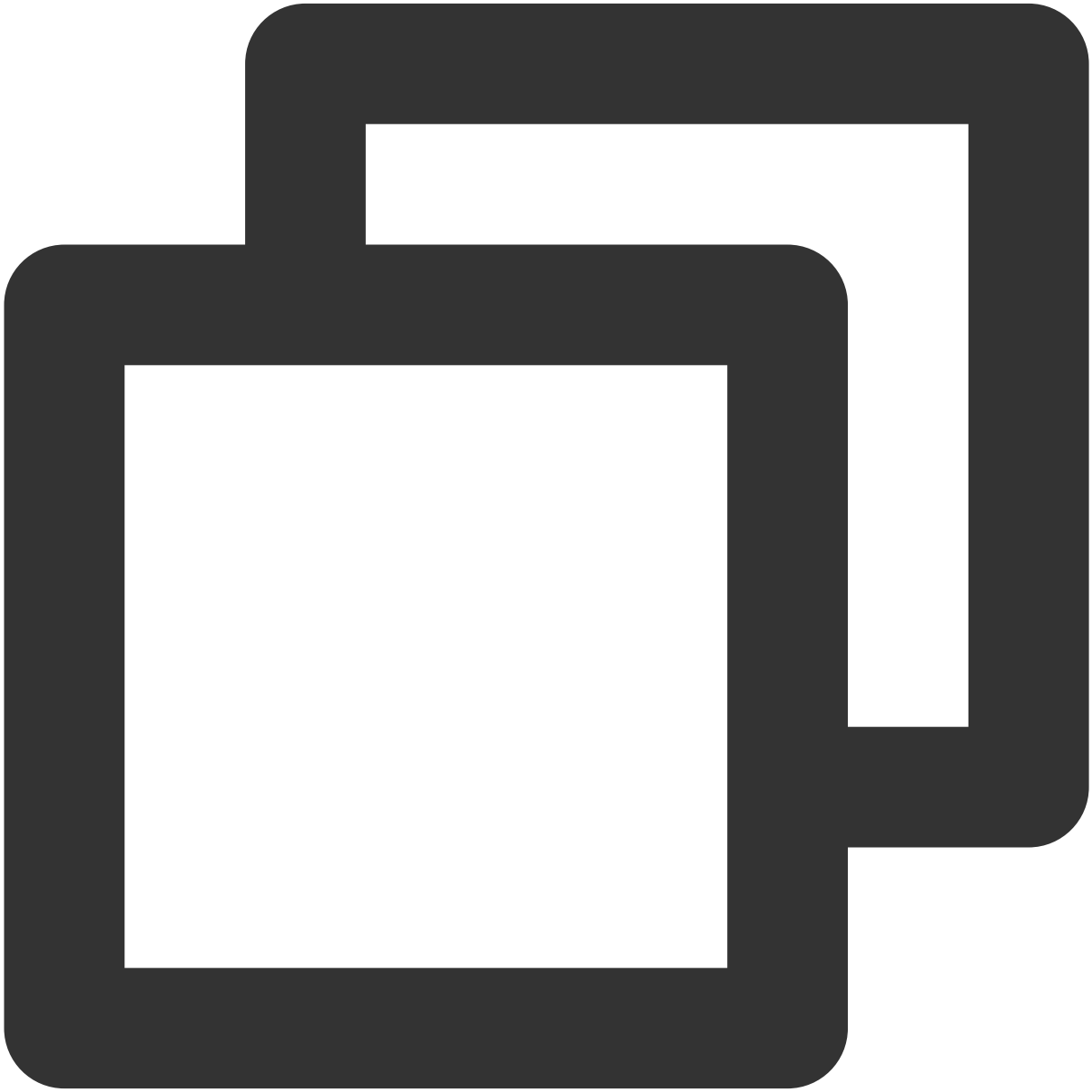
Use the psql client to connect to pgpool, showing status as normal.



```
[root@VM-0-15-tencentos ~]# /usr/local/pgsql/bin/psql -h127.0.0.1 -p9989 -Udbadmin
Password for user dbadmin:
psql (15.1)
Type "help" for help.

postgres=> show pool_nodes;
 node_id |  hostname    | port | status | pg_status | lb_weight |  role    | pg_role
atus_change
---------+-------------+------+--------+-----------+-----------+---------+---------
-------------
 0       | 172.16.0.30  | 5432 | up     | unknown   | 0.500000  | primary | unknown
```

```
-27 20:04:13
 1        | 172.16.0.16 | 5432 | up      | unknown    | 0.500000   | standby | unknown
-27 20:04:13
(2 rows)

postgres=>
```

Use read-write SQL on the client side. Due to the prior distinction between read-write and read-only instances, it can be seen that the read-write separation is successful.



```
postgres=> insert into pgpool1(id,name)values(3,'b');
```

```
INSERT 0 1
postgres=> select * from pgpool1;
 id | name
----+------
  1 | a
  2 | b
  3 | a
  4 | b
  3 | a
(5 rows)

postgres=>
```