云数据仓库 PostgreSQL

数据接入

产品文档

腾讯云

文档目录

数据接入
使用 DataX 离线导入 TencentDB 数据

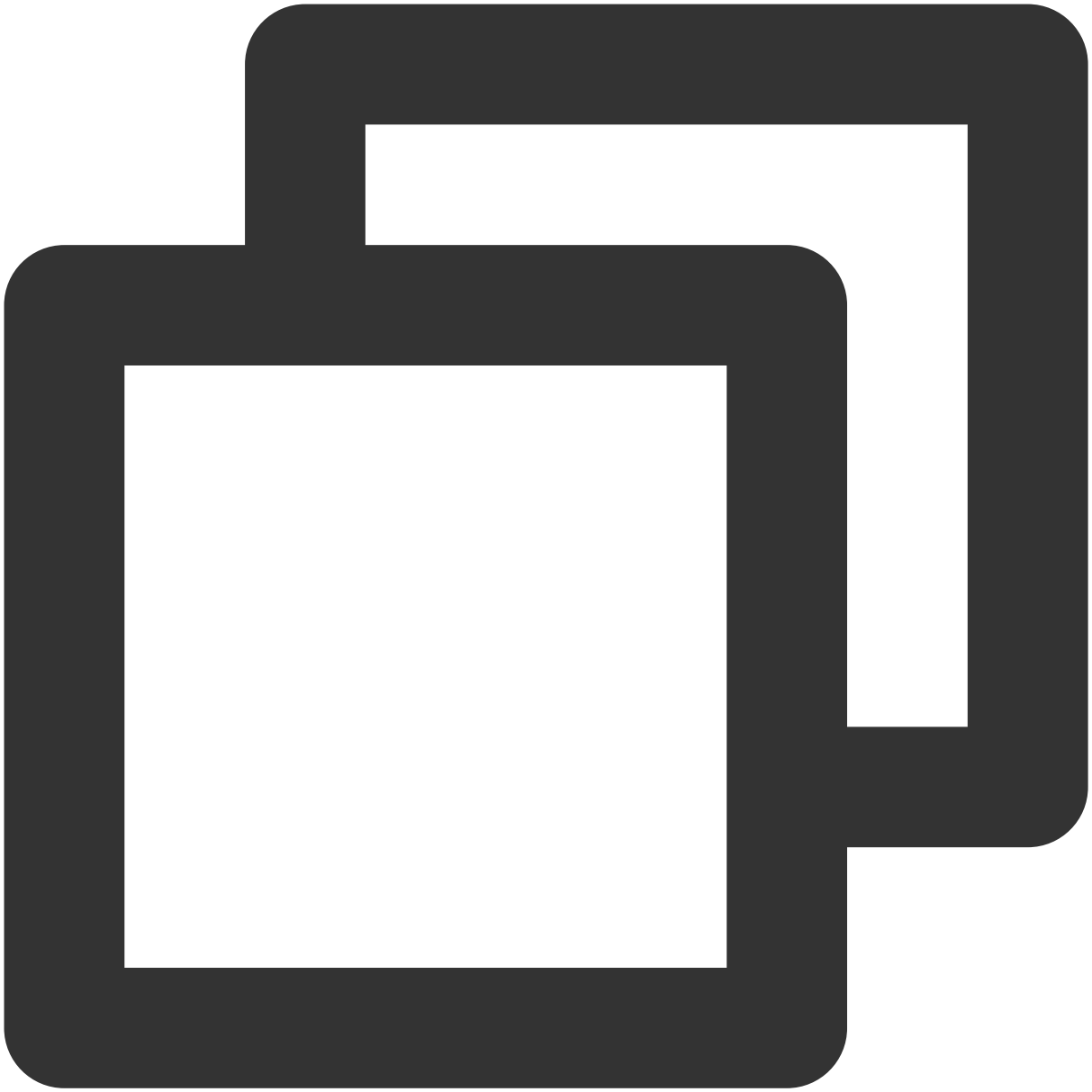
最近更新时间：2024-02-19 15:23:20

DataX 是一个开源的命令行工具，支持将 TencentDB 中全量或增量数据导入到云数据仓库 PostgreSQL 中。工具采用 Java 开发，用 JDBC 连接源数据库与目标数据库，可在 Windows 与 Linux 下运行，使用前需安装 Java 运行环境。

DataX 工具安装：

1. 在 DataX 官网 下载源码进行编译。
2. 直接使用已编译好的版本，datax-v1.0.4-hashdata.tar.gz。

下文主要介绍由 HashData 公司修改过的 DataX，其导入云数据仓库 PostgreSQL 效率更高，经测试可达到每秒10W条以上。以 MySQL 导入到云数据仓库 PostgreSQL 为例，配置文件如下：



```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 3,
                "byte": 1048576,
                "record": 1000
            },
            "errorLimit": {
                "record": 2,
                "percentage": 0.02
```

```
            }
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "****",
                        "password": "****",
                        "column": [
                            "*"
                        ],
                        "splitPk": "id",
                        "connection": [
                            {
                                "table": [
                                    "test1"
                                ],
                                "jdbcUrl": [
                                    "jdbc:mysql://***:***/db1?serverTimezone=Asia/S
                                ]
                            }
                        ]
                    }
                },
                "writer": {
                    "name": "gpdbwriter",
                    "parameter": {
                        "username": "******",
                        "password": "******",
                        "column": [
                            "*"
                        ],
                        "preSql": [
                            "truncate table test1"
                        ],
                        "postSql": [
                            "select count(*) from test2"
                        ],
                        "segment_reject_limit": 0,
                        "copy_queue_size": 2000,
                        "num_copy_processor": 1,
                        "num_copy_writer": 1,
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:postgresql://****:**/db1",
                                "table": [
```

```
                                    "test1"
                                ]
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

参数说明：

1. writer 需选择 gpdbwriter。使用 postgresqlwriter 也可写入云数据仓库 PostgreSQL，但插入效率会很低。
2. 参数具体含义和调优可以参考 DataX。
3. `mysqlreader` 的 jdbc url 建议加上 `serverTimezone=Asia/Shanghai` 参数，避免时区问题导致的数据不一致。

DataX 增量同步导入 MySQL 数据

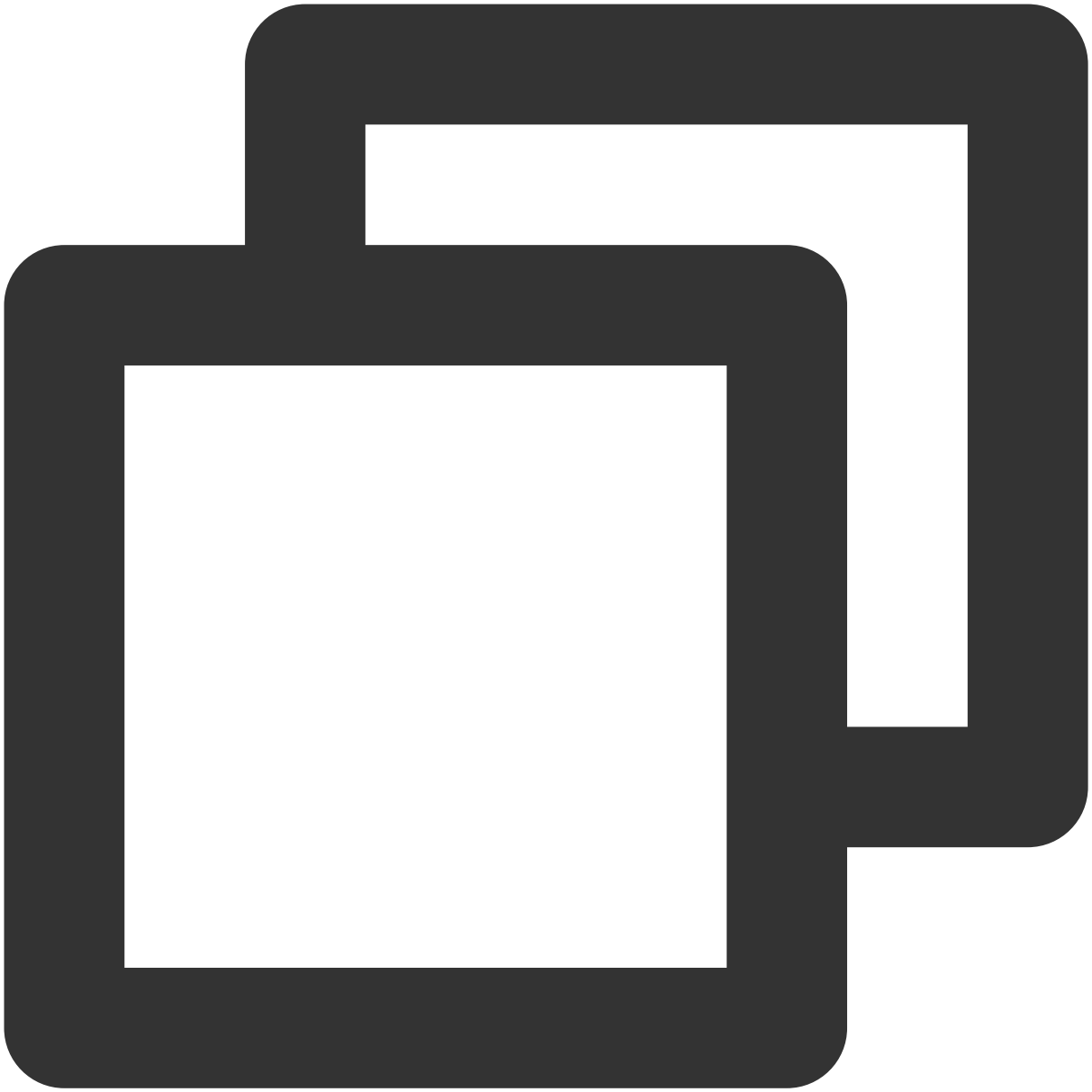
最近更新时间：2024-02-19 15:23:20

本文主要介绍使用 HashData 公司修改过的 DataX，将其 MySQL 中的数据增量同步到云数据仓库 PostgreSQL。

使用 DataX 将 MySQL 中的数据增量同步到云数据仓库 PostgreSQL 中，具体步骤如下：

1. 从本地文件读取上次同步成功之后的最大时间 MaxTime（初始同步时，可以结合业务选取指定一个初始时间值）。
2. 将 MaxTime 作为本次同步时间 LastTime（增量同步的下限），将当前时间 CurTime 作为同步增量的上限。
3. 修改 datax.json 配置，指定同步表的时间区间（SQL 的 where 条件）为：`[LastTime, CurTime)` 。
4. 执行 datax 同步，同步成功后，将 CurTime 写入本地文件供下次同步使用。
5. 循环执行1 - 4实现多次增量同步。

datax.json 配置文件示例如下：



```
{
    "job": {
        "setting": {
            "speed": {
                "channel": 3,
                "byte": 1048576,
                "record": 1000
            },
            "errorLimit": {
                "record": 2,
                "percentage": 0.02
```

```
            }
        },
        "content": [
            {
                "reader": {
                    "name": "mysqlreader",
                    "parameter": {
                        "username": "******",
                        "password": "******",
                        "connection": [
                            {
                                "jdbcUrl": [
                                    "jdbc:mysql://***:***/test?serverTimezone=Asia/
                                ],
                                "querySql": [
                                    "select * from cdw_test_table where updateTime
                                ]
                            }
                        ]
                    }
                },
                "writer": {
                    "name": "gpdbwriter",
                    "parameter": {
                        "username": "******",
                        "password": "******",
                        "column": [
                                 "*"
                        ],
                        "segment_reject_limit": 0,
                        "copy_queue_size": 2000,
                        "num_copy_processor": 1,
                        "num_copy_writer": 1,
                        "connection": [
                            {
                                "jdbcUrl": "jdbc:postgresql://***:***/***",
                                "table": [
                                    "ods_cdw_test_table"
                                ]
                            }
                        ]
                    }
                }
            }
        ]
    }
}
```

使用外表高速导入或导出 COS 数据

最近更新时间：2024-02-19 15:23:20

使用 COS_EXT 查询 COS 数据

COS_EXT 是访问 COS 文件的外部数据访问插件，通过 DDL 定义外部表，可以按照普通的数据表执行 DML，实现对 COS 数据的操作。目前支持：
作为外表，读取 COS 数据。
作为外表，将结果导出到 COS。
作为外表，执行简单分析功能，分析 COS 数据。

注意事项

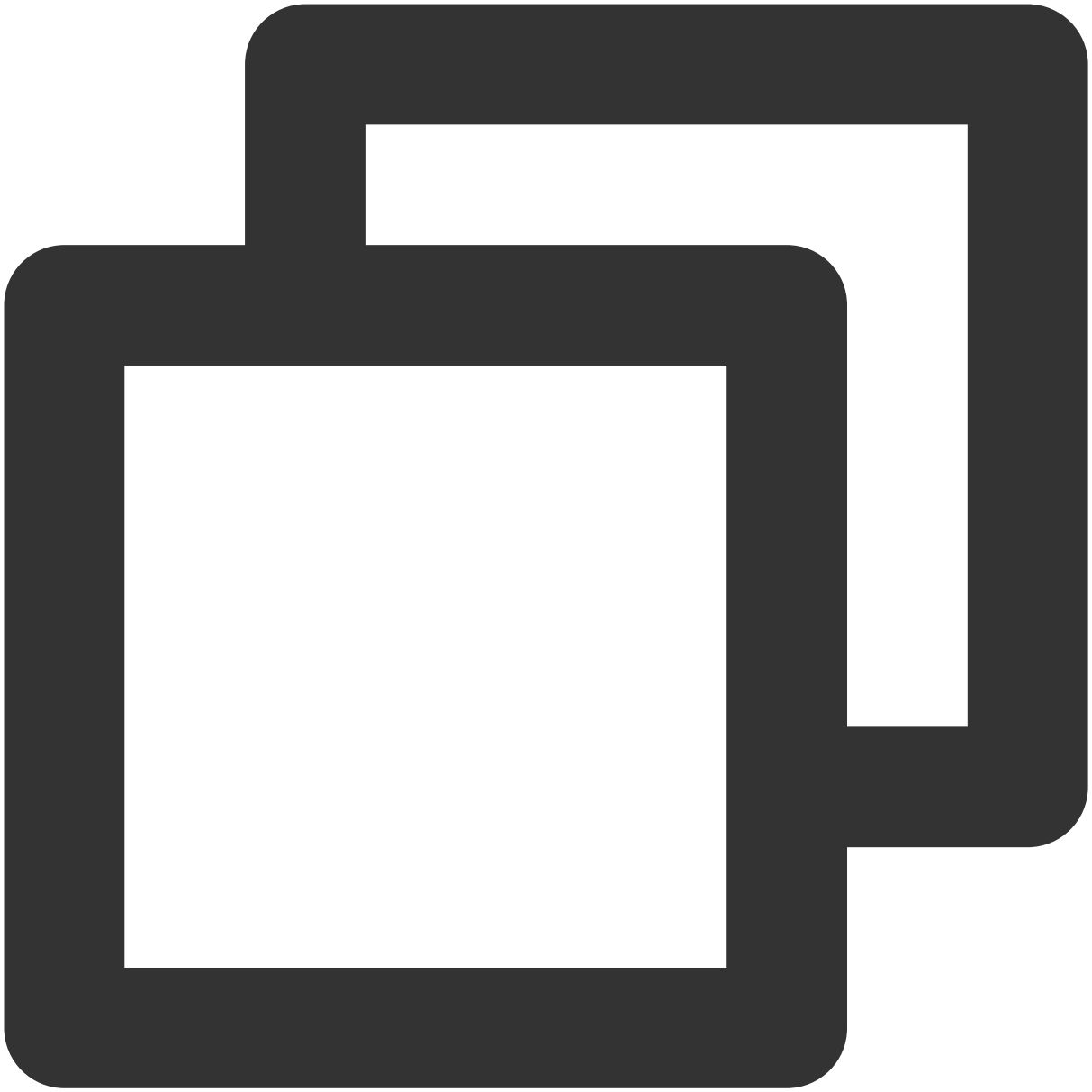
1. 支持 CSV 等文本格式文件，以及 GZIP 压缩格式文件。
2. 只能读取本地域的 COS 数据，例如，广州四区的集群只能读取广州地域的 COS 数据。
3. 只能读取用户自己的 COS 数据（这里用户是指创建集群的用户）。
4. 只写外表只能用于 INSERT 语句，不能用于 UPDATE/DELETE 语句，不能用于 SELECT 查询语句。
5. 删除外表，不会删除 COS 上的数据。

使用步骤

1. 定义 cos_ext 插件。

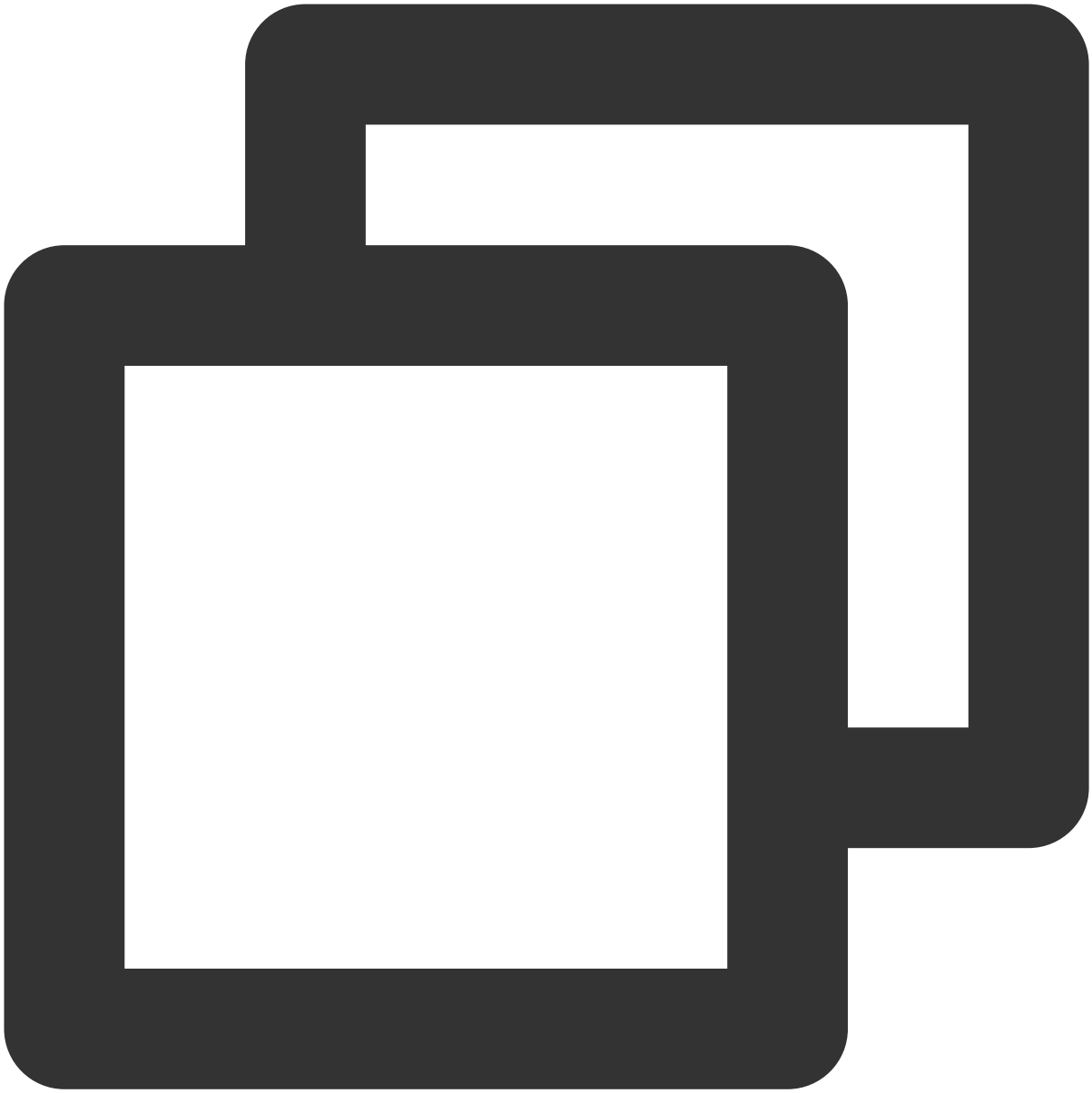
注意：

COS 外表插件的作用域为库。
创建命令如下：



```
CREATE EXTENSION IF NOT EXISTS cos_ext SCHEMA public;
```

删除命令如下：

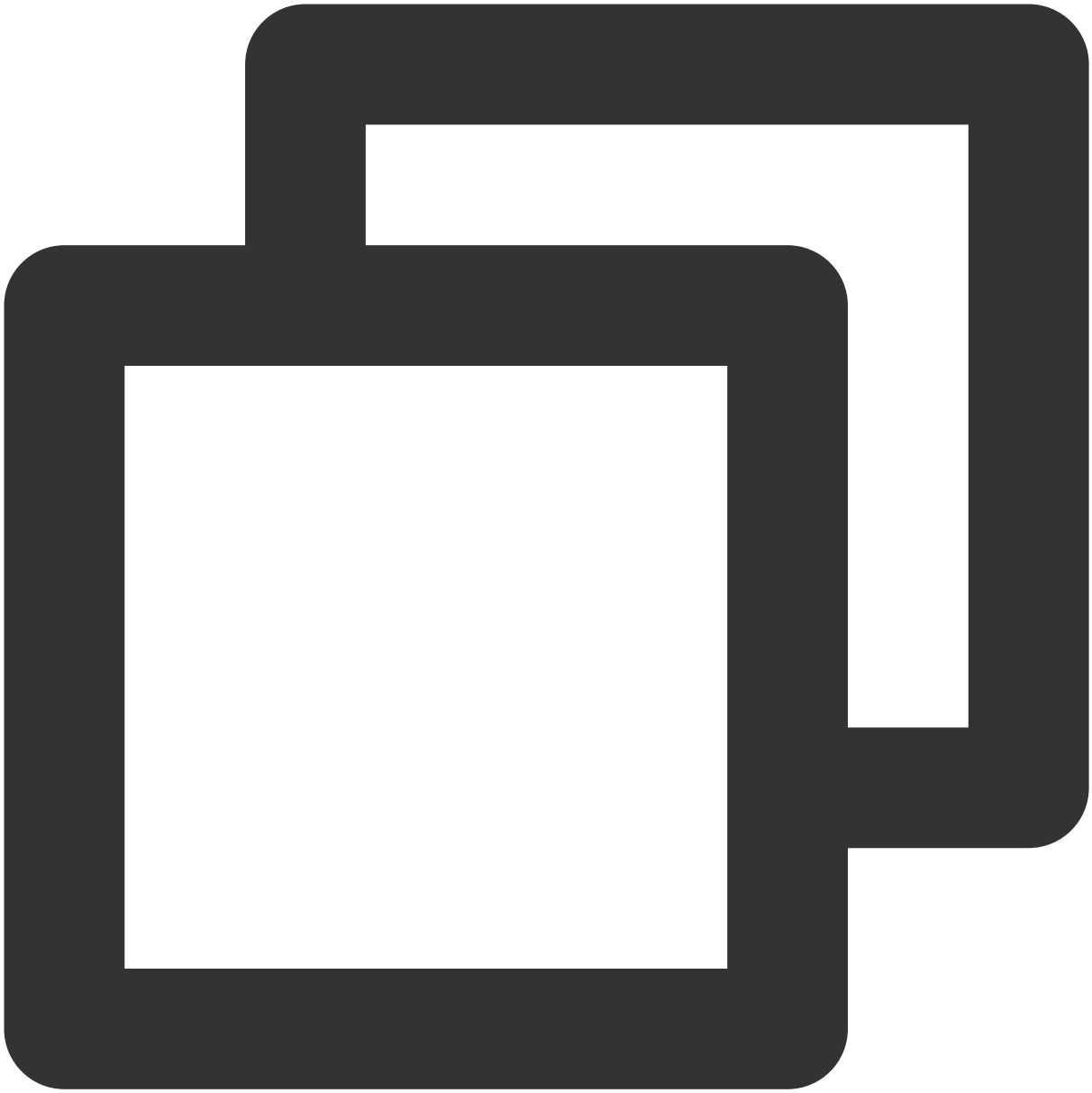


```
DROP EXTENSION IF EXISTS cos_ext;
```

2. 定义 COS 外表，语法参考 语法说明。
3. 操作 COS 外表数据。

语法说明

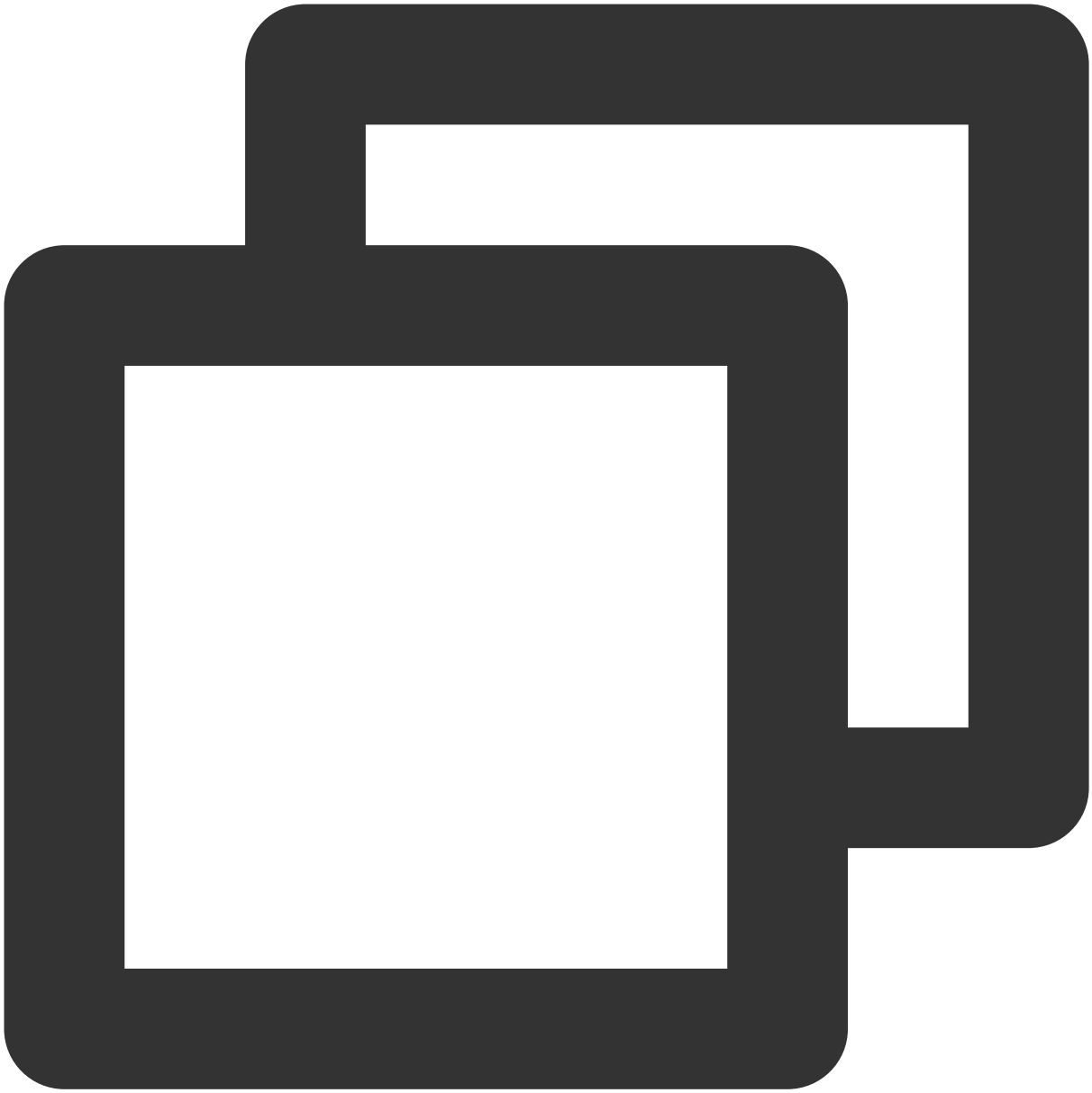
只读输入表定义：



```
CREATE [READABLE] EXTERNAL TABLE tablename
( columnname datatype [, ...] | LIKE othertable )
LOCATION (cos_ext_params)
FORMAT 'TEXT'
      [( [HEADER]
             [DELIMITER [AS] 'delimiter' | 'OFF']
             [NULL [AS] 'null string']
             [ESCAPE [AS] 'escape' | 'OFF']
             [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
             [FILL MISSING FIELDS] )]
     | 'CSV'
```

```
        [( [HEADER]
               [QUOTE [AS] 'quote']
               [DELIMITER [AS] 'delimiter']
               [NULL [AS] 'null string']
               [FORCE NOT NULL column [, ...]]
               [ESCAPE [AS] 'escape']
               [NEWLINE [ AS ] 'LF' | 'CR' | 'CRLF']
               [FILL MISSING FIELDS] )]
  [ ENCODING 'encoding' ]
  [ [LOG ERRORS [INTO error_table]] SEGMENT REJECT LIMIT count
         [ROWS | PERCENT] ]
```

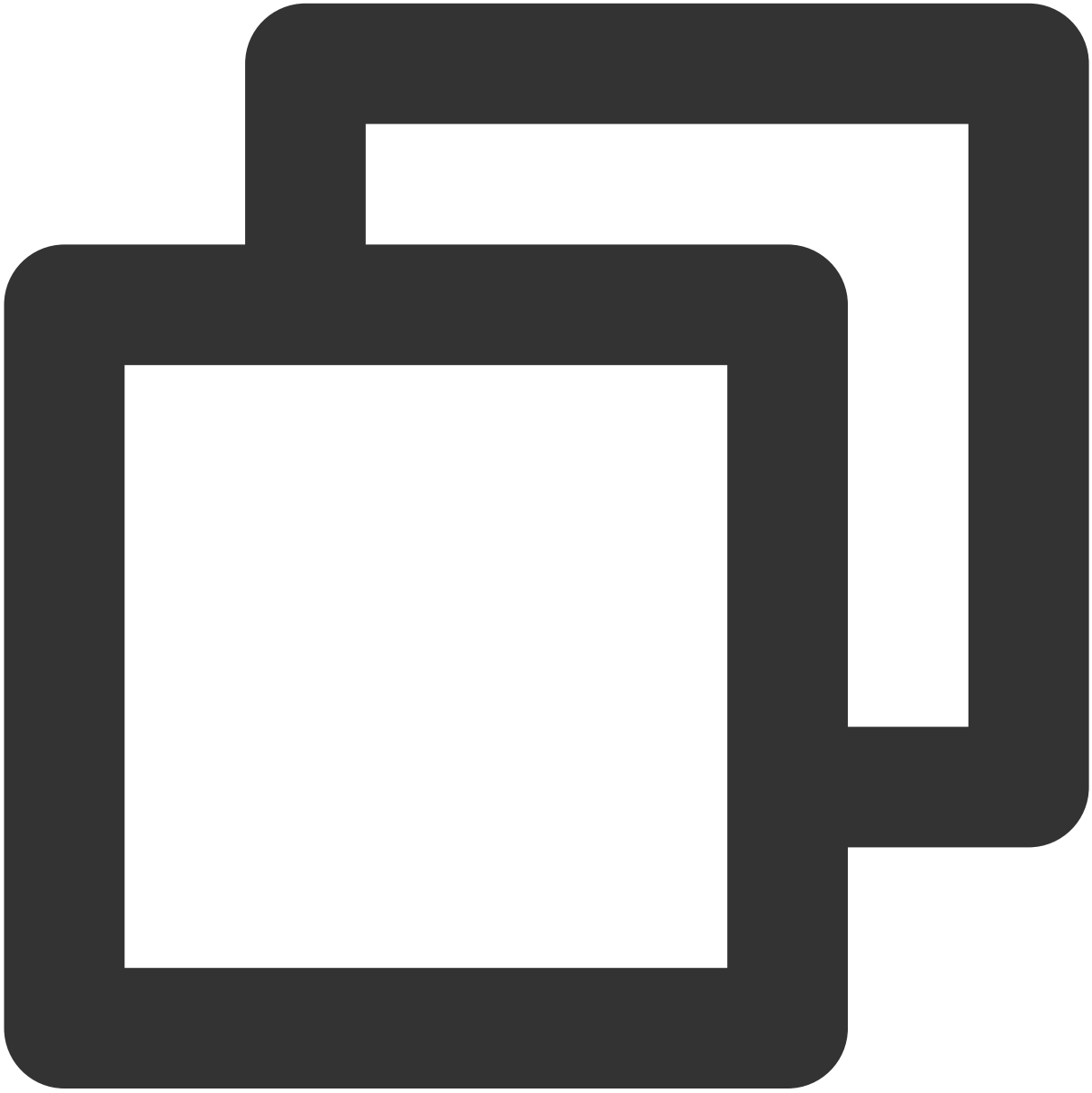
只写输出表定义：



```
CREATE WRITABLE EXTERNAL TABLE table_name
( column_name data_type [, ...] | LIKE other_table )
LOCATION (cos_ext_params)
FORMAT 'TEXT'
          [( [DELIMITER [AS] 'delimiter']
          [NULL [AS] 'null string']
          [ESCAPE [AS] 'escape' | 'OFF'] )]
      | 'CSV'
          [([QUOTE [AS] 'quote']
          [DELIMITER [AS] 'delimiter']
          [NULL [AS] 'null string']
```

```
          [FORCE QUOTE column [, ...] ]
          [ESCAPE [AS] 'escape'] )]
  [ ENCODING 'encoding' ]
  [ DISTRIBUTED BY (column, [ ... ] ) | DISTRIBUTED RANDOMLY ]
```

cos_ext_params 说明：



```
cos://cos_endpoint/bucket/prefix secretId=id secretKey=key compressType=[none|gzip]
```

参数说明

参数	格式	必填	说明
URL	COS V4：`cos://cos.{REGION}.myqcloud.com/{BUCKET}/{PREFIX}` COS V5：`cos:// {BUCKET}-{APPID}.cos.{REGION}.myqcloud.com/{PREFIX}`	是	参见 URL 参数说明
secretId	无	是	访问 API 使用的密钥 ID，参见 API 密钥管理
secretKey	无	是	访问 API 使用的密钥 Key，参见 API 密钥管理
HTTPS	true &lota; false	否	是否使用 HTTPS 访问 COS，默认为 true
compressType	gzip	否	COS 文件是否压缩，默认为空，不压缩

URL 参数说明

REGION：COS 支持的地域，需要和实例在相同地域，可选值参见 地域和访问域名。

BUCKET：COS 存储桶名称。可参见 存储桶列表，**此处名称为不包含 APPID 的名称**，如您在存储桶列表中看到存储桶名称为“test-123123123”，此处填写“test”即可。

PREFIX：COS 对象名称前缀。prefix 可以为空，可以包括多个斜杠。

在只读表场景下，prefix 指定需要读取的对象名前缀。

prefix 为空时，读取 bucket 下所有文件；prefix 以斜杠(/) 结尾时，则匹配该文件夹下面的所有文件及子文件夹中的文件；否则，读取前缀匹配的所有文件夹及子文件夹中的文件。例如，COS 对象包括：read-bucket/simple/a.csv、read-bucket/simple/b.csv、read-bucket/simple/dir/c.csv、read-bucket/simple_prefix/d.csv。

prefix 指定 simple 则读取所有文件，包括目录名称前缀匹配的 simple_prefix，对象列表：

read-bucket/simple/a.csv

read-bucket/simple/b.csv

read-bucket/simple/dir/c.csv

read-bucket/simple_prefix/d.csv

prefix 指定 simple/ 则读取包括 simple/ 的所有文件，包括：

read-bucket/simple/a.csv

read-bucket/simple/b.csv

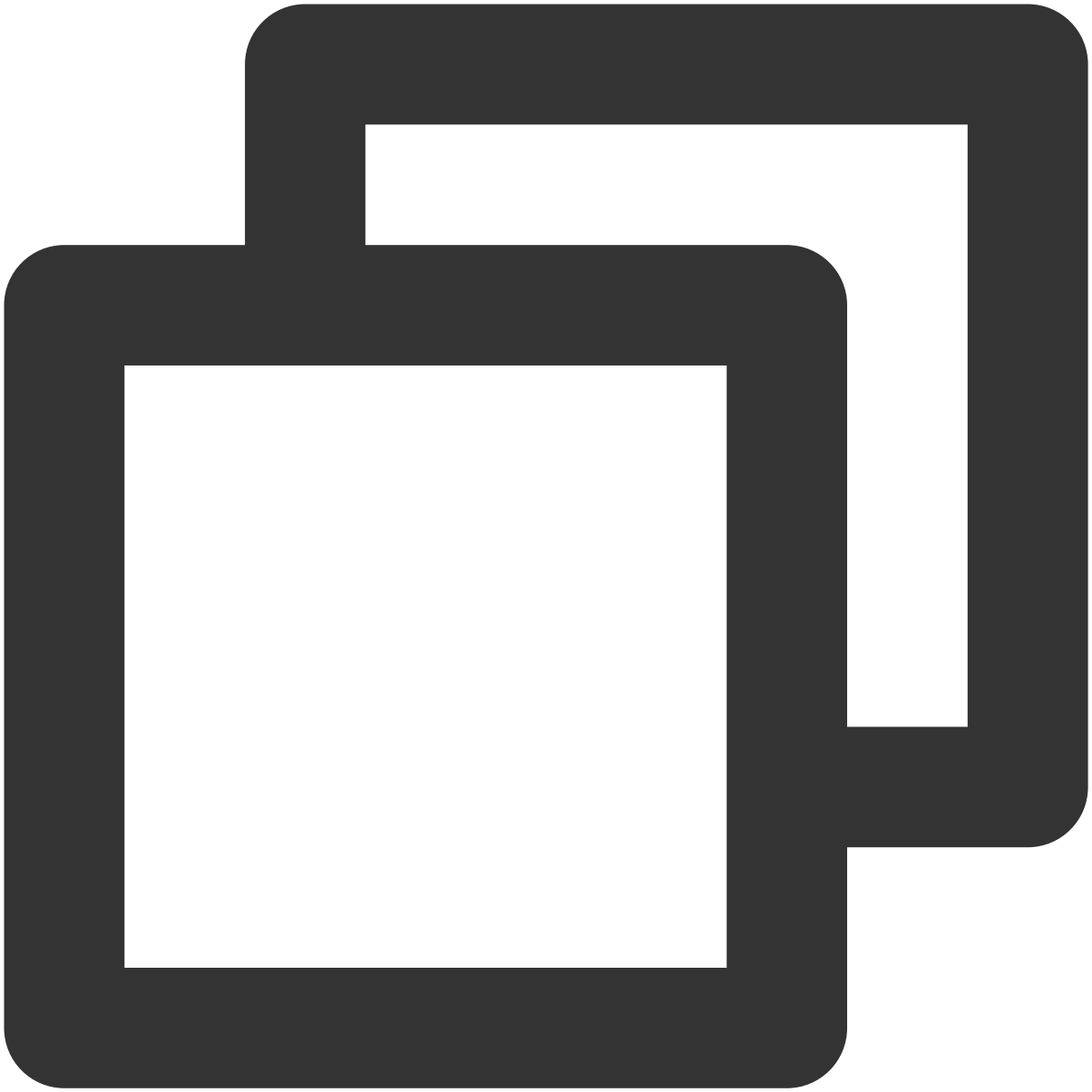
read-bucket/simple/dir/c.csv

在只写表场景下，prefix 指定输出文件前缀。
不指定 prefix 时，文件写入到 bucket 下；prefix 以斜杠（/）结尾时，文件写入到 prefix 指定的目录下，否则以给定的 prefix 作为文件前缀。例如，需要创建的文件包括：a.csv、b.csv、c.csv。
指定 prefix 为 simple/ 则生成的对象为：
read-bucket/simple/a.csv
read-bucket/simple/b.csv
read-bucket/simple/b.csv
指定 prefix 为 simple_，则生成的对象为：
read-bucket/simple_a.csv
read-bucket/simple_b.csv
read-bucket/simple_b.csv

使用示例

导入 COS 数据

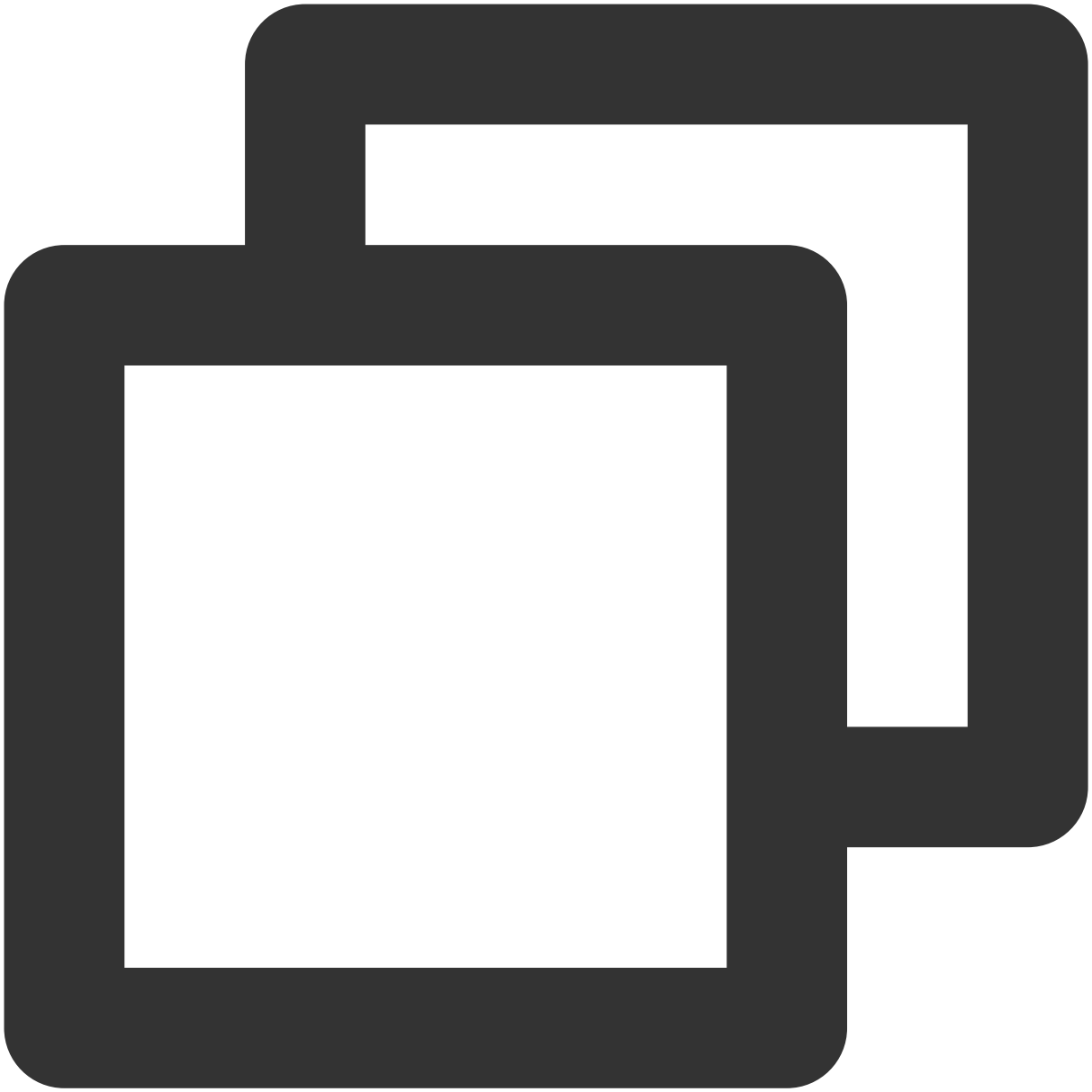
1. 定义 COS 扩展。



```
CREATE EXTENSION IF NOT EXISTS cos_ext SCHEMA public;
```

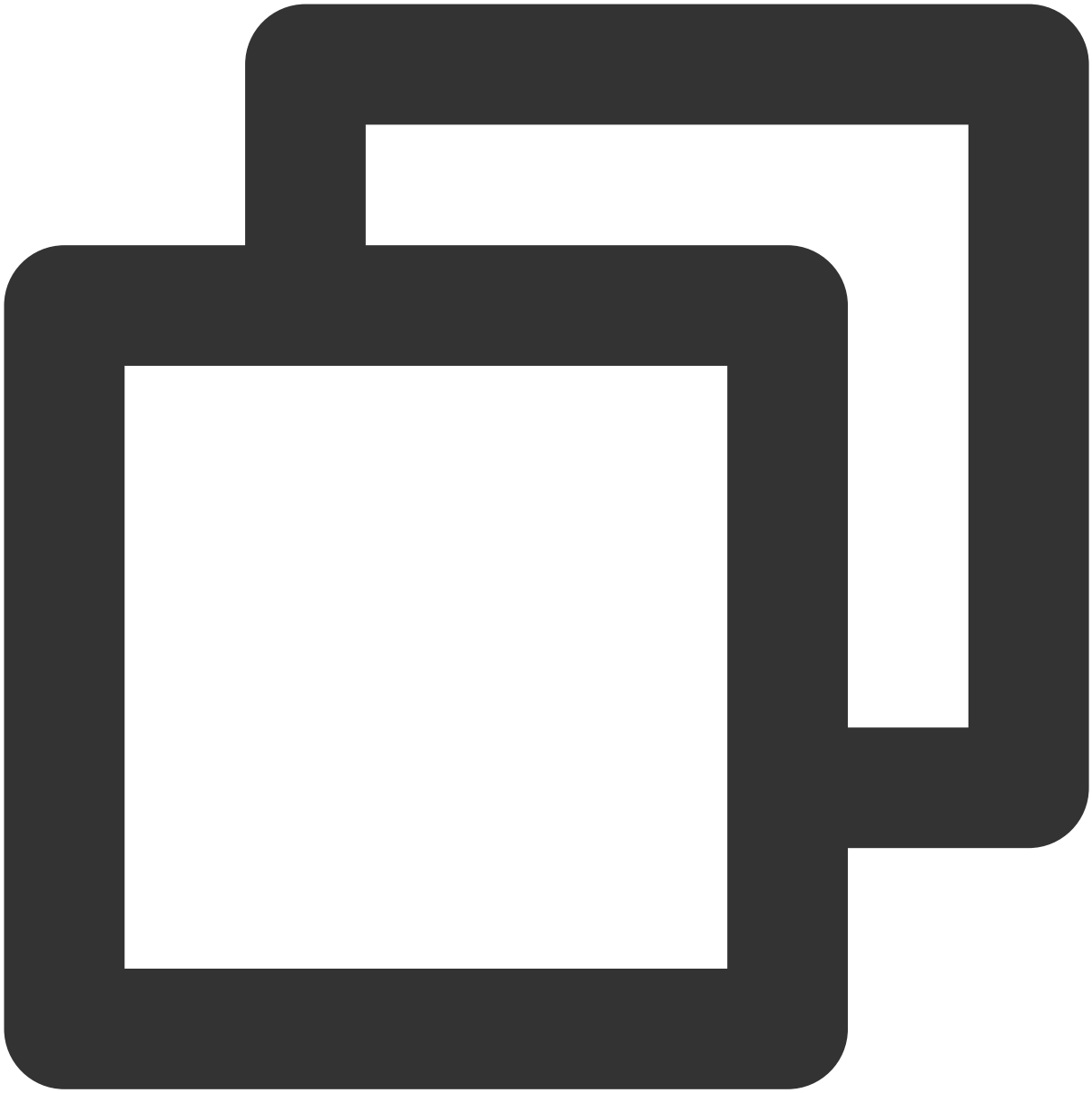
2. 定义只读 COS 外表和本地表。

本地表：



```
CREATE TABLE cos_local_tbl (c1 int, c2 text, c3 int)
DISTRIBUTED BY (c1);
```

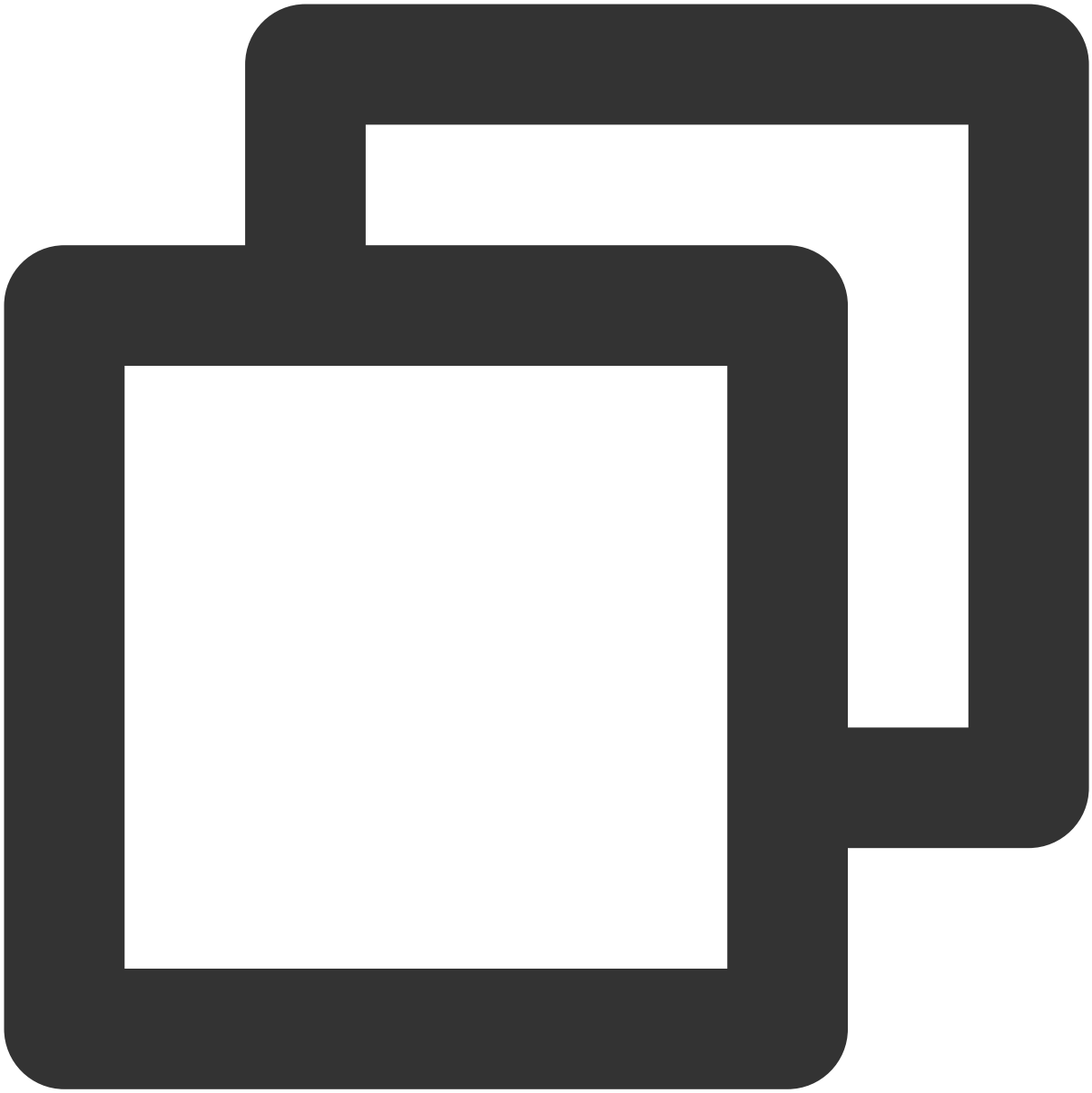
COS 外表：指定读取广州 simple-bucket 下的所有文件。



```
CREATE READABLE EXTERNAL TABLE cos_tbl (c1 int, c2 text, c3 int)
LOCATION('cos://cos.ap-guangzhou.myqcloud.com/simple-bucket/from_cos/ secretKey=xxx
FORMAT 'csv';
```

3. 准备本地表数据。

将文件上传到 simple-bucket 下 from_cos 目录下，文件内容：

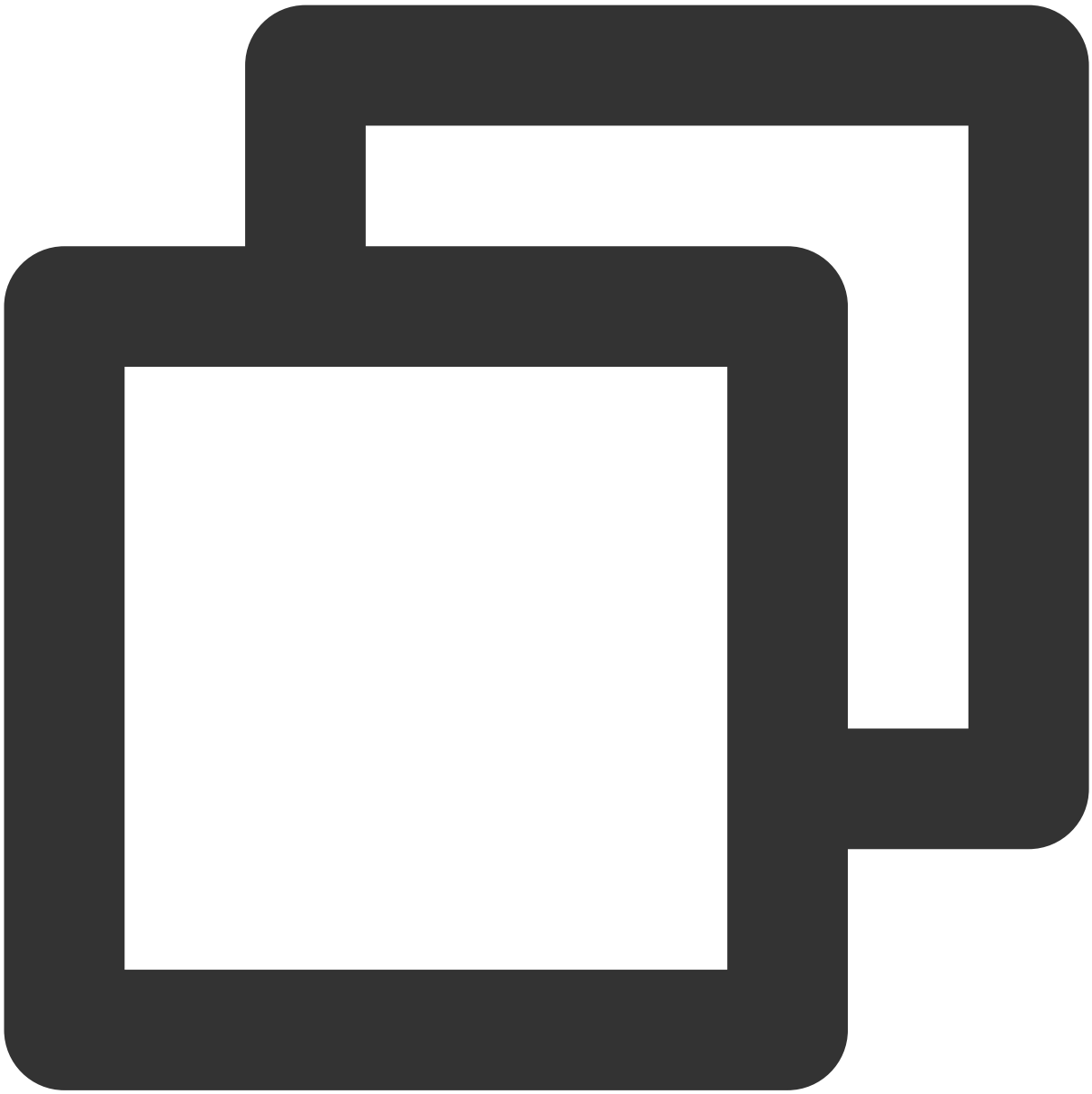


```
1,simple line 1,1
2,simple line 1,1
3,simple line 1,1
4,simple line 1,1
5,simple line 1,1
6,simple line 2,1
7,simple line 2,1
8,simple line 2,1
9,simple line 2,1
```

注意：

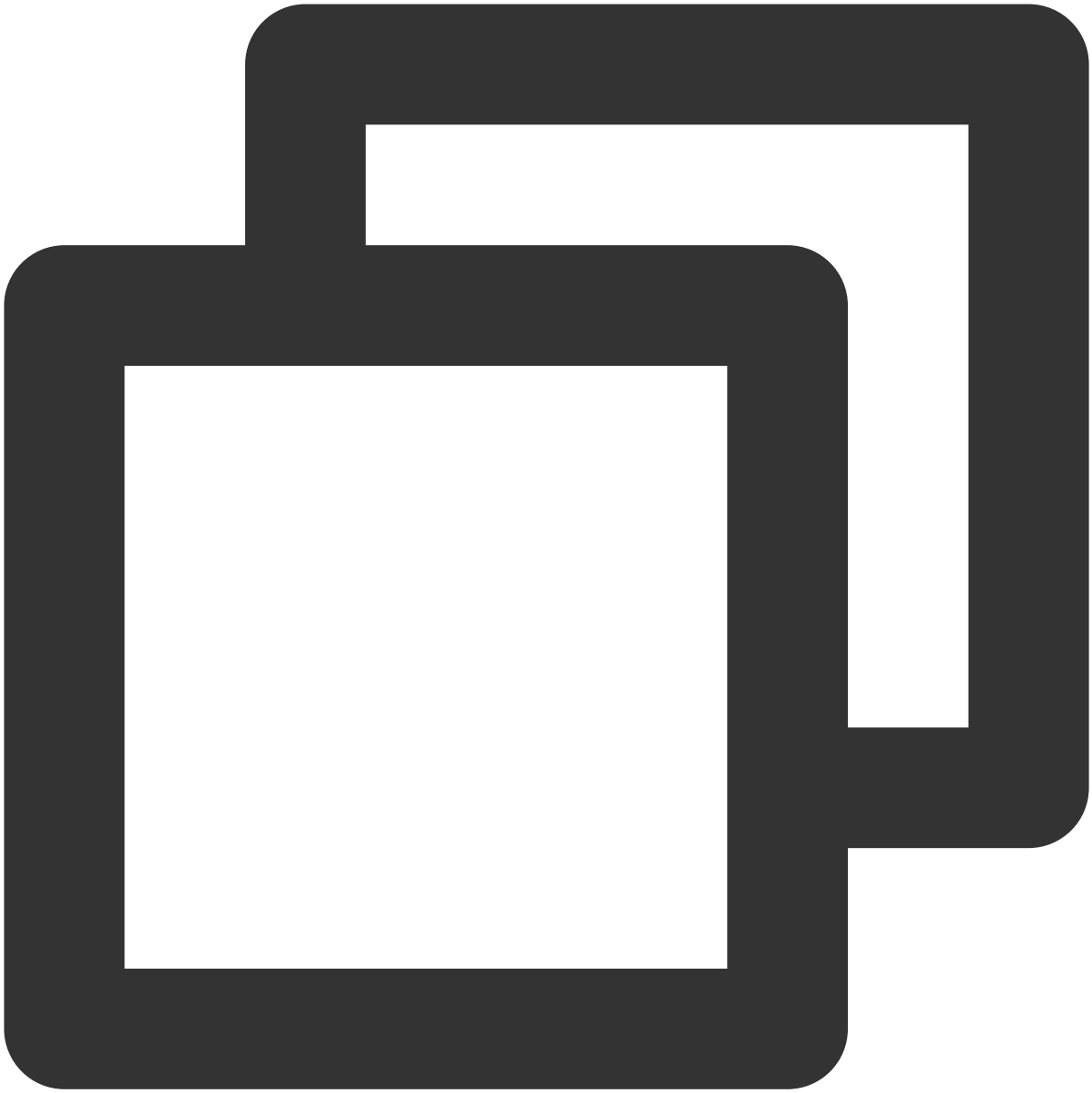
导入数据不包含表头字段行。

4. 导入 COS 数据。



```
INSERT INTO cos_local_tbl SELECT * FROM cos_tbl;
```

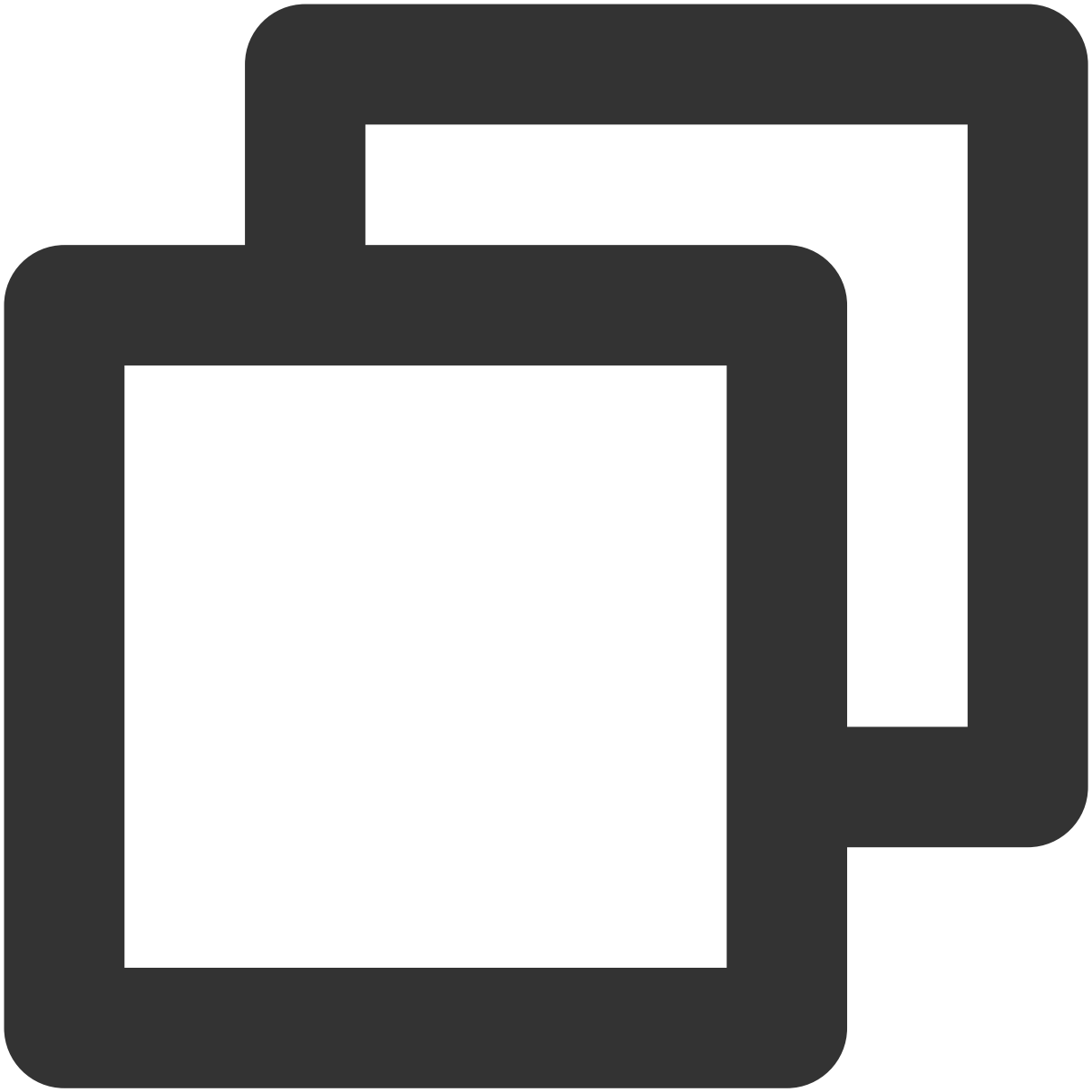
5. 查看结果，对比数据是否一致。



```
SELECT count(1) FROM cos_local_tbl;
SELECT count(1) FROM cos_tbl;
```

数据导出到 COS

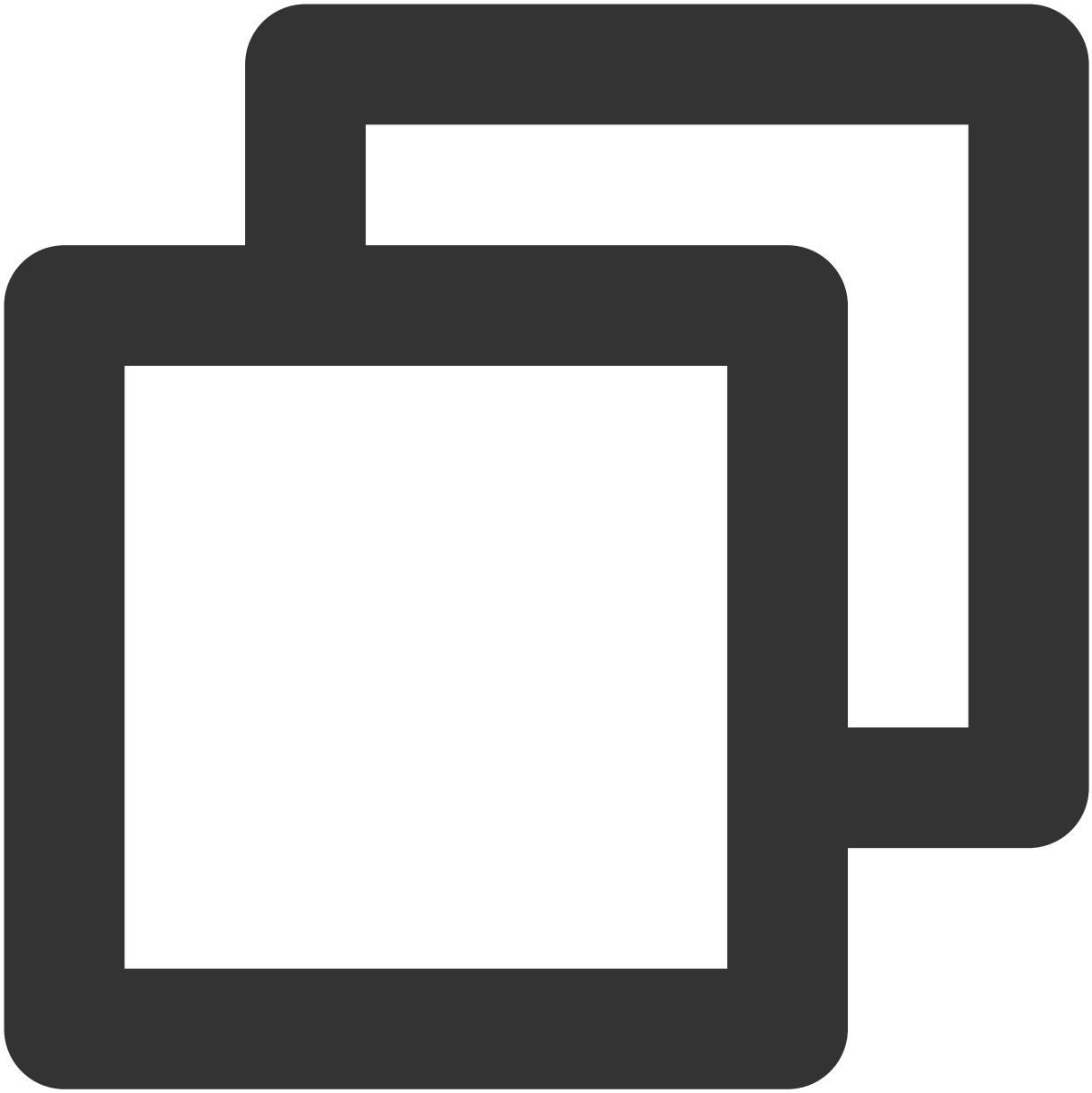
1. 定义 COS 扩展。



```
CREATE EXTENSION IF NOT EXISTS cos_ext SCHEMA public;
```

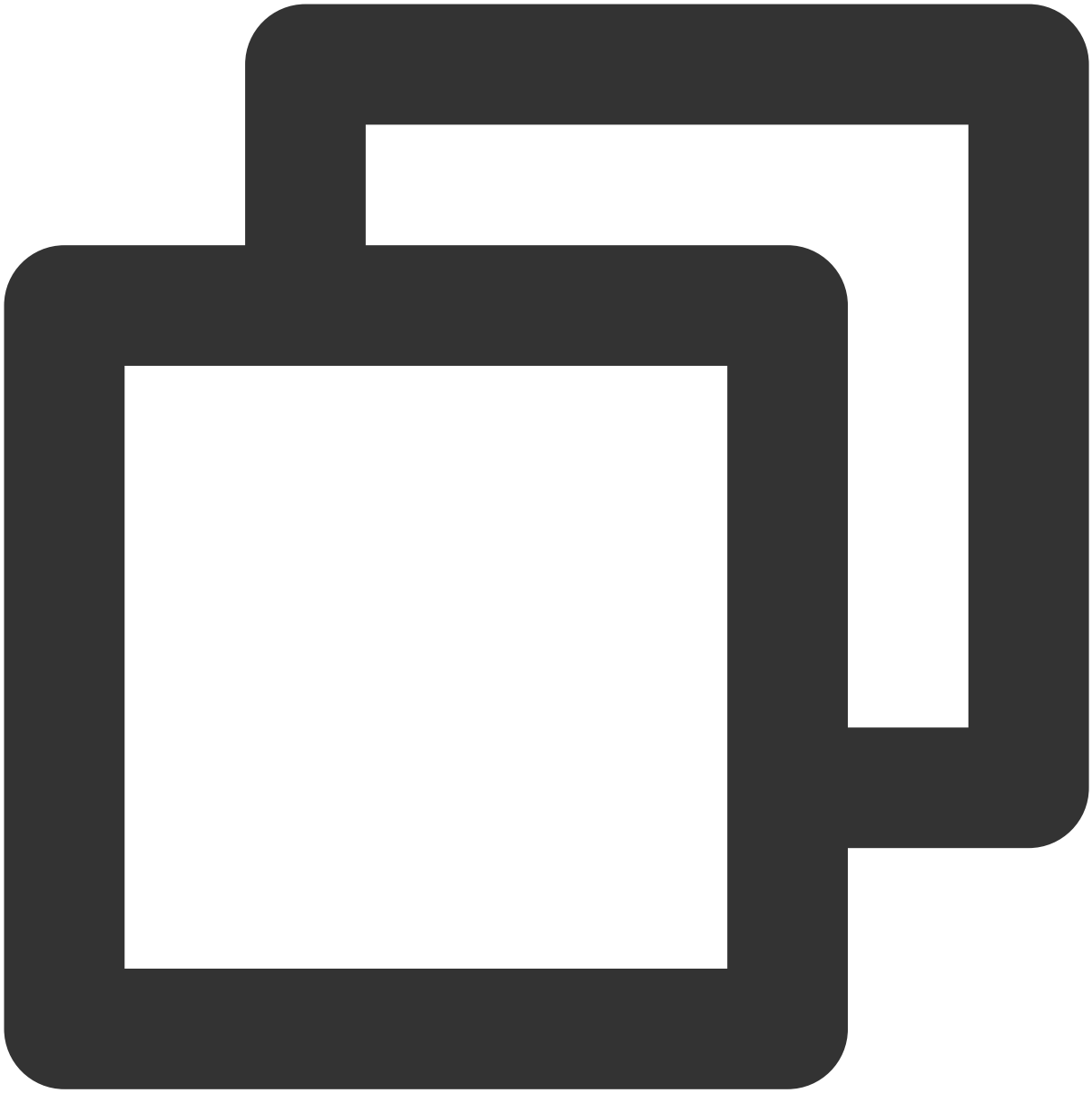
2. 定义只写 COS 外表。

本地表：



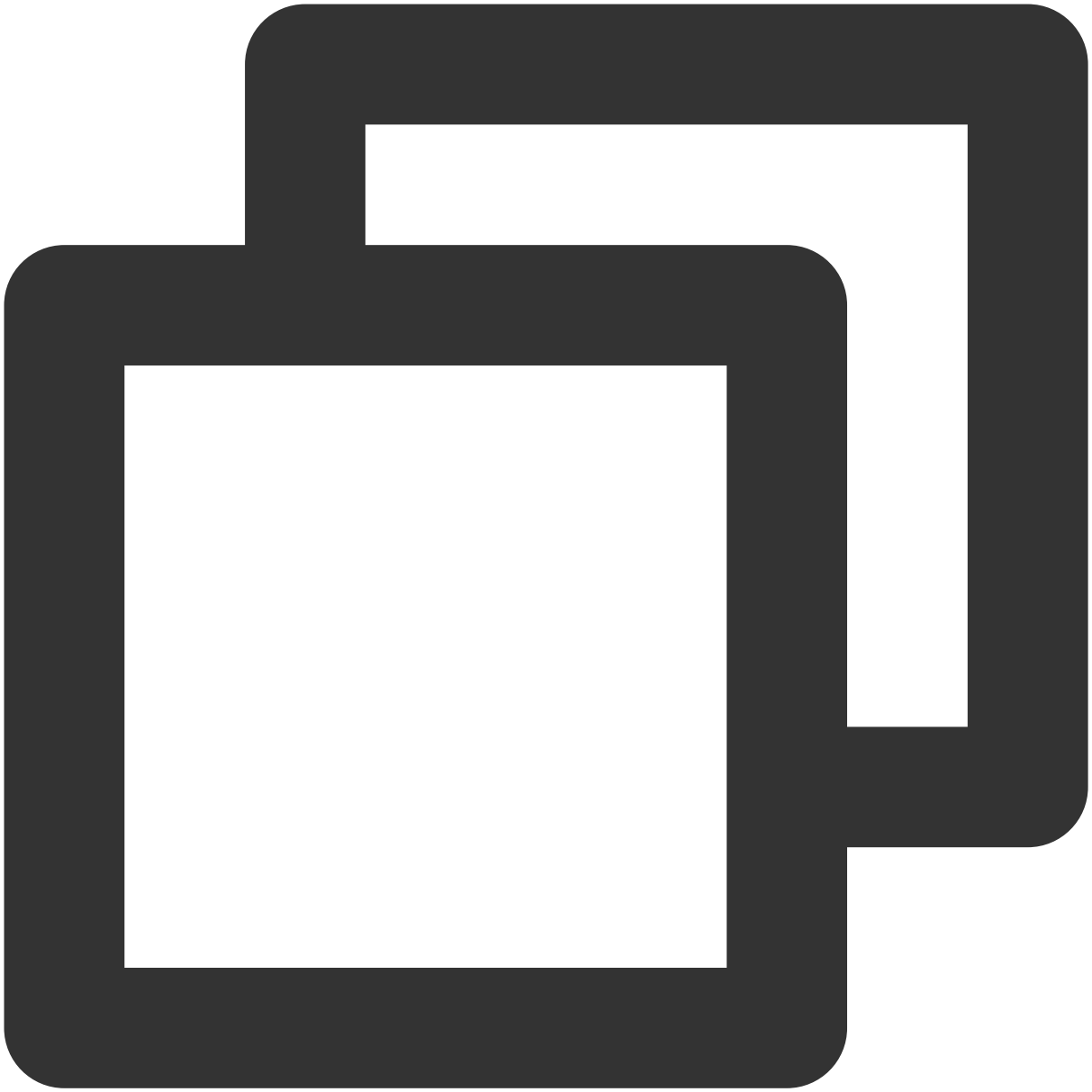
```
CREATE TABLE cos_local_tbl (c1 int, c2 text, c3 int)
DISTRIBUTED BY (c1);
```

COS 外表：指定写入广州 simple-bucket 下的所有文件。



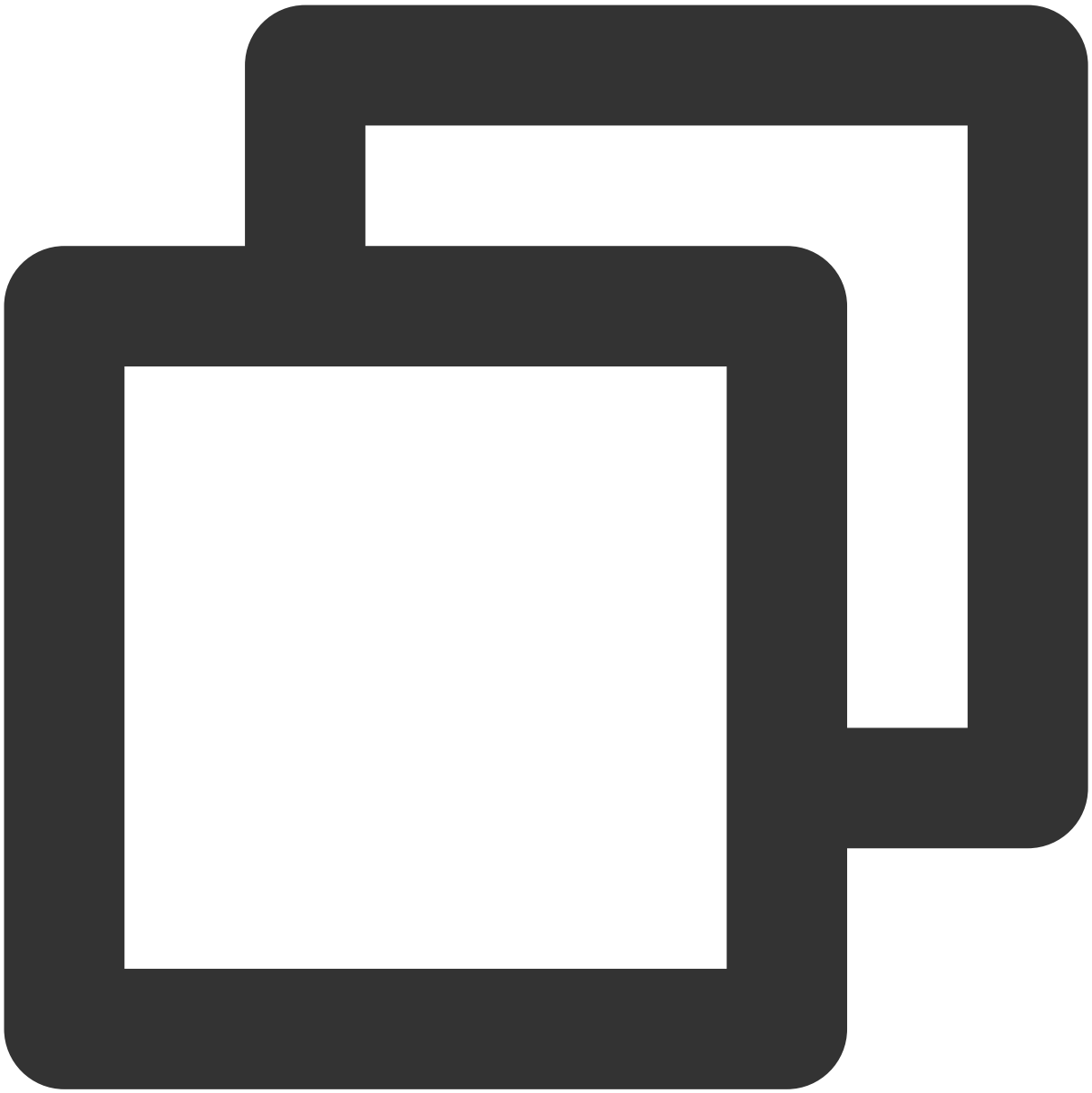
```
CREATE WRITABLE EXTERNAL TABLE cos_tbl_wr (c1 int, c2 text, c3 int)
LOCATION('cos://cos.ap-guangzhou.myqcloud.com/simple-bucket/to-cos/ secretKey=xxx s
FORMAT 'csv';
```

3. 构造测试数据。



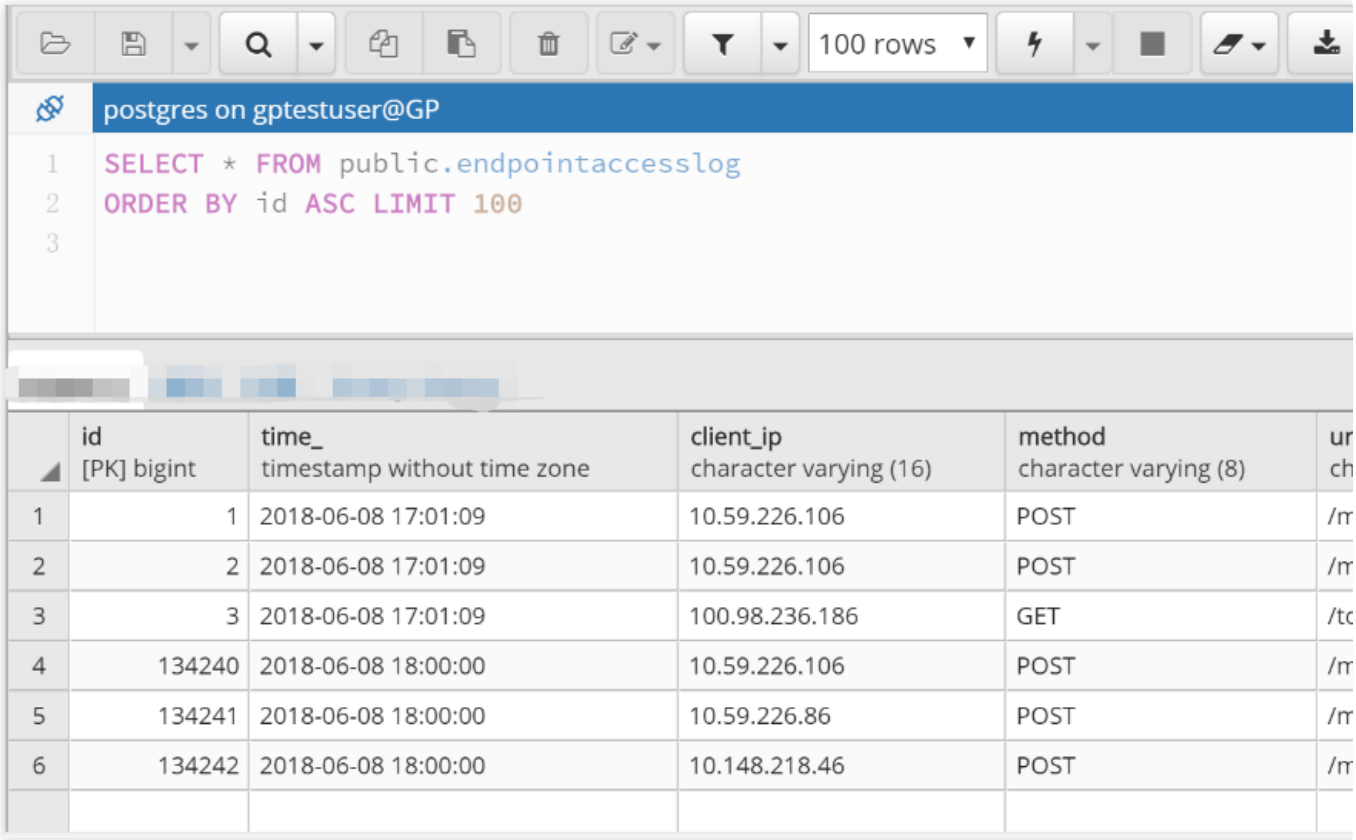
```
insert into cos_local_tbl values
(1, 'simple line 1' , 1),
(2, 'simple line 2', 2),
(3, 'simple line 3', 3) ,
(4, 'simple line 4', 4) ,
(5, 'simple line 5', 5) ,
(6, 'simple line 6', 6) ,
(7, 'simple line 7', 7) ,
(8, 'simple line 8', 8) ,
(9, 'simple line 9', 9);
```

4. 导出数据到 COS。



```
INSERT INTO cos_tbl_wr SELECT * FROM cos_local_tbl;
```

5. 查看结果。



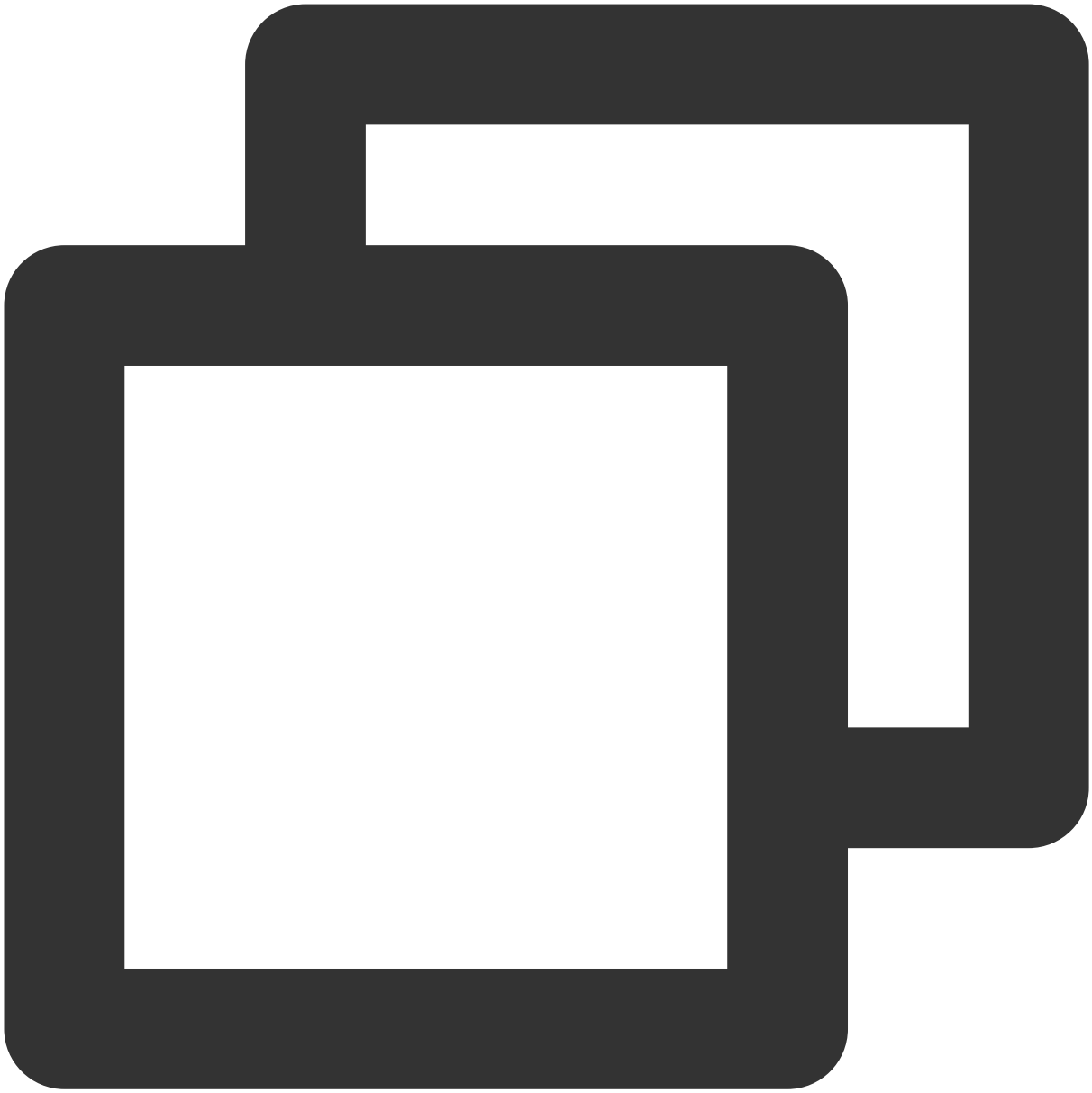
	id [PK] bigint	time_ timestamp without time zone	client_ip character varying (16)	method character varying (8)	ur ch
1	1	2018-06-08 17:01:09	10.59.226.106	POST	/m
2	2	2018-06-08 17:01:09	10.59.226.106	POST	/m
3	3	2018-06-08 17:01:09	100.98.236.186	GET	/to
4	134240	2018-06-08 18:00:00	10.59.226.106	POST	/m
5	134241	2018-06-08 18:00:00	10.59.226.86	POST	/m
6	134242	2018-06-08 18:00:00	10.148.218.46	POST	/m

简单分析 COS 数据

注意：

使用 COS 外表做查询分析时，未进行查询优化，复杂查询建议先将数据导入到本地。

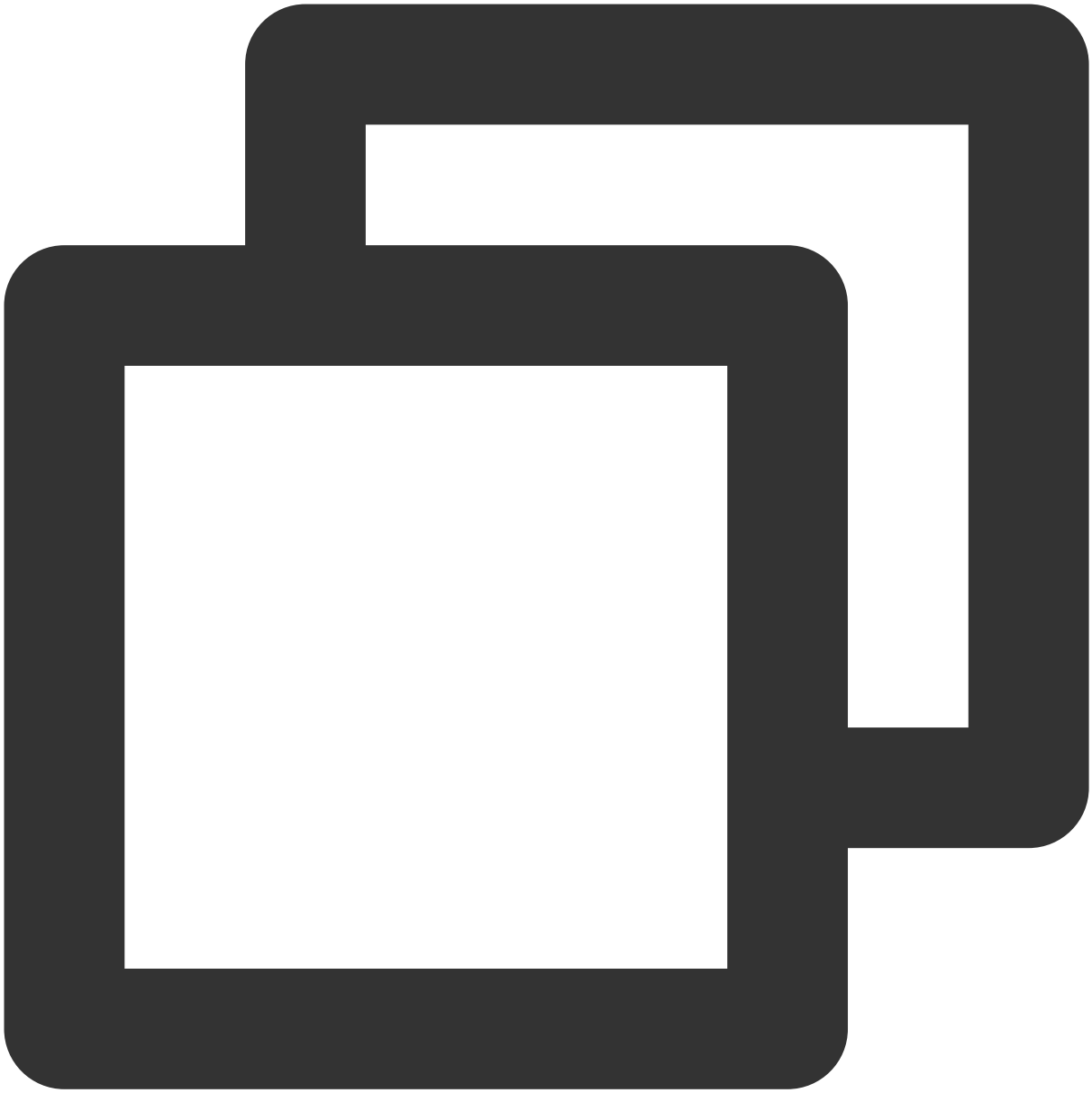
1. 定义 COS 扩展。



```
CREATE EXTENSION IF NOT EXISTS cos_ext SCHEMA public;
```

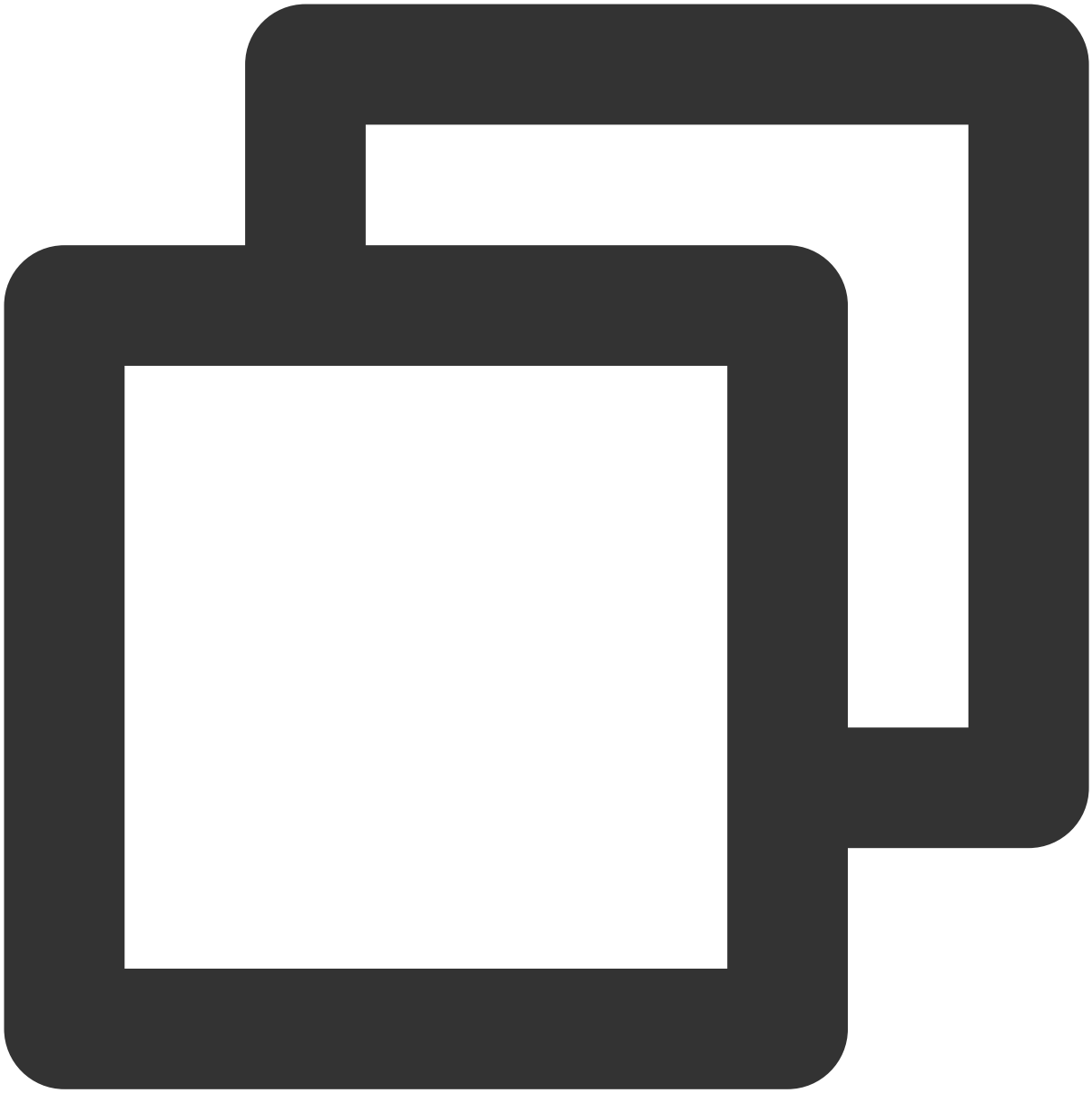
2. 准备数据。

将文件上传到 simple-bucket 的 for-dml 目录下，文件内容：



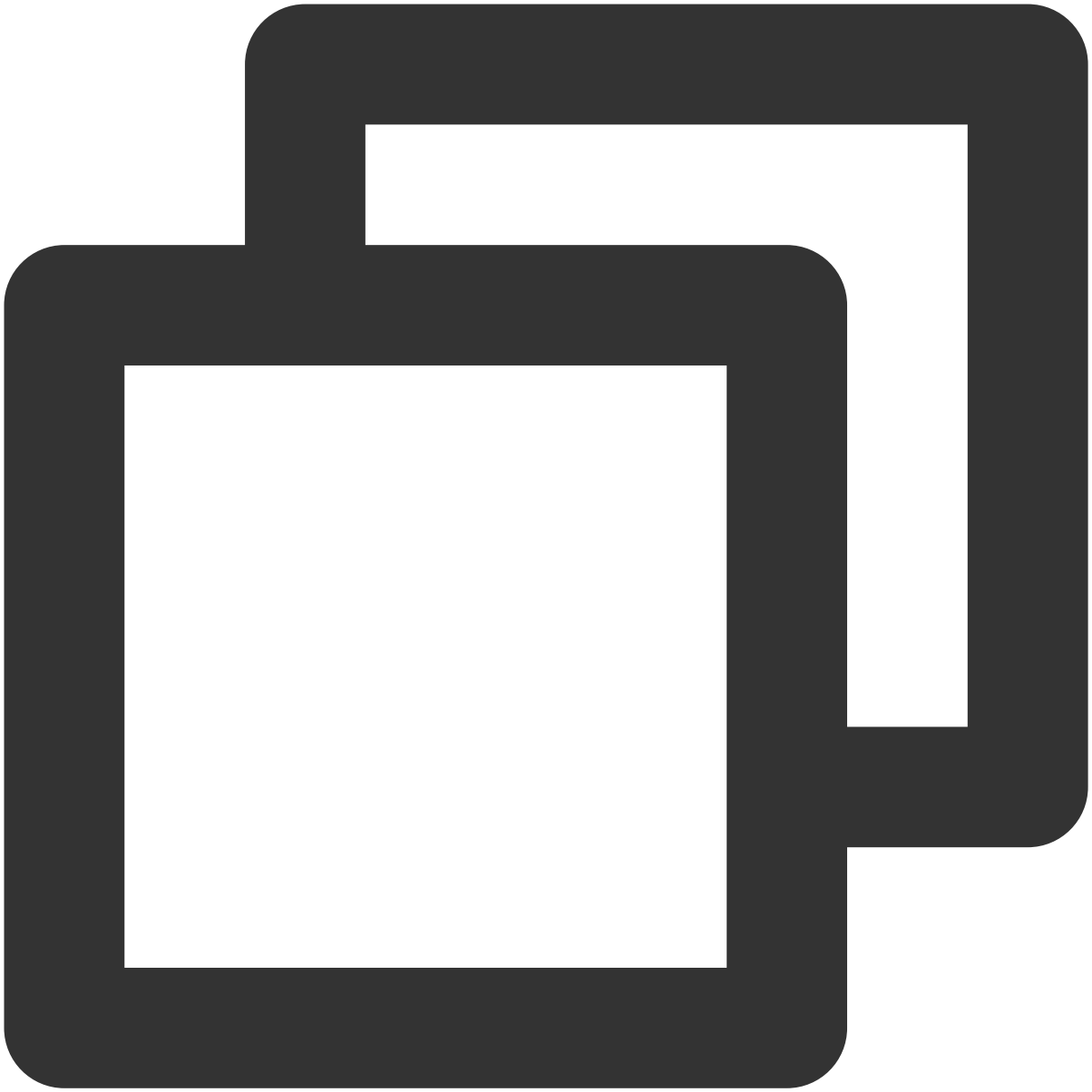
```
1,simple line 1,1
2,simple line 1,1
3,simple line 1,1
4,simple line 1,1
5,simple line 1,1
6,simple line 2,1
7,simple line 2,1
8,simple line 2,1
9,simple line 2,1
```

3. 定义只读 COS 外表。



```
CREATE READABLE EXTERNAL TABLE cos_tbl_dml (c1 int, c2 text, c3 int)
LOCATION('cos://cos.ap-guangzhou.myqcloud.com/simple-bucket/for-dml/ secretKey=xxx
FORMAT ‘csv’;
```

4. 分析 COS 外表数据。



```
SELECT c2, sum(c1) FROM cos_tbl GROUP BY c2;
```

使用外表同步 EMR 数据

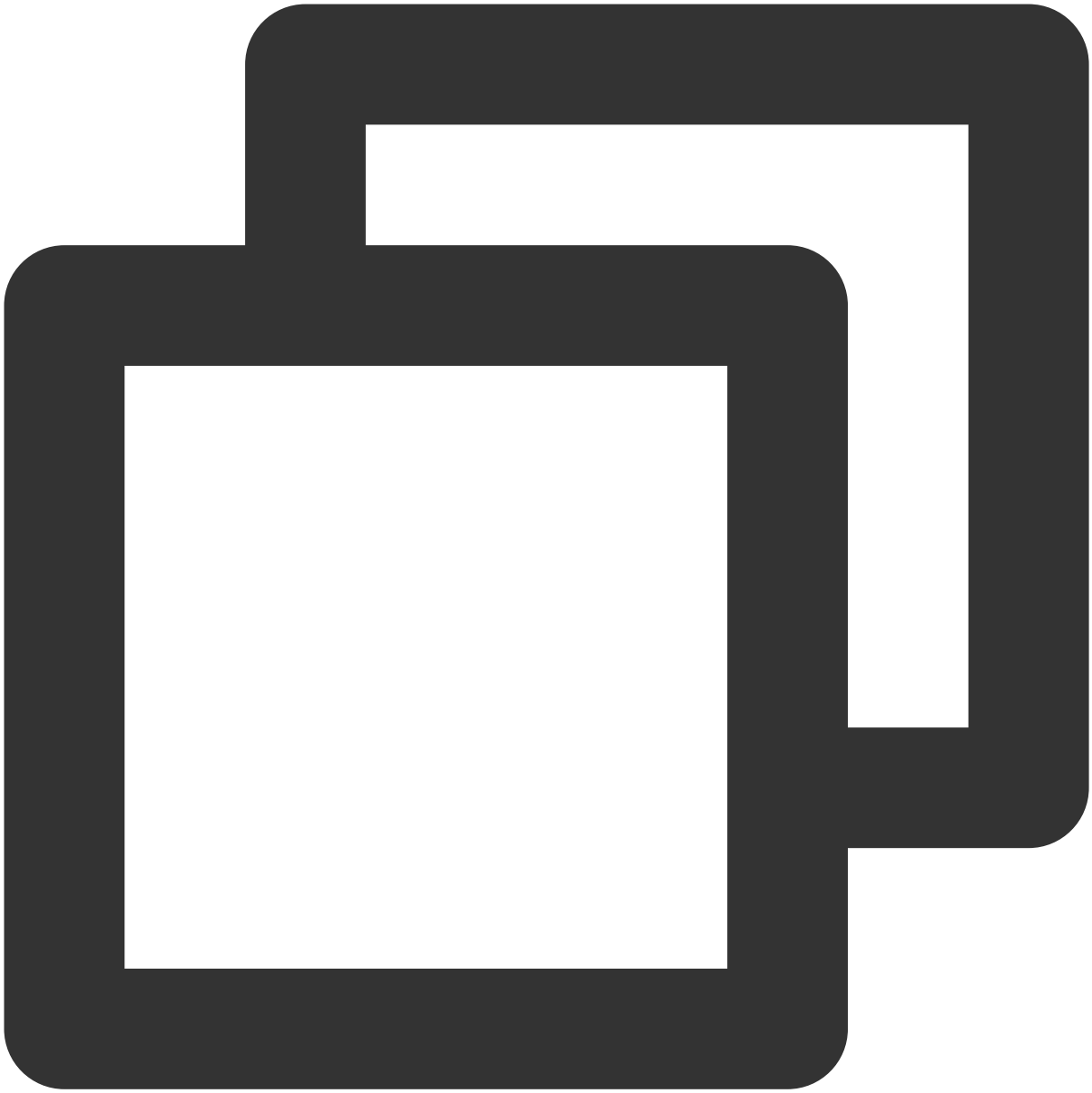
最近更新时间：2024-02-19 15:23:20

背景说明

在数据仓库的建设中，通常我们使用 Hive 处理原始数据（PB 级别），进行耗时较长的 ETL 工作，再将结果数据（TB 级别）交由准实时的计算引擎（例如云数据仓库 PostgreSQL）对接 BI 工具，保证报表的准实时展现。
本文介绍了如何将 EMR 上 Hive 的数据通过 COS 导入到云数据仓库 PostgreSQL 的过程。

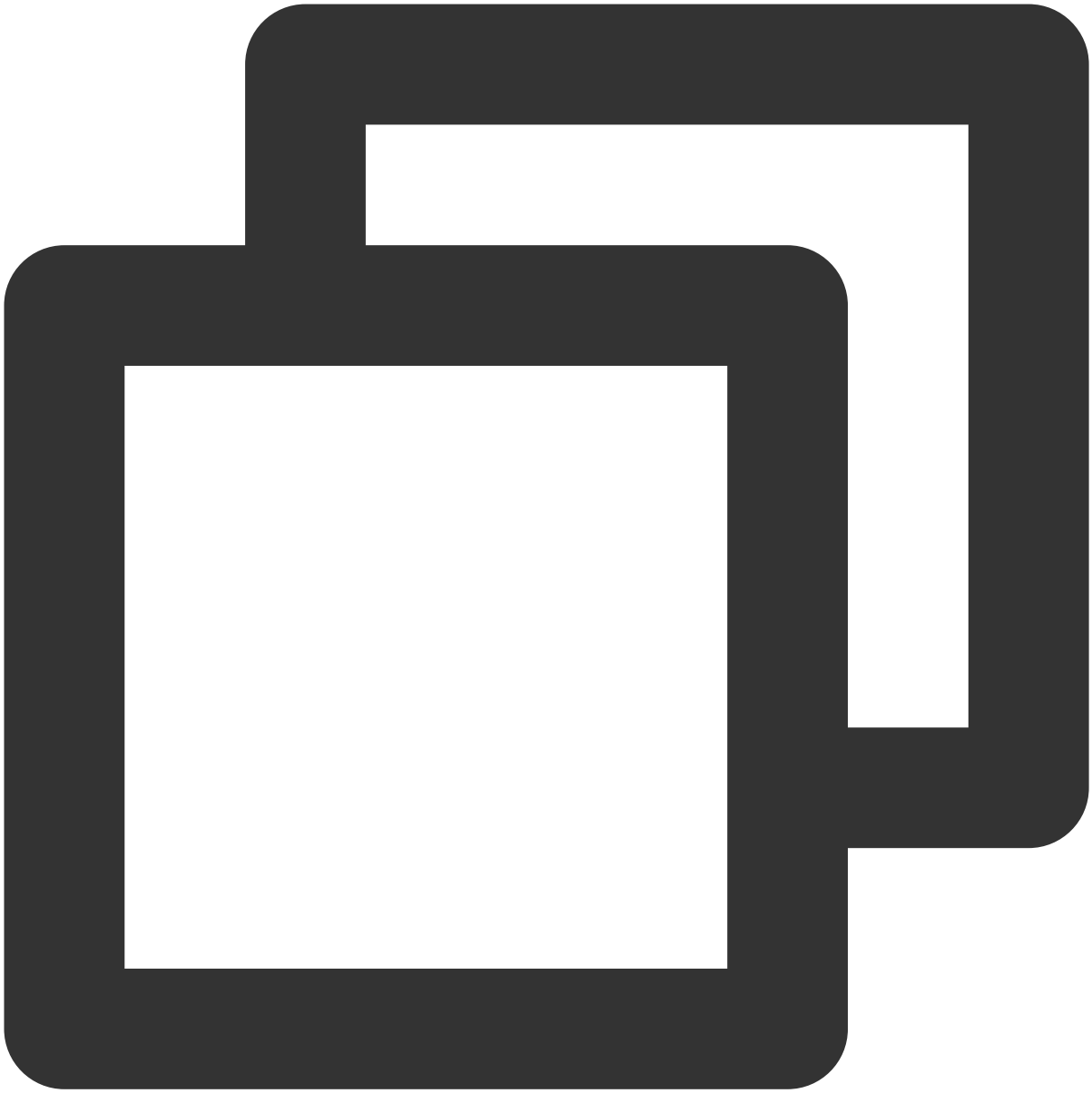
操作步骤

注意：
云数据仓库 PostgreSQL 不支持 ORC、Parquet 等格式，仅支持 CSV 等文本格式及其对应的 GZIP 压缩格式。
云数据仓库 PostgreSQL 侧导入 COS 数据的效率与文件的个数有一定关系，建议个数为云数据仓库 PostgreSQL 计算节点个数的 N 倍。
1. 开启 EMR 读写对象存储能力
首先需要保证 EMR 具备读写 COS 的能力，可在创建 EMR 时，勾选**开启**对象存储。
2. 创建 Hive 本地表并写入数据



```
create table hive_local_table(c1 int, c2 string, c3 int, c4 string);
insert into hive_local_table values(1001, 'c2', 99, 'c4'),(1002, 'c2', 100, 'c4'),(
```

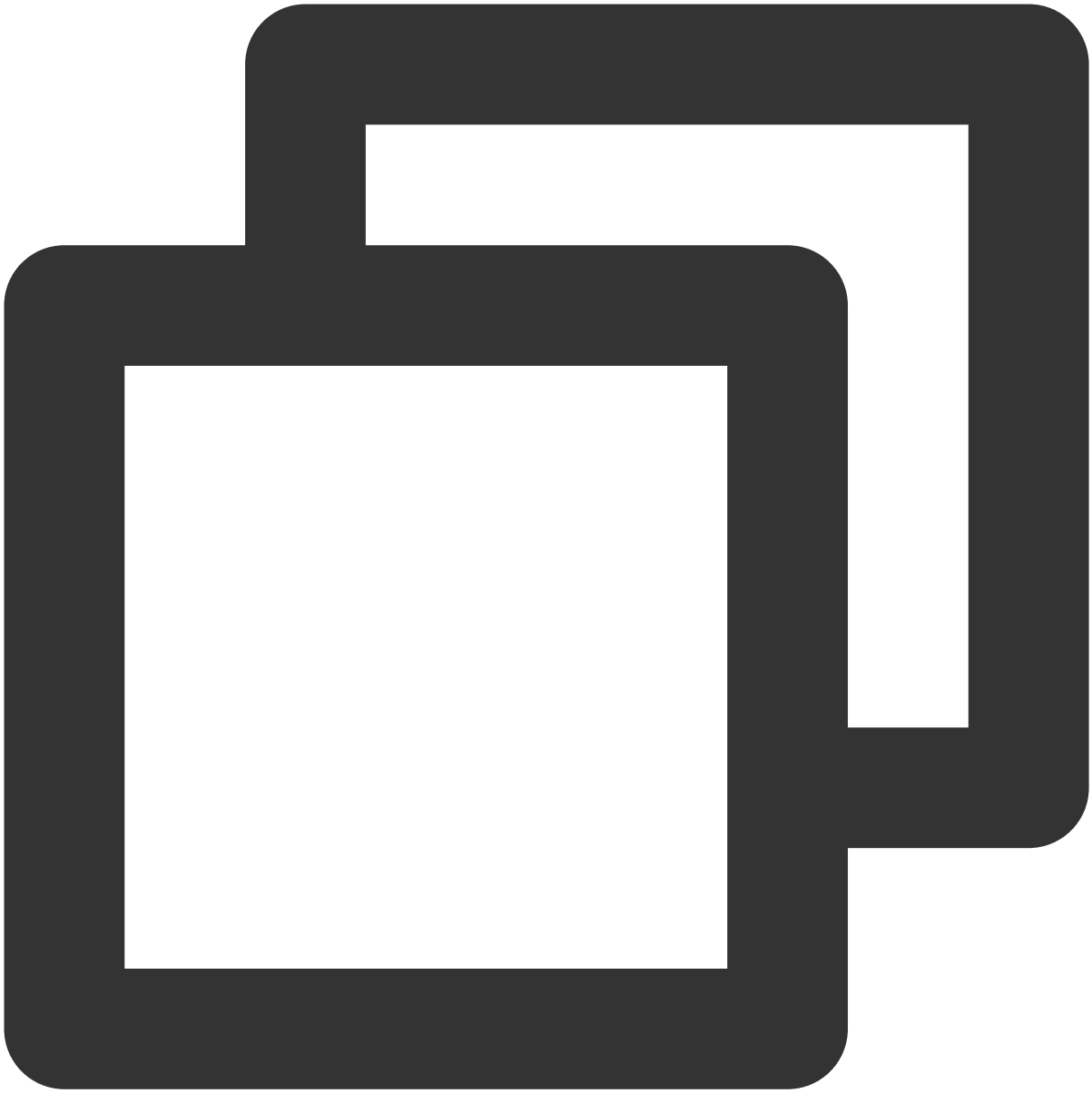
3. 创建 Hive COS 外表



```
create table hive_cos_table(c1 int, c2 string, c3 int, c4 string)
row format delimited fields terminated by ','
LINES TERMINATED BY '\\n'
stored as textfile location 'cosn://{bucket_name}/{dir_name}';
```

详细信息可以参考 EMR 文档 基于对象存储 COS 的数据仓库。

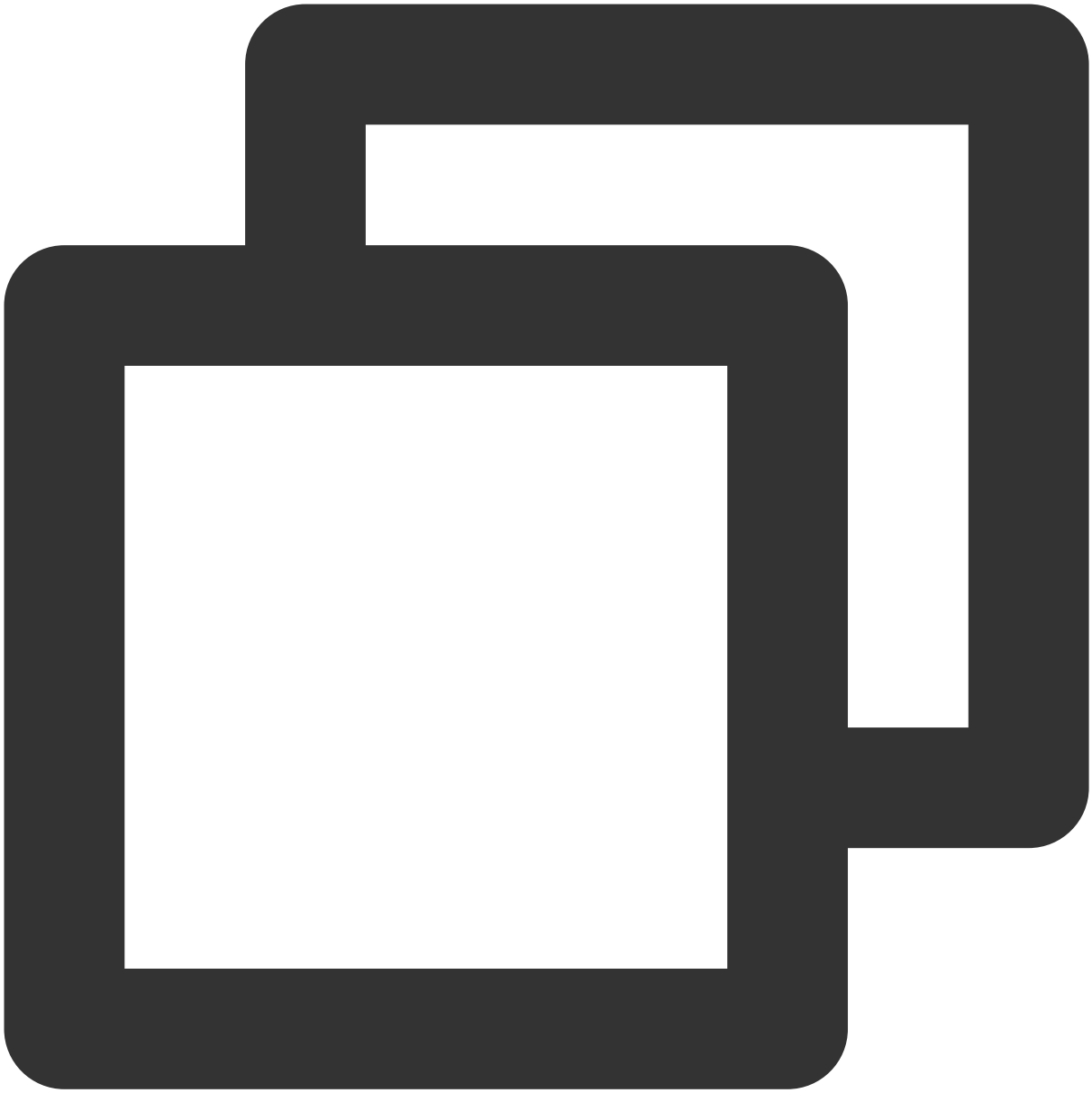
4. 将本地数据导入 COS



```
insert into hive_cos_table select * from hive_local_table;
```

成功写入后，可以在对应的 COS 目录下看到文件。

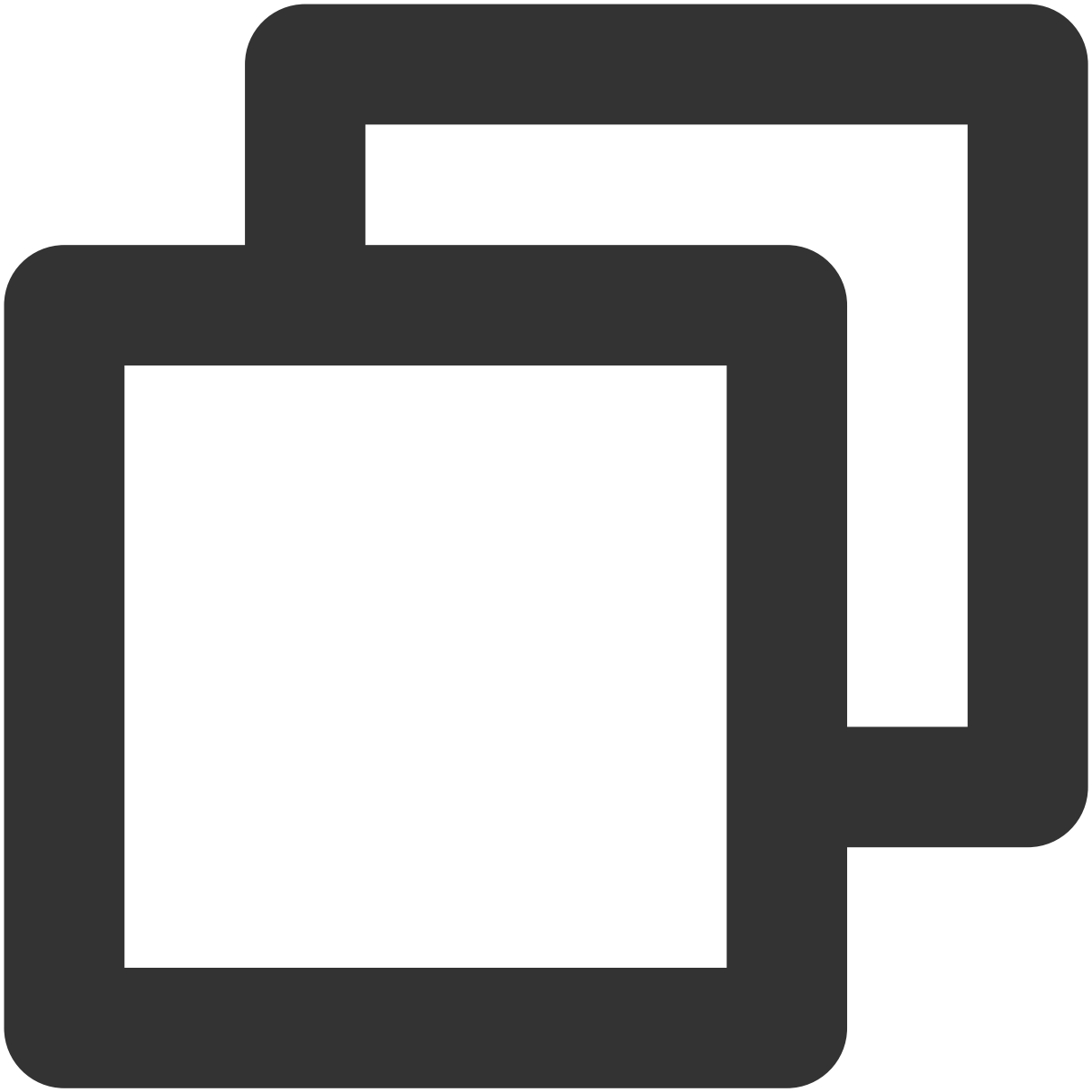
5. 在云数据仓库 PostgreSQL 侧创建 COS 外表



```
CREATE READABLE EXTERNAL TABLE  snova_cos_table (c1 int, c2 varchar(32), c3 int, c4
LOCATION('cos:// {BUCKET}-{APPID}.cos.{REGION}.myqcloud.com/{PREFIX} secretKey=****
FORMAT 'csv';
```

详细内容可以参见 使用外表高速导入或导出 COS 数据。

6. 在云数据仓库 PostgreSQL 侧创建本地表并导入数据



```
create table snova_local_table(c1 int, c2 text, c3 int, c4 text);
insert into snova_local_table select * from snova_cos_table;
```

使用 rule 规则实现云数据仓库 PostgreSQL upsert 操作

最近更新时间：2024-02-19 15:23:20

背景说明

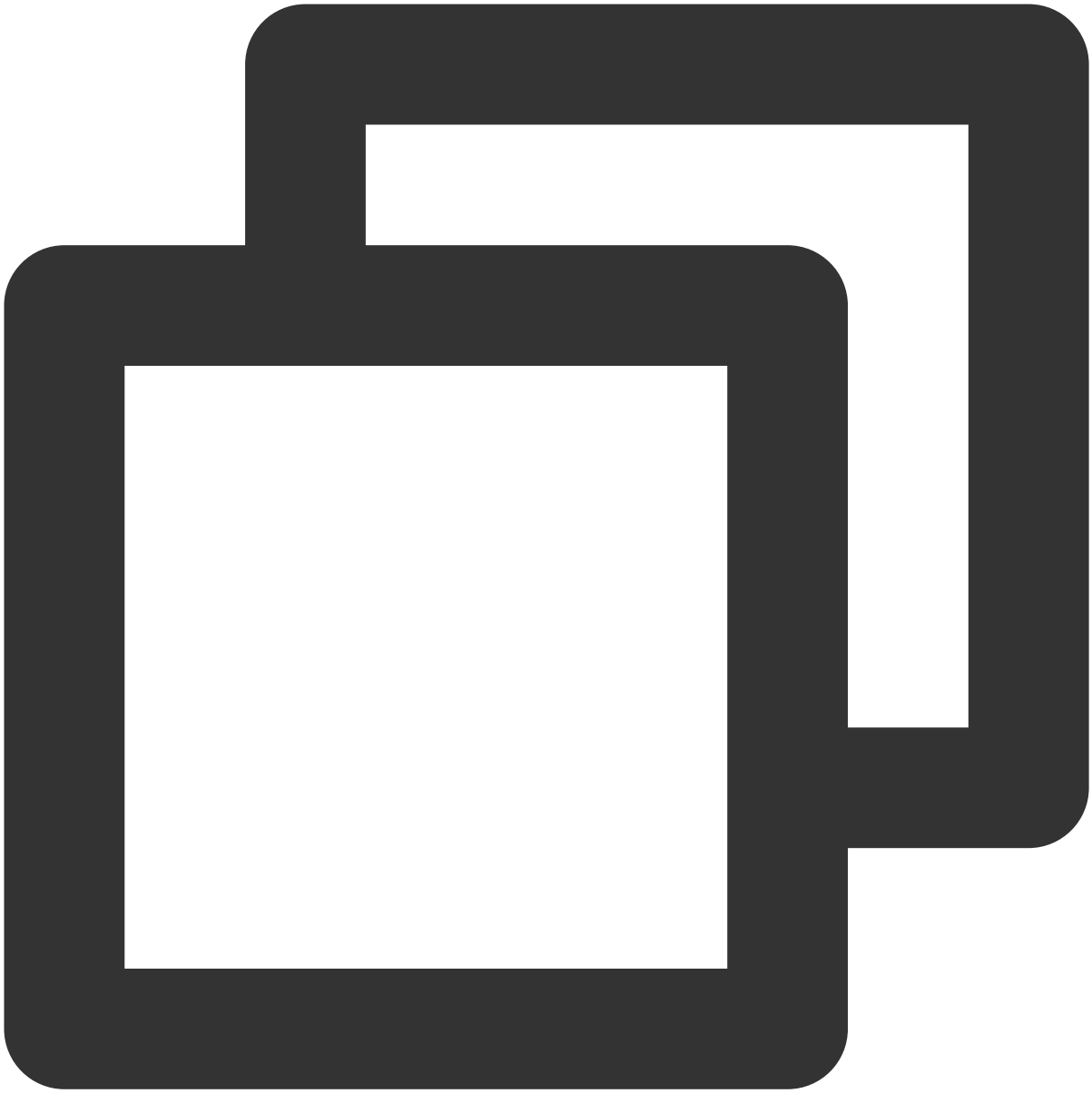
云数据仓库 PostgreSQL 底层是基于 greenplum6 来构建，postgresql 内核为9.4版本，目前并不能很好支持 postgresql 的 `insert .. on conflict` 特性，所以对于 upsert 场景需要采用额外的方式来进行处理，这里提供一种利用 postgresql rule 特性来进行 upsert 的方法。

规则介绍

PostgreSQL 规则系统允许在更新、插入、删除时执行一个其它的预定义动作。简单的说，规则就是在指定表上执行指定动作的时候，将导致一些额外的动作被执行。另外，一个 `INSTEAD` 规则可以用另外一个命令取代特定的命令，或者完全不执行该命令。规则还可以用于实现表视图。规则实际上只是一个命令转换机制，或者说命令宏。这种转换发生在命令开始执行之前。
详细信息可参考 rule 使用手册。

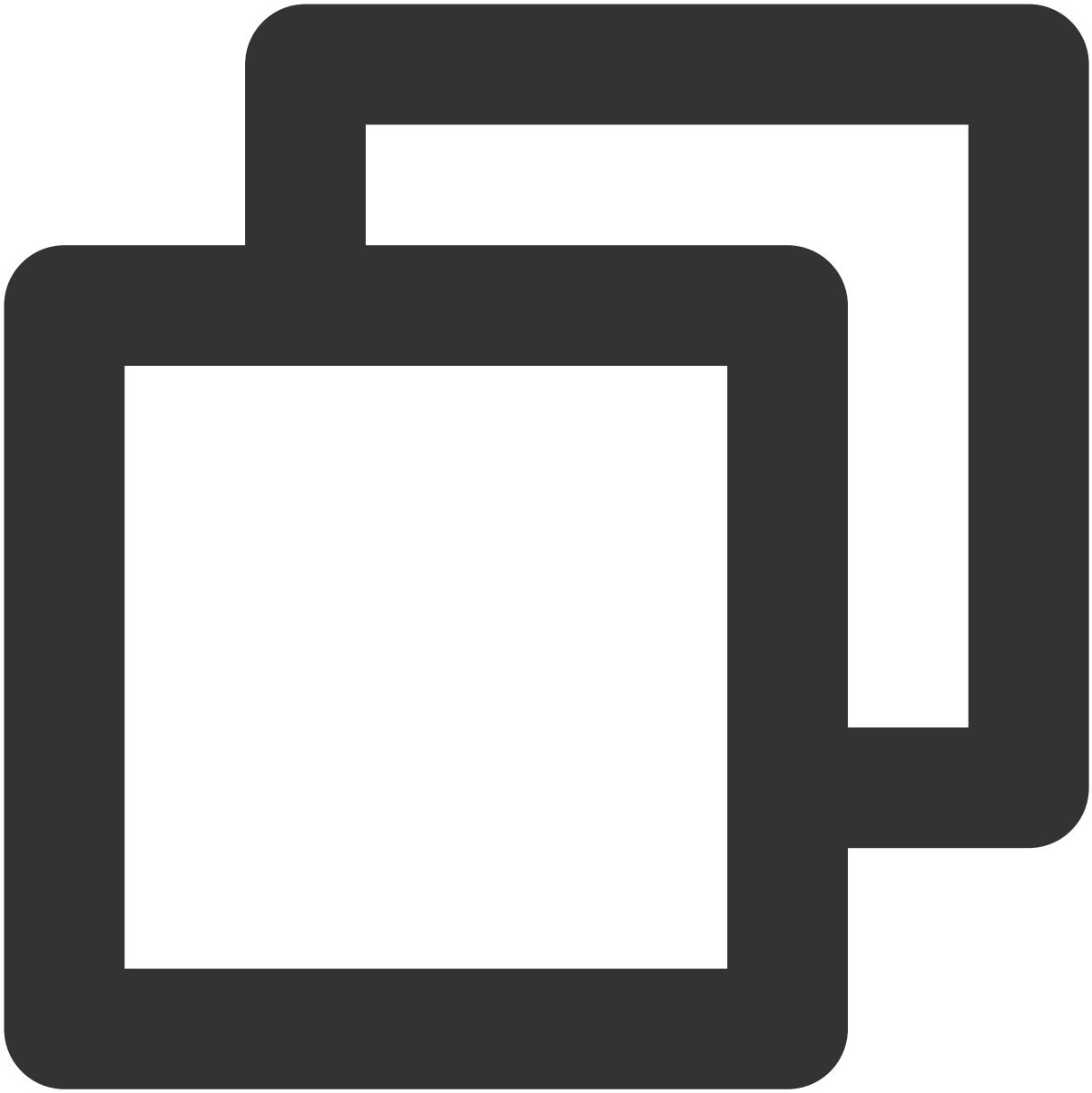
upsert rule

如果需要实现 upsert 的操作，那么需要这样一条规则：当进行 insert 操作时，判断是否已经有相应的记录，如果存在记录则改为进行 update 操作，如果不存在记录则进行正常 insert 操作。
下面以一个数据库实例来进行说明：
创建一个测试数据库。



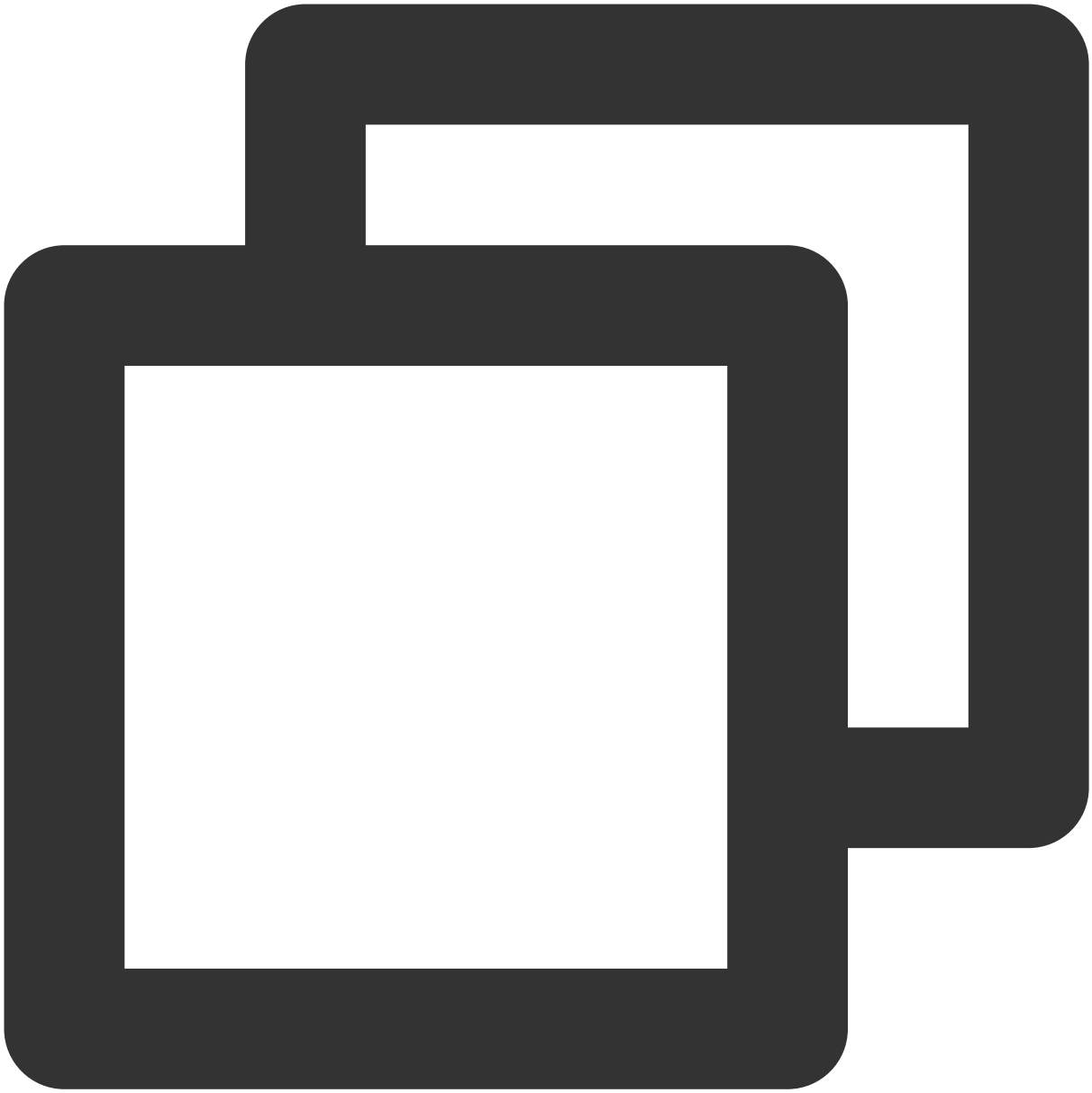
```
CREATE TABLE my_test(
   id integer,
   num1 integer,
   num2 decimal,
   str1 varchar(20),
   str2 text,
   PRIMARY KEY(id)
) distributed by (id);
```

然后给表增加 rule 规则。



```
create rule r1 as on insert to my_test where exists (select 1 from e t1 where t1.id
```

这条 rule 命令的含义就是针对 insert 操作，如果新的 insert 语句的 id 是存在，那么就直接用新 insert 里面的值 update 原来的数据，语句中的 NEW.XXX，即新 insert 语句的值，操作完成后可以看到。数据表中存在 rule 规则，接着进行 insert 操作，如果 id 存在，那么不会因为主键约束报错，而是进行 update 操作。



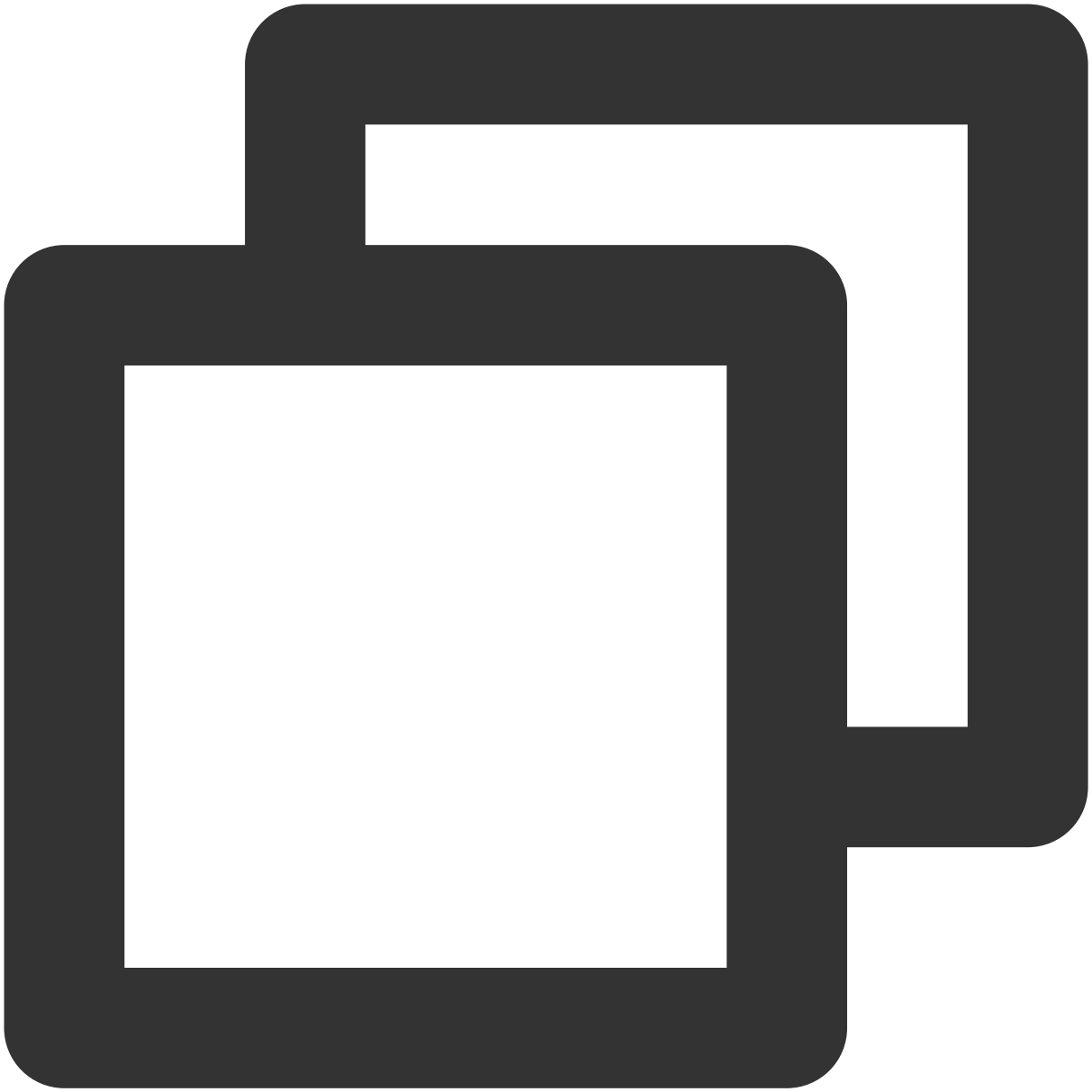
```
\\d my_test
                                  Table "public.my_test"
 Column |         Type          | Collation | Nullable |                Default
--------+-----------------------+-----------+----------+-----------------------------
 id     | integer               |           | not null | nextval('my_test_id_seq'::
 num1   | integer               |           |          |
 num2   | numeric               |           |          |
 str1   | character varying(20) |           |          |
 str2   | text                  |           |          |
Indexes:
    "my_test_pkey" PRIMARY KEY, btree (id)
```

```
Rules:
    r1 AS
    ON INSERT TO my_test
   WHERE (EXISTS ( SELECT 1
     FROM my_test my_test_1
     WHERE my_test_1.id = new.id
     LIMIT 1)) DO INSTEAD  UPDATE my_test SET num1 = new.num1, num2 = new.num2, str
```

使用注意

rule 规则使用存在一定局限，如下所示：

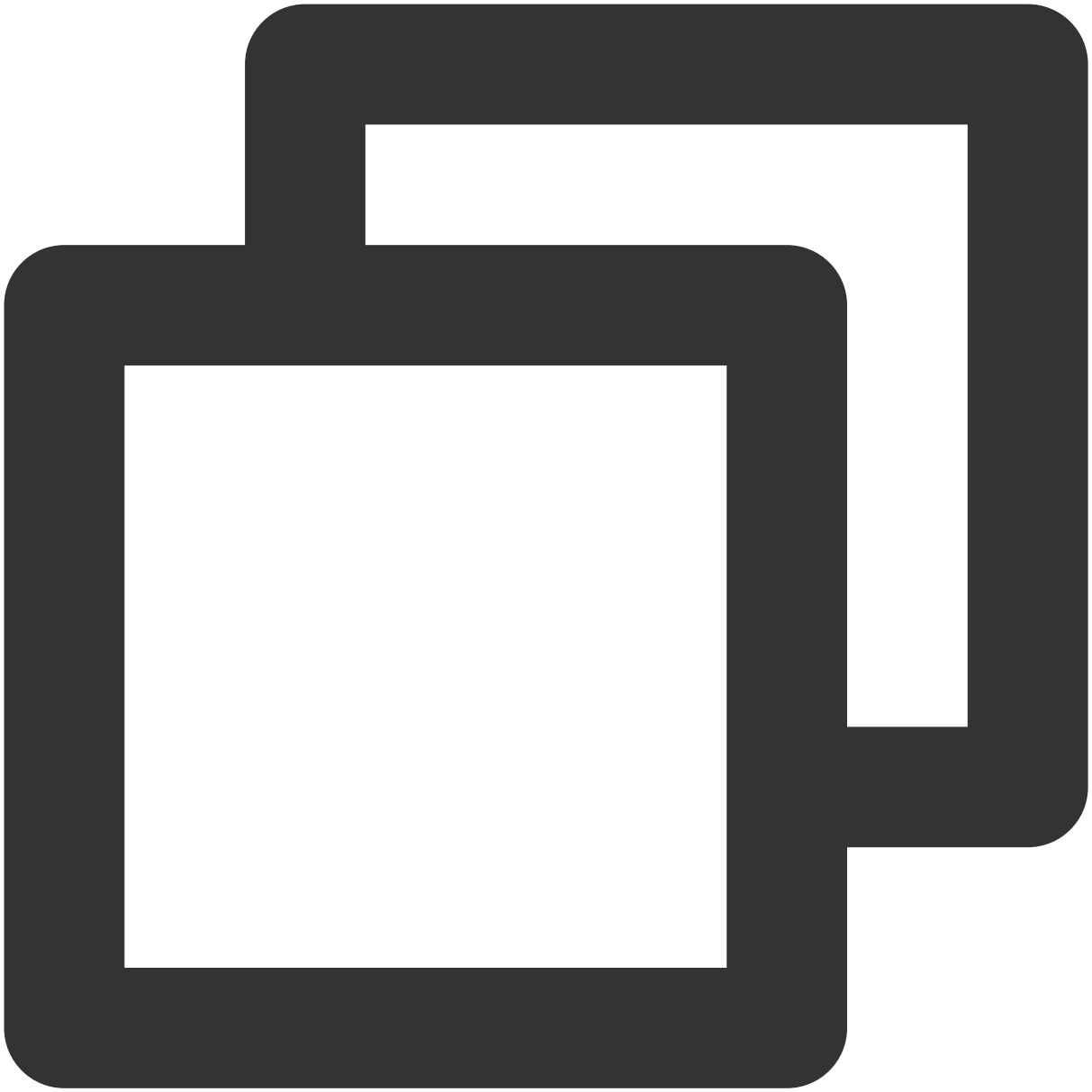
1. 因为 exists 对于 insert 批量插入处理不完善，如果语句没有设置唯一约束或者主键约束，可能在 insert 批量插入时产生重复数据，应尽量避免使用批量 insert，使用时也要避免需要判断 upsert 的字段不重复，或者对需要判断的字段增加唯一约束。例如下面这种批量操作，如果 id 没有主键约束，那么可能执行后存在重复数据。



```
insert into my_test (id,num1,num2,str1,str2)values(1,2,1.0,'111','555'),(1,3,2.0,'1
```

2. rule 不支持 COPY 语句，COPY 语句效果和 insert 批处理类似，都可能导致重复数据。

3. 在设置 update 规则时，如果设置了 rule 用法，但是 insert 语句没有传 num1 和 num2 字段，这两个字段更新后会为空值，导致原数据丢失。



```
update my_test set num1=NEW.num1,num2=NEW.num2,str1=NEW.str1,str2=NEW.str2
```