

# Data Lake Compute

# Development Guide

# Product Documentation



## Copyright Notice

©2013-2024 Tencent Cloud. All rights reserved.

Copyright in this document is exclusively owned by Tencent Cloud. You must not reproduce, modify, copy or distribute in any way, in whole or in part, the contents of this document without Tencent Cloud's the prior written consent.

## Trademark Notice



All trademarks associated with Tencent Cloud and its services are owned by Tencent Cloud Computing (Beijing) Company Limited and its affiliated companies. Trademarks of third parties referred to in this document are owned by their respective proprietors.

## Service Statement

This document is intended to provide users with general information about Tencent Cloud's products and services only and does not form part of Tencent Cloud's terms and conditions. Tencent Cloud's products or services are subject to change. Specific products and services and the standards applicable to them are exclusively provided for in Tencent Cloud's applicable terms and conditions.

# Contents

## Development Guide

SparkJar Job Development Guide

PySpark Job Development Guide

Query Performance Optimization Guide

UDF Function Development Guide

Materialized View

# Development Guide

## SparkJar Job Development Guide

Last updated : 2024-08-15 17:54:48

### Use Cases

DLC is fully compatible with open-source Apache Spark, allowing users to write business programs for reading, writing, and analyzing data on the DLC platform. This example demonstrates how to write Java code to read and write data on COS and perform detailed operations such as creating databases and tables, and reading and writing tables on DLC, helping users complete job development on DLC.

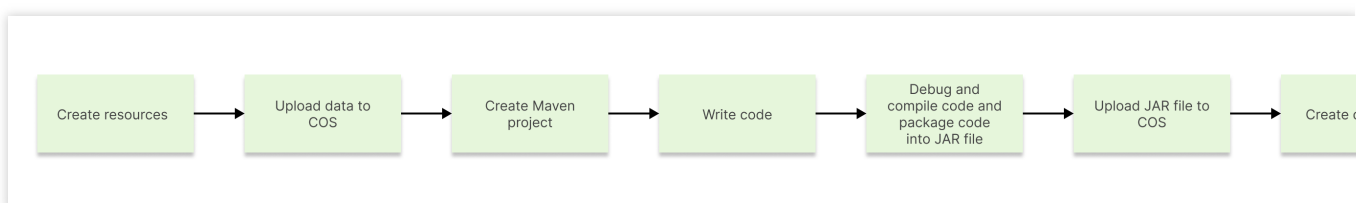
### Environment Preparation

Dependencies: JDK1.8 Maven IntelliJ IDEA

### Development Process

#### Development Flowchart

The development flowchart for DLC Spark JAR jobs is as follows:



#### Creating Resource

The first time you run a job on DLC, you need to create Spark job computing resources. For example, create a Spark job resource named "dlc-demo".

1. Log in to the [Data Lake Compute DLC Console](#). Select your service region, and click **Data Engine** in the navigation menu.
2. Click **Create Resource** in the upper left corner to enter the resource configuration purchase page.
3. In **Cluster Configuration > Calculation Engine Type**, select Spark Job Engine.

## Data Lake Compute [Back](#)

Engine edition: **SuperSQL engine** Standard engine Beta

Billing mode: **Pay-as-you-go** Monthly subscription [Detailed comparison](#)

In this mode, a cluster is billed based on the CUs used and can be suspended when no task is in progress. A suspended cluster incurs no cost. It is suitable for data compute applications with certain task loads and irregular task c

Region: North China South China East China Southwest China West US So

Beijing **Guangzhou** Nanjing Shanghai Shanghai Finance Chengdu Chongqing Silicon Valley S

Europe North China region Hong Kong/Macao/TaiWan (China Region)

Frankfurt Beijing Finance Hong Kong

Cloud products in different regions are not interconnected over private networks and the region cannot be changed after you purchase the service. Please proceed with caution. We recommend you select the region nearest to you

---

### Cluster configuration

**Basic configuration**

Compute engine type: SparkSQL **Spark job** Presto

This is a pay-as-you-go engine for batch jobs. It is suitable for Spark jar and pyspark batch jobs and data processing.  
Fees are charged based on resource usage in data jobs. Bills are generated hourly, and the unit price is 0.35 CNY/CU/hour for a standard cluster and 0.45 CNY/CU/hour for a memory cluster.

Kernel version: **Spark 3.2**

The batch job feature depends on Spark version capabilities. Different versions correspond to different dependency packages. For dependency details, see [Spark Environments](#)

In **Information Configuration > Resource Name**, enter "dlc-demo". For detailed instructions on creating resources, see [Purchasing Private Data Engine](#).

### Advanced configuration

Parameter configuration + Add

IP range of cluster: 10.255.0.0/16 Modify

This option affects the network interconnection between services. In case of non-federated queries, default configuration is recommended; in federated queries, the IP range of the engin

Auto-granting of engine permissions

If this option is enabled, all users are granted the following permissions on this engine:  
USE: Use this engine to execute tasks  
OPERATE: Pause or suspend the engine  
MONITOR: Monitor and maintain the engine based on its usage  
For more engine permissions, see [here](#)

---

### Info configuration

Resource name:

It can contain up to 100 Chinese characters, letters, digits, hyphens (-) and underscores (\_) only. A duplicate name is not allowed.

Description:

Optional, up to 250 characters.

Tag: No tag

Tags are used to categorize resources. To learn more, see [Tag Documentation](#)

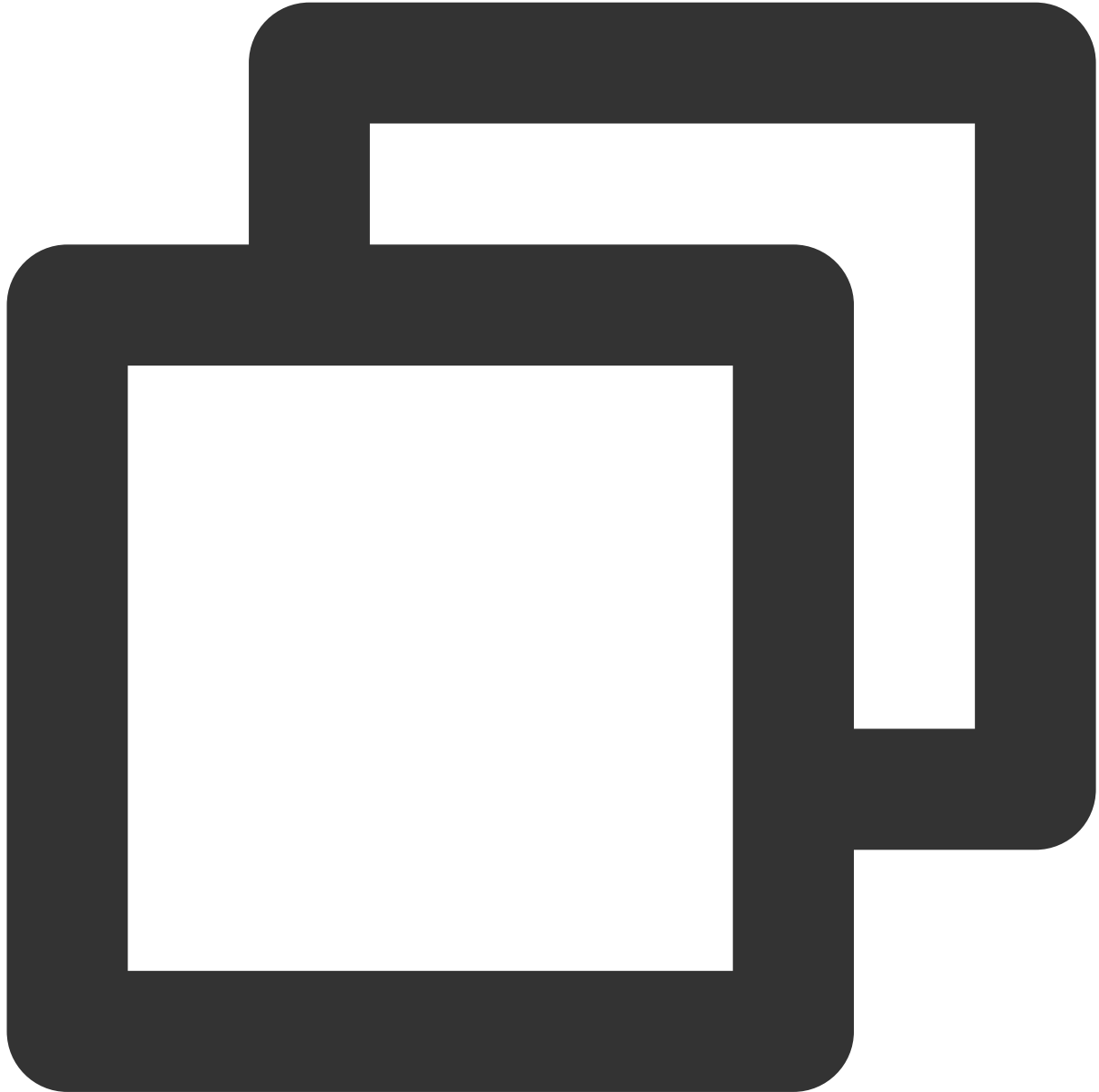
Terms of agreement:  I have read and agree to the [Service Level Agreement for Data Lake Compute](#) and [Refund Policy](#)

4. Click **Activate Now** and confirm the resource configuration information.

5. After confirming the information is correct, click **Submit** to complete the resource configuration.

## Uploading Data to COS

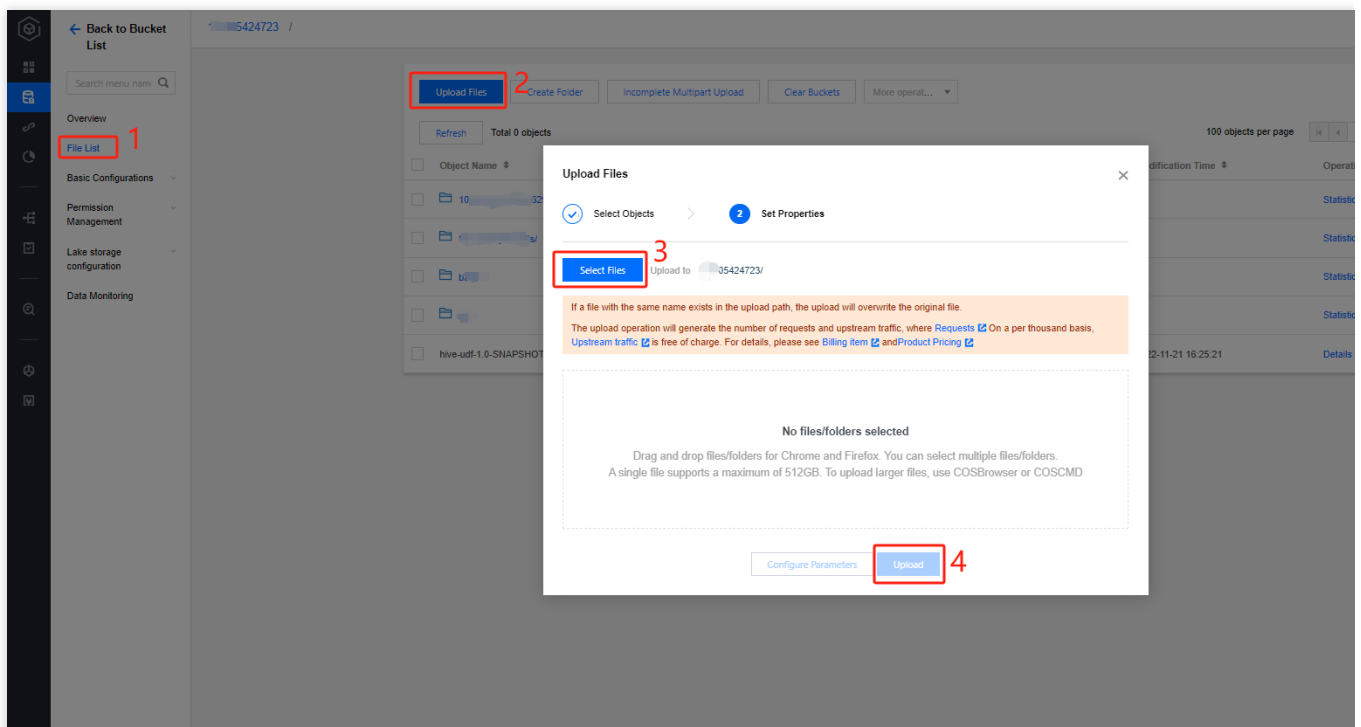
Create a bucket named "dlc-demo" and upload the people.json file to use as an example for reading and writing data from COS. The content of the people.json file is as follows:



```
{"name": "Michael"}  
{"name": "Andy", "age": 30}  
{"name": "Justin", "age": 3}  
{"name": "WangHua", "age": 19}  
{"name": "ZhangSan", "age": 10}  
{"name": "LiSi", "age": 33}  
{"name": "ZhaoWu", "age": 37}
```

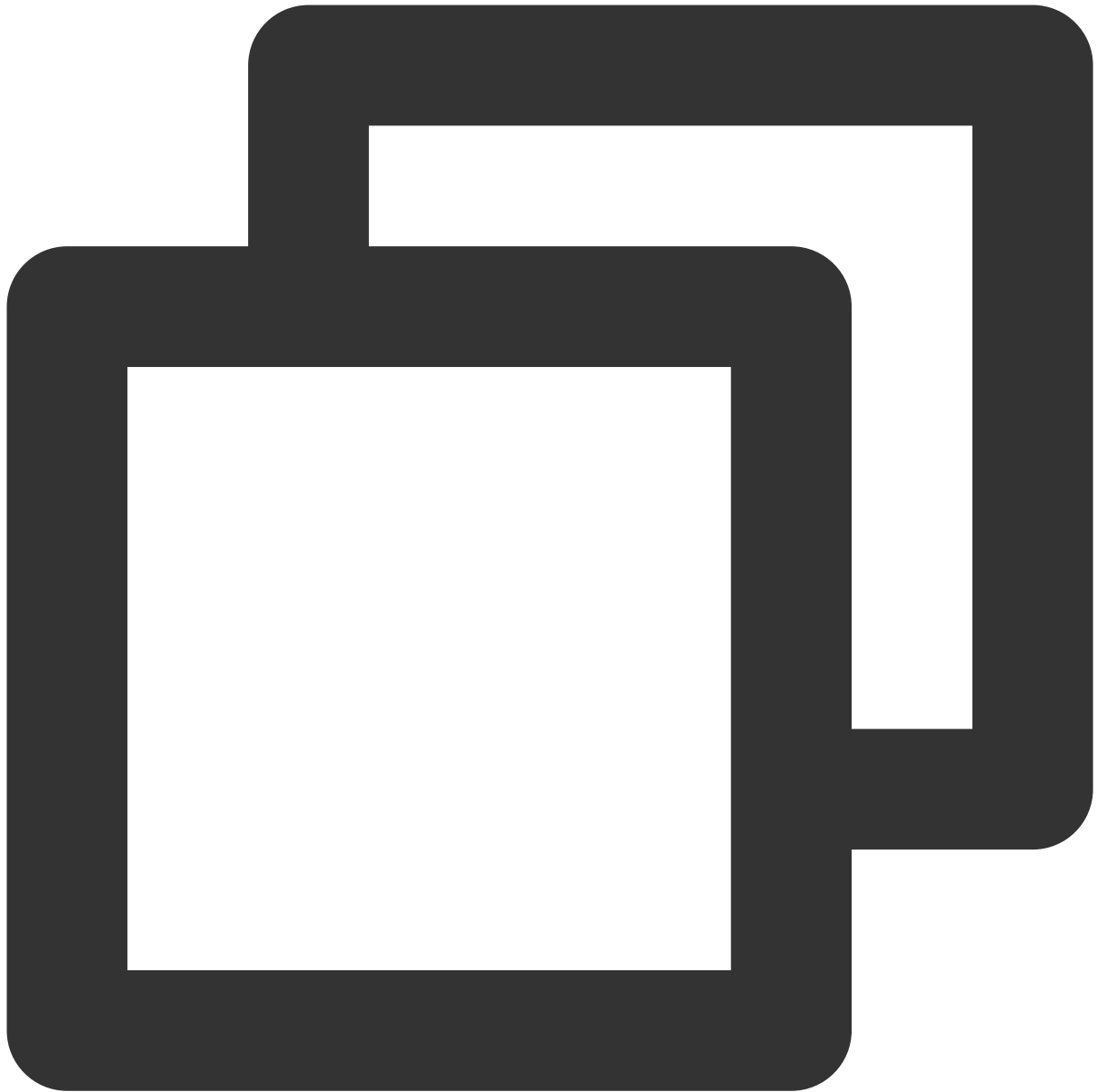
```
{"name": "MengXiao", "age": 68}
{"name": "KaiDa", "age": 89}
```

1. Log in to the [Cloud Object Storage \(COS\) console](#) and click **Bucket List** in the left navigation menu.
2. Create a bucket: Click **Create Bucket** in the upper left corner, enter "dlc-demo" for the name, and click **Next** to complete the configuration.
3. Upload files: Click **File List > Upload File**, select the local "people.json" file, and upload it to the "dlc-demo-1305424723" bucket (-1305424723 is a randomly generated string by the platform when creating the bucket), then click **Upload** to complete the file upload. For detailed instructions on creating a new bucket, see [Create Bucket](#).



## Creating a Maven Project

1. Create a new Maven project named "demo" through IntelliJ IDEA.
2. Add dependencies: Add the following dependencies to the pom.xml file:



```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.12</artifactId>
  <version>3.2.1</version>
  <scope>provided</scope>
</dependency>
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.12</artifactId>
  <version>3.2.1</version>
  <scope>provided</scope>
```

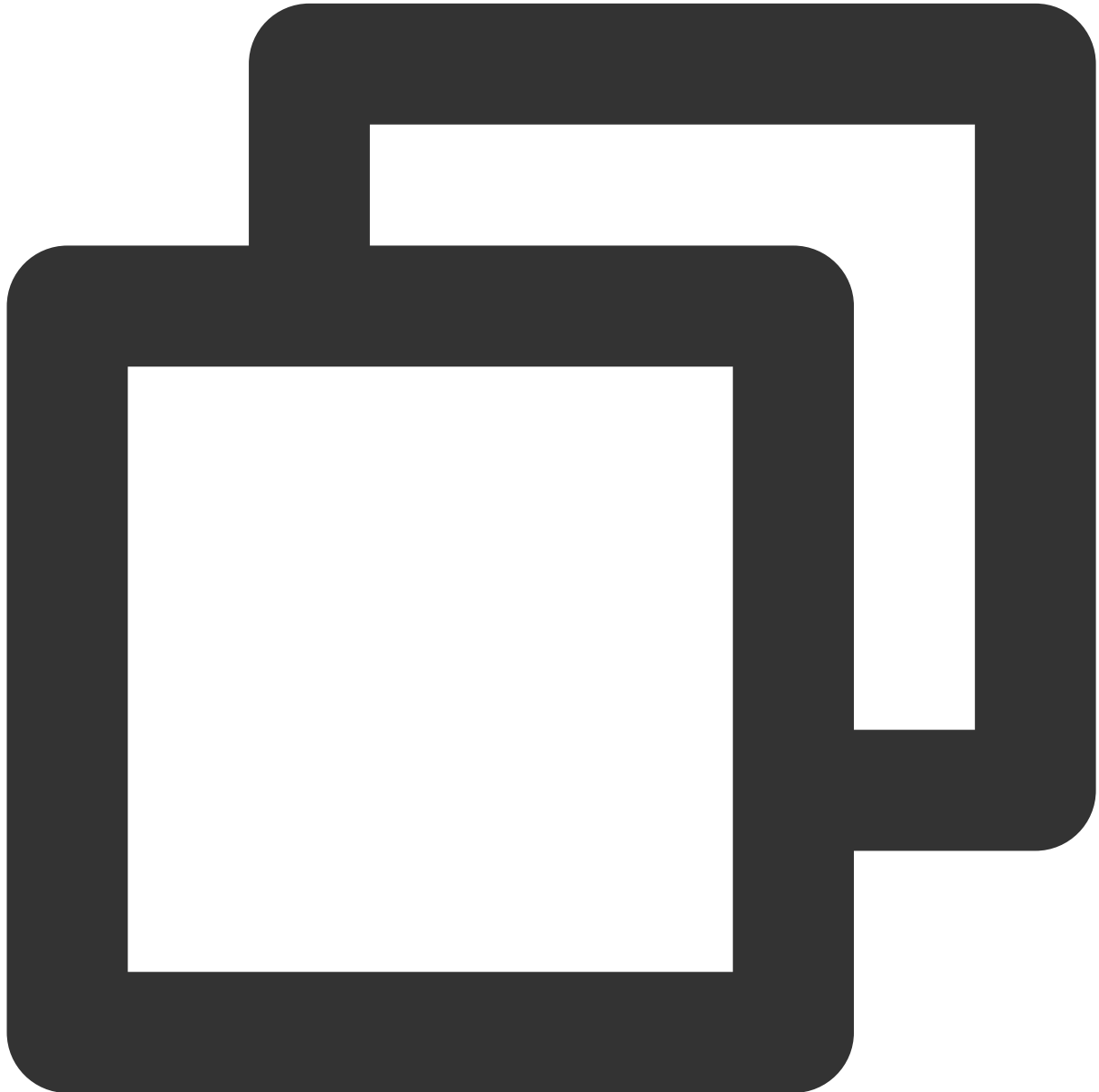


```
</dependency>
```

## Writing Codes

The code features include reading and writing data from COS, as well as creating databases and tables, querying data, and writing data in DLC.

1. Example code for reading and writing data from COS:



```
package com.tencent.dlc;  
  
import org.apache.spark.sql.Dataset;
```

```
import org.apache.spark.sql.Row;
import org.apache.spark.sql.SaveMode;
import org.apache.spark.sql.SparkSession;

public class CosService {

    public static void main( String[] args )
    {
        //1. Create SparkSession
        SparkSession spark = SparkSession
            .builder()
            .appName("Operate data on cos")
            .config("spark.some.config.option", "some-value")
            .getOrCreate();
        //2. Read the json file from COS to generate a data set, supporting various
        String readPath = "cosn://dlc-demo-1305424723/people.json";
        Dataset<Row> readData = spark.read().json(readPath);
        //3. Perform business computations on the data set to generate result data,
        readData.createOrReplaceTempView("people");
        Dataset<Row> result = spark.sql("SELECT * FROM people where age > 3");
        //4. Save the result data back to COS
        String writePath = "cosn://dlc-demo-1305424723/people_output";
        //Supports writing various file types such as json, csv, parquet, orc, text
        result.write().mode(SaveMode.Append).json(writePath);
        spark.read().json(writePath).show();
        //5. Close the session
        spark.stop();
    }
}
```

## 2. Create databases, tables, query data, and write data on DLC:



```
package com.tencent.dlc;

import org.apache.spark.sql.Session;

public class DbService {

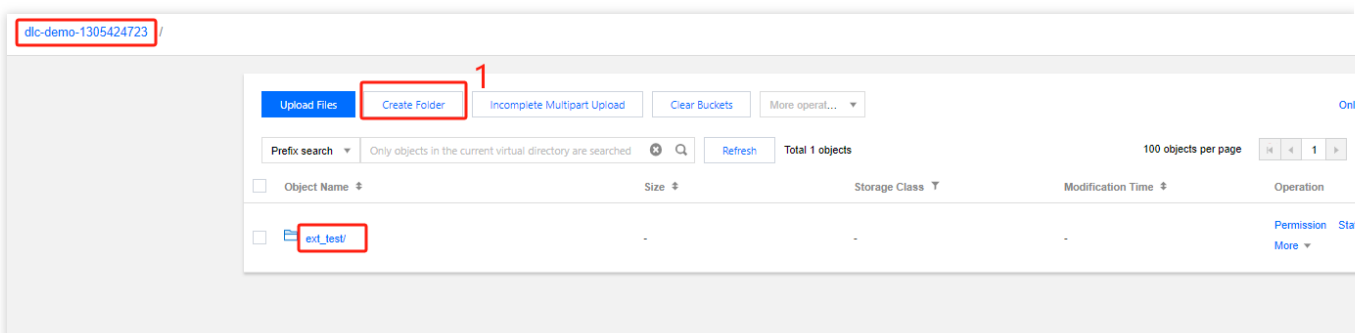
    public static void main(String[] args) {
        //1. Initialize SparkSession
        Session spark = Session
            .builder()
    }
```

```

        .appName("Operate DB Example")
        .getOrCreate();
//2. Create a database
String dbName = " `DataLakeCatalog`.`dlc_db_test` ";
String dbSql = "CREATE DATABASE IF NOT EXISTS" + dbName + " COMMENT 'demo t
spark.sql(dbSql);
//3. Create an internal table
String tableName = "`test`";
String tableSql = "CREATE TABLE IF NOT EXISTS " + dbName + "." + tableName
        + "(`id` int,`name` string, `age` int)";
spark.sql(tableSql);
//4. Write data
spark.sql("INSERT INTO " + dbName + "." + tableName + "VALUES (1,'Andy',12)");
//5. Query data
spark.sql(" SELECT * FROM " + dbName + "." + tableName).show();
//6. Create an external table
String extTableName = "`ext_test`";
spark.sql(
        "CREATE EXTERNAL TABLE IF NOT EXISTS " + dbName + "." + extTableName
        + " (`id` int, `name` string, `age` int) "
        + "ROW FORMAT SERDE 'org.apache.hive.hcatalog.data.JsonSerD
        + "STORED AS TEXTFILE LOCATION 'cosn://dlc-demo-1305424723/
//7. Write data to the external table
spark.sql("INSERT INTO " + dbName + "." + extTableName + "VALUES (1,'LiLy',
//8. Query data from the external table
spark.sql(" SELECT * FROM " + dbName + "." + extTableName).show();
//9. Close the session
spark.stop();
}
}

```

When you create an external table, follow the **steps to upload data to COS** and first create a corresponding folder named after the table in the bucket to store the table files.



## Debugging, Compiling Codes, and Packaging Them as JAR Files

Compile and packet the demo project through IntelliJ IDEA to generate the JAR packet demo-1.0-SNAPSHOT.jar in the project's target folder.

## Uploading JAR Files to COS

Log in to the [COS console](#) and follow the steps in **Uploading Data to COS** to upload demo-1.0-SNAPSHOT.jar to COS.

## Creating a Spark Jar Data Job

Before creating a data job, you need to configure CAM role arn to ensure the data job can securely access the data. For details on configuring CAM role arn, see [Configuring Data Access Policy](#). If you have already configured a data policy, the policy name might be: qcs::cam::uin/100018379117:roleName/dlc-demo.

1. Log in to the [Data Lake Compute DLC Console](#). Select your service region, and click **Data Jobs** in the navigation menu.
2. Click **Create Job** in the upper left corner to enter the creation page.
3. On the job configuration page, configure the job running parameters as follows:

Configuration Parameter	Note
Job Name	Custom Spark Jar job name, for example: cosn-demo.
Job Type	Select <b>Batch Processing Type</b>
Data Engine	Select the dlc-demo computing engine created in the <b>Create Resource</b> step.
Program Packet	Select COS, and choose the JAR file demo-1.0-SNAPSHOT.jar uploaded in the <b>Upload JAR file to COS</b> step.
Main Class	Fill in according to the program code, such as: For reading and writing data from COS, fill in: com.tencent.dlc.CosService For creating databases and tables on DLC, fill in: com.tencent.dlc.DbService.
CAM role arn	Select the policy created in the previous step, qcs::cam::uin/100018379117:roleName/dlc-demo.

Keep other parameter values as default.

**Create job**
✕

---

**Basic info** ▲

Job name \*   
It can contain up to 100 characters in Chinese characters, letters, digits, and underscores ( ).

Job type \* Batch processing Stream processing SQL job

Data engine \*   
The billing mode of the selected data engine prevails. For more info, see [Data engine](#) . For network configuration of the data engine, see [Network configuration](#) .

Program package \*  COS  Upload

[Select a COS path](#)  
COS permissions are required, and .jar/.py files are supported.

Main class \*

Program entry parameter

Job parameter (--config)  
  
--config info, the parameter info started with "spark.", one entry per line.

CAM role arn \*  [Refresh](#)  
It determines the data access scope of a Spark job. For configurations, see [Configure CAM role arn](#) .

**Network configuration** ▲

Enhanced network configuration --  
Select up to one enhanced network configuration if needed.

Cross-source network --  
All cross-source network configurations bound will apply.

Create job
Cancel

4. Click **Save** and you can see the created job on the **Spark Job** page.

## Running and Viewing Job Results

1. Running the Job: On the **Spark Job** page, find the newly created job and click **Run** to run the job.
2. Viewing Job Running Results: You can view the job running logs and results.


## Viewing Job Running Logs

1. Click **Job Name** > **Tasks history** to view the task's running status.

### Spark job details

Job info **Task history** Monitoring and alerting

Select an executi... **Last 7 days** Last 30 days 2023-12-14 ~ 2023-12-20

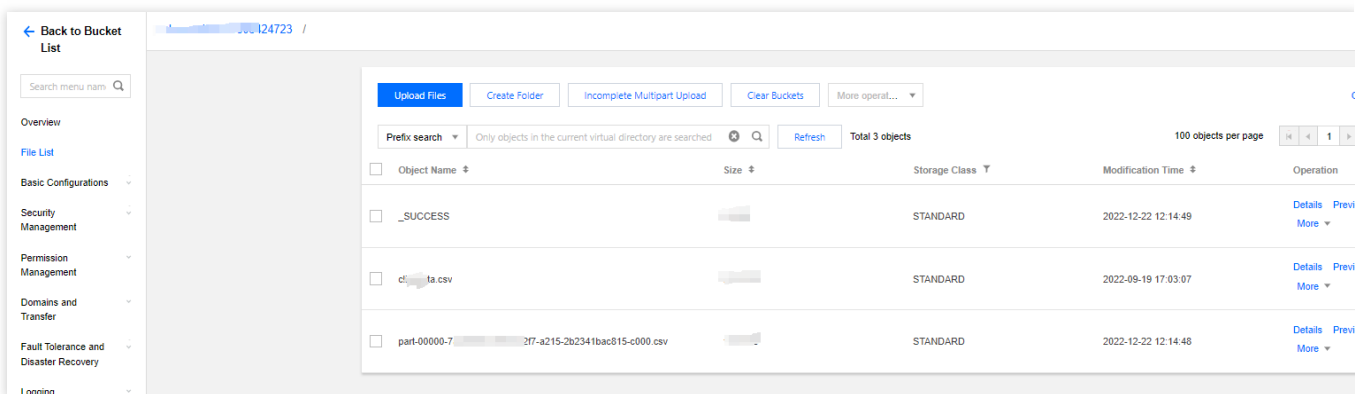
Task ID	Executi...	Task submissi...	Comput...	Operation
	Successful	2023-12-12 20:53:42	47.8s	<a href="#">Learn more</a> <a href="#">Spark UI</a>

Total items: 1 10 / page

2. Click **Task ID > Run Log** to view the job running logs.

### Viewing Job Running Results

1. If you run the example of reading and writing data from COS, you should check the [COS Console](#) for writing results.



Object Name	Size	Storage Class	Modification Time	Operation
_SUCCESS		STANDARD	2022-12-22 12:14:49	<a href="#">Details</a> <a href="#">Previ</a>
data.csv		STANDARD	2022-09-19 17:03:07	<a href="#">Details</a> <a href="#">Previ</a>
part-00000-7-217-a215-2b2341bac815-c000.csv		STANDARD	2022-12-22 12:14:48	<a href="#">Details</a> <a href="#">Previ</a>

2. For jobs that create tables and databases on DLC, check the DLC **Data Exploration** page to view the created databases and tables.

The screenshot displays the Tencent Cloud Data Explorer interface. At the top, the 'Data Explorer' header includes the region 'Guangzhou'. Below this, the 'Database' section shows 'DataLakeCatalog' as the selected catalog. A query editor displays a SQL query: `SELECT * FROM 'DataLakeCatalog'.dl_...'`. The query status is 'Running'. Below the query editor, a sidebar on the left shows a tree view of the database structure, including 'Table', 'View', and 'Function'. The 'Query result' section at the bottom provides performance metrics: 'Task ID', 'SQL details', 'Export', and 'Suggestions'. It also shows 'Query time: 1.63s', 'Scanned data volume: 42 B', and 'Billable scanned volume: 34.0 MB'. A note indicates '4 entries in total (up to 1,000 entries shown in the console)'. A table with columns 'id', 'name', and 'age' is visible, showing four rows of data.



# PySpark Job Development Guide

Last updated : 2024-07-31 18:02:53

## Scenarios

Data Lake Compute supports the execution of programs written in Python. This example demonstrates the detailed operations of reading and writing data on Cloud Object Storage (COS), creating libraries and tables on Data Lake Compute, and reading and writing tables, assisting users in job development on Data Lake Compute.

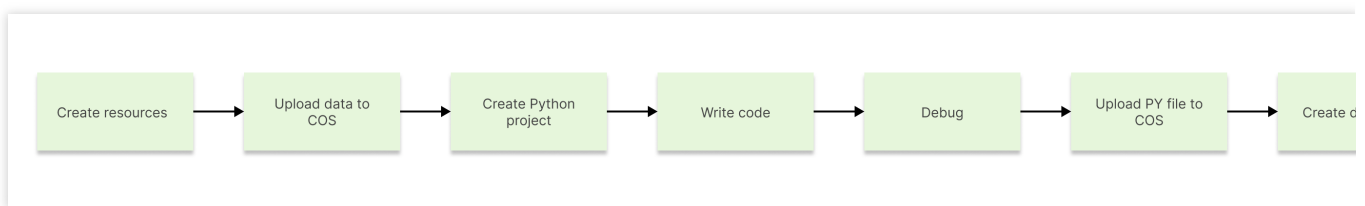
## Environment Preparation

Dependencies: PyCharm or other Python programming development tools.

## Development Process

### Development Flowchart

The development process for Data Lake Compute Spark JAR jobs is as follows:



### Resource Creation

For the first time running a job on Data Lake Compute, you need to create new Spark job compute resources, for instance, creating a Spark job resource named "dlc-demo".

1. Log in to the [Data Lake Compute DLC Console](#), select the service region, and click on **Data Engine** in the navigation menu.
2. Click **Create Resource** in the upper left corner to enter the resource configuration purchase page.
3. In the **Cluster Configuration > Calculation Engine Type** option, select Spark as the job engine.

### Data Lake Compute [Back](#)

Engine edition: **SuperSQL engine** Standard engine Beta

Billing mode: **Pay-as-you-go** Monthly subscription [Detailed comparison](#)

In this mode, a cluster is billed based on the CUs used and can be suspended when no task is in progress. A suspended cluster incurs no cost. It is suitable for data compute applications with certain task loads and irregular task cy

Region: North China South China East China Southwest China West US Sou

Beijing **Guangzhou** Nanjing Shanghai Shanghai Finance Chengdu Chongqing Silicon Valley SI

Europe North China region Hong Kong/Macao/TaiWan (China Region)

Frankfurt Beijing Finance Hong Kong

Cloud products in different regions are not interconnected over private networks and the region cannot be changed after you purchase the service. Please proceed with caution. We recommend you select the region nearest to your

#### Cluster configuration

**Basic configuration**

Compute engine type: SparkSQL **Spark job** Presto

This is a pay-as-you-go engine for batch jobs. It is suitable for Spark jar and pyspark batch jobs and data processing.  
Fees are charged based on resource usage in data jobs. Bills are generated hourly, and the unit price is 0.35 CNY/CU/hour for a standard cluster and 0.45 CNY/CU/hour for a memory cluster.

Kernel version: **Spark 3.2**

The batch job feature depends on Spark version capabilities. Different versions correspond to different dependency packages. For dependency details, see [Spark Environments](#)

Fill in "dlc-demo" for **Information Configuration > Resource Name**. For a detailed introduction to creating new resources, please refer to [Purchasing a Dedicated Data Engine](#).

### Advanced configuration

Parameter configuration [+ Add](#)

IP range of cluster 10.255.0.0/16 [Modify](#)

This option affects the network interconnection between services. In case of non-federated queries, default configuration is recommended; in federated queries, the IP range of the engine

Auto-granting of engine permissions

If this option is enabled, all users are granted the following permissions on this engine:  
USE: Use this engine to execute tasks  
OPERATE: Pause or suspend the engine  
MONITOR: Monitor and maintain the engine based on its usage  
For more engine permissions, see [here](#)

### Info configuration

Resource name

It can contain up to 100 Chinese characters, letters, digits, hyphens (-) and underscores (\_) only. A duplicate name is not allowed.

Description

Optional, up to 250 characters.

Tag

Tags are used to categorize resources. To learn more, see [Tag Documentation](#)

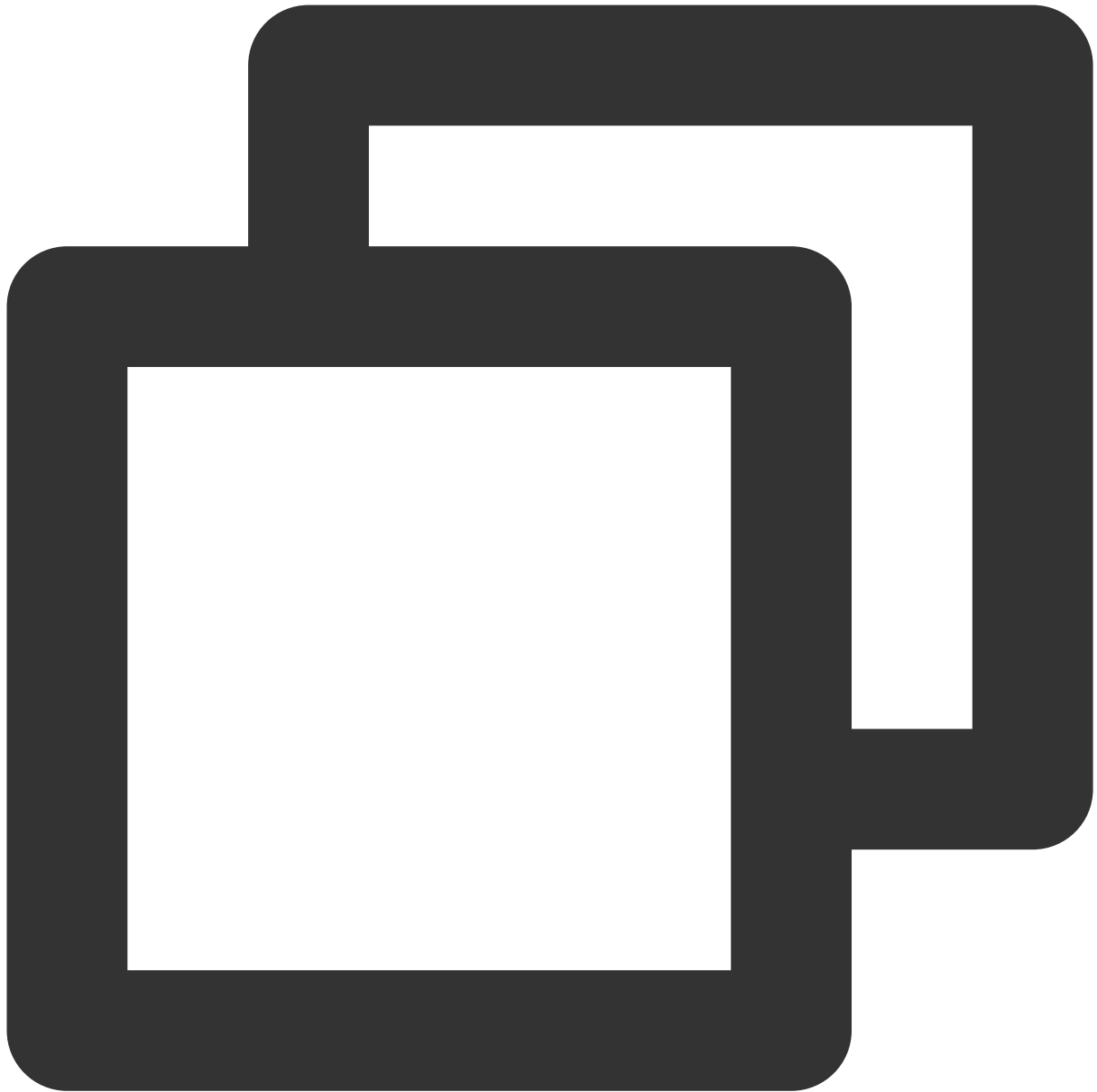
Terms of agreement  I have read and agree to the [Service Level Agreement for Data Lake Compute](#) and [Refund Policy](#)

4. Click **Activate Now** to confirm the resource configuration information.

5. Upon verifying that the information is accurate, click **Submit** to complete the resource configuration.

## Uploading Data to COS

Create a bucket named "dlc-demo" and upload the file people.json for the example of reading and writing data from COS. The content of the people.json file is as follows:



```
{"name": "Michael"}  
{"name": "Andy", "age": 30}  
{"name": "Justin", "age": 3}  
{"name": "WangHua", "age": 19}  
{"name": "ZhangSan", "age": 10}  
{"name": "LiSi", "age": 33}  
{"name": "ZhaoWu", "age": 37}  
{"name": "MengXiao", "age": 68}  
{"name": "KaiDa", "age": 89}
```

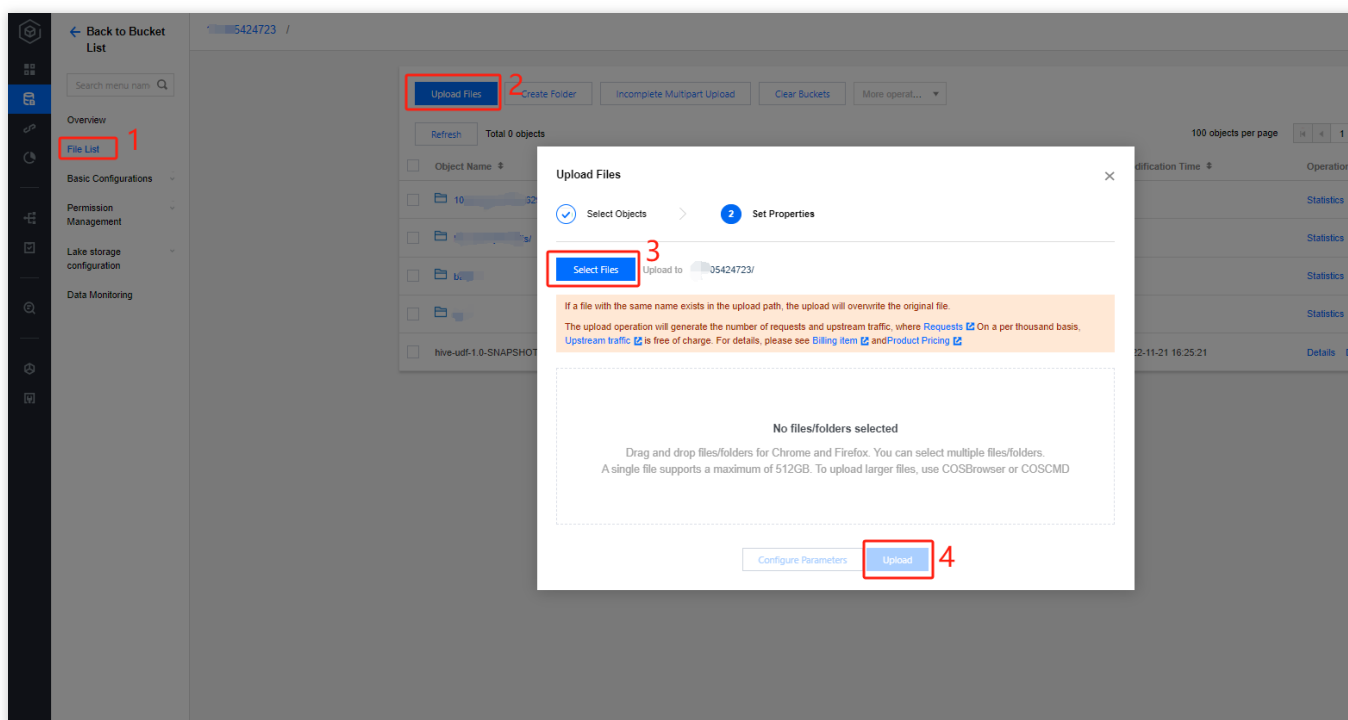
1. Log in to the [Cloud Object Storage \(COS\) console](#) and click on **Bucket List** in the left navigation menu.

## 2. Creating a Bucket:

Click **Create Bucket** in the upper left corner, fill in the name field with "dlc-dmo", and click **Next** to complete the configuration.

## 3. Upload File:

Click on **File List > Upload File**, select the local "people.json" file to upload to the "dlc-demo-1305424723" bucket (-1305424723 is a random string generated by the platform when creating the bucket), click **Upload** to complete the file upload. For details on creating a new bucket, please refer to [Create Bucket](#).



## Creating a Python Project

Create a new project named "demo" using PyCharm.

## Writing Code

1. Create a new cos.py file, write code with the functionality to read and write data from COS, create libraries and tables on DLC, query data, and write data.



```
import sys
from pyspark.sql import SparkSession
from pyspark.sql import Row

if __name__ == "__main__":
    spark = SparkSession \
        .builder \
        .appName("Operate data on cos")\
        .getOrCreate()

    # 1. Read data from COS, supporting various file types such as JSON, CSV, Parqu
```

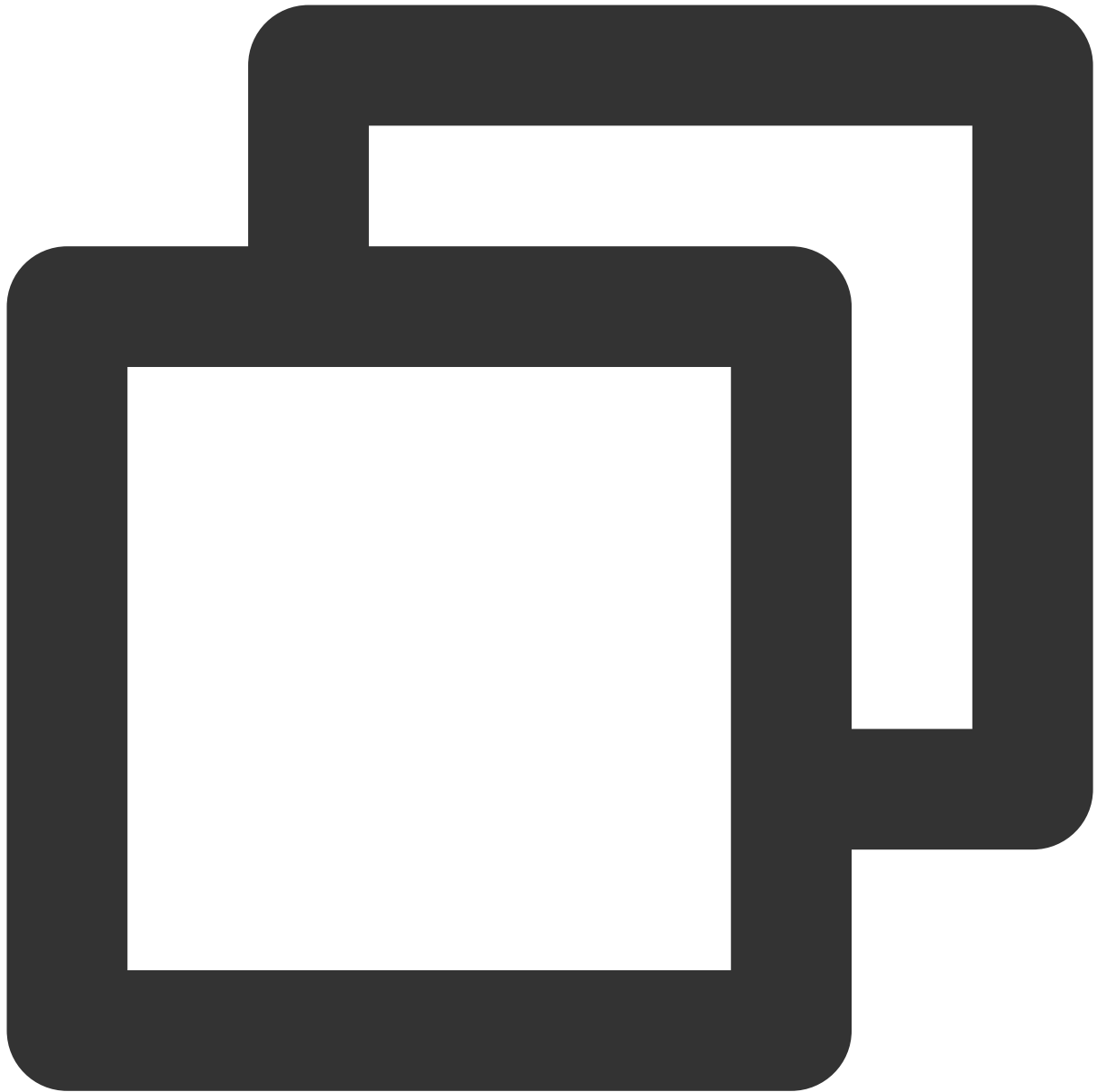
```
read_path = "cosn://dlc-demo-1305424723/people.json"
peopleDF = spark.read.json(read_path)

# 2. Operate on the data
peopleDF.createOrReplaceTempView("people")
data_src = spark.sql("SELECT * FROM people WHERE age BETWEEN 13 AND 19")
data_src.show()

# 3. Writing Data
write_path = "cosn://dlc-demo-1305424723/people_output"
data_src.write.csv(path=write_path, header=True, sep=",", mode='overwrite')

spark.stop()
```

2. Create a new db.py file, write code, the functions of which include creating libraries, tables, querying data, and writing data on Data Lake Compute.



```
from os.path import abspath

from pyspark.sql import SparkSession

if __name__ == "__main__":

    spark = SparkSession \
        .builder \
        .appName("Operate DB Example") \
```



```

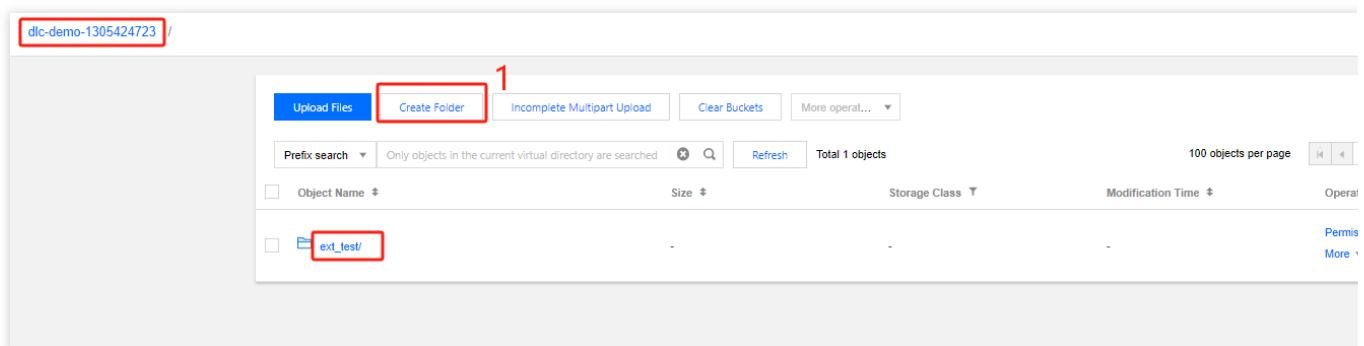
    .getOrCreate()

# 1. Create a Database
spark.sql("CREATE DATABASE IF NOT EXISTS DataLakeCatalog.dlc_db_test_py COMMENT")
# 2. Create Internal Table
spark.sql("CREATE TABLE IF NOT EXISTS DataLakeCatalog.dlc_db_test_py.test(id in
# 3. Writing Internal Data
spark.sql("INSERT INTO DataLakeCatalog.dlc_db_test_py.test VALUES (1, 'Andy', 12)")
# 4. Inspect Internal Data
spark.sql("SELECT * FROM DataLakeCatalog.dlc_db_test_py.test ").show()

# 5. Create External Table
spark.sql("CREATE EXTERNAL TABLE IF NOT EXISTS DataLakeCatalog.dlc_db_test_py.e
# 6. Write external data
spark.sql("INSERT INTO DataLakeCatalog.dlc_db_test_py.ext_test VALUES (1, 'Andy'
# 7. Inspect External Data
spark.sql("SELECT * FROM DataLakeCatalog.dlc_db_test_py.ext_test ").show()
spark.stop()

```

When creating an external table, you can follow the **steps to upload data to COS** and first create a corresponding table name folder in the bucket to save the table files.



## Debugging

Ensure PyCharm debugging is free of syntax errors.

## Upload PY Files to COS

Log in to the [COS console](#) and follow the steps in the previous section **Upload data to COS** to upload `cos.py` and `db.py` to COS.

## Create a New Spark Jar Data Job

Before creating a data job, you need to complete the data access policy configuration to ensure that the data job can safely access the data. For details on configuring the data access policy, please refer to [Configuring Data Access](#)

**Policy.** If the data policy name has been configured as: `qcs::cam::uin/100018379117:roleName/dlc-demo`

1. Log in to the [Data Lake Compute DLC Console](#), select the service region, and click on **Data Jobs** in the navigation menu.
2. Click the **Create Job** button in the upper left corner to enter the creation page.
3. On the job configuration page, set the job running parameters as detailed below:

Parameter Configuration	Note
Job name	Specify a custom Spark job name, for instance: <code>cosn_py</code>
Job type	Select <b>Batch Processing Type</b>
Data engine	Select the <code>dlc-demo</code> compute engine created in the <b>Create Resource</b> step.
Application Package	Select COS, and in the step of <b>uploading a py file to COS</b> , upload the py file: To read and write data from COS, select: <code>cosn://dlc-demo-1305424723/cos.py</code> To create a library, table, etc. on Data Lake Compute, select: <code>cosn://dlc-demo-1305424723/db.py</code>
CAM role arn	Select the policy created in the previous step: <code>qcs::cam::uin/100018379117:roleName/dlc-demo</code>

Retain the default values of other parameters.

### Create job ✕

**Basic info** ▲

Job name \*   
It can contain up to 100 characters in Chinese characters, letters, digits, and underscores (\_).

Job type \*  Batch processing  Stream processing  SQL job

Data engine \*   
The billing mode of the selected data engine prevails. For more info, see [Data engine](#). For network configuration of the data engine, see [Network configuration](#).

Program package \*  COS  Upload

[Select a COS path](#)  
COS permissions are required, and .jar/.py files are supported.

Program entry parameter

Job parameter (--config)   
--config info, the parameter info started with "spark.", one entry per line.

CAM role arn \*  [Refresh](#)  
It determines the data access scope of a Spark job. For configurations, see [Configure CAM role arn](#).

**Network configuration** ▲

Enhanced network configuration --  
Select up to one enhanced network configuration if needed.

Cross-source network configuration --  
All cross-source network configurations bound will apply

**Dependencies** ▶

4. Click **Save** to view the created job on the **Spark Job** page.


## Execute and View Job Results

1. Run the job: On the **Spark Job** page, locate the newly created job and click **Run** to execute the job.
2. Viewing Job Execution Results: You can view the job execution logs and results.

## Viewing Job Execution Logs

1. Click **Job Name >Tasks history** to view the task execution status:

The screenshot shows the 'Spark job details' page with the 'Task history' tab highlighted in a red box. Below the tabs, there are filters for 'Last 7 days', 'Last 30 days', and a date range '2023-12-14 ~ 2023-12-20'. A table lists task execution details:

Task ID	Executi...	Task submissi...	Comput...	Operation
	Successful	2023-12-12 20:53:42	47.8s	<a href="#">Learn more</a> <a href="#">Spark UI</a>

At the bottom, it shows 'Total items: 1' and a pagination control set to '10 / page' with page number '1'.

2. Click **Task ID > Run Log** to view the job execution log.

### View Job Execution Results

1. To run the example of reading and writing data from COS, go to the COS console to view the data write results.

The screenshot shows the COS console interface. On the left is a navigation menu with options like 'Back to Bucket List', 'Overview', 'File List', 'Basic Configurations', 'Security Management', 'Permission Management', 'Domains and Transfer', 'Fault Tolerance and Disaster Recovery', and 'Logging'. The main area displays a table of objects:

Object Name	Size	Storage Class	Modification Time	Operation
_ <u>SUCCESS</u>		STANDARD	2022-12-22 12:14:49	<a href="#">Details</a> <a href="#">Prev</a> <a href="#">More</a>
cf...ta.csv		STANDARD	2022-09-19 17:03:07	<a href="#">Details</a> <a href="#">Prev</a> <a href="#">More</a>
part-00000-7...2f7-a215-2b2341bac815-c000.csv		STANDARD	2022-12-22 12:14:48	<a href="#">Details</a> <a href="#">Prev</a> <a href="#">More</a>

2. To create tables and libraries on Data Lake Compute, navigate to the Data Exploration page on Data Lake Compute to view the creation of libraries and tables.

The screenshot displays the Tencent Cloud Data Explorer interface. At the top, the 'Data Explorer' header includes the region 'Guangzhou'. Below this, the 'Database' section shows 'DataLakeCatalog' as the selected catalog. A search bar for table names is present. The left sidebar shows a tree view of the database structure, with 'Table' selected. The main query editor shows a running query: `SELECT * FROM `DataLakeCatalog`.`dl_...``. Below the query editor, the 'Query result' tab is active, displaying query statistics: 'Task ID', 'SQL details', 'Export', and 'Suggestions'. The statistics include 'Query time: 1.63s', 'Scanned data volume: 42 B', and 'Billable scanned volume: 34.0 MB'. Below the statistics, a table with 4 entries is shown, with columns 'id' and 'name'.

id	name

# Query Performance Optimization Guide

Last updated : 2024-07-31 18:03:07

## Foreword

To enhance task execution efficiency, the DLC engine employs numerous optimization measures during computation, such as Data Governance, Iceberg indexing, Cache, etc. Proper use can not only reduce unnecessary scan costs but can even increase efficiency by several times or even dozens of times. Below, some optimization ideas are provided at different levels.

## Optimize SQL Statements

Scenario: The SQL statement itself is unreasonable, leading to poor execution efficiency.

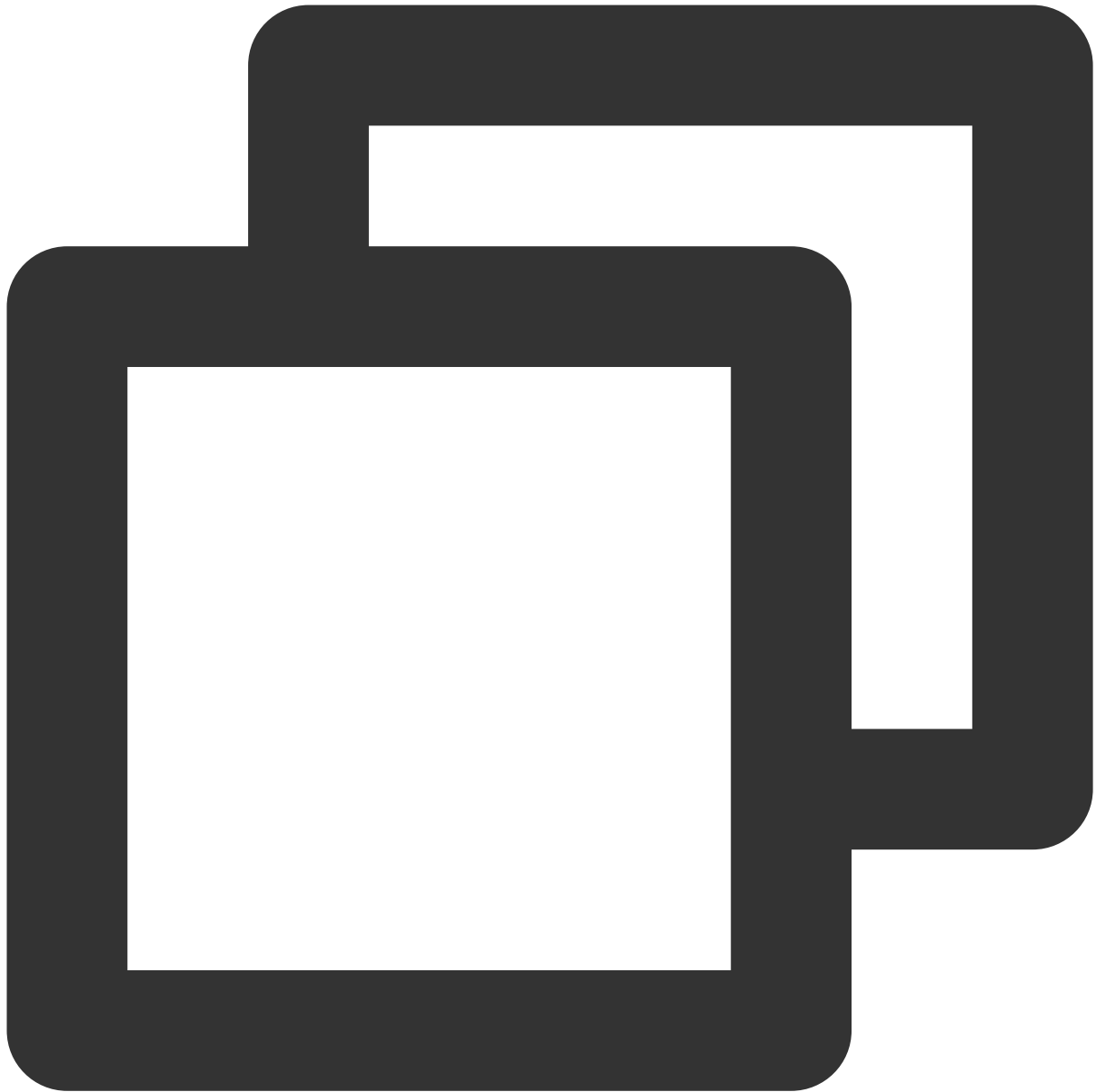
### Optimize JOIN Statements

When a query involves JOINS with multiple tables, the Presto engine prioritizes completing the JOIN operation for the table on the right side of the query. Generally, completing the JOIN for the smaller table first, then joining the result set with the larger table, leads to higher execution efficiency. Therefore, the order of JOINS directly affects the query's performance. DLC Presto automatically collects statistics for inner tables and uses CBO to reorder the tables in the query.

For outer tables, users can usually collect statistics through the analyze statement or manually specify the order of JOINS. If manual specification is needed, please order the tables by size, placing the smaller table on the right and the larger table on the left, as in tables A > B > C, for example: `select * from A Join B Join C`. It is important to note that this does not guarantee increased efficiency in all scenarios, as it actually depends on the size of the data set resulting from the JOIN.

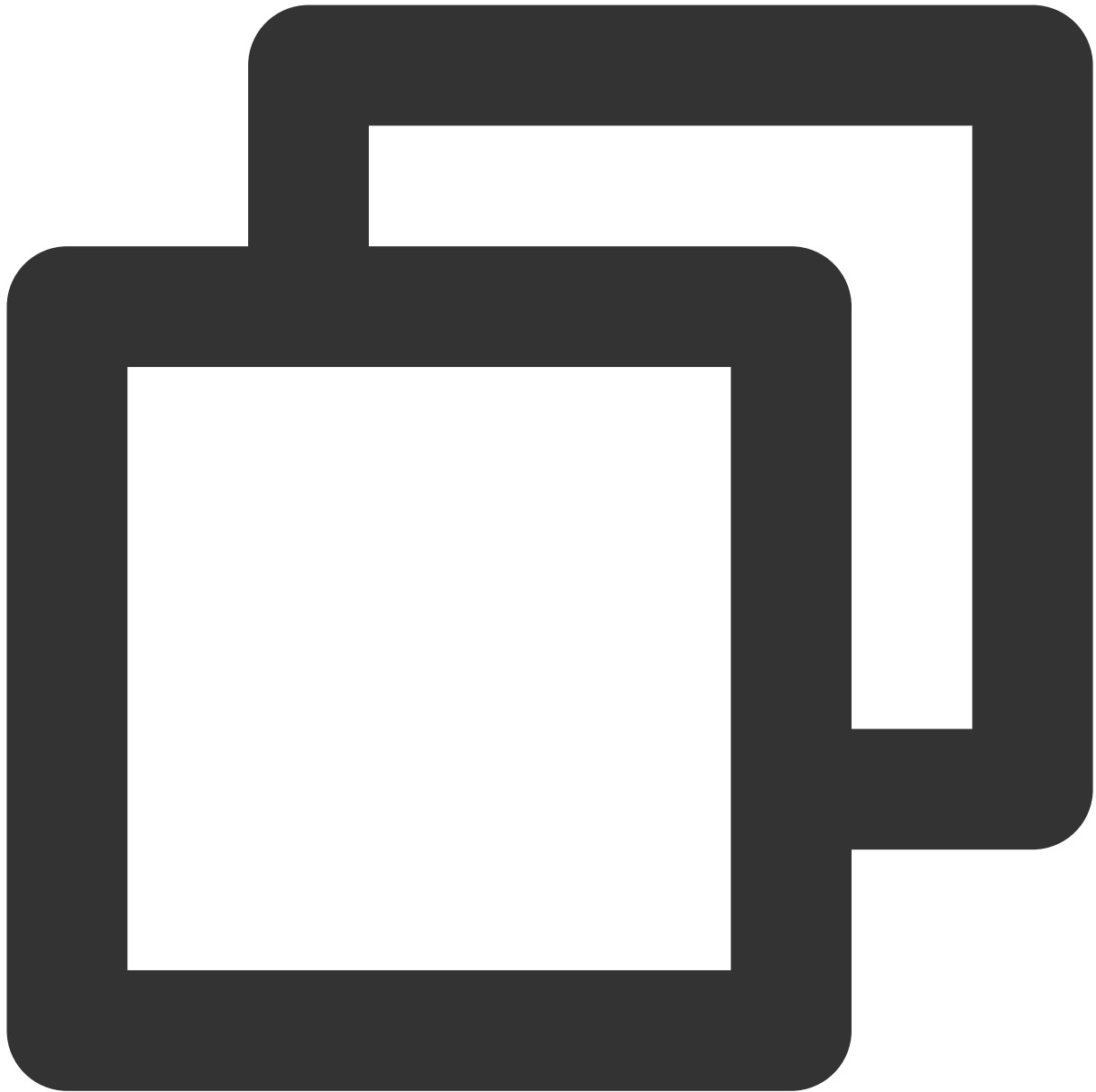
### Optimize GROUP BY Statements

Arranging the order of fields in the GROUP BY statement can improve performance to a certain extent. Please sort the aggregation fields by cardinality from highest to lowest, for example:



```
// Efficient approach
SELECT id,gender,COUNT(*) FROM table_name GROUP BY id, gender;
// Inefficient approach
SELECT id,gender,COUNT(*) FROM table_name GROUP BY gender, id;
```

Another optimization method is to use numbers to replace the specific grouping fields as much as possible. These numbers represent the positions of the column names following the SELECT keyword, for example, the above SQL can be replaced as follows:



```
SELECT id,gender,COUNT(*) FROM table_name GROUP BY 1, 2;
```

## Use Approximate Aggregate Functions

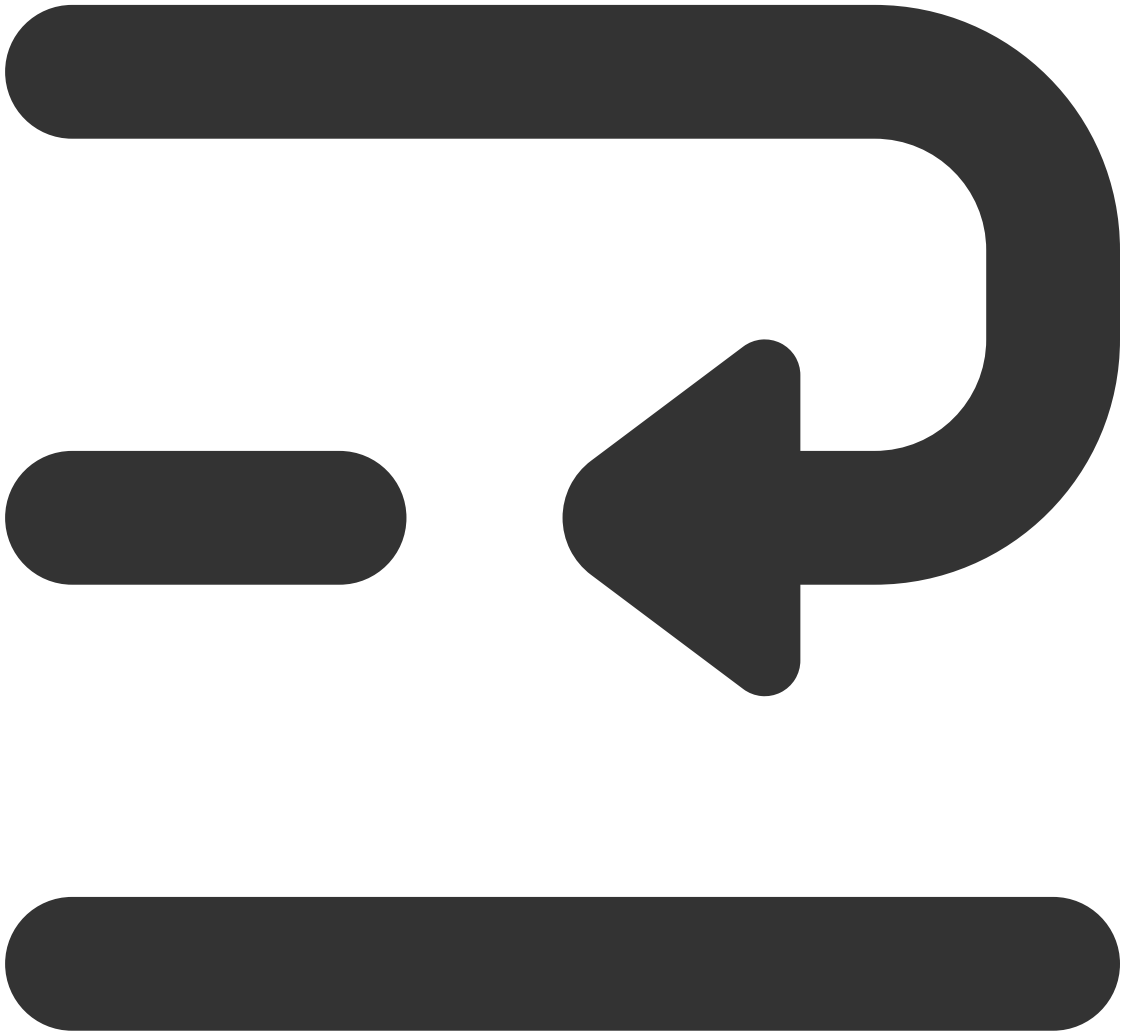
For query scenarios that can tolerate a small amount of error, using these approximate aggregate functions can significantly improve query performance.

For example, Presto can use the `APPROX_DISTINCT()` function instead of `COUNT(distinct x)`, and the corresponding function in Spark is `APPROX_COUNT_DISTINCT`. The drawback of this approach is that approximate aggregate functions have an error margin of about 2.3%.



## Use REGEXP\_LIKE instead of multiple LIKE statements

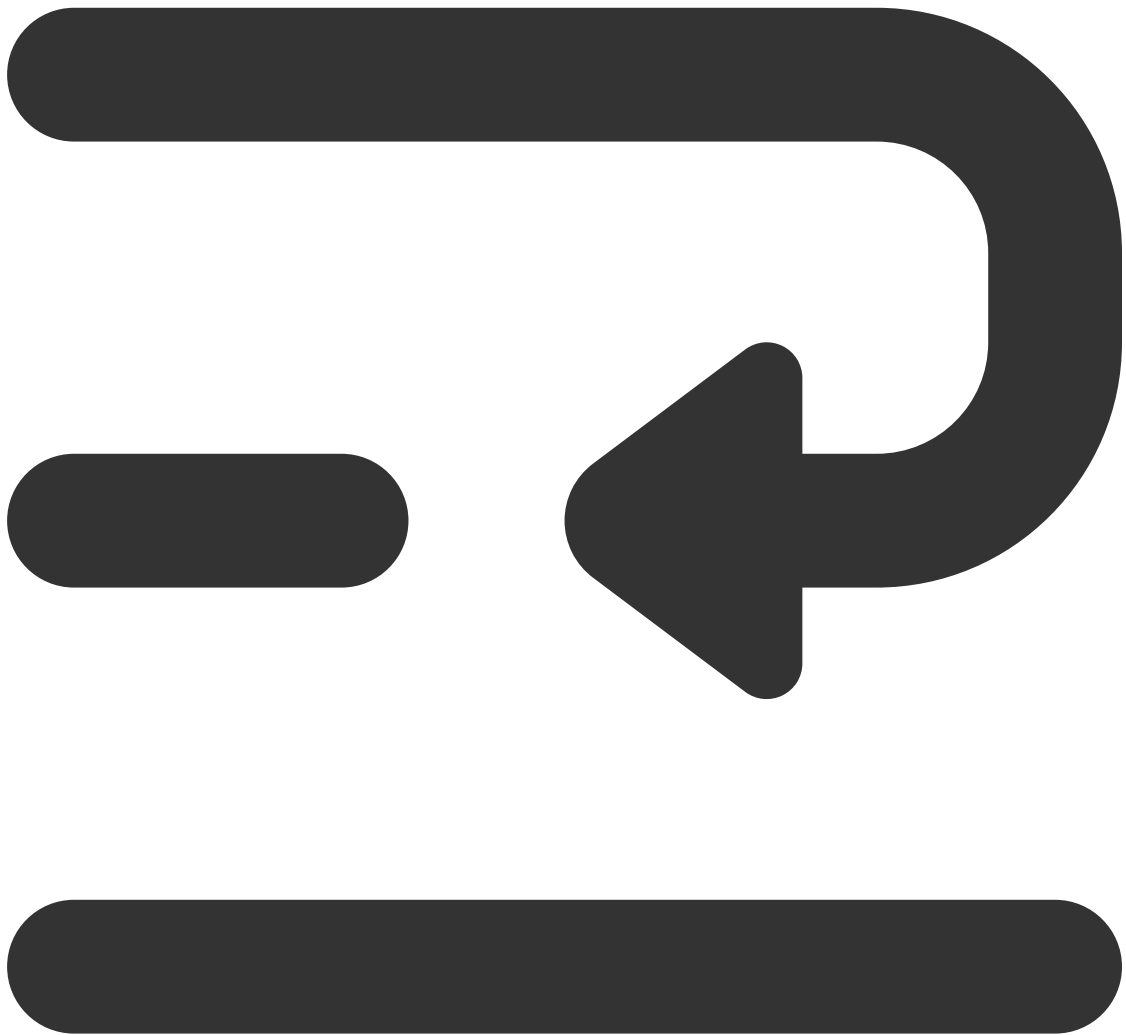
When there are multiple LIKE statements in SQL, you can often use regular expressions to replace multiple LIKEs, which can significantly improve execution efficiency. For example:

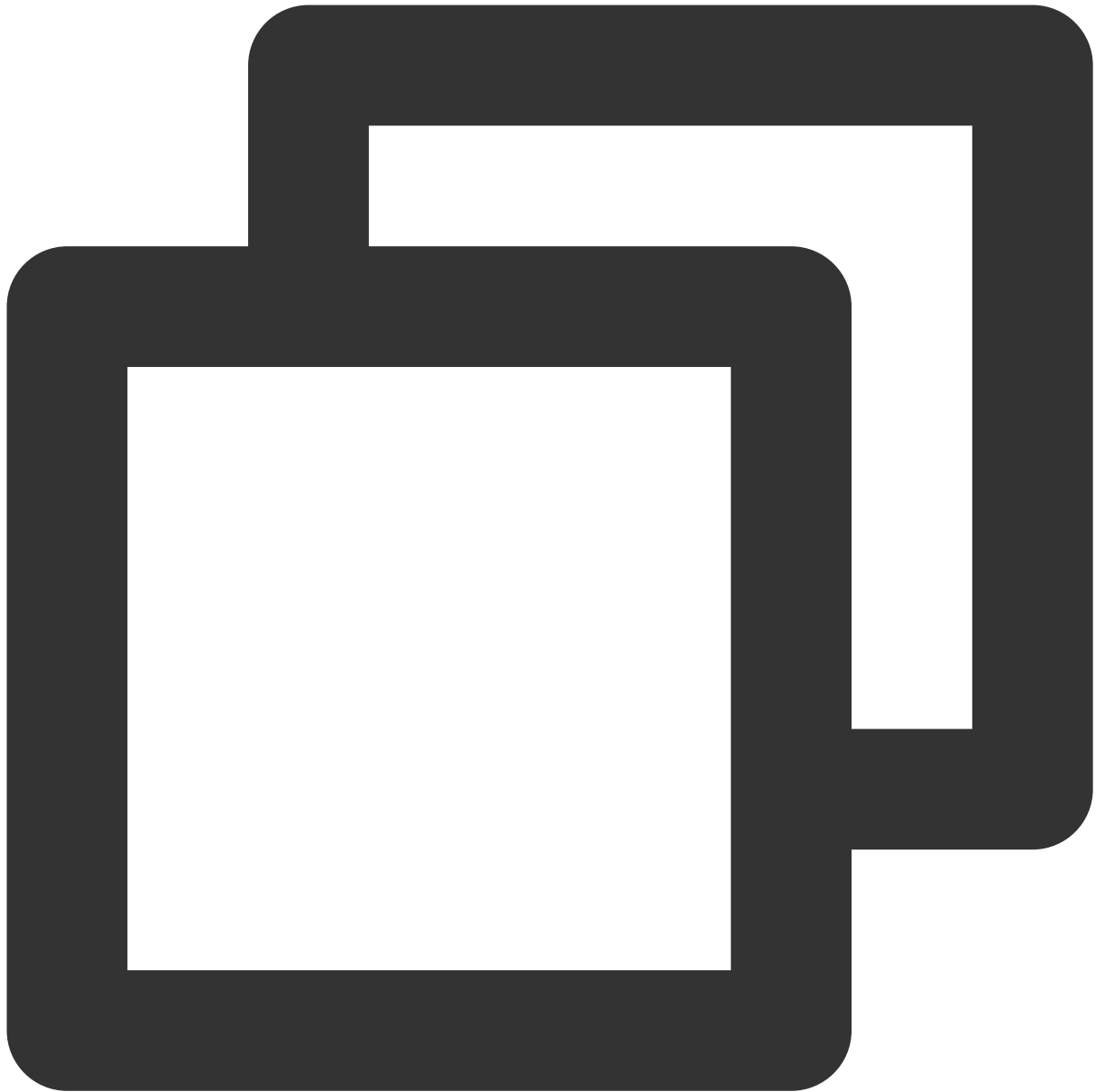




```
SELECT COUNT(*) FROM table_name WHERE field_name LIKE '%guangzhou%' OR LIKE '%beiji'
```

Can be optimized to:





```
SELECT COUNT(*) FROM table_name WHERE regexp_like(field_name, 'guangzhou|beijing|ch
```

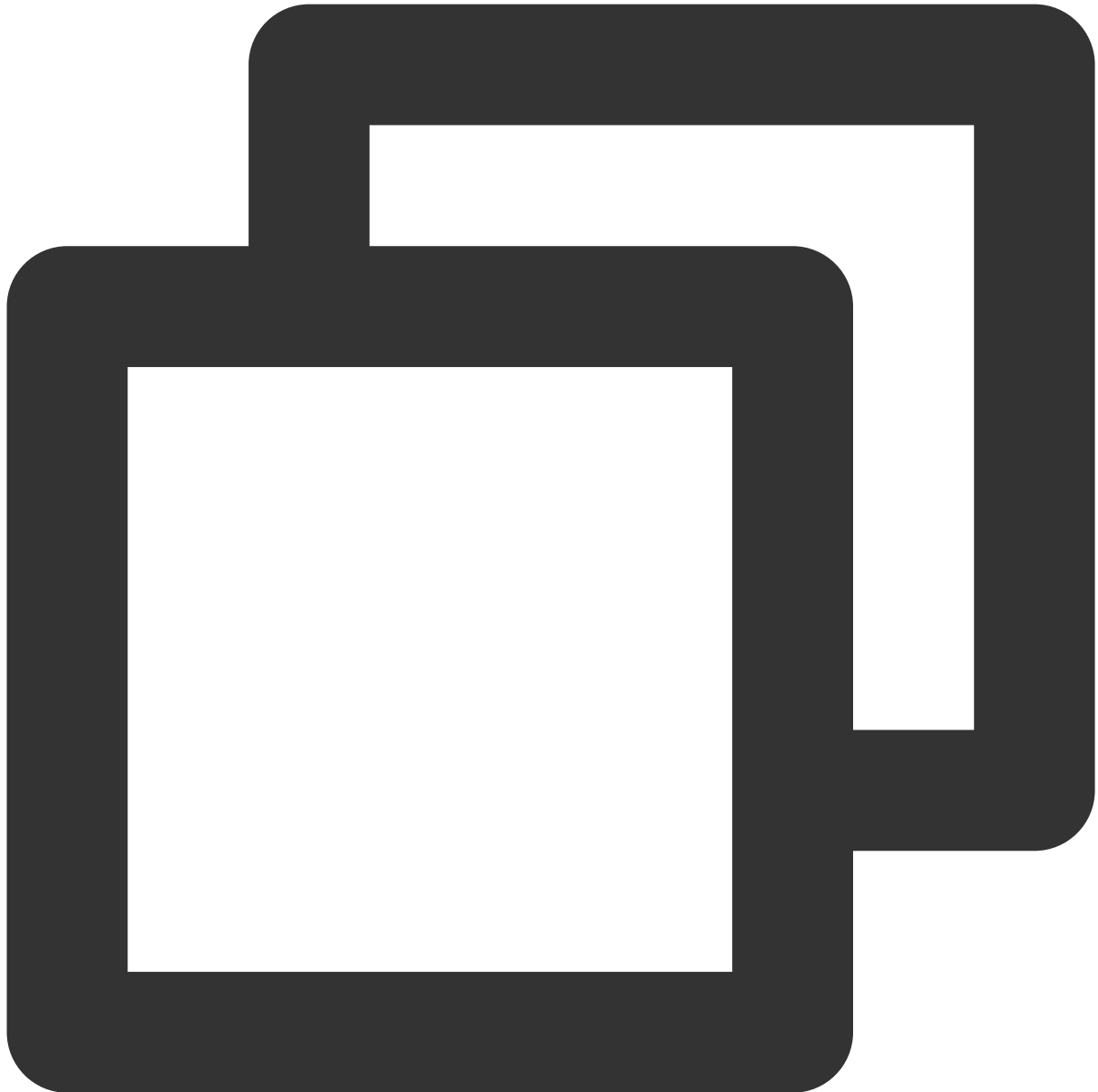
## Data Governance

### Data Governance Use cases

Scenario: Real-time writing. Flink CDC real-time writing usually adopts the upsert method, which generates a large number of small files during the writing process. When small files accumulate to a certain extent, it can cause data

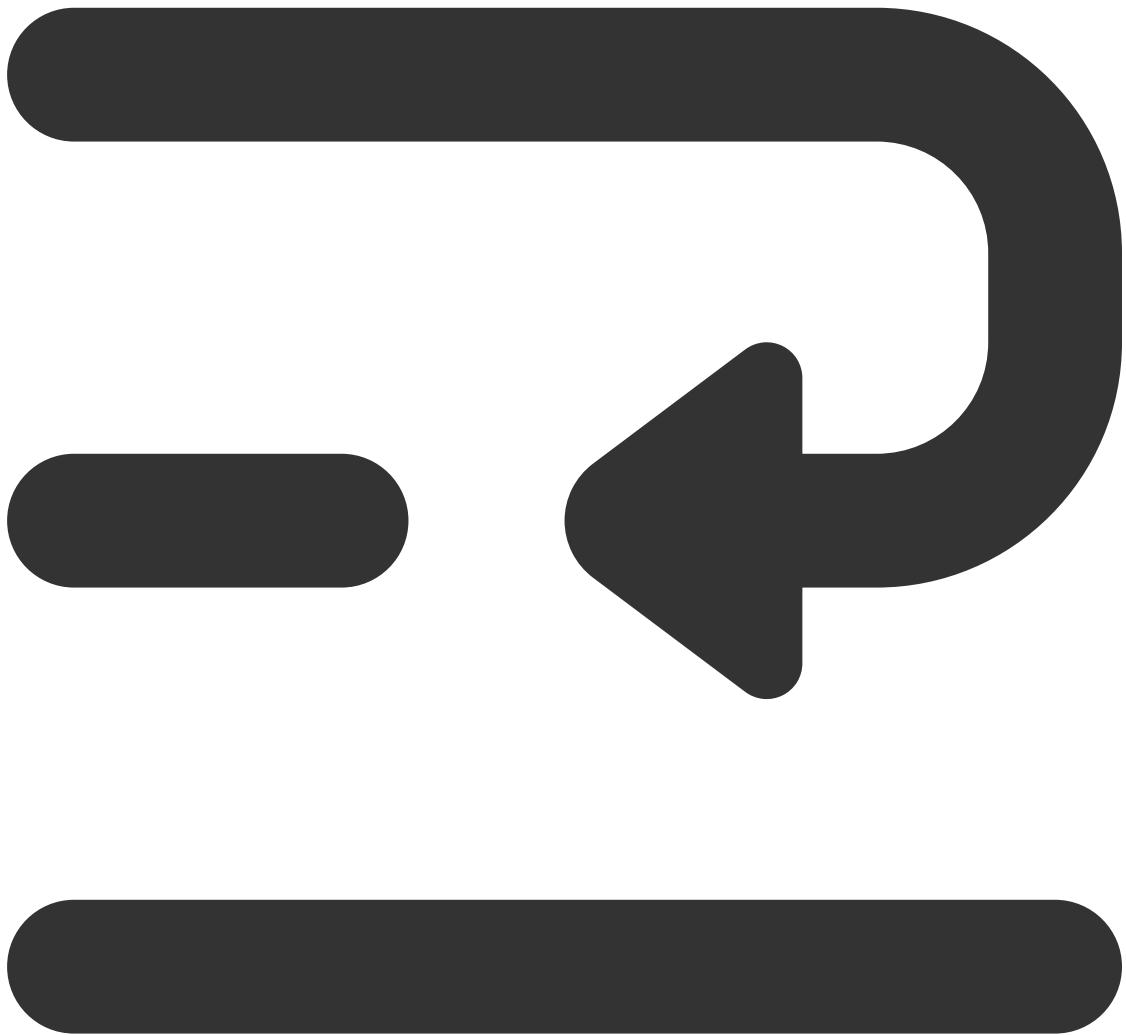
queries to slow down, or even result in timeout failures.

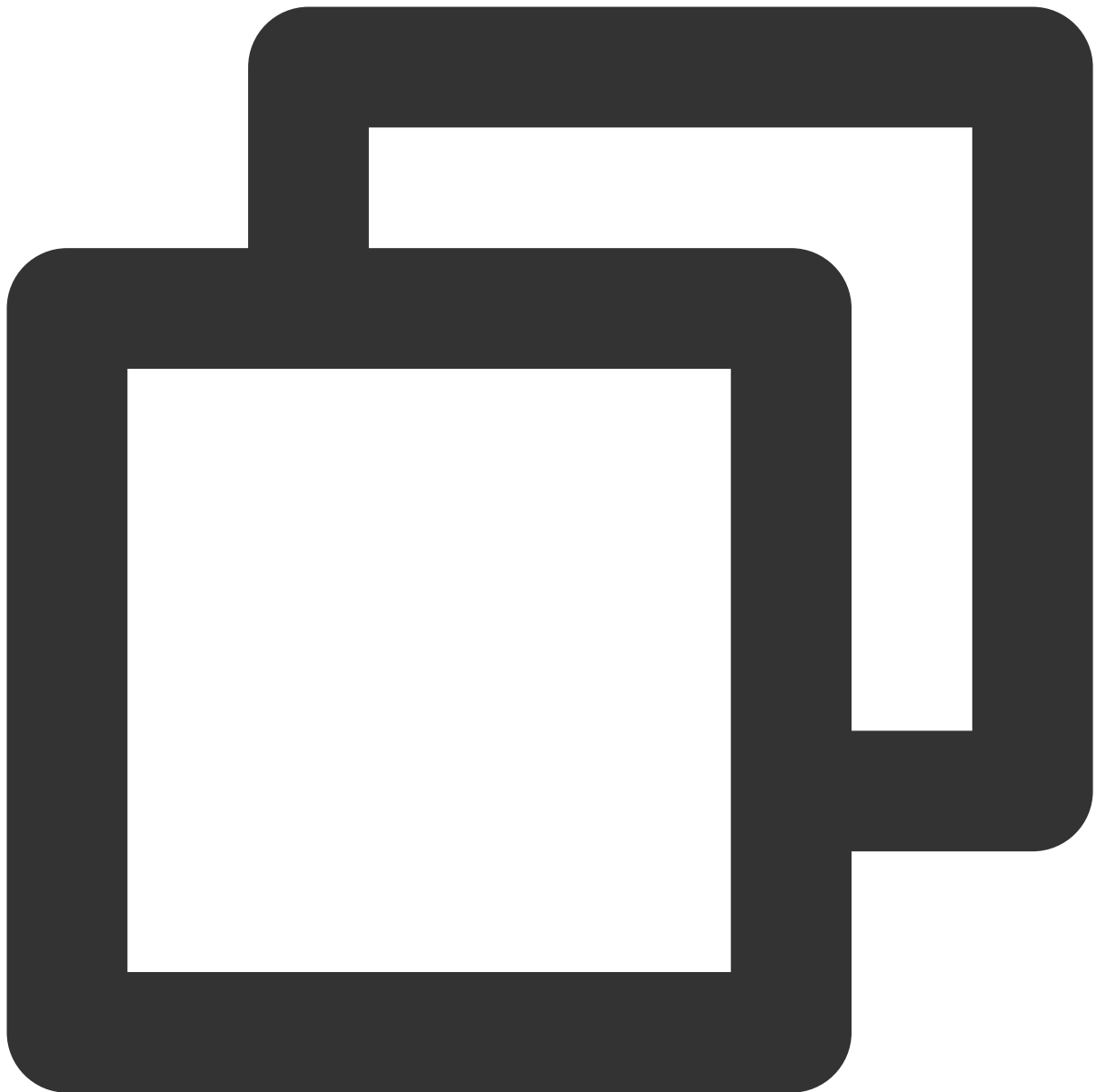
You can check the number of table files and snapshot information in the following way.



```
SELECT COUNT(*) FROM [catalog_name.][db_name.]table_name$files;  
SELECT COUNT(*) FROM [catalog_name.][db_name.]table_name$snapshots;
```

For example:





```
SELECT COUNT(*) FROM `DataLakeCatalog`.`db1`.`tb1$files`;  
SELECT COUNT(*) FROM `DataLakeCatalog`.`db1`.`tb1$snapshots`;
```

When the number of table files and snapshots is excessive, refer to the document [Enable data optimization](#) to activate the data governance feature.

### Data Governance Effectiveness

After enabling data governance, there was a significant improvement in query efficiency. For example, the table below compares the query time before and after merging files. The experiment used a 16CU Presto, with a data volume of

14M, 2921 files, and an average of 0.6KB per file.

Executed Statement	Merge Files	Number of files	Number of records	Query time consumption	Effect
SELECT count(*) FROM tb	No	2921 items	7895 entries	32s	93% faster speed
SELECT count(*) FROM tb	Yes	1 item	7895 entries	2s	

## Partition

Partitioning enables the classification and storage of related data based on column values with different characteristics such as time and region. This significantly reduces scan volume and improves query efficiency. For more details on DLC external table partitioning, please refer to [Quick Start with Partition Table](#). The table below shows a comparison of query time consumption and scan volume between partitioned and unpartitioned states in a single table with a data volume of 66.6GB, 1.4 billion data records, and an ORC data format. Within it, `dt` is a partition field containing 1,837 partitions.

Query statement	Unpartitioned		Partition		Time Consumption Comparison	Scan Volume Comparison
	Time Consumption	Scan Volume	Time Consumption	Scan Volume		
SELECT count(*) FROM tb WHERE dt='2001-01-08'	2.6s	235.9MB	480ms	16.5 KB	81% Faster	Reduce by 99.9%
SELECT count(*) FROM tb WHERE dt<'2022-01-08' AND dt>'2001-07-08'	3.8s	401.6MB	2.2s	2.8MB	Faster by 42%	Reduce by 99.3%

As can be seen from the above table, partitioning can effectively reduce Query Latency and Scan Volume, but Excessive partitioning may backfire. As shown in the table below.

Query statement	Unpartitioned		Partition		Time Consumption Comparison	Scan Volume Comparison
	Time	Scan	Time	Scan		



	Consumption	Volume	Consumption	Volume		
SELECT count(*) FROM tb	4s	24MB	15s	34.5MB	Slower by 73%	30% More

It is recommended to filter partitions in your SQL statements using the WHERE keyword.

## Cache

In today's trend of Distributed Computing and Compute-Storage Separation, accessing Metadata and Huge Data through Network will be restricted by Network I/O. DLC significantly reduces Response Latency by defaulting to the following Caching Technologies, without the need for your intervention.

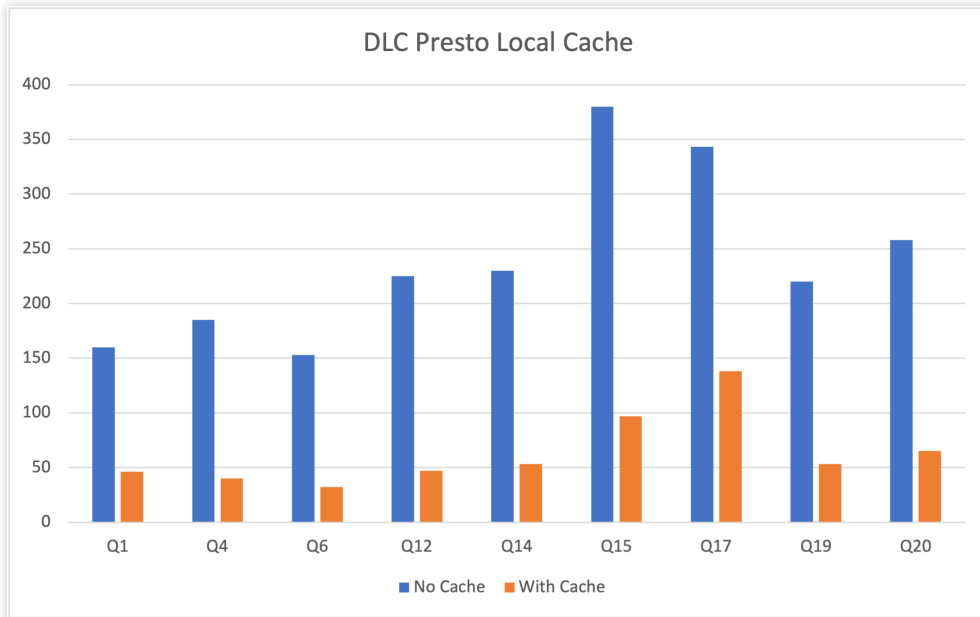
Alluxio: is a Data Orchestration Technology. It provides a Cache, moving data from the Storage Layer to a location closer to Data-Driven Applications, making it more accessible. Alluxio's Memory-First Hierarchical Architecture allows data access to be several orders of magnitude faster than existing solutions.

RaptorX: is a Linker for Presto. It runs on top of storage like Presto, providing Sub-Second Latency. The goal is to offer a Unified, Cost-Effective, Rapid, and Scalable solution for OLAP and Interactive Use Cases.

Result Cache: Caches the same repeated queries, greatly improving speed and efficiency

The DLC Presto engine by default supports Tiered Cache with RaptorX and Alluxio, effectively reducing latency in similar task scenarios within a short period. Both Spark and Presto engines support Result Cache.

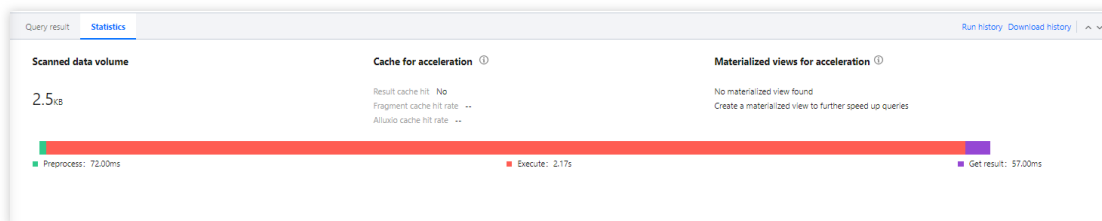
The following table shows TPCH benchmark data in a total data volume of 1TB Parquet files, using 16CU Presto for this test. Since the test focuses on the caching feature, it primarily selects SQLs with significant IO consumption from TPCH. The tables involved mainly include lineitem, orders, customer, etc. The SQLs involved are Q1, Q4, Q6, Q12, Q14, Q15, Q17, Q19, and Q20. The horizontal axis represents the SQL statement, and the vertical axis represents the running time (in seconds).



It's important to note that the DLC Presto engine dynamically loads the cache based on Data Access Frequency. Therefore, cache hits cannot be achieved during the engine's first task execution after startup, leading to initial performance still being limited by network IO. However, this limitation is significantly mitigated as the **Number of executions** increases. The table below shows a performance comparison of three queries in a presto 16cu cluster.

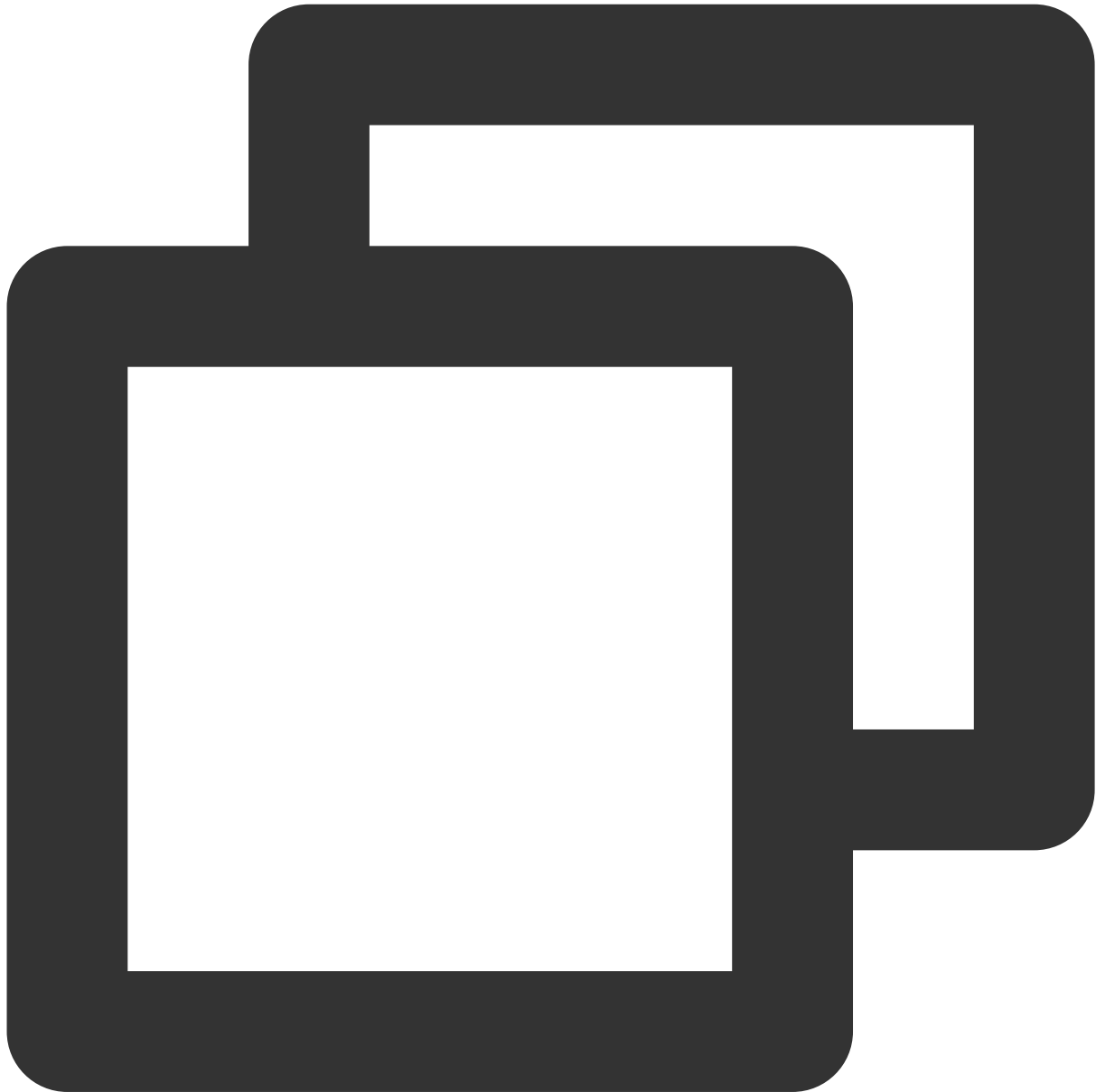
Query statement	Query	Time Consumption	Data Scan Volume
SELECT * FROM table_name WHERE udid='xxx';	First Query	3.2s	40.66MB
	Second Query	2.5s	40.66MB
	Third Query	1.6s	40.66MB

You can view the cache hit ratio of executed SQL tasks in the 'Data Exploration' feature of the DLC Console.



## Index

Compared to external tables, the table creation method using internal tables + indexes will significantly reduce both time and scan volume. For more detailed information about creating tables, please refer to [Data Table Management](#). After creating a table, build an index before inserting based on the business usage frequency, after `WRITE ORDERED BY` for the indexed fields.



```
alter table `DataLakeCatalog`.`dbname`.`tablename` WRITE ORDERED BY udid;
```

The table below shows a comparison of query performance on external and internal tables (with indexes) in a presto 16cu cluster

Table Types	Query	Time	Data Scan Volume
-------------	-------	------	------------------

		Consumption	
Exterior	First Query	16.5s	2.42GB
	Second Query	15.3s	2.42GB
	Third Query	14.3s	2.42GB
Inner Table (Index)	First Query	3.2s	40.66MB
	Second Query	2.5s	40.66MB
	Third Query	1.6s	40.66MB

It is evident from the table that, compared to external tables, the table creation method using inner tables + indexes significantly reduces both time and scan volume. Moreover, due to cache acceleration, **the execution time will also decrease as the number of executions increases.**

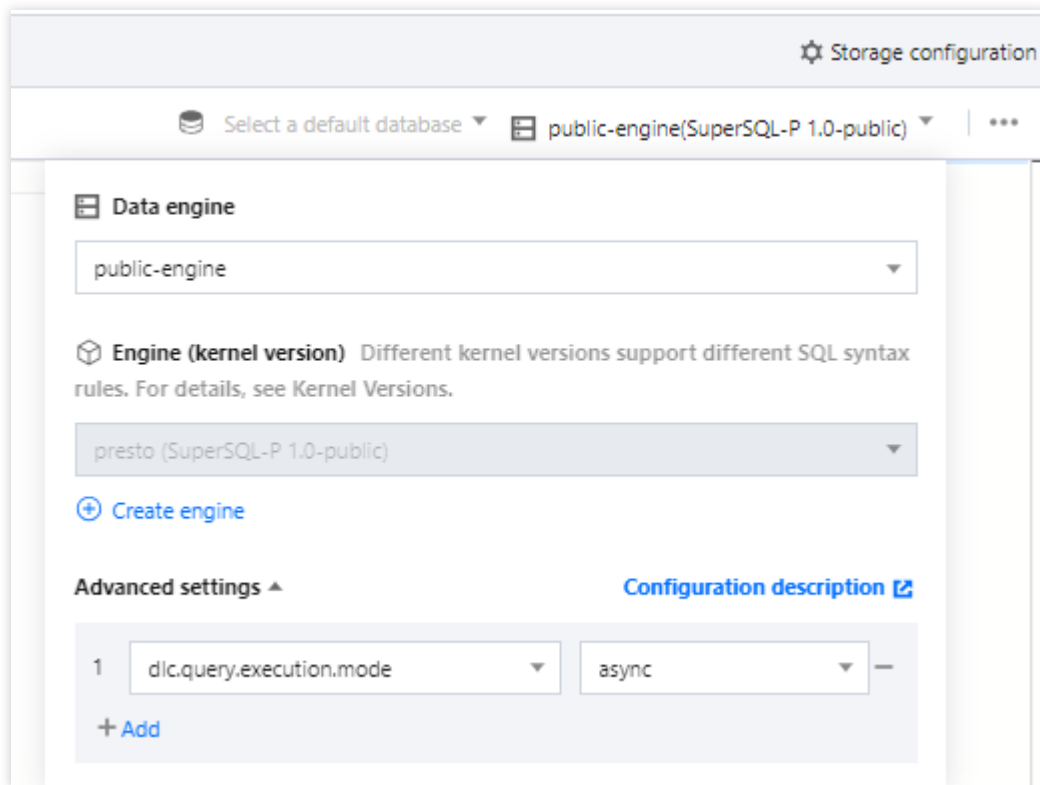
## Synchronous Query and Asynchronous Query

DLC has undergone special optimization for BI scenarios. It can be set to synchronize mode or asynchronous mode (supports only the Presto engine) by configuring the engine parameter `dlc.query.execution.mode`. The value descriptions are as follows.

**async (default):** In this mode, tasks complete full query calculations, and the results are saved to COS before being returned to the user, allowing users to download the query results after the query has completed.

**sync:** Under this mode, it is not necessary to perform full calculation. Once partial results are available, they are directly returned to the user by the engine, without being saved to COS. Thus, users can achieve lower query latency and reduced time consumption, but the results are only stored in the system for 30s. It is recommended to use this mode when complete query results from COS are not needed, but lower query latency and time consumption are desired, such as during the query exploration phase or for BI result presentation.

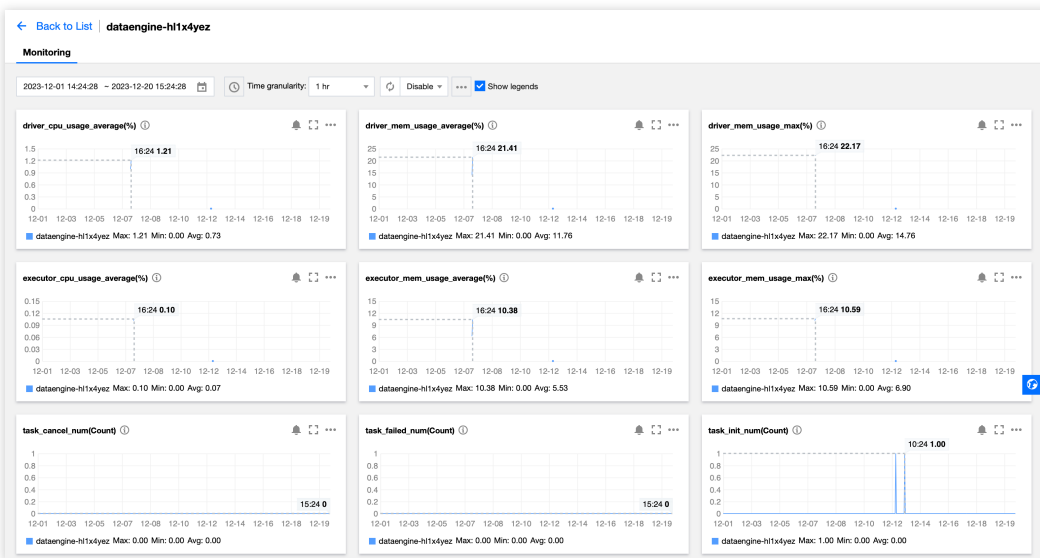
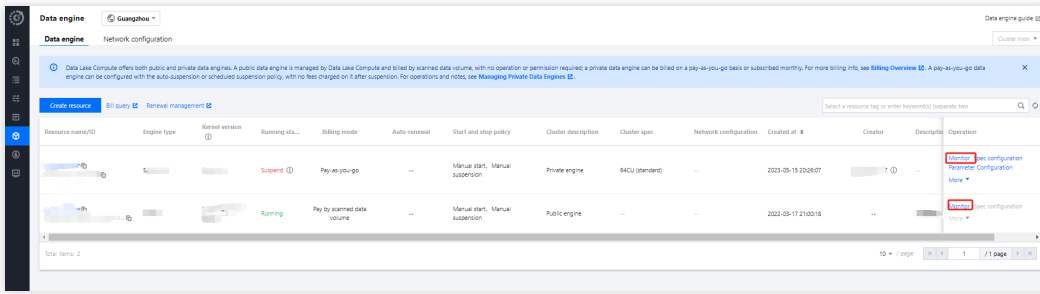
**Configuration Method:** After selecting the Data Engine, it supports parameter configuration for the data engine. After selecting the data engine, click add in Advanced settings to configure.



## Resource Bottleneck

To assess whether resources have reached a bottleneck, DLC provides monitoring of engine resources such as CPU, memory, cloud disks, and network. You can adjust resource specifications according to business scale. For adjustments, refer to the [Adjustment Configuration Fee Explanation](#). Steps to view engine resource usage are as follows:

1. Open the Data Engine Tag page on the left.
2. Click the **Monitoring** button on the right side of the respective engine.
3. Navigate to TCOP, where you can see all monitoring metrics as shown below. For detailed operations and monitoring metrics, refer to Data Engine Monitoring. You can also configure alarms for each metric. For a detailed introduction, refer to [Monitoring and Alarm Configuration](#).



## Other Factors

### Adaptive Shuffle

To enhance stability, DLC by default enables Adaptive Shuffle, which supports regular shuffle with limited local disk space while ensuring stability in scenarios of large shuffle and data skew. Advantages of Adaptive Shuffle include:

### Cluster cold start

The DLC supports the automatic or manual suspension of a cluster. After the suspension, no charges are incurred. Therefore, the message "Queuing" may be displayed when a task is executed for the first time after the cluster is started, because resources are being pulled up during the cold start of the cluster. If you submit tasks frequently, it is recommended to [Purchase a package year/month cluster](#), which does not have a cold start and can quickly execute tasks at any time.

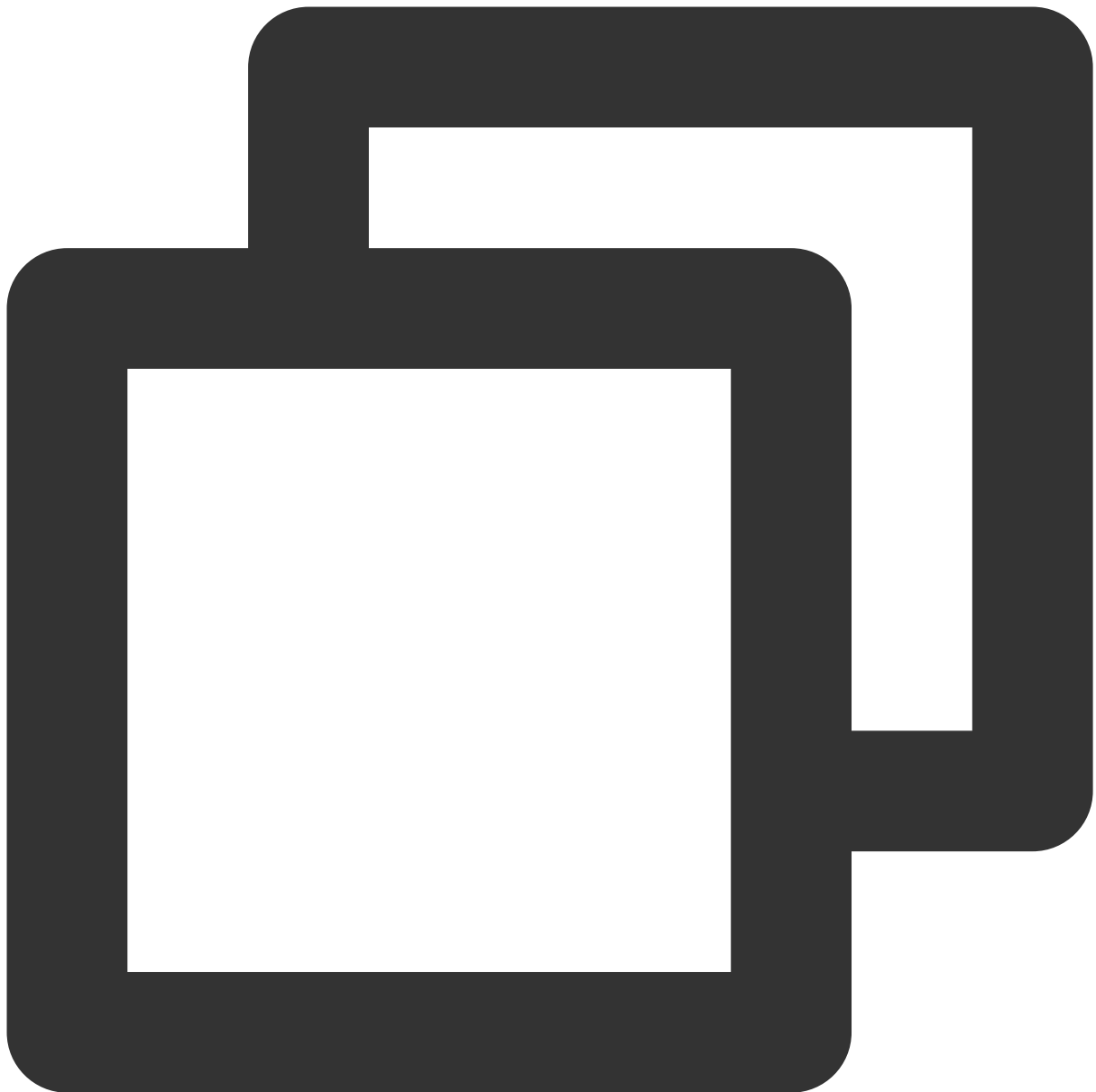
# UDF Function Development Guide

Last updated : 2024-07-31 18:03:19

## UDF Description

Users can write UDF functions, package them into JAR files, and then use them in query analysis by defining them as functions in Data Lake Compute. Currently, DLC's UDFs are in HIVE format, inheriting from `org.apache.hadoop.hive.ql.exec.UDF` and implementing the `evaluate` method.

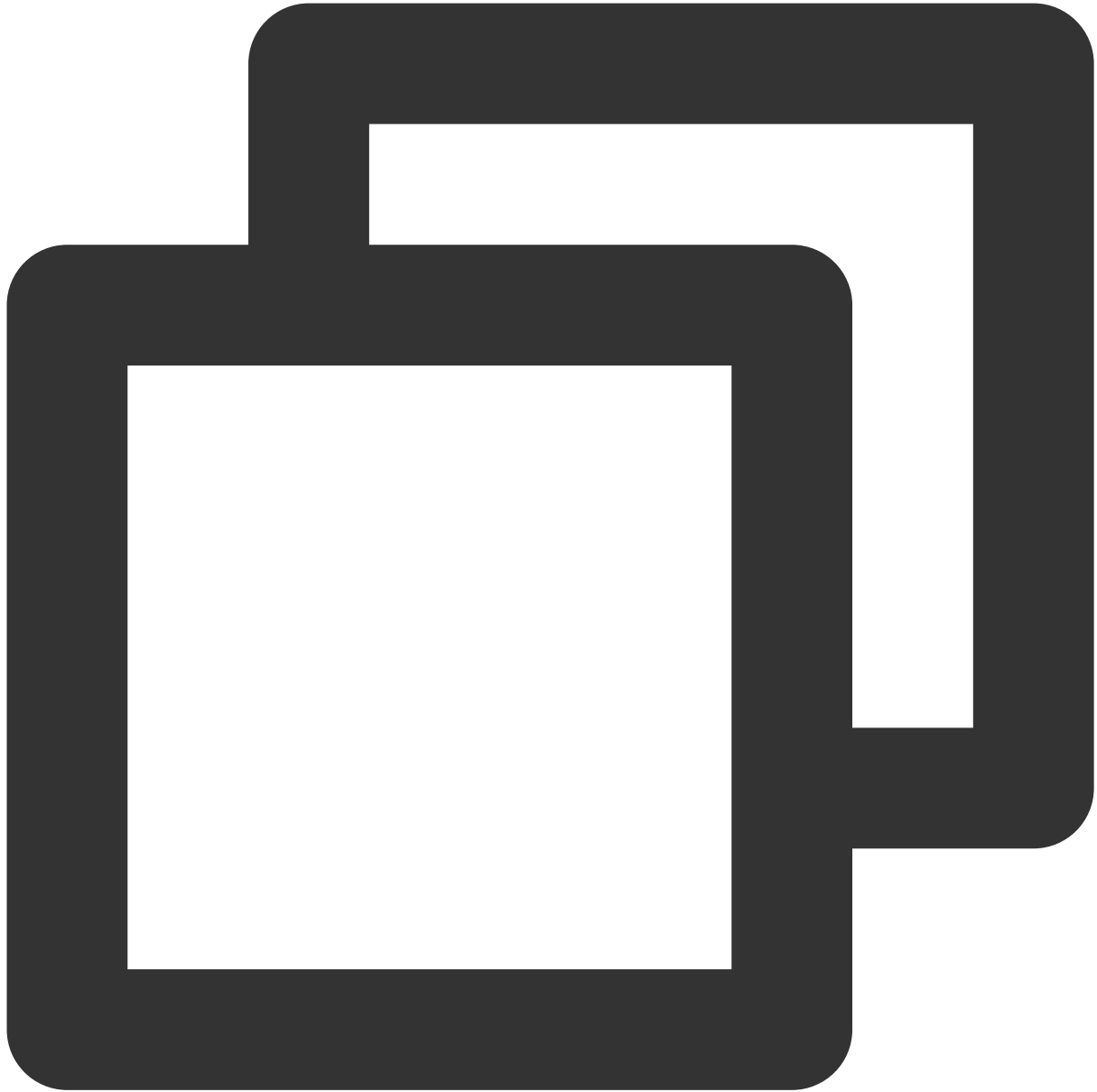
Example: Simple Array UDF Function.



```
public class MyDiff extends UDF {  
    public ArrayList<Integer> evaluate(ArrayList<Integer> input) {  
        ArrayList<Integer> result = new ArrayList<Integer>();  
        result.add(0, 0);  
        for (int i = 1; i < input.size(); i++) {  
            result.add(i, input.get(i) - input.get(i - 1));  
        }  
        return result;  
    }  
}
```



Reference for POM file:



```
<dependencies>
  <dependency>
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-log4j12</artifactId>
    <version>1.7.16</version>
    <scope>test</scope>
  </dependency>
  <dependency>
    <groupId>org.apache.hive</groupId>
```

```
<artifactId>hive-exec</artifactId>
<version>1.2.1</version>
</dependency>
</dependencies>
```

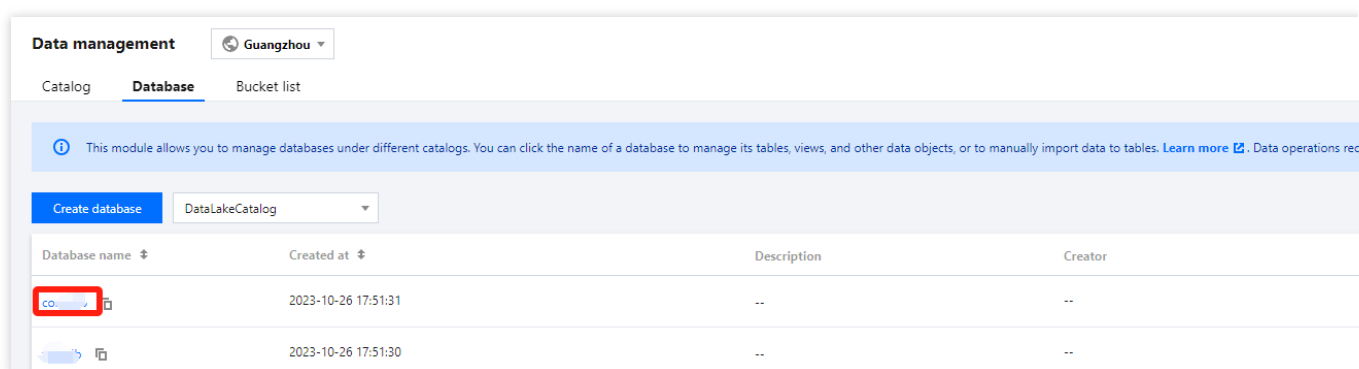
## Creating function

### Note:

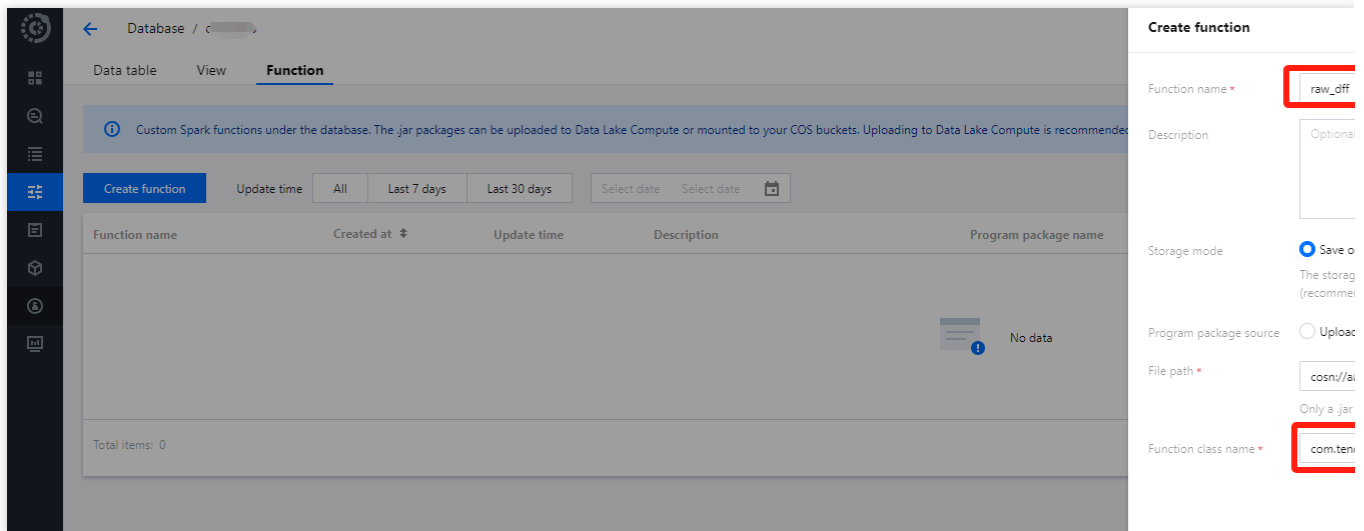
If you are creating a UDAF/UDTF function, you need to add the `_udaf/_udtf` suffix to the function name accordingly.

If you are familiar with SQL syntax, you can create a function by executing the CREATE FUNCTION syntax via **Data Exploration**, or by using the visual interface. The process is as follows:

1. Log in to the [Data Lake Compute Console](#) and select the service region.
2. Enter **Data Management** through the left sidebar, select the database for the function you need to create. If you need to create a new database, refer to [Data Catalog and DMC](#).



3. Click **Function** to enter the function management page.
4. Click **Create Function** to proceed with creation.



UDF's application package can be uploaded locally or a COS path can be selected (requires COS-related permissions), for instance, creating by selecting a COS path.

Function Class Name includes "Package Information" and "Function Execution Class Name".

## Function Usage


1. Log in to the [Data Lake Computing Console](#) and select the service region.
2. Enter Data Exploration via the left navigation menu, select a Compute Engine, and then you can use SQL to invoke the function.

Running    Complet ▾    Save    Refresh    Format    SQL

```
1 select raw_diff(Array[1,2,3])
```

Query result    Statistics

Task ID    SQL details    Export    Suggestions [🔗](#)



# Materialized View

Last updated : 2024-07-31 18:03:31

## Note:

Currently, DLC materialized views only support the SparkSQL and Presto engines.

A Materialized View is a special object in a database, which is a pre-calculated and stored query result set.

Materialized views can provide fast query performance when dealing with large amounts of data and complex queries.

While materialized views improve query performance, they also introduce storage and compute costs. We recommend using materialized views in the following scenarios:

Source table changes are infrequent

Compared to the source table, the fields and result quantity in the materialized view table are significantly reduced

DLC supports both regular materialized views and mapped materialized views. Below is an introduction and a complete set of usage examples. The supported syntax list can be found in [Materialized View Syntax](#).

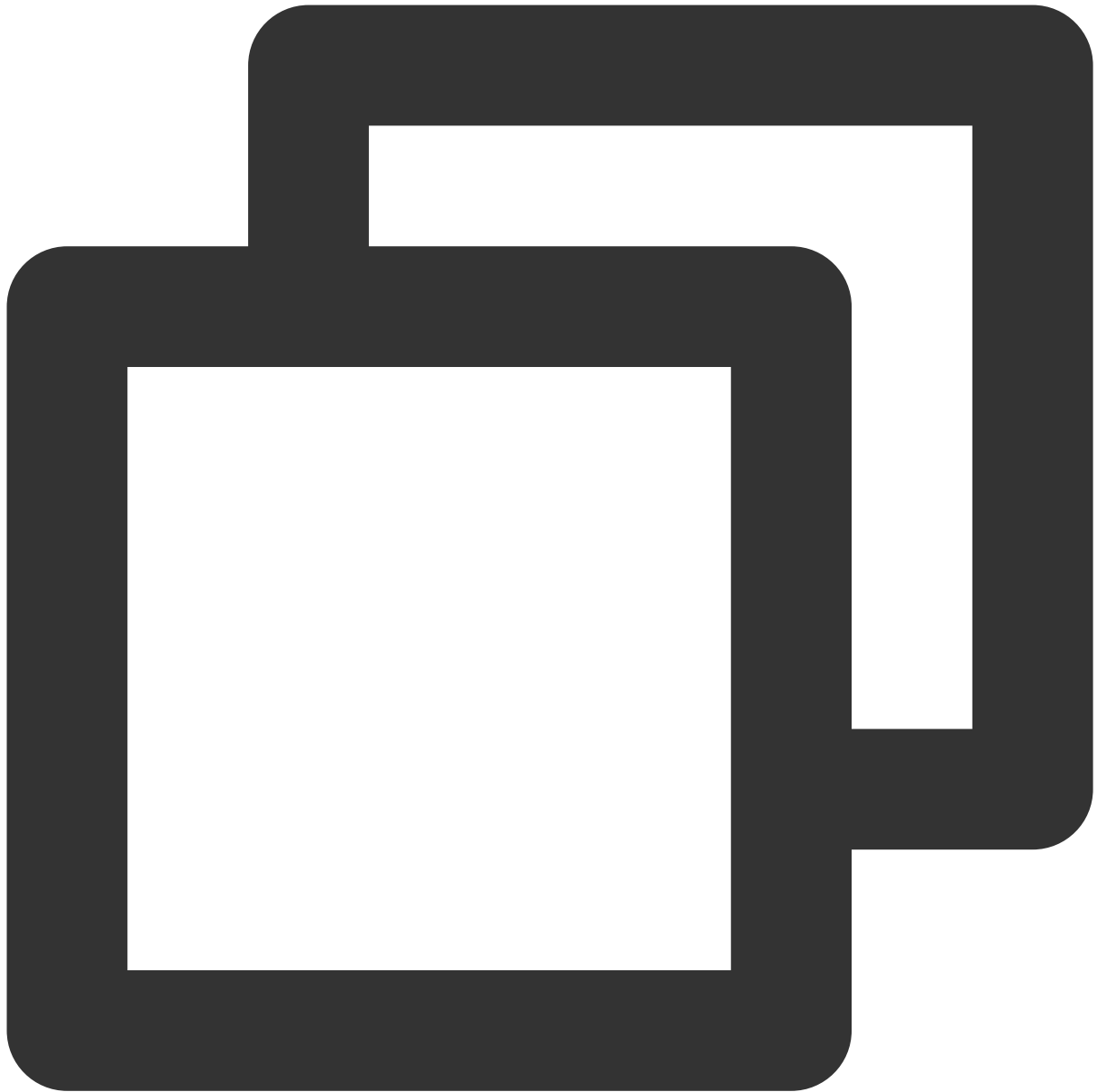
## Regular Materialized View

The basic usage process of a regular materialized view includes creation, refresh, and use.

The following is a complete process example based on the Presto Engine.

### Data Preparations

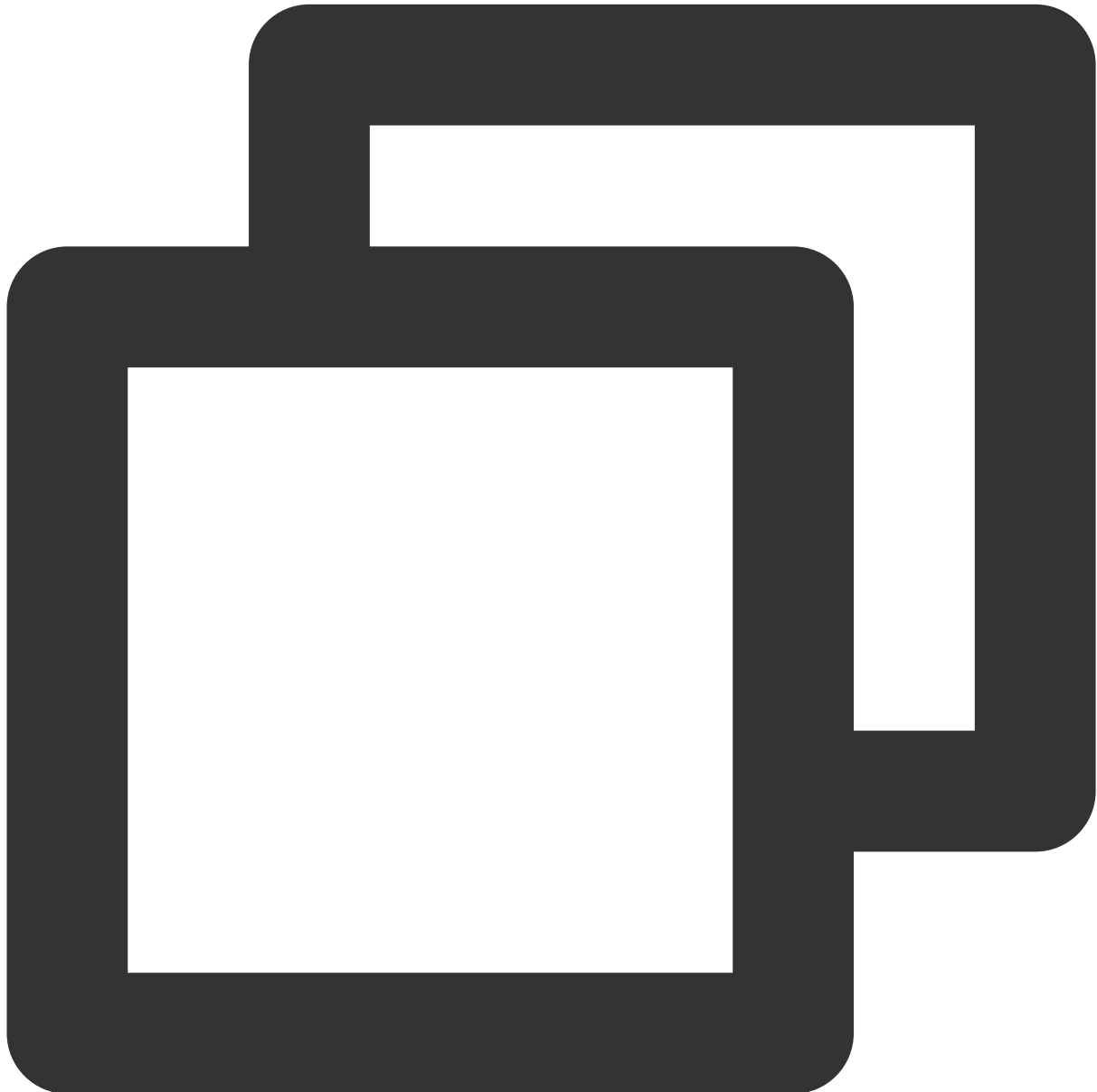
Execute SQL to create a database table and insert data. The following statement creates a table named `student` .



```
CREATE DATABASE IF NOT EXISTS mv_test3;
create table student(id int, name string, score int);
insert into student values (1,'zhangsan', 90);
insert into student values (2,'lisi', 100);
insert into student values (3,'wangwu', 80);
insert into student values (4,'zhaoliu', 30);
select * from student order by id;
```

## Creating a Regular Materialized View

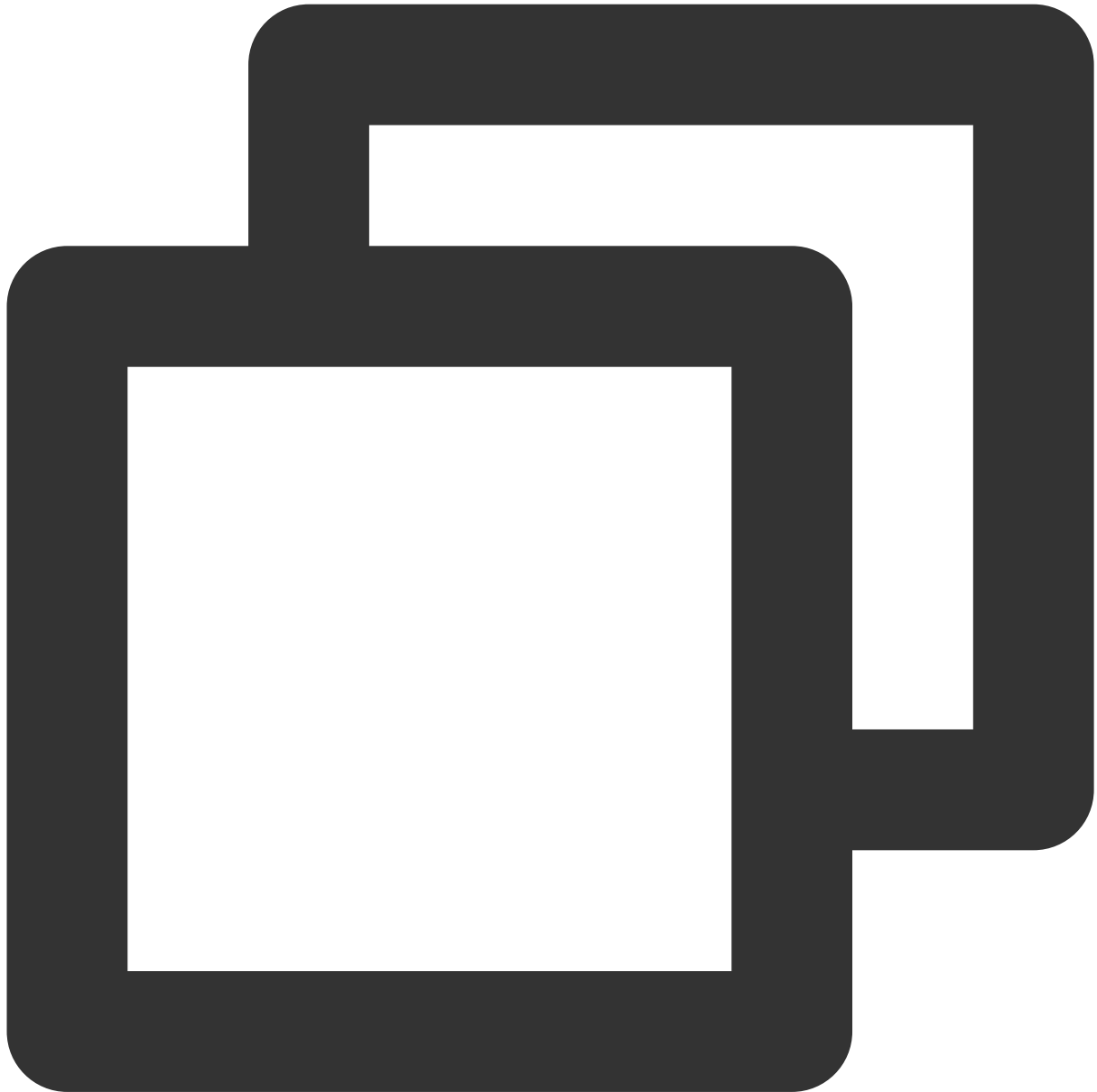
Use the `CREATE MATERIALIZED VIEW` statement to create a materialized view. Specify the name of the materialized view and the query statement, and optionally specify the source table and condition of the query. In the following example, a simple `SELECT` statement is used to select all scores from the table `student` and perform a summation operation on them. Then, this summation result serves as the content of the materialized view `mv_student_sum`.



```
CREATE MATERIALIZED VIEW mv_student_sum AS (  
  select sum(score) from student  
);
```

## Viewing Materialized View Details

Use the `DESCRIBE MATERIALIZED VIEW` statement to view the detailed information of the materialized view, including the name, query statement, and refresh status, among others.



```
DESCRIBE MATERIALIZED VIEW mv_student_sum;
```



Query result    Statistics

Task ID    SQL details    Export    Suggestions [🔗](#)

Query time: 216.00ms    Scanned data volume: 0 B    Billable scanned volume: 34.0 MB [①](#)

1 entries in total (up to 1,000 entries shown in the console)

#	col_name	data_type	comment
a		int	
b		string	

MaterializedView Detail:

state    NORMAL

mvType    SINGLE

viewOriginalText    SELECT \*

FROM `tbl\_2`

autoRewrite    ENABLE

dataLatest    FRESH

freshType    RUNTIME

updateType    FULL

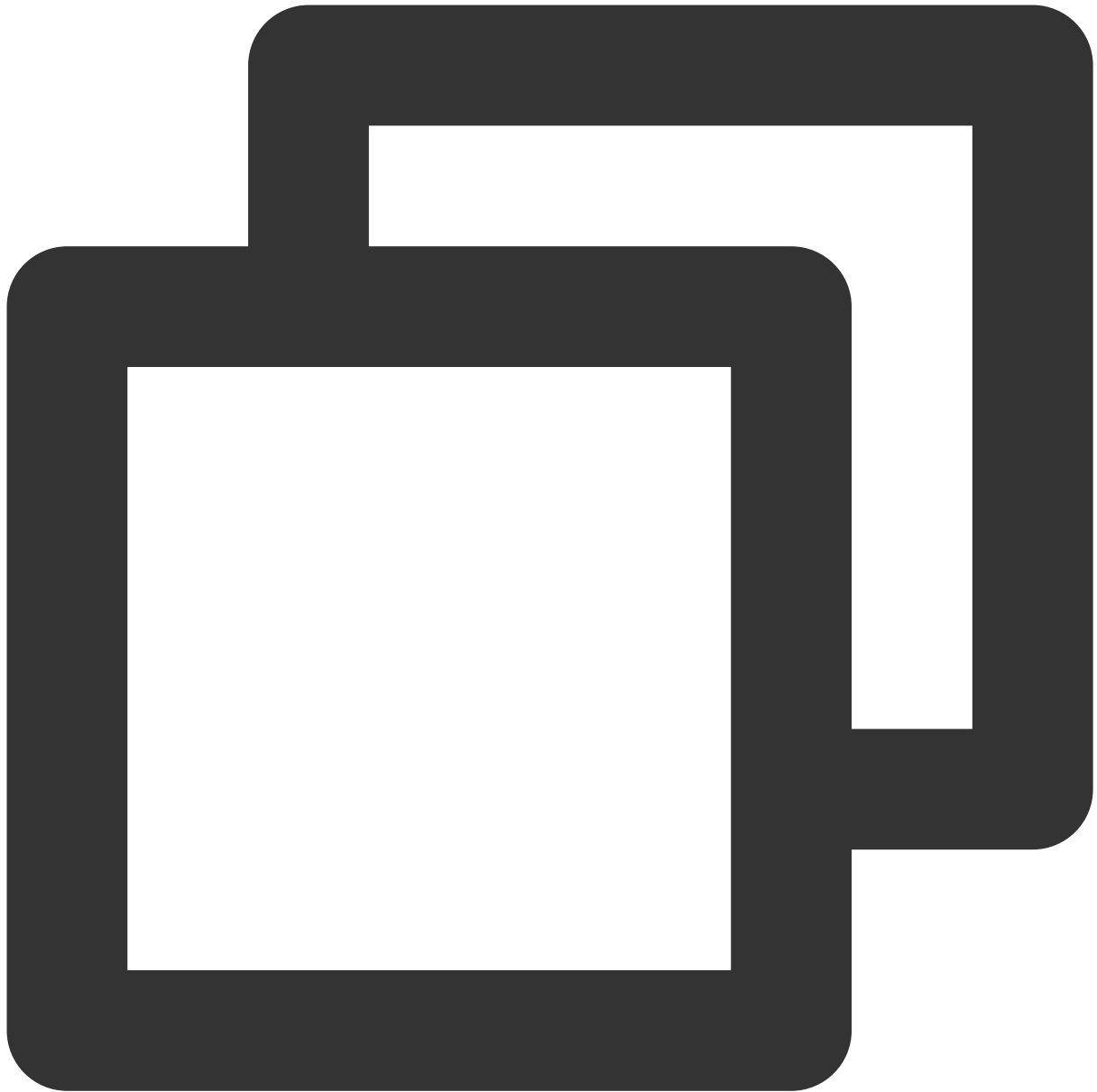
createTime    Wed Oct 25 21:01:46 CST 2023

modifiedTime    Wed Oct 25 21:01:49 CST 2023

## Manual Refresh of Materialized View

Use the `REFRESH MATERIALIZED VIEW` statement to manually refresh the data of the materialized view.

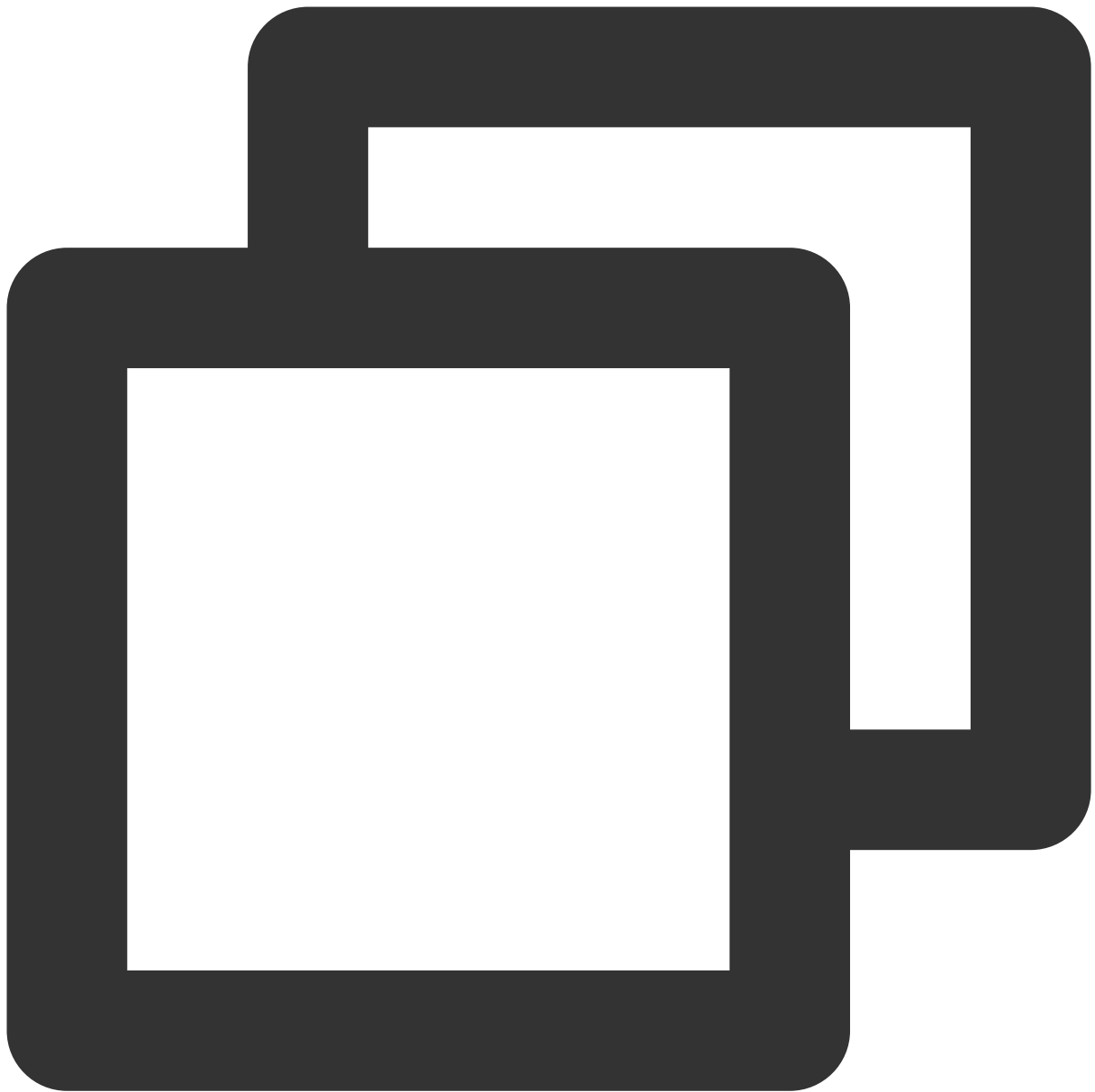
This demonstration is just for illustrative purposes; in most cases, you do not need to manually refresh the materialized view. The view will automatically refresh if the SQL hits a source table that has changed.



```
REFRESH MATERIALIZED VIEW mv_student_sum;
```

### Viewing the Execute Task List of Materialized View

Use the `SHOW MATERIALIZED VIEW JOBS` statement to view the execute task list of the materialized view, enabling you to understand the refresh history and status of the materialized view.

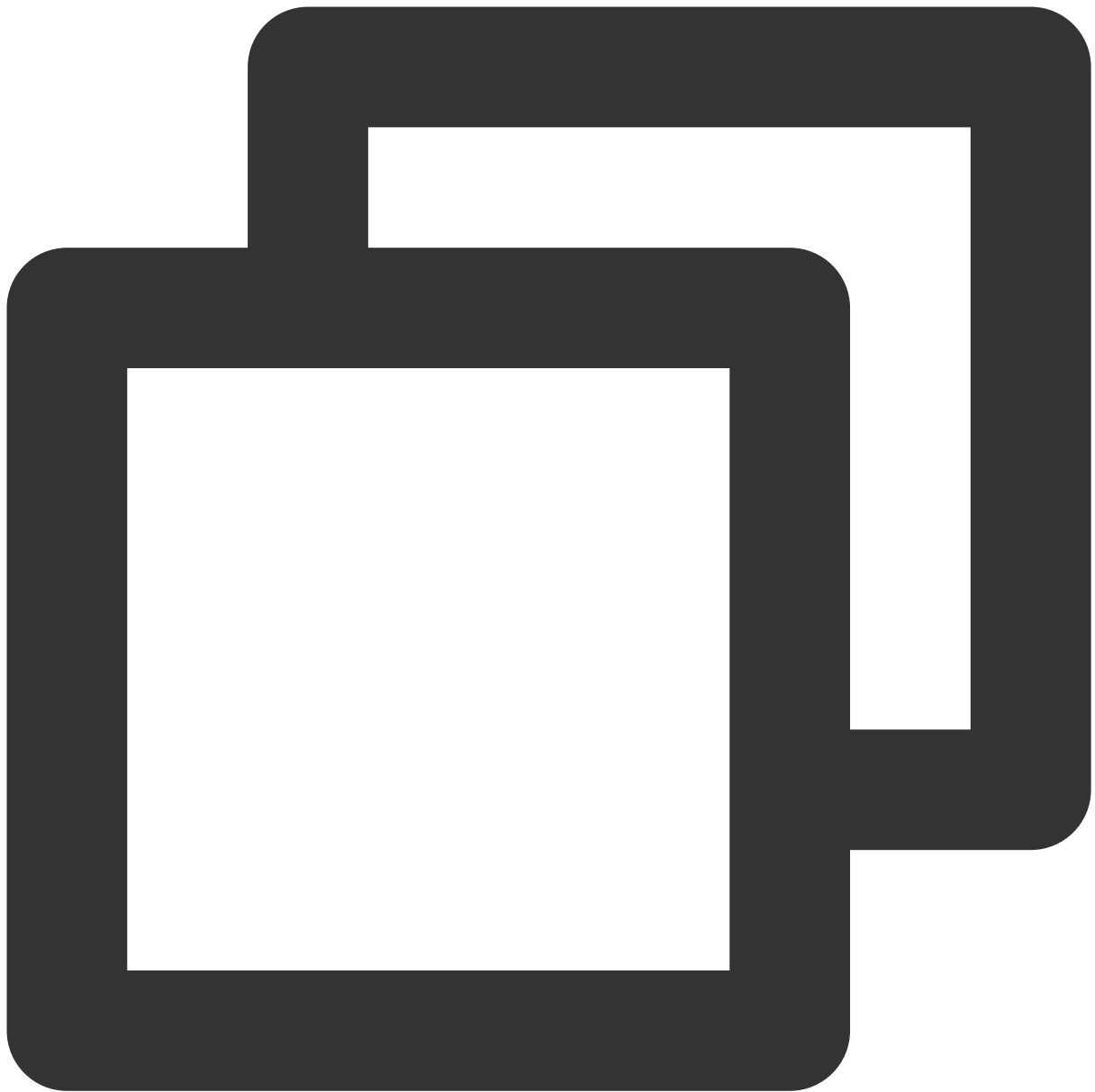


```
SHOW MATERIALIZED VIEW JOBS IN mv_student_sum;
```

Query result		Statistics		
<a href="#">Task ID</a> <a href="#">SQL details</a> <a href="#">Export</a> <a href="#">Suggestions</a> <a href="#">🔗</a>				
Query time: 144.00ms Scanned data volume: 0 B Billable scanned volume: 34.0 MB <a href="#">①</a>				
1 entries in total (up to 1,000 entries shown in the console) <a href="#">Copy</a> <a href="#">🔗</a>				
xid	taskId	state	buildType	execEngine
1429befafd553e8c9aa8	8100254009bc421	FINISHED	CREATE	SPARK

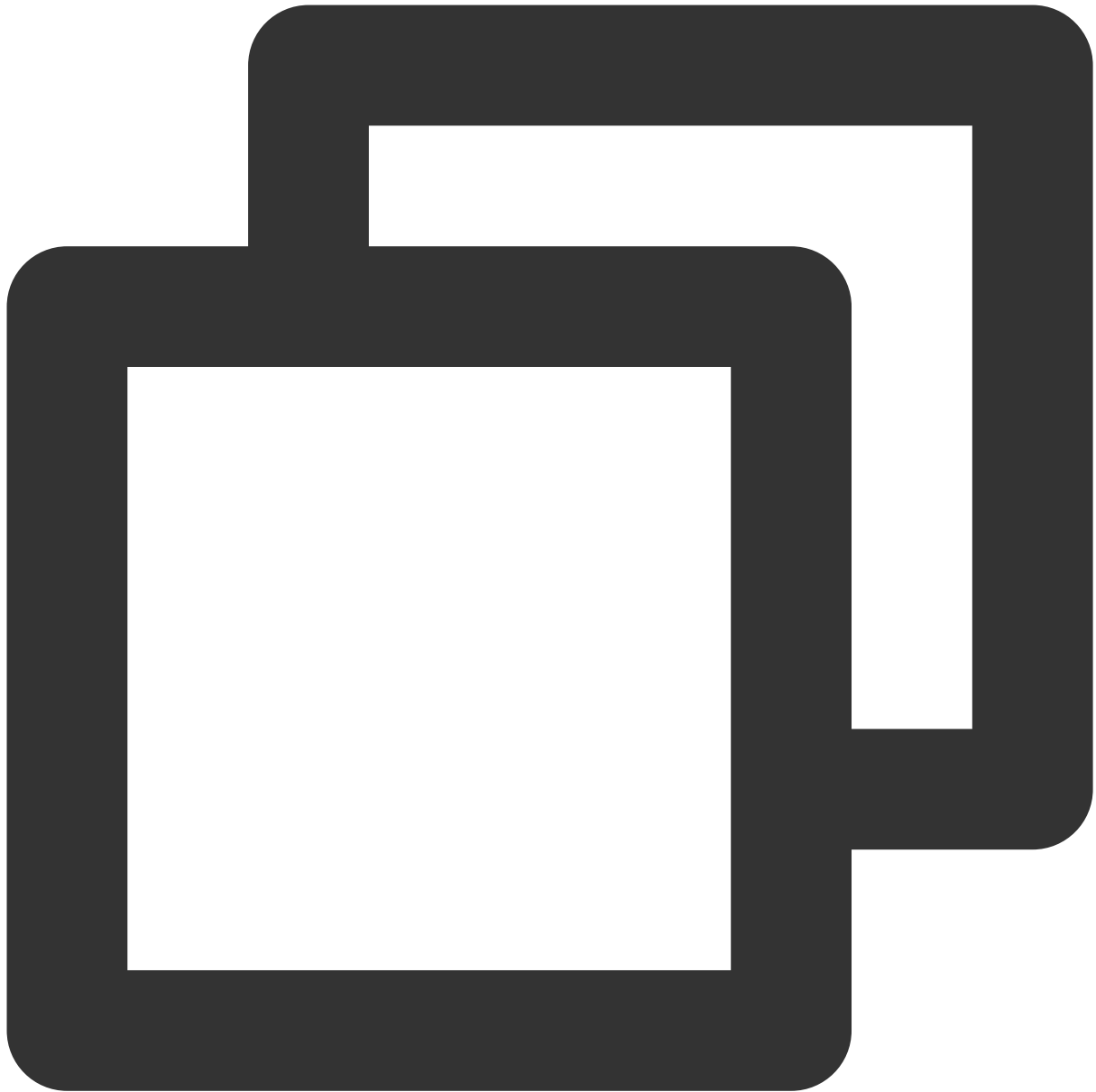
## SQL Rewrite Execution

When querying data with the SELECT statement, it is expected to be automatically rewritten and hit the materialized view. You can check if it was automatically rewritten to the materialized view through the statistics in the query results.



```
select sum(score) from student;
```

## Dropping Materialized View



```
DROP MATERIALIZED VIEW mv_student_sum;
```

## Mapped Materialized View

Mapped Materialized View is a special type of materialized view that is associated with an existing table through mapping. By using a mapped materialized view, you can associate the query results of the materialized view with the data of an existing table, thereby optimizing the query performance of the existing table.

## Limit

Materialized views have the following restrictions compared to ordinary materialized views:

Mapped Materialized Views do not support refresh operations, meaning it's not possible to refresh the data of the materialized view through the REFRESH MATERIALIZED VIEW statement. As a result, the data of the materialized view can only remain consistent with that of the mapped table and cannot be automatically updated.

Mapped Materialized Views do not perform automatic SQL rewriting, meaning query statements are not automatically converted to use the materialized view. It is necessary to manually specify the query statements that use the materialized view.

When a Mapped Materialized View is deleted, only the association relationship with the mapped table is removed, not the mapped table itself. The mapped table still exists and can continue to be used.

## Recommended scenarios

It is recommended to use Mapped Materialized Views in the following scenarios:

When there is already a large table with low query performance, query performance can be optimized through the use of Mapped Materialized Views.

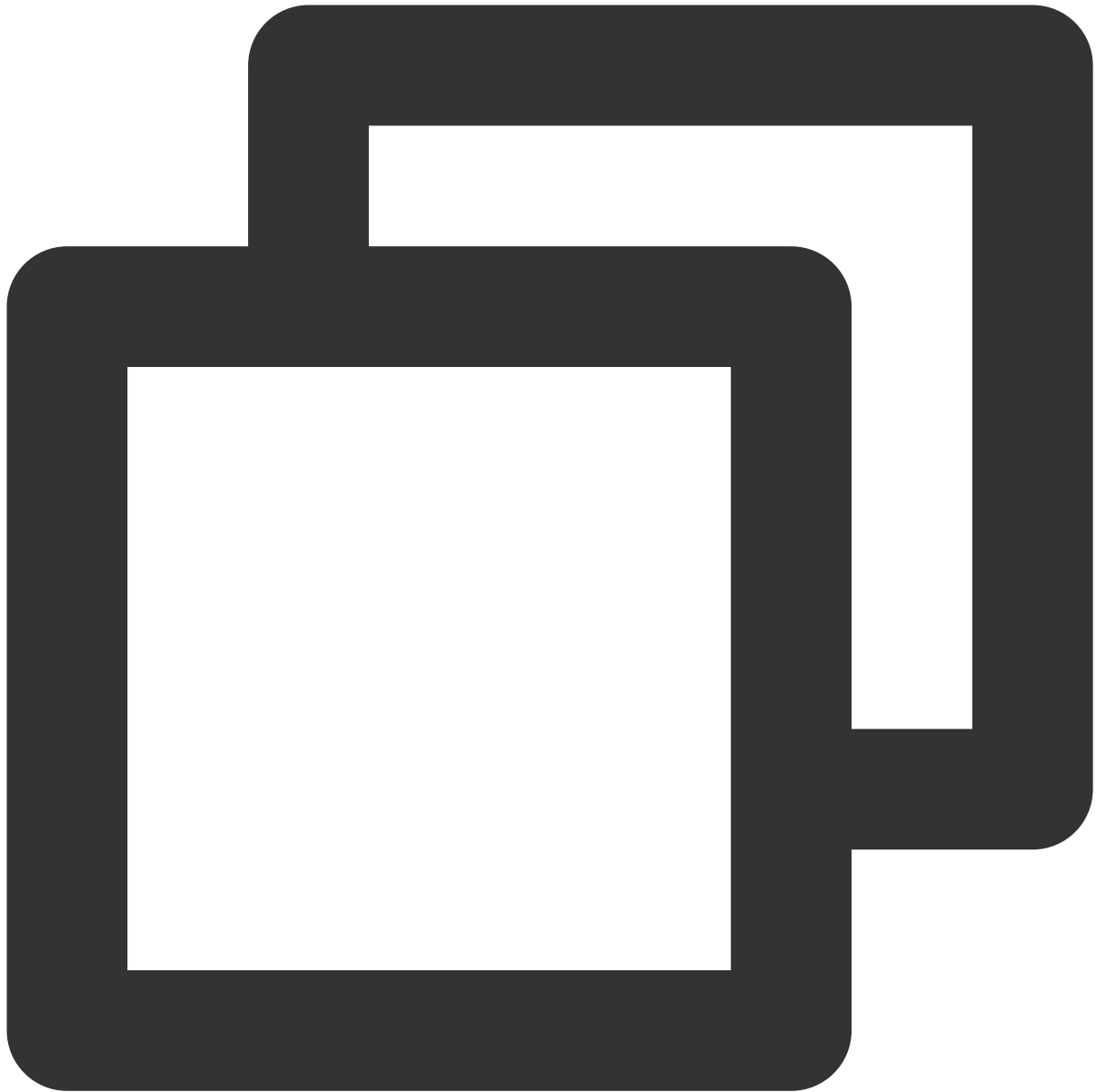
When it is necessary to keep the data of the materialized view consistent with that of an existing table, and there is no need for automatic refresh of the materialized view, Mapped Materialized Views can be used.

## Iceberg type of source table

When an Iceberg table is the source table, a complete example is as follows:

### Creating a Mapped Materialized View based on CTAS

The mapped materialized view needs to maintain consistent naming with the table to be mapped. The following example first creates a table based on CTAS for the creation of the Mapping MV. The data preparation can refer to the section on data preparation in the complete example of an ordinary materialized view.

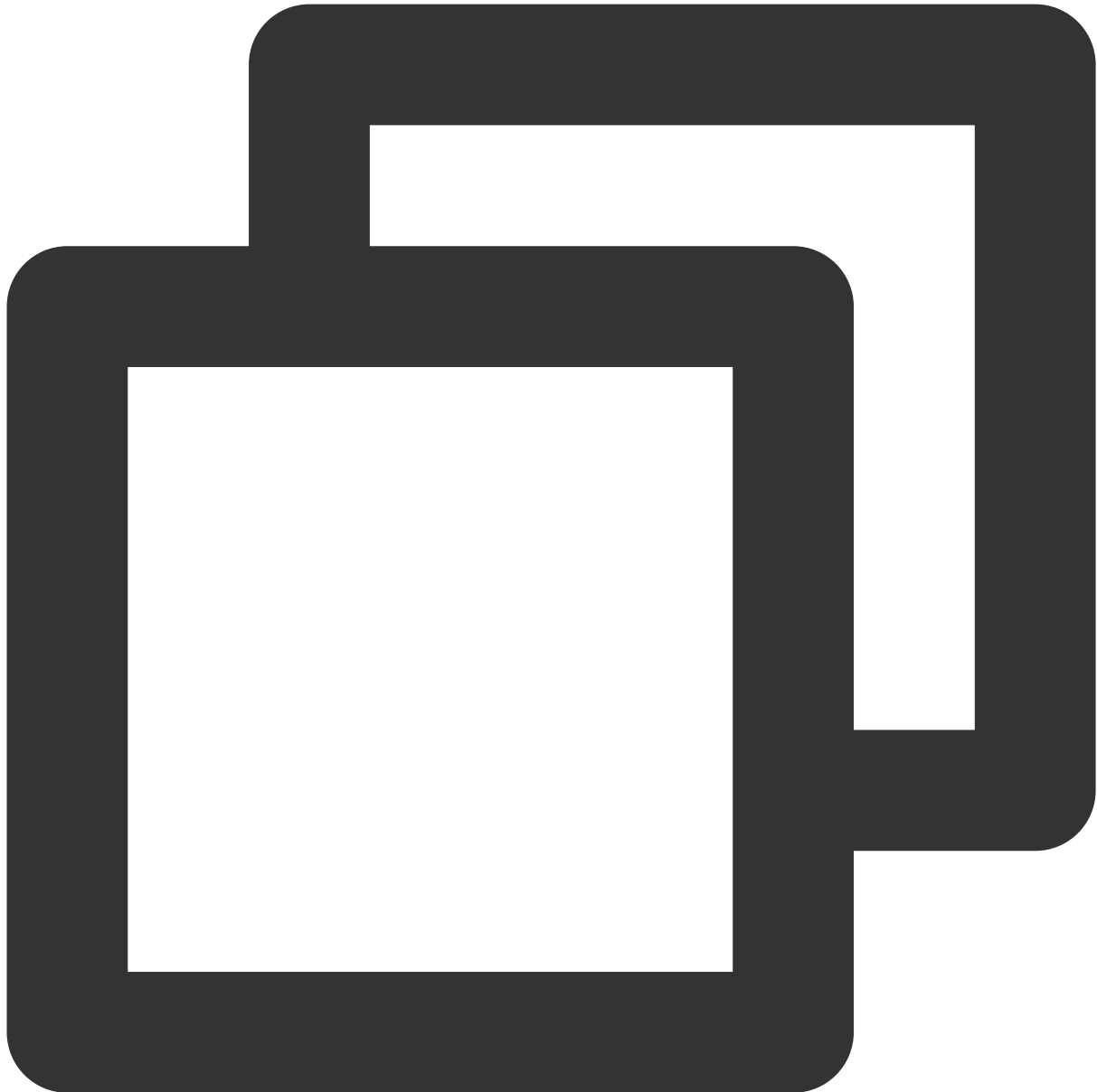


```
CREATE TABLE link_mv_student AS (  
  select sum(score) from student  
)  
--Create Mapped Materialized View: Use the CREATE MATERIALIZED VIEW statement to cr  
--When creating the materialized view, use the WITH META LINK clause and specify th  
CREATE MATERIALIZED VIEW link_mv_student WITH META LINK AS (  
  select sum(score) from student  
)
```

### View Mapping Materialized View



Using the DESCRIBE MATERIALIZED VIEW statement, you can view the detailed information of the mapped materialized view, including the name, query statement, and refresh status, among others.



```
DESCRIBE MATERIALIZED VIEW link_mv_student;  
SHOW MATERIALIZED VIEW JOBS IN link_mv_student;
```

### **Mapped Materialized Views do not support refresh operations**

Mapped Materialized Views do not support REFRESH operations, meaning it's not possible to refresh the data of the materialized view through the REFRESH MATERIALIZED VIEW statement. As a result, the data of the materialized

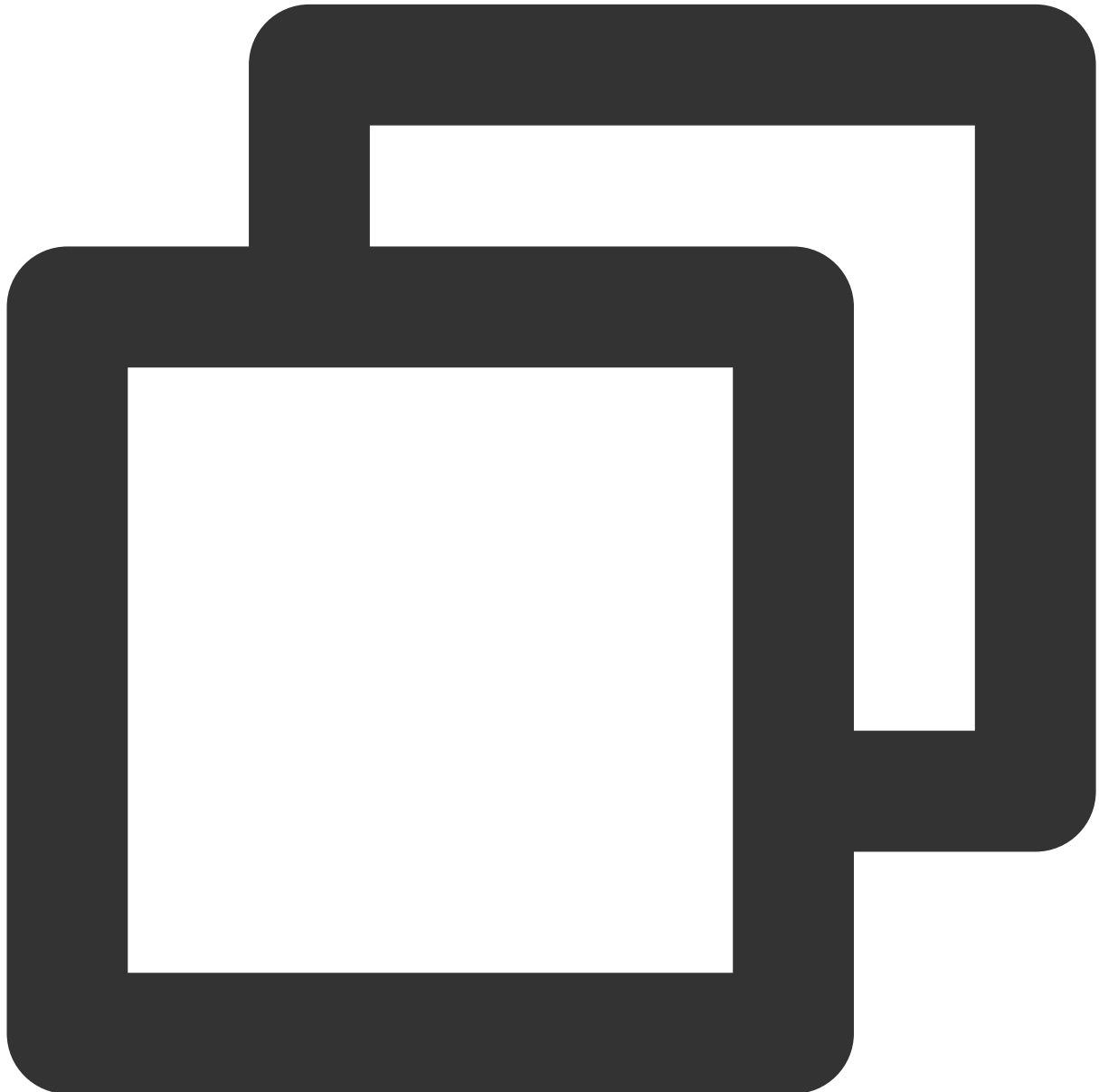
view can only remain consistent with that of the mapping table and cannot be automatically updated.

### SQL Rewriting

Mapped Materialized Views do not automatically perform SQL rewriting of query statements.

Executing `select sum(score) from student;` will not hit the mapped materialized view.

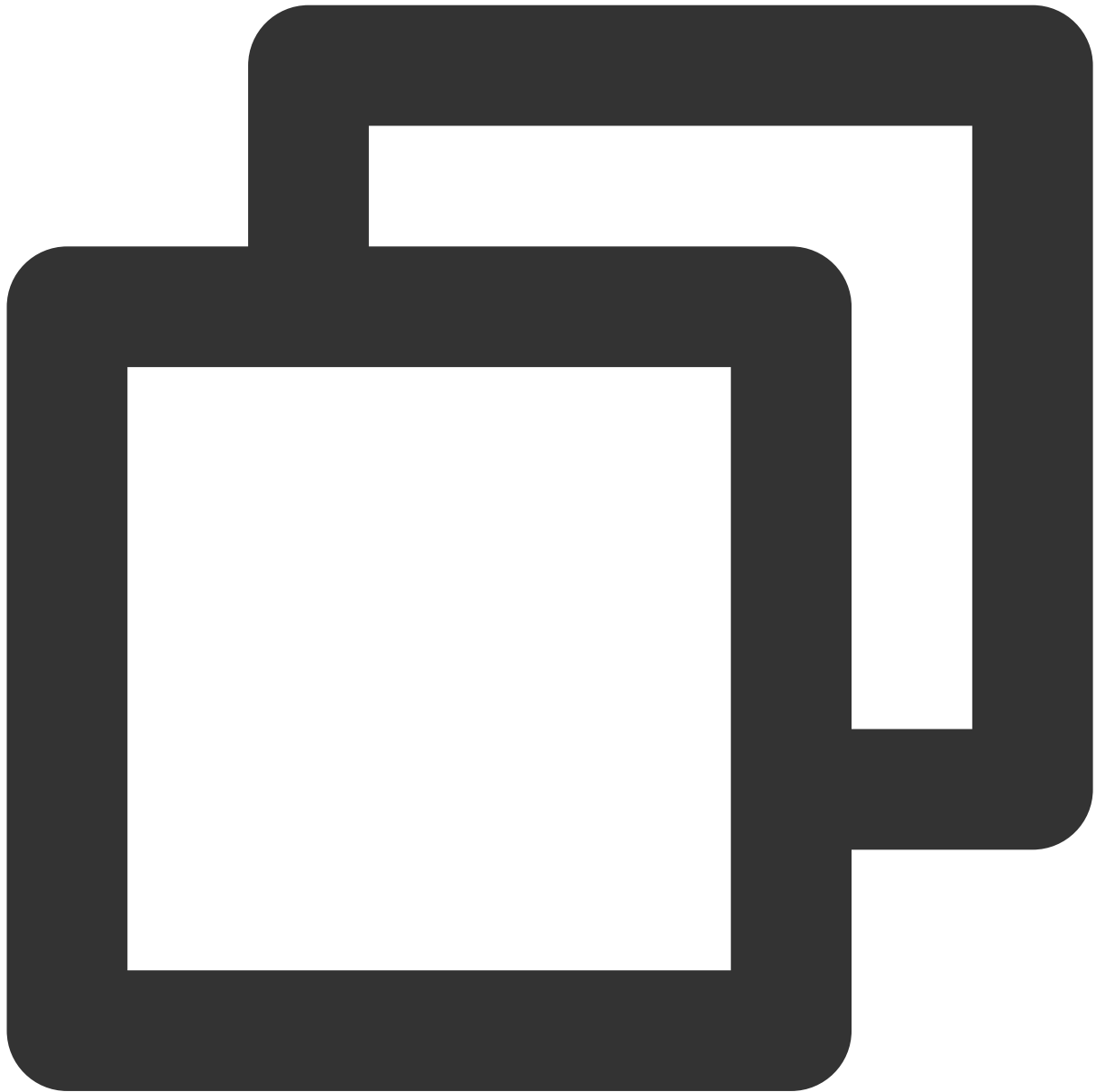
SQL rewriting based on the mapped materialized view can be specified by using the Hint or TaskConf parameters.



```
--Manually Specify the SQL to be rewritten
select /*+ OPTIONS('eos.sql.materializedView.enableRewrite'='true') */
sum(score) from student;
```

## Delete mapped materialized view

Use the DROP MATERIALIZED VIEW statement to delete the mapped materialized view. After deleting the mapped materialized view, only the association with the mapping table will be deleted, the mapping table itself will still exist.



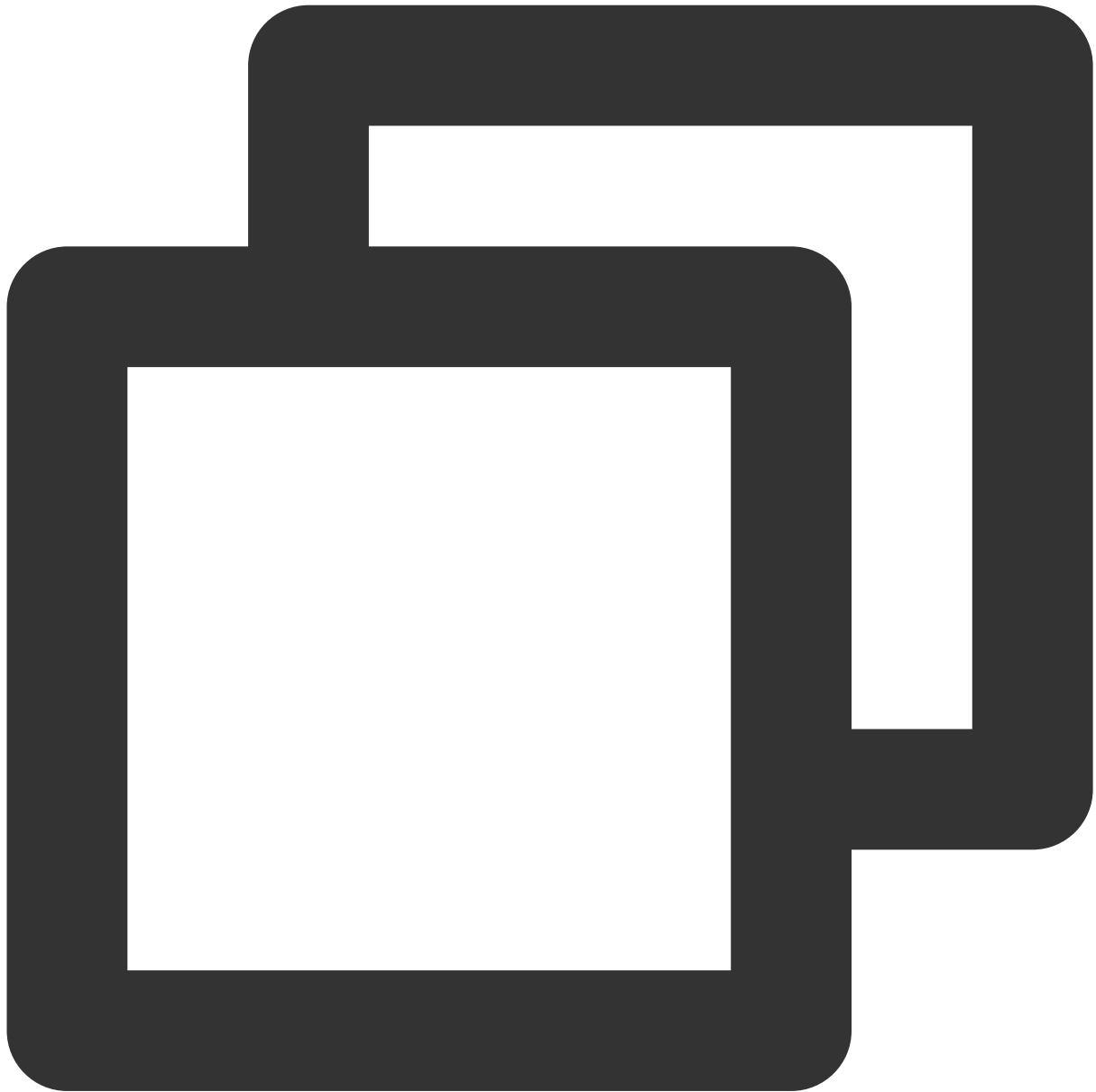
```
DROP MATERIALIZED VIEW link_mv_student;  
DESCRIBE link_mv_student; --The source table still exists
```

## Hive type source table

When the Hive table is the source table, a complete example is as follows:

## Prepare to initialize data

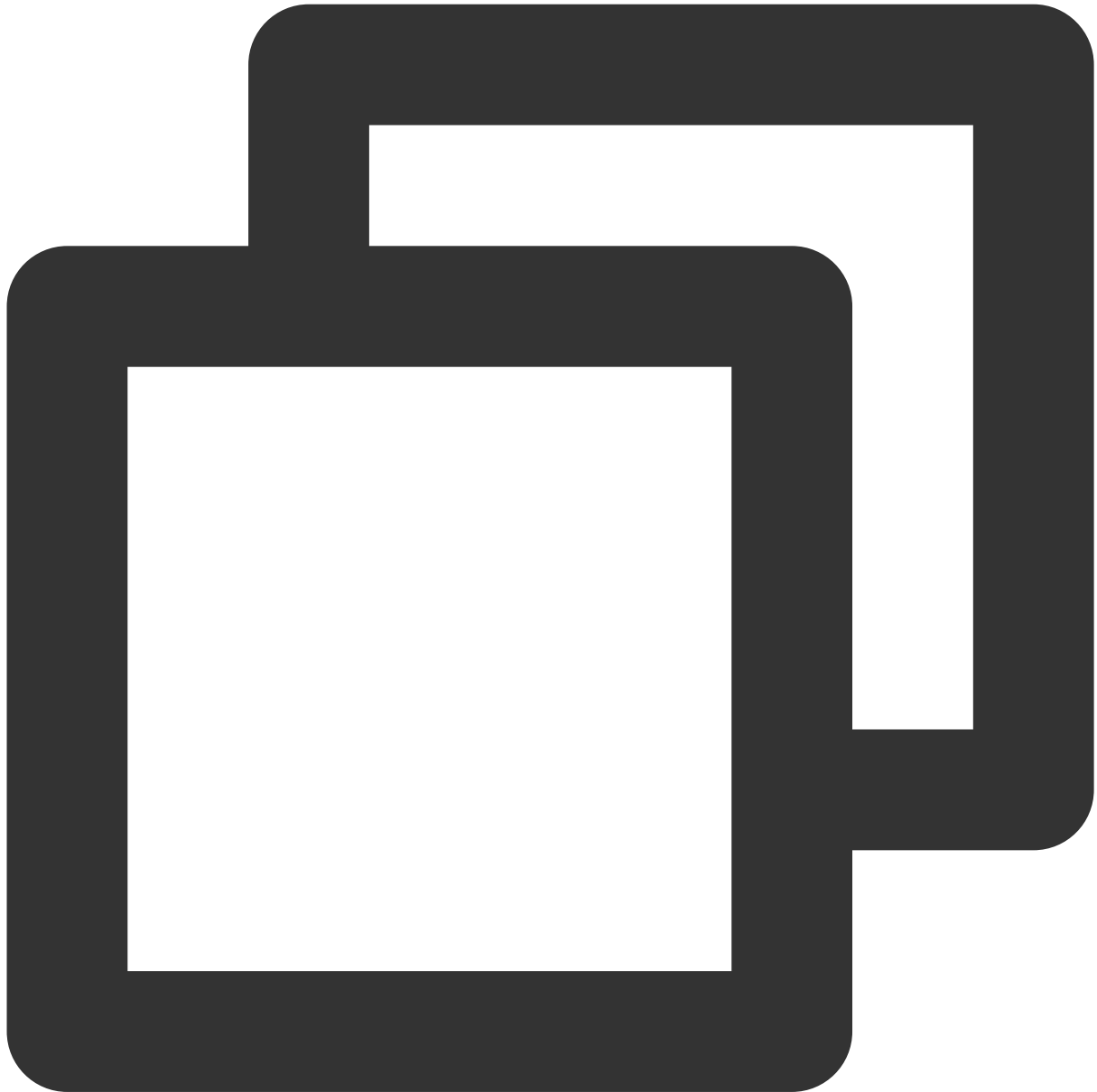
First, you need to prepare the initial data and create a Hive base table. Use the CREATE EXTERNAL TABLE statement to create a Hive base table, and manually insert data via the INSERT statement.



```
CREATE EXTERNAL TABLE student_2(id int, name string, score int)
LOCATION 'cosn://guangzhou-test-1305424723/mv_test4/student_2';
insert into student_2 values (1,'zhangsan', 90);
insert into student_2 values (2,'lisi', 100);
insert into student_2 values (3,'wangwu', 80);
insert into student_2 values (4,'zhaoliu', 30);
select * from student_2;
```

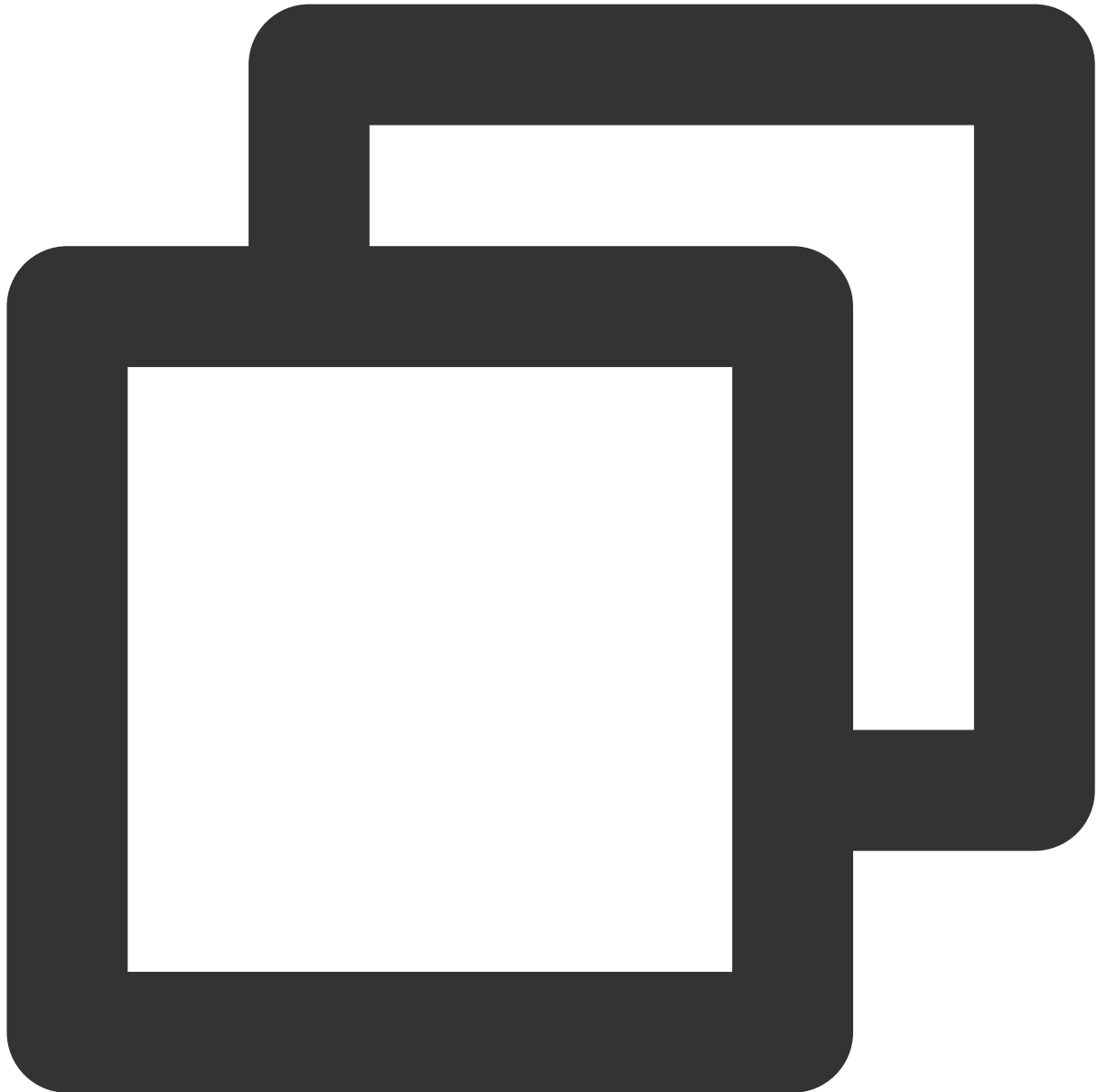
## Create a mapped Hive external table

Use the CREATE EXTERNAL TABLE statement to create a mapped Hive external table.



```
CREATE EXTERNAL TABLE link_mv_student_hive (  
  sum_score BIGINT  
) LOCATION 'cosn://guangzhou-test-1305424723/mv_test4/link_mv_student_hive';
```

To insert data into the mapping table, use the `INSERT OVERWRITE` statement to insert the query results into the mapping table, ensuring the data in the mapping table is consistent with the data in the Hive base table.

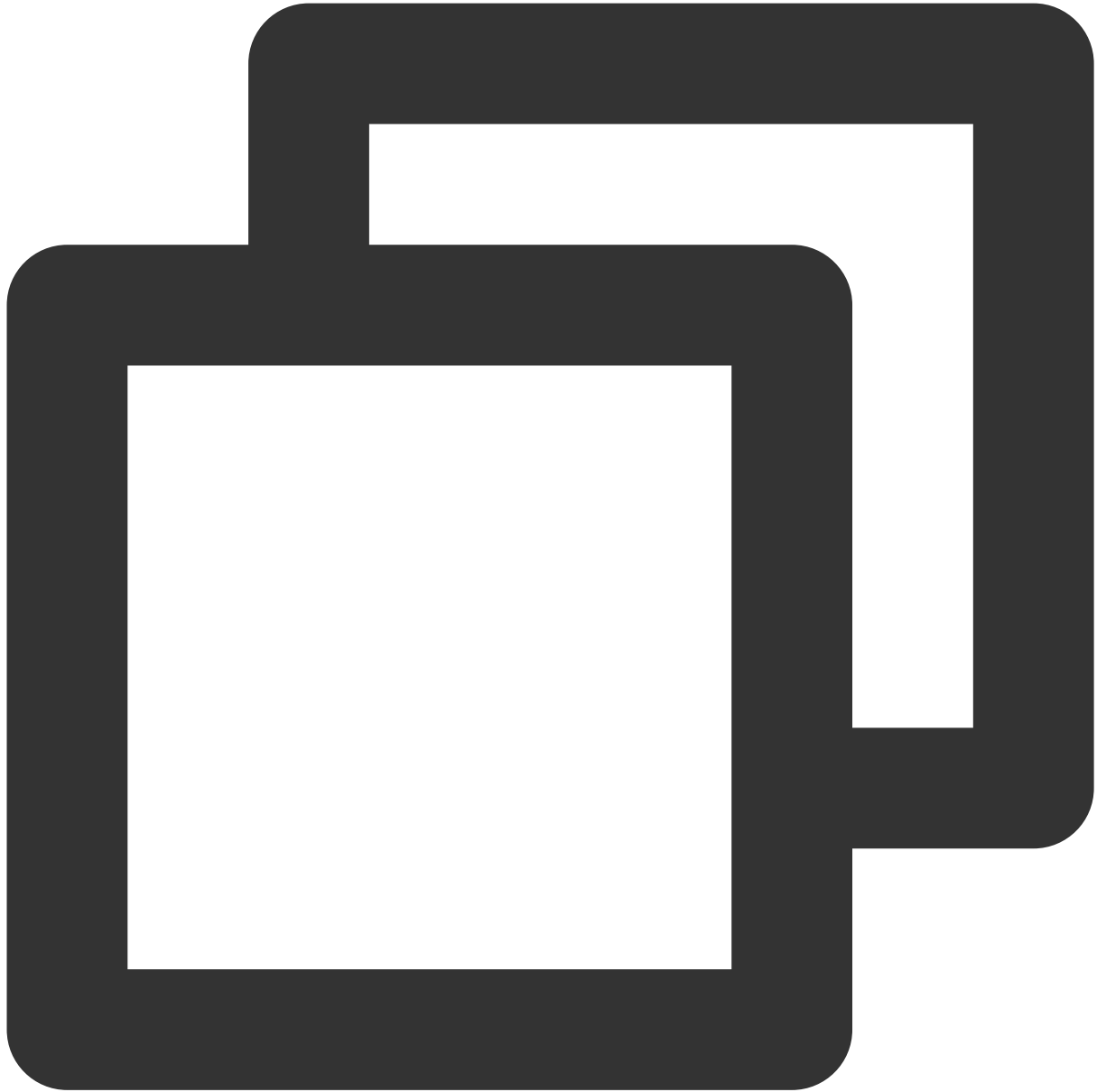


```
--Insert data into the mapping table
INSERT OVERWRITE link_mv_student_hive
select sum(score) from student;
```

### Create a mapped materialized view based on the Hive external table

Use the `CREATE MATERIALIZED VIEW` statement to create a mapped materialized view. When creating a materialized view, use the `WITH META LINK` clause and specify the name of the above Hive external table for

association.



```
CREATE MATERIALIZED VIEW link_mv_student_hive WITH META LINK AS (  
    select sum(score) from student_2  
);
```