

**OCEANBASE**



# OceanBase 数据库

## 实践教程

| 产品版本：V4.3.5


| 文档版本：20250106

# 声明

**北京奥星贝斯科技有限公司版权所有©2024，并保留一切权利。**

未经北京奥星贝斯科技有限公司事先书面许可，任何单位、公司或个人不得擅自摘抄、翻译、复制本文档内容的部分或全部，不得以任何方式或途径进行传播和宣传。

## 商标声明

 **OCEANBASE** 及其他 OceanBase 相关的商标均为北京奥星贝斯科技有限公司所有。本文档涉及的第三方的注册商标，依法由权利人所有。

## 免责声明

由于产品版本升级、调整或其他原因，本文档内容有可能变更。北京奥星贝斯科技有限公司保留在没有任何通知或者提示下对本文档的内容进行修改的权利，并在北京奥星贝斯科技有限公司授权通道中不时发布更新后的用户文档。您应当实时关注用户文档的版本变更并通过北京奥星贝斯科技有限公司授权渠道下载、获取最新版的用户文档。如因文档使用不当造成的直接或间接损失，本公司不承担任何责任。

# 通用约定

格式	说明	样例
 <b>危险</b>	该类警示信息将导致系统重大变更甚至故障，或者导致人身伤害等结果。	 <b>危险</b> 重置操作将丢失用户配置数据。
 <b>警告</b>	该类警示信息可能会导致系统重大变更甚至故障，或者导致人身伤害等结果。	 <b>警告</b> 重启操作将导致业务中断，恢复业务时间约十分钟。
 <b>注意</b>	用于警示信息、补充说明等，是用户必须了解的内容。	 <b>注意</b> 权重设置为0，该服务器不会再接受新请求。
 <b>说明</b>	用于补充说明、最佳实践、窍门等，不是用户必须了解的内容。	 <b>说明</b> 您也可以通过按Ctrl+A选中全部文件。
>	多级菜单递进。	单击 <b>设置</b> > <b>网络</b> > <b>设置网络类型</b> 。
<b>粗体</b>	表示按键、菜单、页面名称等UI元素。	在 <b>结果确认</b> 页面，单击 <b>确定</b> 。
Courier字体	命令或代码。	执行 <code>cd /d C:/window</code> 命令，进入Windows系统文件夹。
斜体	表示参数、变量。	<code>bae log list --instanceid</code> <code>Instance_ID</code>
[ ] 或者 [a b]	表示可选项，至多选择一个。	<code>ipconfig [-all -t]</code>
{ } 或者 {a b}	表示必选项，至多选择一个。	<code>switch {active stand}</code>

# 目录

1 基于 OceanBase 构建图搜图应用 .....	5
1.1 背景信息 .....	5
1.2 图搜图架构 .....	5
1.3 前提条件 .....	5
1.4 操作步骤 .....	6
1.5 应用展示 .....	7
2 基于 OceanBase 构建智能问答机器人 .....	9
2.1 背景信息 .....	9
2.2 文档智能助手架构 .....	9
2.3 前提条件 .....	9
2.4 操作步骤 .....	10
2.4.0.1 注意 .....	10
2.4.0.2 说明 .....	12
2.5 应用展示 .....	13
3 基于 OceanBase 搭建离线数仓和可视化看板 .....	14
3.1 背景信息 .....	14
3.2 前提条件 .....	14
3.3 准备工作 .....	14
3.3.1 创建数据库并建表 .....	14
3.3.2 构建 flink 同步链路 .....	16
3.3.3 计算层构建 .....	19
3.3.3.1 构建 DBT 项目 .....	20
3.3.3.2 创建 DAG 调度任务 .....	22
3.3.4 配置 Grafana 大盘 .....	25
3.4 实施 .....	25
3.4.1 模拟购买行为 .....	25
3.4.2 Airflow 调度 .....	28
3.4.3 查看数据大盘 .....	28

# 1 基于 OceanBase 构建图搜图应用

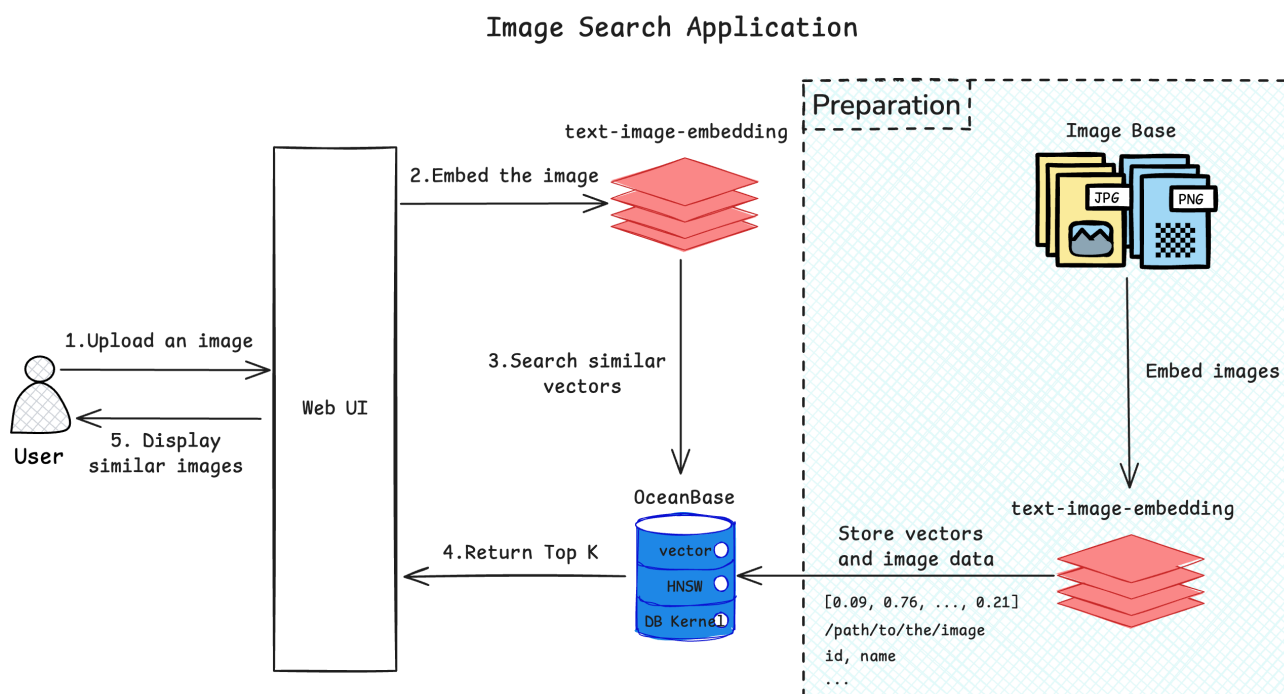
## 1.1 背景信息

在当今信息爆炸的时代，用户常需要从海量数据中迅速检索所需信息。例如在线文献数据库、电商平台产品目录、以及不断增长的多媒体内容库，都需要高效的检索系统来快速定位到用户感兴趣的内容。随着数据量不断激增，传统的基于关键字的检索方法已经无法满足用户对于检索精度和速度的需求，向量检索技术应运而生。它通过将文本、图片、音频等不同类型的数据编码为数学上的向量，并在向量空间中进行检索。这种方法允许系统捕捉数据的深层次语义信息，从而提供更为准确和高效的检索结果。

本文通过 OceanBase 的向量检索技术构建您的图搜图应用。

## 1.2 图搜图架构

图搜图应用是把图片库以向量形式，存储在数据库内，用户在对应的 UI 界面，上传需要查询的图片，图片会被应用转换为向量，在数据库内查询相似向量，并返回结果，最终以图片形式，在 UI 页面上展示相似图片。



## 1.3 前提条件

- 您已部署 OceanBase V4.3.3 及以上版本的集群并且创建了 MySQL 模式租户。更多有关部署 OceanBase 集群的信息，请参见 [部署概述](#)
- 您所创建的 MySQL 模式租户需要拥有插入及查询的权限。更多有关权限设置的信息，请参见 [直接授予权限](#)。
- 您已创建数据库。更多有关创建数据库的信息，请参见 [创建数据库](#)。

- 数据库已开启向量检索功能。更多关于向量检索功能的信息，请参见 [使用 SQL 快速进行向量检索](#)。

```
obclient> ALTER SYSTEM SET ob_vector_memory_limit_percentage = 30;
```

- 准备自己所需的图片，如果没有足够的图片用于测试效果，可参考各大开源网站的图片数据集。
- 安装 Python 3.9 及以上版本。
- 安装 Poetry：

```
python3 -m ensurepip  
python3 -m pip install poetry
```

## 1.4 操作步骤

1. 克隆代码仓库。

```
git clone https://gitee.com/oceanbase-devhub/image-search.git  
cd image-search
```

2. 安装依赖。

```
poetry install
```

3. 设置环境变量。

```
cp .env.example .env  
# 更新 .env 文件中的数据库信息  
vi .env
```

更新 .env 中内容。

```
HF_ENDPOINT=https://hf-mirror.com  
  
DB_HOST="127.0.0.1" ## 设置对应租户的 IP  
DB_PORT="2881" ## 设置对应的端口  
DB_USER="root@test" ## 设置对应的租户及用户名
```

```
DB_NAME="test" ## 设置对应的数据库名  
DB_PASSWORD="" ## 设置对应的租户用户的密码
```

4. 上传准备的图片至服务器上。
5. 启动图搜图程序。

```
poetry run streamlit run --server.runOnSave false image_search_ui.py
```

返回结果如下：

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.

You can now view your Streamlit app in your browser.


Local URL: <http://localhost:8501>

Network URL: <http://xxx.xxx.xxx.xxx:8501>

External URL: <http://xxx.xxx.xxx.xxx:8501>

6. 打开图搜图对应的 UI 界面。可根据实际情况，打开上述第 5 步中对应的 URL。
7. 在 **图片加载设置**下，**图片加载目录**内填写服务器上图片所存放目录的绝对路径。
8. 点击 **加载图片**。
9. 等图片加载完成后即可进行图搜图操作。

## 1.5 应用展示

 OCEANBASE  
海量数据 极致性能

应用设置

应用搜索

应用名称

image\_search

召回数量

1.0

130

☒ 显示距离

☒ 显示文件路径

图片加载设置

图片加载目录

图片目录的绝对路径, 如 /data/imgs

加载图片

基于 OceanBase 向量检索能力构建的相似图像搜索应用

选择一张图片...

Drag and drop file here

Limit 200MB per file • JPG, JPEG, PNG

Browse files

00f56e5c92.jpg

174.4KB

您上传的图片

上传图片

相似图片

图片 1

图片 2

图片 3

图片 4

图片 5

图片 6

图片 7

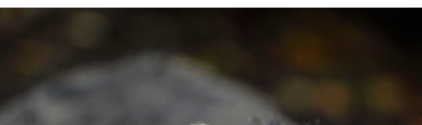
图片 8

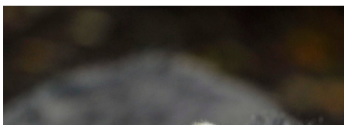
图片 9

图片 10

距离: 0.00000000

文件路径:







## 2 基于 OceanBase 构建智能问答机器人

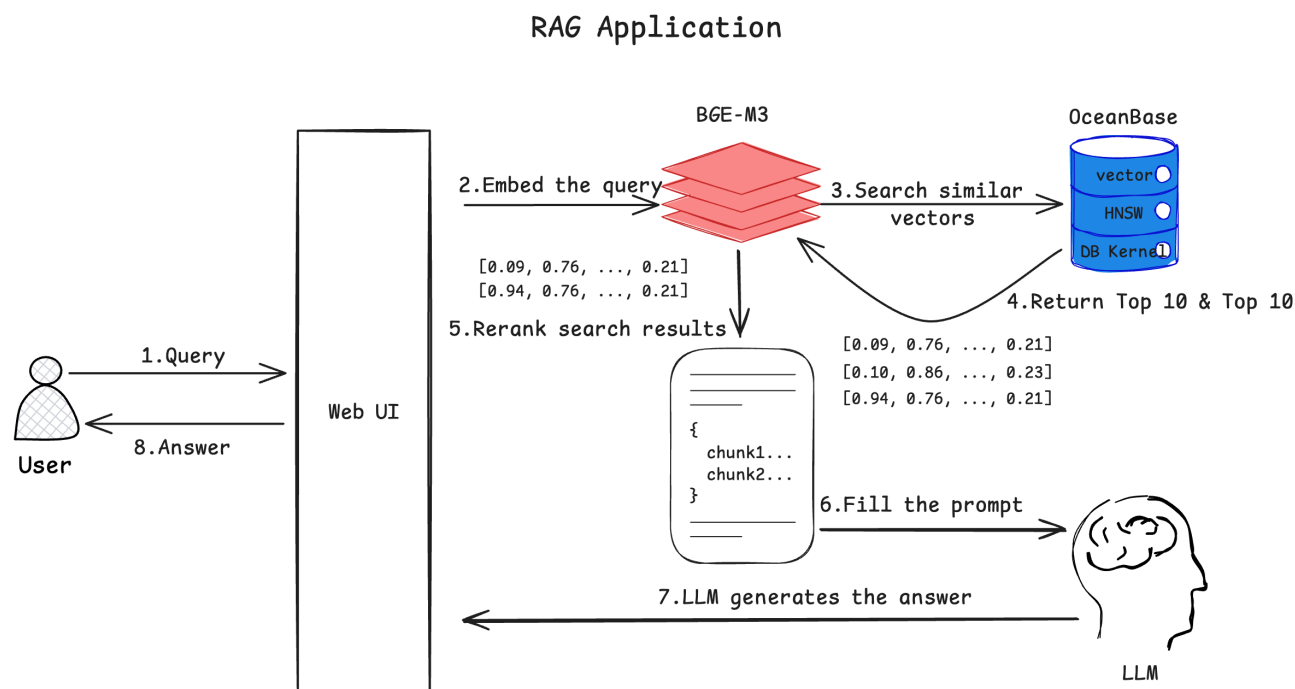
### 2.1 背景信息

在当今信息爆炸的时代，用户常需要从海量数据中迅速检索所需信息。例如在线文献数据库、电商平台产品目录、以及不断增长的多媒体内容库，都需要高效的检索系统来快速定位到用户感兴趣的内容。随着数据量不断激增，传统的基于关键字的检索方法已经无法满足用户对于检索精度和速度的需求，向量检索技术应运而生。它通过将文本、图片、音频等不同类型的数据编码为数学上的向量，并在向量空间中进行检索。这种方法允许系统捕捉数据的深层次语义信息，从而提供更为准确和高效的检索结果。

本文通过 OceanBase 的向量检索能力构建文档智能助手。

### 2.2 文档智能助手架构

文档智能助手将文档以向量的形式批量存储在 OceanBase 数据库内。用户通过 UI 界面提问，程序使用 BGE-M3 模型将提问内容嵌入成为向量并在数据库中检索相似向量，得到相似向量对应的文档内容后，应用将它们会同用户提问一起发送给 LLM，LLM 会根据提供的文档生成更加准确的回答。



### 2.3 前提条件

- 您已部署 OceanBase V4.3.3 及以上版本的集群并且创建了 MySQL 模式租户。更多有关部署 OceanBase 集群的信息，请参见 [部署概述](#)
- 您所创建的 MySQL 模式租户需要拥有插入及查询的权限。更多有关权限设置的信息，请参见 [直接授予权限](#)。

- 您已创建数据库。更多有关创建数据库的信息，请参见 [创建数据库](#)。
- 数据库已开启向量检索功能。更多关于向量检索功能的信息，请参见 [使用 SQL 快速进行向量检索](#)。

```
obclient> ALTER SYSTEM SET ob_vector_memory_limit_percentage = 30;
```

- 安装 Python 3.9 及以上版本。
- 安装 Poetry:

```
python3 -m ensurepip
python3 -m pip install poetry
```

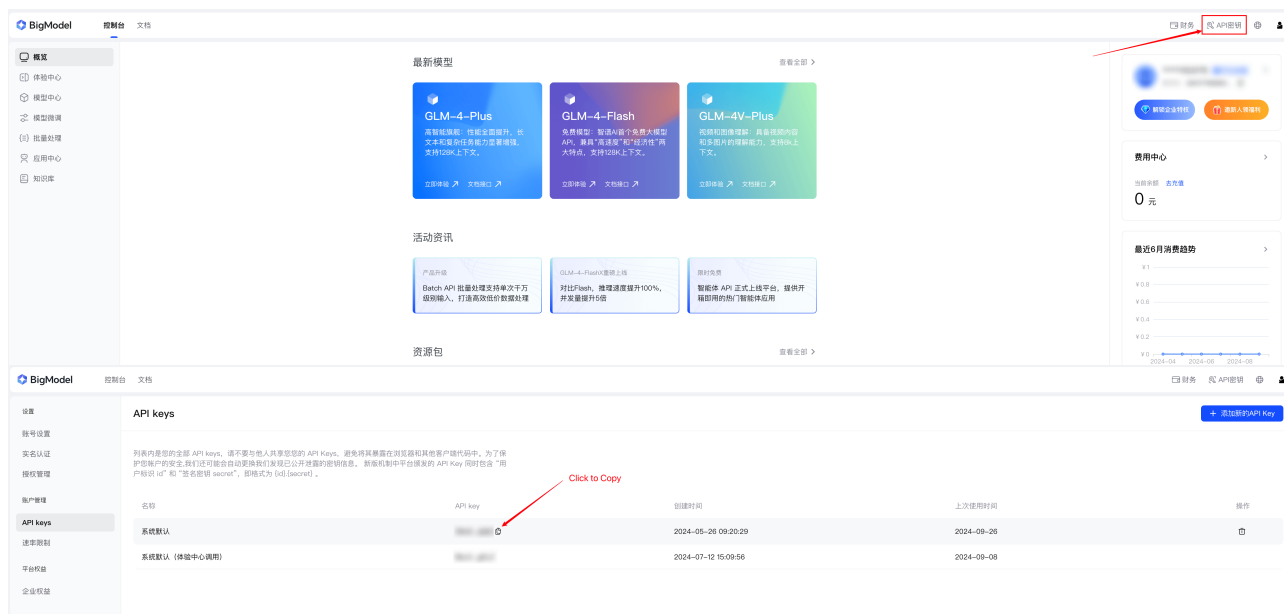
## 2.4 操作步骤

1. 注册 Zhipu LLM 平台账号。

注册 [Zhipu LLM](#)，获取 API 密钥。

### 2.4.0.1 注意

- Zhipu LLM 提供了一定的免费使用额度。使用过程中请关注免费额度使用情况，超出将会产生费用。
- 本教程以 Zhipu LLM 为例来介绍问答机器人的搭建，您也可以选择使用其他 LLM 进行搭建，选用其他 LLM 需要更新 `.env` 文件中的 `API_KEY`、`LLM_BASE_URL` 和 `LLM_MODEL`。



## 2. 克隆代码仓库。

```
git clone https://github.com/oceanbase-devhub/ai-workshop-2024.git
cd ai-workshop-2024
```

### 3. 安装依赖。

```
poetry install
```

### 4. 设置环境变量。

```
cp .env.example .env
# 更新 .env 文件中的数据库信息
vi .env
```

更新 .env 中内容。

```
API_KEY=xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx ## 此处为上述 Zhipu LLM 平台中获取的 API
密钥
LLM_BASE_URL="https://open.bigmodel.cn/api/paas/v4/" ## 如果选用非 Zhipu 的
其他 LLM，则需要更新此字段
LLM_MODEL="glm-4-flash" ## 如果选用非 Zhipu 的其他 LLM，则需要更新此字段

HF_ENDPOINT=https://hf-mirror.com
BGE_MODEL_PATH=BAAI/bge-m3

DB_HOST="127.0.0.1" ## 设置对应租户的 IP
DB_PORT="2881" ## 设置对应租户的端口
DB_USER="root@mysql_tenant" ## 设置对应的租户及用户名
DB_NAME="test" ## 设置对应的数据库名
DB_PASSWORD="" ## 设置对应的租户用户的密码
```

### 5. 准备 BGE-M3 模型。

```
poetry run python utils/prepare_bgem3.py
```

返回如下结果，说明模型加载成功。

```
# =====  
# BGEM3FlagModel loaded successfully.  
# =====
```

## 6. 准备文档语料。

### 2.4.0.2 说明

此过程需要将 OceanBase 开源的文档转换为向量并且储存在 OceanBase 数据库中，需要花费较长时间。

#### a. 克隆文档仓库。

```
cd doc_repos  
git clone --single-branch --branch V4.3.3 https://github.com/oceanbase  
/oceanbase-doc.git  
cd ..
```

#### b. 将标题转换为标准 Markdown 格式。

```
poetry run python convert_headings.py \  
doc_repos/oceanbase-doc/zh-CN
```

#### c. 将文档文本嵌入为向量。

```
poetry run python embed_docs.py --doc_base doc_repos/oceanbase-doc/zh-  
CN
```

## 7. 启动 UI 聊天界面。

```
poetry run streamlit run --server.runOnSave false chat_ui.py
```

返回结果如下：

Collecting usage statistics. To deactivate, set browser.gatherUsageStats to false.

You can now view your Streamlit app in your browser.

Local URL: <http://localhost:8501>

Network URL: http://xxx.xxx.xxx.xxx:8501

External URL: http://xxx.xxx.xxx.xxx:8501

## 2.5 应用展示

根据实际情况，通过上述启动 UI 聊天界面中的地址，在浏览器内打开，即可向助手提问。

由于本应用基于 OceanBase 文档语料构建，请就 OceanBase 相关问题向您的助手提问。



## 3 基于 OceanBase 搭建离线数仓和可视化看板

### 3.1 背景信息

随着社会向数字化转型的加速，企业对于数据分析的要求日益提高。数据处理主要集中在处理大量历史数据的离线场景，可以通过搭建数据仓库来解决。OceanBase 可以通过搭建数据仓库解决离线数据的分析问题。通过定时任务来实现从原始数据层 (ODS) 到明细数据层 (DWD) 最终到达应用数据层 (ADS) 的数据层级构建，创建基于 OceanBase 的数据仓库。同时通过配套的生态工具搭建可视化看板。

### 3.2 前提条件

- 您使用的源端数据库为 MySQL 或已部署 OceanBase V4.3.3 及以上版本的集群并且创建了 MySQL 模式租户。更多有关部署 OceanBase 集群的信息，请参见 [部署概述](#)
- 您所创建的 MySQL 模式租户需要拥有插入及查询的权限。更多有关权限设置的信息，请参见 [直接授予权限](#)。
- 您已创建数据库。有关创建数据库的信息，请参见 [创建数据库](#)。
- 您使用的源端数据库已开启 Binlog 服务。
- 您已部署 Flink CDC、DBT、airflow、Grafana。

### 3.3 准备工作

#### 创建数据库并建表

1. 将源端数据库作为 TP 库，将目标端数据库作为 AP 库，分别创建 tpctest 和 aptest 数据库。

```
create database aptest;
```

```
create database tpctest;
```

2. 分别在 TP 库和 AP 库中建表。建表语句如下。

```
CREATE TABLE `orders` (  
  order_id bigint not null primary key,  
  user_id varchar(50) not null,  
  shop_id bigint not null,
```

```
product_id bigint not null,  
buy_fee numeric(20,2) not null,  
create_time timestamp not null,  
update_time timestamp not null default now(),  
state int not null  
);
```

```
CREATE TABLE `orders_pay` (  
pay_id bigint not null primary key,  
order_id bigint not null,  
pay_platform varchar(64) not null,  
create_time timestamp not null  
);
```

```
CREATE TABLE `product_catalog` (  
product_id bigint not null primary key,  
catalog_name varchar(50) not null  
);
```

### 3. 给 TP 库中的表插入几条数据。

```
INSERT INTO product_catalog VALUES(1, 'iphone 14'),(2, 'iphone 14 pro max'),(3,  
'iphone 15'),(4, 'huawei mate 60'),(5, 'huawei pura 70');  
insert into `tpctest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)  
values(1,1,'test','2024-10-01 00:00:00');  
insert into `tpctest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)  
values(2,2,'test','2024-10-02 00:00:00');  
insert into `tpctest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)  
values(3,3,'test','2024-10-03 00:00:00');  
insert into `tpctest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)  
values(4,4,'test','2024-10-04 00:00:00');
```

```
insert into `tptest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)
values(1,1,'test','2024-10-01 00:00:00');
insert into `tptest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)
values(2,2,'test','2024-10-02 00:00:00');
insert into `tptest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)
values(3,3,'test','2024-10-03 00:00:00');
insert into `tptest`.`orders_pay`(`pay_id`,`order_id`,`pay_platform`,`create_time`)
values(4,4,'test','2024-10-04 00:00:00');
```

## 构建 flink 同步链路

启动 flink CDC 后，进入部署 flink 文件夹下，执行 `./bin/sql-client.sh` 命令，打开 Flink SQL 界面。依次执行如下 SQL。

```
``sql
CREATE TABLE mysql_orders (
order_id bigint not null primary key NOT ENFORCED,
user_id varchar(50) not null,
shop_id bigint not null,
product_id bigint not null,
buy_fee numeric(20,2) not null,
create_time timestamp not null,
update_time timestamp not null,
state int not null
) WITH (
'connector' = 'mysql-cdc',
'hostname' = 'xxx.xxx.xxx.xxx',
'port' = '3306',
'username' = 'wktest',
'password' = '*****',
```



```
'database-name' = 'tpctest',
'table-name' = 'orders');

CREATE TABLE `mysql_orders_pay` (
  pay_id bigint not null primary key NOT ENFORCED,
  order_id bigint not null,
  pay_platform varchar(64) not null,
  create_time timestamp not null
) WITH (
  'connector' = 'mysql-cdc',
  'hostname' = 'xxx.xxx.xxx.xxx',
  'port' = '3306',
  'username' = 'wktest',
  'password' = '*****',
  'database-name' = 'tpctest',
  'table-name' = 'orders_pay');

CREATE TABLE `mysql_product_catalog` (
  product_id bigint not null primary key NOT ENFORCED,
  catalog_name varchar(50) not null
) WITH (
  'connector' = 'mysql-cdc',
  'hostname' = 'xxx.xxx.xxx.xxx',
  'port' = '3306',
  'username' = 'wktest',
  'password' = '*****',
  'database-name' = 'tpctest',
  'table-name' = 'product_catalog');
```

```
CREATE TABLE `orders` (  
  order_id bigint not null primary key NOT ENFORCED,  
  user_id varchar(50) not null,  
  shop_id bigint not null,  
  product_id bigint not null,  
  buy_fee numeric(20,2) not null,  
  create_time timestamp not null,  
  update_time timestamp not null,  
  state int not null  
) WITH (  
  'connector' = 'jdbc',  
  'url' = 'jdbc:mysql://*****:3306/aptest',  
  'username' = 'wktest@mysql001',  
  'password' = '*****',  
  'table-name' = 'orders');
```

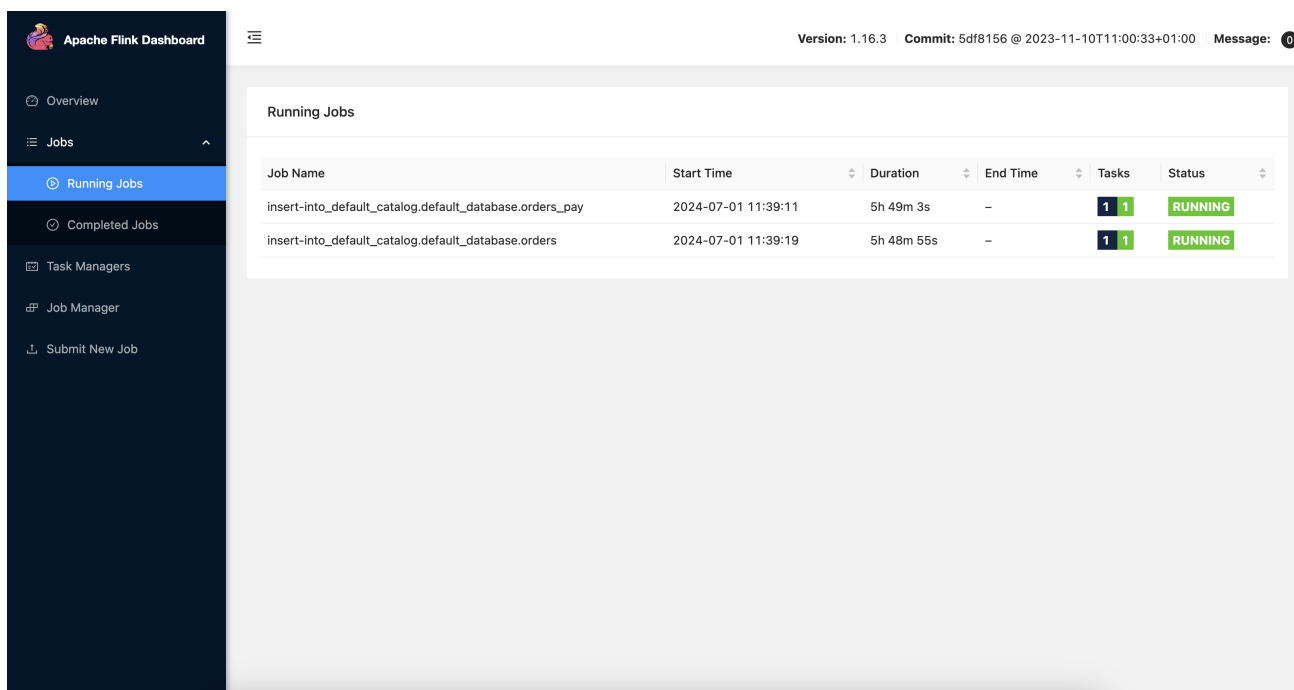
```
CREATE TABLE `orders_pay` (  
  pay_id bigint not null primary key NOT ENFORCED,  
  order_id bigint not null,  
  pay_platform varchar(64) not null,  
  create_time timestamp not null  
) WITH (  
  'connector' = 'jdbc',  
  'url' = 'jdbc:mysql://*****:3306/aptest',  
  'username' = 'wktest@mysql001',  
  'password' = '*****',  
  'table-name' = 'orders_pay');
```

```
CREATE TABLE `product_catalog` (  

```

```
product_id bigint not null primary key NOT ENFORCED,  
catalog_name varchar(50) not null  
) WITH (  
'connector' = 'jdbc',  
'url' = 'jdbc:mysql://*****:3306/aptest',  
'username' = 'wktest@mysql001',  
'password' = '*****',  
'table-name' = 'product_catalog',  
'sink.buffer-flush.max-rows' = '0',  
'sink.buffer-flush.interval' = '0');  
  
INSERT INTO product_catalog SELECT * FROM mysql_product_catalog;  
INSERT INTO orders_pay SELECT * FROM mysql_orders_pay;  
INSERT INTO orders SELECT * FROM mysql_orders;  
...
```

创建上述 Flink CDC 同步链路后，数据会实时从 `tpctest` 数据库同步到 `aptest` 数据库。可以从 SQL 控制台查看对应的表数据。



The screenshot shows the Apache Flink Dashboard interface. The left sidebar contains navigation links: Overview, Jobs (expanded), Running Jobs (selected), Completed Jobs, Task Managers, Job Manager, and Submit New Job. The main panel displays the 'Running Jobs' section with a table of active jobs.

Job Name	Start Time	Duration	End Time	Tasks	Status
insert-into_default_catalog.default_database.orders_pay	2024-07-01 11:39:11	5h 49m 3s	-	1 1	RUNNING
insert-into_default_catalog.default_database.orders	2024-07-01 11:39:19	5h 48m 55s	-	1 1	RUNNING

## 计算层构建

## 构建 DBT 项目

原始的 TP 表结构不适用于直接进行数据分析与展示，需要一定的转换。这里通过使用 DBT 项目进行数据转换。根据 Demo 的原本范例我们构建一个 DBT 项目，并定义模型。详细步骤如下。

1. DBT 安装部署后，执行 `dbt init my_project` 创建一个名为 `my_project` 的 DBT 项目，在弹出的对话框中按照提示输入数据库信息。这里要注意的是 `user` 只填用户名即可。创建后会在文件 `~/.dbt/profiles.yml` 记录刚才输入的信息，并用于数据库连接。
2. 执行 `cd my_project` 进入项目目录。
3. 在 `/my_project/models/example` 目录下编写 SQL 文件来定义数据模型。

```
# models/example/dwd_orders.sql

{{ config(
  materialized='view'
)}}

select
  o.order_id as order_id,
  o.user_id as order_user_id,
  o.shop_id as order_shop_id,
  o.product_id as order_product_id,
  o.buy_fee as order_fee,
  o.create_time as order_create_time,
  o.update_time as order_update_time,
  o.state as order_state,
  c.catalog_name as order_product_catalog_name,
  p.pay_id as pay_id,
  p.pay_platform as pay_platform,
  p.create_time as pay_create_time
```

```
from
orders o
left join product_catalog c on o.product_id = c.product_id
left join orders_pay p on o.order_id = p.order_id
```

```
# models/example/dwd_shops.sql
{{
config(materialized='table')
}}

select
order_shop_id,
str_to_date(date_format(pay_create_time, '%Y%m%d'), '%Y%m%d') as ds,
sum(order_fee) as sum_fee
from
{{ ref('dwd_orders') }}
where
order_state=1
GROUP BY
order_shop_id,
date_format(pay_create_time, '%Y%m%d')
```

```
# models/example/dwd_shops_all.sql
{{
config(materialized='table')
}}

select
str_to_date(date_format(pay_create_time, '%Y%m%d'), '%Y%m%d') as ds,
sum(order_fee) as sum_fee
```

```
from
{{ ref('dwd_orders') }}
where
order_state=1
GROUP BY
date_format(pay_create_time, '%Y%m%d')
```

```
# models/example/dwd_users.sql

{{
  config(materialized='table')
}}

select
order_user_id,
str_to_date(concat(date_format(pay_create_time, '%Y%m'), '01'), '%Y%m%d') as
ds,
sum(order_fee) as sum_fee
from
{{ ref('dwd_orders') }}
where
order_state = 1
group by
order_user_id,
date_format(pay_create_time, '%Y%m')
```

4. 执行 `dbt compile` ，编译模型会将 SQL 转换为数据库特定的语句。
5. 执行 `dbt run` ，运行所有模型。

## 创建 DAG 调度任务

这里采用 Airflow 作为基本的 DAG 和调度平台。首次安装部署后，需要执行 `airflow db init` 初始化 Airflow 的数据库。部署 Airflow 后，执行 `airflow scheduler` 启动 Scheduler，可以对 DAG 做实时调度；执行 `airflow webserver`，启动 Web 服务器，在 web 界面上能够完整看到整个调度任务。

配置好访问的数据库后，从后台登录部署 Airflow 的机器，将你的 DAG 文件（.py 文件）放在 Airflow 的 `dags` 目录下，Airflow 会自动发现并加载这些 DAG。这里定义 2 个计算节点，一个用于运行 DBT 项目，完成数仓计算任务，另一个用于完成计算任务后发送邮件提示使用者。DAG 定义如下。

```
# Copyright (c) 2023 OceanBase.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
from datetime import datetime, timedelta

from airflow import DAG
from airflow.operators.email import EmailOperator
from airflow_dbt import DbtRunOperator

default_args = {
    "owner": "airflow",
    "depends_on_past": False,
```

```
"start_date": datetime(2024, 7, 12, 8, 35),
"email": ["*****@oceanbase.com"],
"email_on_failure": True,
"email_on_retry": False,
"retries": 1,
"retry_delay": timedelta(minutes=5),
}

dag = DAG("warehouse_demo", default_args=default_args, schedule=timedelta(
(minutes=1))

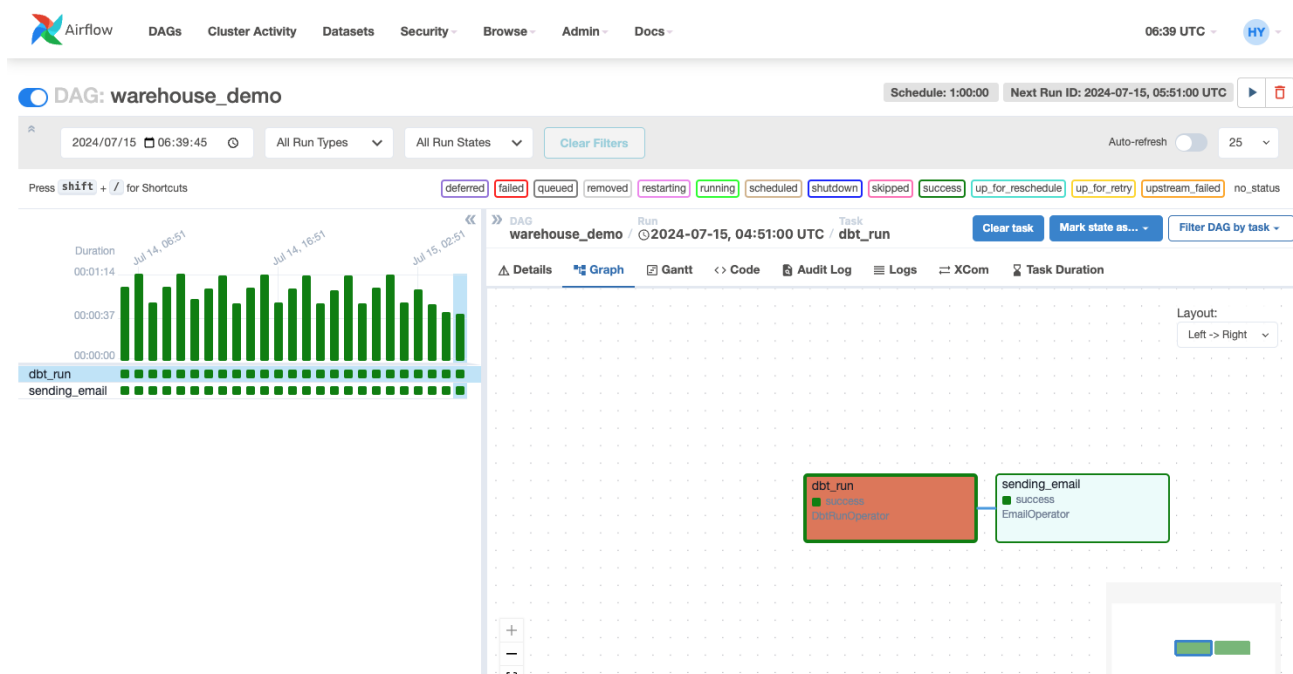
t4 = EmailOperator(
task_id="sending_email",
to="*****@oceanbase.com",
subject="AirFlow Notice",
html_content="<h1>Your Airflow Has been completed</h1>",
dag=dag,
)

dbt_operator = DbtRunOperator(
task_id="dbt_run", dir="xxx", dag=dag
)

dbt_operator >> t4
```

上述文件部署后，在 Airflow 的 web 界面能够完整看到整个调度任务，可以根据需要对该任务进行调度。





## 配置 Grafana 大盘

这里仅展示店铺按日的销售额，只关注其中一个店铺的销售额以及所有店铺的销售额。因此 Grafana 配置 AP 库的连接，图形化展示对应 SQL 即可。SQL 如下。

```
```sql
SELECT
ds AS "time",
sum_fee AS "sum_fee"
FROM dwd_shops
WHERE
order_shop_id = 35
ORDER BY ds
```
```

## 3.4 实施

### 模拟购买行为

编写一个简单的 python 脚本 `test.py` 用于模拟用户的购买行为。

```
# Copyright (c) 2023 OceanBase.
#
# Licensed under the Apache License, Version 2.0 (the "License");
# you may not use this file except in compliance with the License.
# You may obtain a copy of the License at
#
# http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
# See the License for the specific language governing permissions and
# limitations under the License.
import dataclasses
from typing import Dict

import mysql.connector

PRODUCT_ID_2_FEE: Dict[int, float] = {
    1: 5399,
    2: 10099,
    3: 4599,
    4: 5499,
    5: 6499,
}

@dataclasses.dataclass
class Phone:
    product_id: int
```

```
catalog_name: str
buy_fee: float

def get_max_order_id(cur):
    cur.execute("select order_id from orders order by order_id desc limit 1")
    id = cur.fetchone()
    return next(iter(id))

def get_max_pay_id(cur):
    cur.execute("select pay_id from orders_pay order by pay_id desc limit 1")
    id = cur.fetchone()
    return next(iter(id))

def buy_phone(product_id: int, cursor, user_id=15, shop_id=35):
    cursor.execute("select product_id, catalog_name from product_catalog")
    tuples = cursor.fetchall()
    phones = [
        Phone(
            **{
                "product_id": p_id,
                "catalog_name": c_name,
                "buy_fee": PRODUCT_ID_2_FEE.get(p_id),
            }
        )
        for p_id, c_name in tuples
    ]
    target = next(filter(lambda p: p.product_id == product_id, phones))
    order_id = get_max_order_id(cursor) + 1
    sql = f"insert into `orders`(`order_id`, `user_id`, `shop_id`, `product_id`, `buy_fee`,
```

```
`create_time`, `update_time`, `state`) values({order_id}, {user_id}, {shop_id},
{product_id}, {target.buy_fee}, now(), now(), 0)"
cursor.execute(sql)
pay_id = get_max_pay_id(cursor) + 1
sql = f"insert into `orders_pay`(`pay_id`, `order_id`, `pay_platform`, `create_time`)
values({pay_id}, {order_id}, 'Alipay', now())"
cursor.execute(sql)
sql = f"update orders set state=1 where order_id={order_id}"
cursor.execute(sql)
cursor.execute("commit")
print(target)

if __name__ == "__main__":
    with mysql.connector.connect(
        **{
            "host": "*****",
            "port": "3306",
            "database": "tpctest",
            "user": "wktest@mysql001",
        }
    ) as conn:
        with conn.cursor() as cursor:
            buy_phone(5, cursor)
```

执行 `sh python.py` 命令，运行该脚本。

## Airflow 调度

等待 1 分钟，Airflow 将会调度之前配置的 DAG，可以根据 Web 界面上的状态判断节点的运行状态同时可以查看任务的日志。

## 查看数据大盘

查看数据大盘，可以看到最新的数据。这里仅展示店铺按日的销售额，只关注其中一个店铺的销售额以及所有店铺的销售额，可以看到，7.1 的销售额已经变成 32495。

