ALIBABA CLOUD

阿里云

Databricks 数据洞察

快速入门

文档版本：20220526

阿里云

法律声明

阿里云提醒您在阅读或使用本文档之前仔细阅读、充分理解本法律声明各条款的内容。如果您阅读或使用本文档，您的阅读或使用行为将被视为对本声明全部内容的认可。

1. 您应当通过阿里云网站或阿里云提供的其他授权通道下载、获取本文档，且仅能用于自身的合法合规的业务活动。本文档的内容视为阿里云的保密信息，您应当严格遵守保密义务；未经阿里云事先书面同意，您不得向任何第三方披露本手册内容或提供给任何第三方使用。

2. 未经阿里云事先书面许可，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。

3. 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。阿里云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，并在阿里云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阿里云授权渠道下载、获取最新版的用户文档。

4. 本文档仅作为用户使用阿里云产品及服务的参考性指引，阿里云以产品及服务的“现状”、“有缺陷”和“当前功能”的状态提供本文档。阿里云在现有技术的基础上尽最大努力提供相应的介绍及操作指引，但阿里云在此明确声明对本文档内容的准确性、完整性、适用性、可靠性等不作任何明示或暗示的保证。任何单位、公司或个人因为下载、使用或信赖本文档而发生任何差错或经济损失的，阿里云不承担任何法律责任。在任何情况下，阿里云均不对任何间接性、后果性、惩戒性、偶然性、特殊性或刑罚性的损害，包括用户使用或信赖本文档而遭受的利润损失，承担责任（即使阿里云已被告知该等损失的可能性）。

5. 阿里云网站上所有内容，包括但不限于著作、产品、图片、档案、资讯、资料、网站架构、网站画面的安排、网页设计，均由阿里云和/或其关联公司依法拥有其知识产权，包括但不限于商标权、专利权、著作权、商业秘密等。非经阿里云和/或其关联公司书面同意，任何人不得擅自使用、修改、复制、公开传播、改变、散布、发行或公开发表阿里云网站、产品程序或内容。此外，未经阿里云事先书面同意，任何人不得为了任何营销、广告、促销或其他目的使用、公布或复制阿里云的名称（包括但不限于单独为或以组合形式包含“阿里云”、“Aliyun”、“万网”等阿里云和/或其关联公司品牌，上述品牌的附属标志及图案或任何类似公司名称、商号、商标、产品或服务名称、域名、图案标示、标志、标识或通过特定描述使第三方能够识别阿里云和/或其关联公司）。

6. 如若发现本文档存在任何错误，请与阿里云取得直接联系。

通用约定

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

目录

1.产品入门 05
1.1. Databricks数据洞察快速使用 05
1.2. Databricks数据洞察Notebook演示 07
1.3. Notebook - 航空公司数据分析示例 12
1.4. PySpark及Python库使用 16
1.5. 机器学习开发示例 19
2.Delta Lake 26
2.1. Delta Lake 快速开始一 26
2.2. Delta Lake 快速开始二 32
3.Databricks企业版Spark & Delta引擎 42
3.1. 使用OPTIMIZE和Z-ORDER优化商品库存查询性能 42
3.2. 使用Databricks Delta优化Spark作业性能 52
3.3. 基于TPC-DS测试Databricks引擎性能 57

1.产品入门

1.1. Databricks数据洞察快速使用

本文指引您快速完成一次完整的Databricks数据洞察的使用操作。

前提条件

已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI访问OSS路径结构：oss://BucketName/Object
>
> - BucketName为您的存储空间名称；
> - Object为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val text = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

步骤一：创建集群

1. 使账号登录Databricks管控台。
2. 在Databricks数据洞察控制台页面，选择所在的地域（Region）。创建的集群将会在对应的地域内，一旦创建后不能修改。
3. 输入**集群名称**，勾选**服务条款**。其他参数如无特殊需求，可直接使用默认值。创建集群的具体步骤请参见创建集群。
4. 单击**创建**。

步骤二：添加安全组白名单

1. 要创建NoteBook，需要添加安全组白名单。默认不添加不能访问。
2. 创建好集群之后，进入集群详情页。
3. 在**集群基础信息->集群操作->安全组白名单**进行配置页面。
4. 白名单配置方法请参见-安全组白名单。

步骤三：创建Notebook

1. 在左侧导航栏中，单击**Notebook**。

2. 在Notebook区域，选择待操作的集群。
3. 单击Create new note。
4. 在Create New Note对话框中，输入Note Name、从Default Interpreter列表，选择spark。
5. 单击Create。

步骤四：创建表

1. 单击已创建的Note名称。
2. 在DataInsight Notebook页面，在单元格中创建数据库。

```
%spark.sql
create database db_demo location '/user/hive/warehouse';
```

> **说明**
>
> 在default数据库中建表会提示失败，因此需要新建一个数据库。新建数据库需要指定存储空间location，否则无法导入数据到数据库。本文的location均为示例，请您更换为您实际的location。

1. 在单元格创建表。

```
%spark.sql
use db_demo;
create table db_bank_demo(age string, job string, marital string, education string, def
ault string, balance string, housing string, loan string, contact string, day string, m
onth string, duration string, campaign string, pdays string, previous string, poutcome
string, y string) ROW FORMAT DELIMITED FIELDS TERMINATED BY ';';
```

2. 导入数据到数据库。

   示例文件：bank.csv。

   您可以下载此示例文件，上传到您的OSS目录并导入

```
%spark.sql
use db_demo;
load data inpath 'oss://databricks-dbr/db_demo/bank/bank.csv' overwrite into table db_b
ank_demo;
describe db_bank_demo;
```

   导入成功后，查看表信息如下所示。

步骤五：查询数据

数据表导入后可以通过spark sql读取数据。

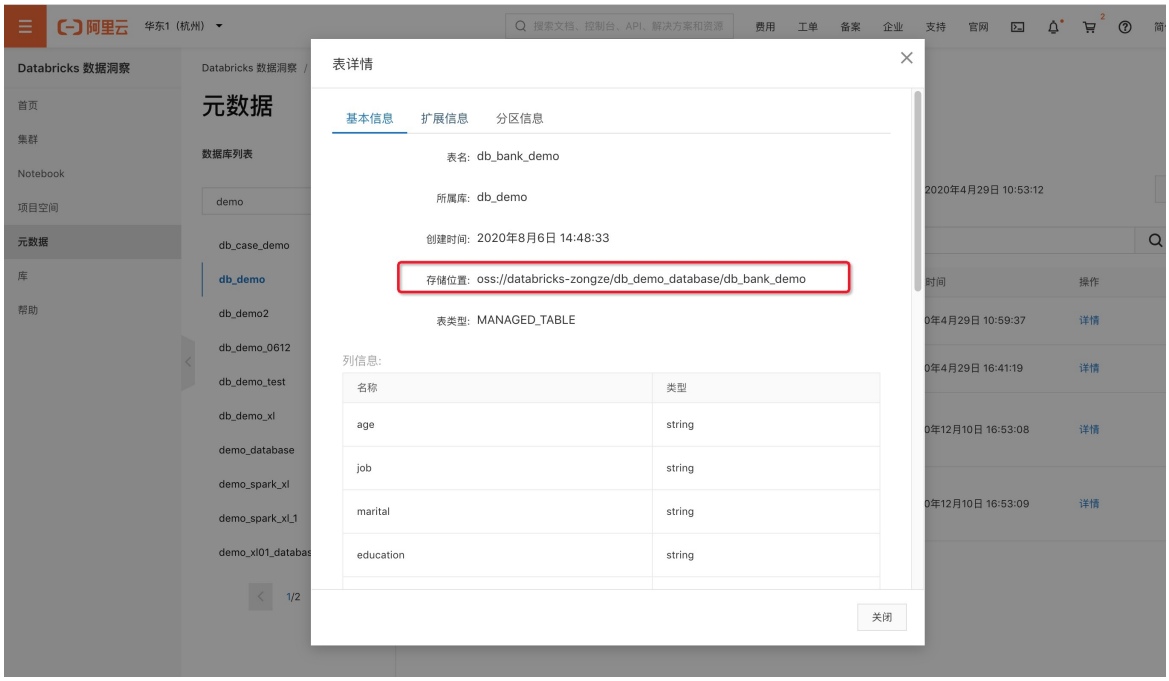
```
%spark.sql
select age, count(1) from db_bank_demo where age < ${maxAge=30} group by age order by age
```

步骤六：查看数据展示

在DataInsight Notebook页面，查看可视化数据展示信息。

步骤七：查看元数据

1. 在左侧导航栏，单击**元数据**。
2. 选择待查看的数据库。在数据库信息页面，查看数据库和表详情。



1.2. Databricks数据洞察Notebook演示

本文针对Dat abricks数据洞察Not ebook基本使用的一个示例。

前提条件

1. 通过主账号登录阿里云 Dat abricks控制台。
2. 已创建集群，具体请参见创建集群。
3. 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> 说明
>
> DDI支持免密访问OSS路径，结构为：oss://BucketName/Object
>
> - BucketName为您的存储空间名称；
> - Object为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val text = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

步骤一：创建集群并通过knox账号访问Notebook

创建集群参考: https://help.aliyun.com/document_detail/167621.html，需注意要设置ram子账号及保存好knox密码，登录WebUI时候需要用到。

步骤二：创建Notebook并做可视化调试

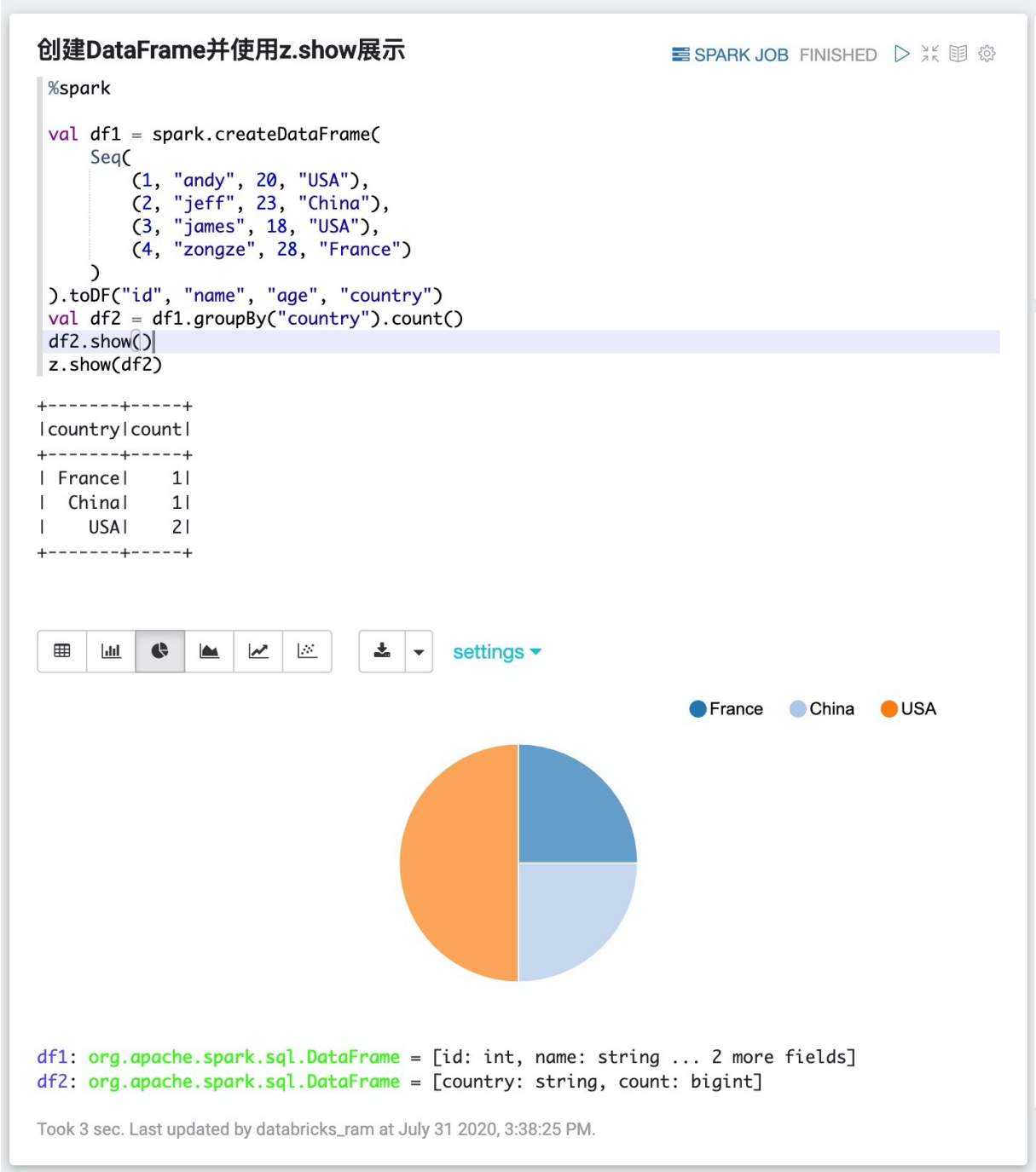
示例Note下载：CASE1-DataInsight-Notebook-Demo.zpln

示例文本下载：The_Sorrows_of_Young_Werther.txt

示例python lib下载：matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl

1. 创建DataFrame并使用z.show展示

```
%spark
val df1 = spark.createDataFrame(
    Seq(
        (1, "andy", 20, "USA"),
        (2, "jeff", 23, "China"),
        (3, "james", 18, "USA"),
        (4, "zongze", 28, "France")
    )
).toDF("id", "name", "age", "country")
val df2 = df1.groupBy("country").count()
df2.show()
z.show(df2)
```



2. 创建DataFrame并通过%spark.sql做可视化查询

```
%spark
val df1 = spark.createDataFrame(Seq((1, "andy", 20, "USA"), (2, "jeff", 23, "China"), (3, "
james", 18, "USA"), (4, "zongze", 28, "France"))).toDF("id", "name", "age", "country")
// register this DataFrame first before querying it via %spark.sql
df1.createOrReplaceTempView("people")
```

3. 测试OSS联通性，基本的WordCount示例

```
// 从oss地址读取文本文档（注意oss文件在账号下上传到对应目录）
val text = sc.textFile("oss://databricks-demo-hangzhou/The_Sorrows_of_Young_Werther.txt")
// 使用Scala做WordCount处理
val counts = text.flatMap(line => line.split(" ")).map(word => (word, 1)).reduceByKey(_+_)
// 将计算结果写入并存档到oss
counts.saveAsTextFile("oss://databricks-demo-hangzhou/WordCount示例-Result-Zeppelin-001.txt"
)
```

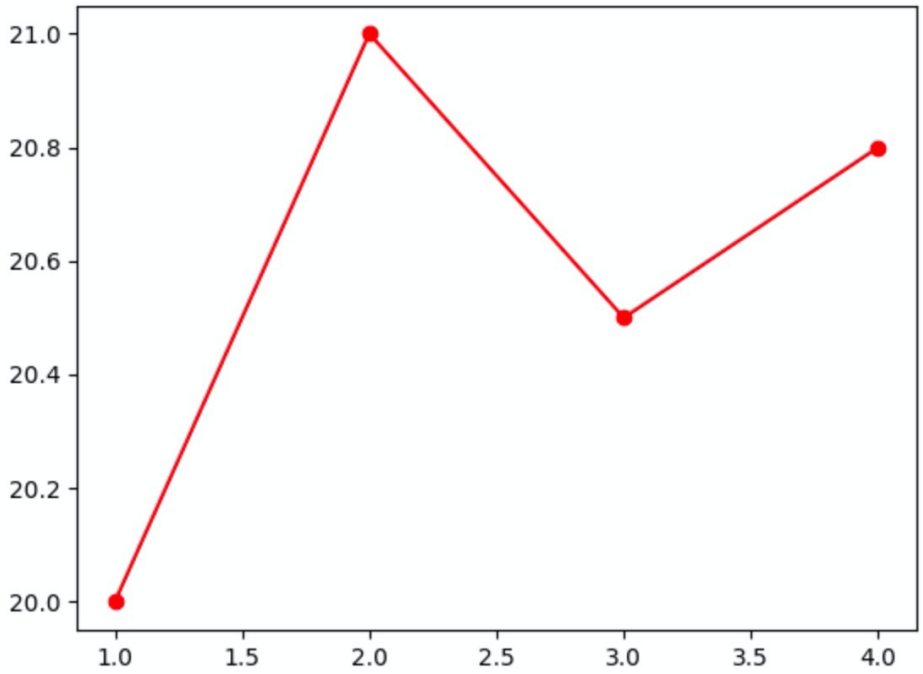
4. 库的功能示例

```
%pyspark
import matplotlib.pyplot as plt
# 1. 当使用pyspark进行开发时候，依赖的库可以通过ddi库能力导入，具体见：https://help.aliyun.com/document_detail/168087.html
# 2. 这里示例导入matplotlib库，并做展示—先download下官方库安装包，在库功能下进行.whl文件的上传，在需要使用库的集群里进行安装
plt.clf()
#define some data
x = [1,2,3,4]
y = [20, 21, 20.5, 20.8]
#plot data
plt.plot(x, y, marker="o", color="red")
z.show(plt)
```

库的功能示例

FINISHED

```
%pyspark
import matplotlib.pyplot as plt

# 1. 当使用pyspark进行开发时候，依赖的库可以通过ddi库能力导入，具体见：https://help.aliyun
    .com/document_detail/168087.html
# 2. 这里示例导入matplotlib库，并做展示--先download下官方库安装包，在库功能下进行
    .whl文件的上传，在需要使用库的集群里进行安装

plt.clf()
#define some data
x = [1,2,3,4]
y = [20, 21, 20.5, 20.8]

#plot data
plt.plot(x, y, marker="o", color="red")
z.show(plt)
```



Took 0 sec. Last updated by databricks_ram at July 31 2020, 3:31:26 PM. (outdated)

1.3. Notebook - 航空公司数据分析示例

本文针对Notebook的使用，做一个具体的场景示例——航空公司数据导入及分析。

前提条件

1. 通过主账号登录阿里云 Dat abricks控制台。
2. 已创建集群，具体请参见创建集群。
3. 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI支持免密访问OSS路径，结构为：oss://BucketName/Object
>
> - **BucketName**为您的存储空间名称；
> - **Object**为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val text = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

步骤一：创建集群并通过knox账号访问Notebook

创建集群参考: https://help.aliyun.com/document_detail/167621.html，需注意要设置RAM子账号及保存好knox密码，登录WebUI时候需要用到。

步骤二：创建Notebook、导入数据、进行数据分析

示例数据下载：airline_statistic_usa.csv

1. 读取OSS数据、打印schema，创建TempView

Load OSS data

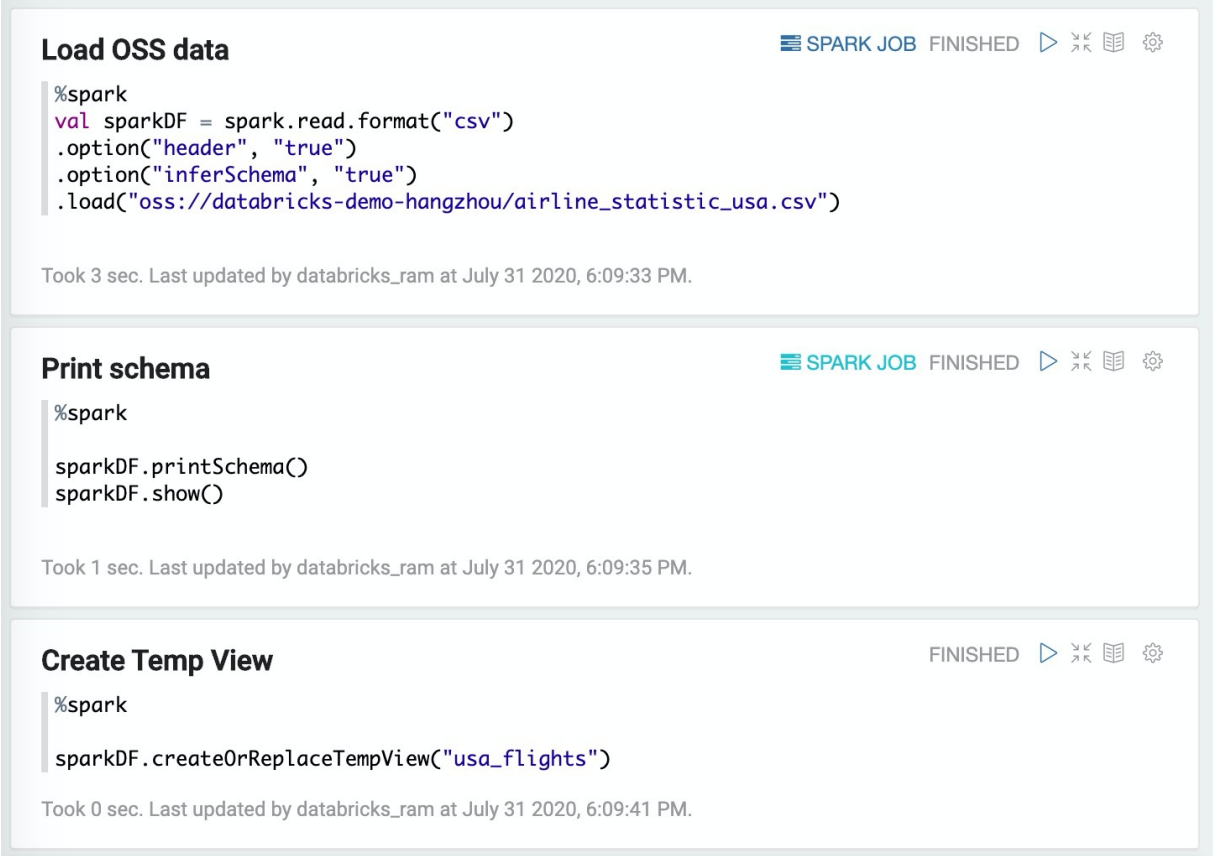
```
%spark
val sparkDF = spark.read.format("csv")
.option("header", "true")
.option("inferSchema", "true")
.load("oss:/
/databricks-demo-hangzhou
/airline_statistic_usa.csv")
```

Print schema

```
%spark
sparkDF.printSchema()
sparkDF.show()
```

Create Temp View

```
%spark
sparkDF.createOrReplaceTempView("usa_flights")
```



2. 查询分析：Analysis，Top 10 Average Distance Traveled By Flight Carrier

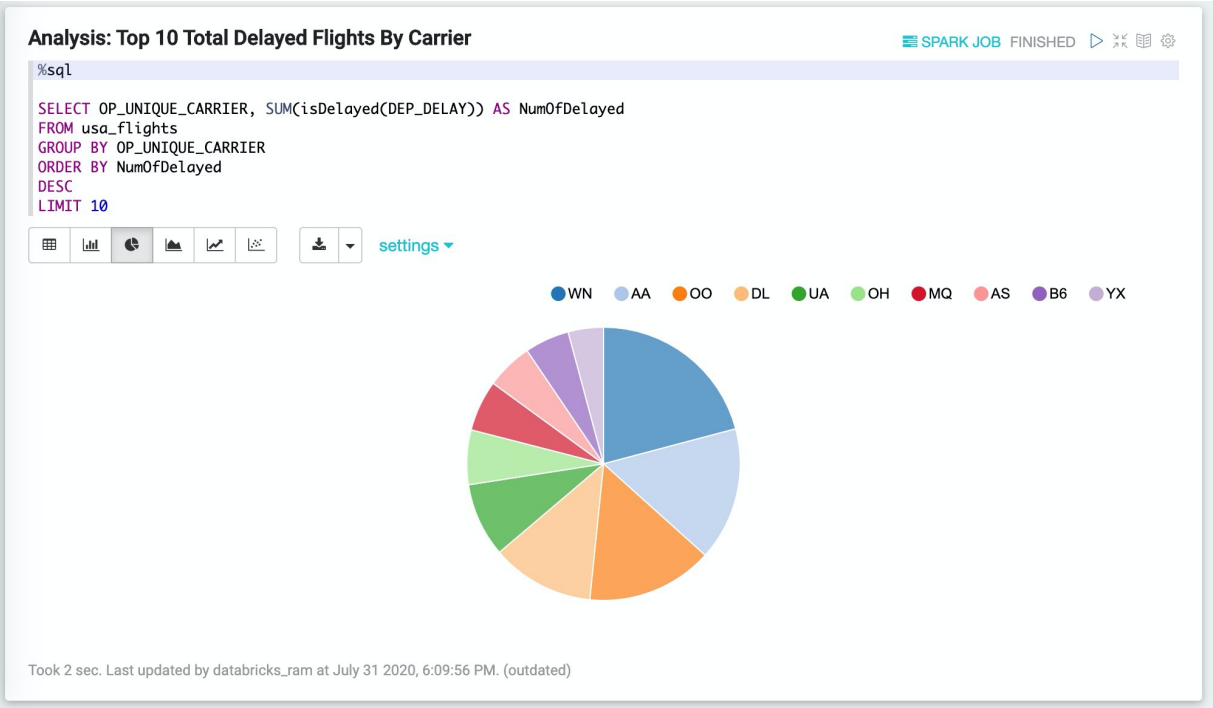
```
%sql
SELECT OP_UNIQUE_CARRIER, CAST(AVG(DISTANCE) AS INT) AS AvgDistance
FROM usa_flights
GROUP BY OP_UNIQUE_CARRIER
ORDER BY AvgDistance
DESC
LIMIT 20
```

3. 定义UDF：Define UDF

```
%pyspark
from pyspark.sql.types import IntegerType
def isDelayed(x):
    if x == None:
        return 0
    elif int(x) > 10:
        return 1
    else:
        return 0
spark.udf.register("isDelayed", isDelayed, IntegerType())
```

4. 使用UDF进行查询：Analysis，Top 10 Total Delayed Flights By Carrier

```
%sql
SELECT OP_UNIQUE_CARRIER, SUM(isDelayed(DEP_DELAY)) AS NumOfDelayed
FROM usa_flights
GROUP BY OP_UNIQUE_CARRIER
ORDER BY NumOfDelayed
DESC
LIMIT 10
```



1.4. PySpark及Python库使用

本文介绍如何使用阿里云 Databricks数据洞察 Notebook 进行 PySpark 开发。

> **注意**
>
> 若要使用其他数据源进行数据开发，需开通相应服务。本示例采用OSS数据源。

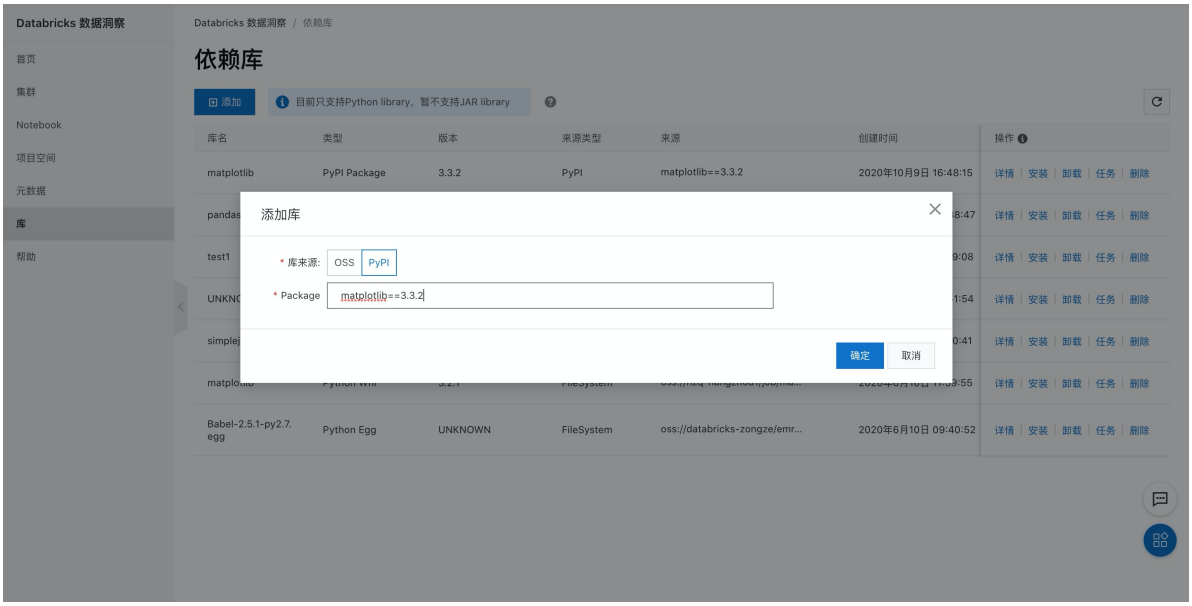
步骤一：创建 Databricks数据洞察集群

- 登录阿里云Databricks数据洞察控制台。
- 创建Databricks 数据洞察集群，详情参见集群创建。

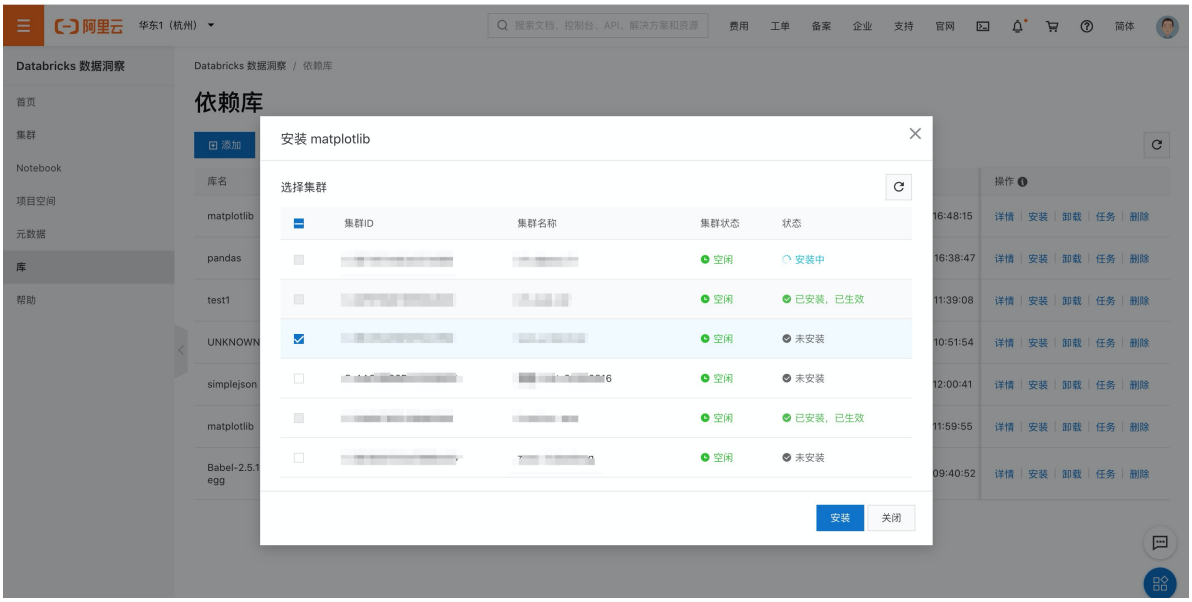
步骤二：添加依赖库并安装

根据开发需要，添加相应的依赖库

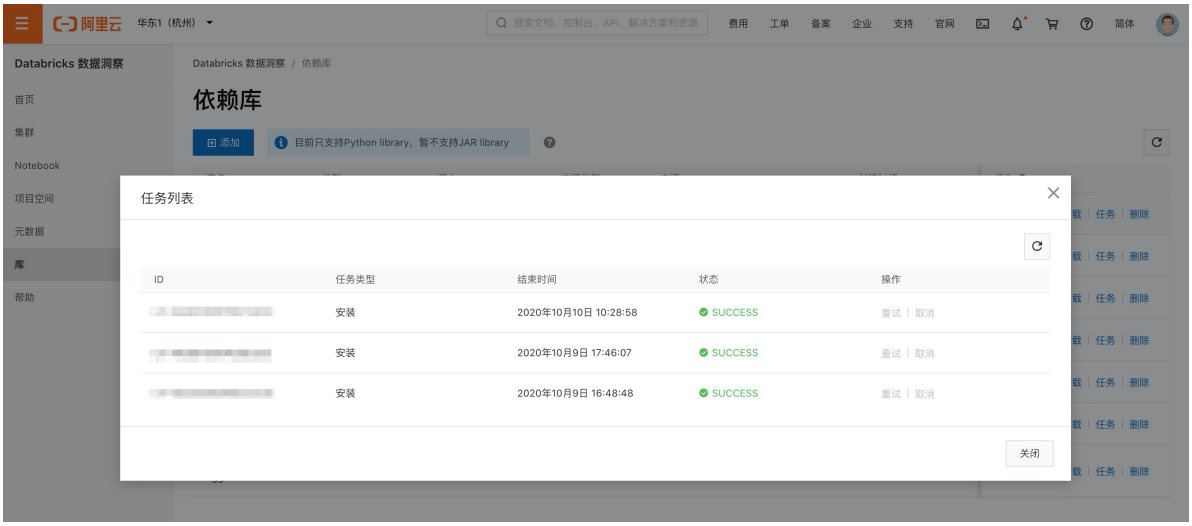
- 添加matplotlib库，本示例使用PyPI方式添加，详情参见管理库。



- 单击**安装**按钮，安装依赖到开发集群。



- 等待安装完成，可单击**任务**按钮查看。



步骤三：获取数据并上传对象存储 OSS

- 登录OSS管理控制台。
- 创建Bucket存储空间，详情请参见存储空间。
- 获取测试文件（TEST.txt）。
- 上传文件，详情请参见上传文件。

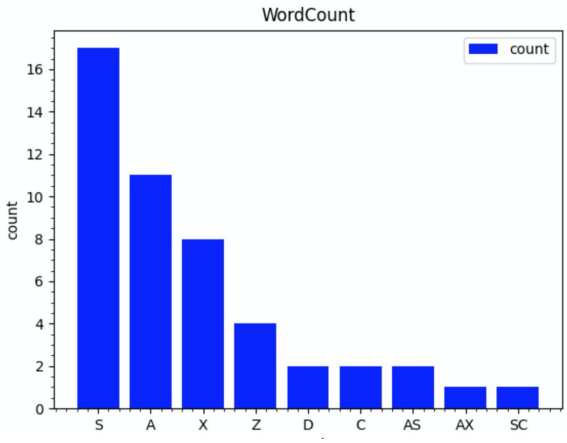
步骤四：数据开发

- Notebook使用，详情参见Notebook使用。
- 数据开发，将以下代码写入note文件，如下图所示。

```
%pyspark
import matplotlib.pyplot as plt

data = sc.textFile('oss://xxx/xxx/TEST.txt')

data = data.flatMap(lambda line: line.split(" "))

resultRdd = data.map(lambda word: (word,1)).reduceByKey(lambda a,b:a+b)

result = resultRdd.sortBy(lambda x: x[1], False)

resultColl = resultRdd.collect()

x = result.keys().collect()
y = result.values().collect()
plt.bar(x, y, color="b", label="count")
plt.minorticks_on()
plt.title("WordCount")
plt.xlabel("word")
plt.ylabel("count")
plt.legend()
z.show(plt)
```

- 单击右上角**运行**按钮，等待任务结束查看结果

```
data = sc.textFile('oss:                                    ST.txt')

data = data.flatMap(lambda line: line.split(" "))

resultRdd = data.map(lambda word: (word,1)).reduceByKey(lambda a,b:a+b)

result = resultRdd.sortBy(lambda x: x[1], False)

resultColl = resultRdd.collect()

x = result.keys().collect()
y = result.values().collect()
plt.bar(x, y, color="b", label="count")
plt.minorticks_on()
plt.title("WordCount")
plt.xlabel("word")
plt.ylabel("count")
plt.legend()
z.show(plt)
```



1.5. 机器学习开发示例

本文介绍如何使用阿里云 Databricks 数据洞察的Notebook进行机器学习开发。

前提条件

- 通过主账号登录阿里云 Databricks控制台。
- 已创建集群，具体请参见创建集群。
- 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI访问OSS路径结构：oss://BucketName/Object
>
> - BucketName为您的存储空间名称。
> - Object为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val dataRDD = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

开发流程

在本教程中，您将执行以下步骤：

1. 创建集群并通过knox账号访问Notebook
2. 创建Notebook、加载样本数据
3. 准备ML算法数据
4. 建立模型、运行线性回归模型
5. 评估线性回归模型
6. 可视化线性回归模型

步骤一：创建集群并通过knox账号访问Notebook

创建集群参考: https://help.aliyun.com/document_detail/167621.html，需注意要设置ram子账号及保存好knox密码，登录WebUI时候需要用到。

步骤二：创建Notebook、加载样本数据

示例Note下载：CASE7-DataInsight-Notebook-ML.zpln

示例数据下载：data_geo.csv

示例python lib下载：matplotlib-3.2.1-cp37-cp37m-manylinux1_x86_64.whl

> **说明**
>
> 开始进行机器学习的最简单方法是使用对象存储工作区中可访问的文件夹中的示例Databricks数据集 。例如，要访问将城市人口数量与房屋平均价格进行比较的文件，您可以通过oss路径访问文件。例如：oss://databricks-datasets/samples/data_geo.csv。

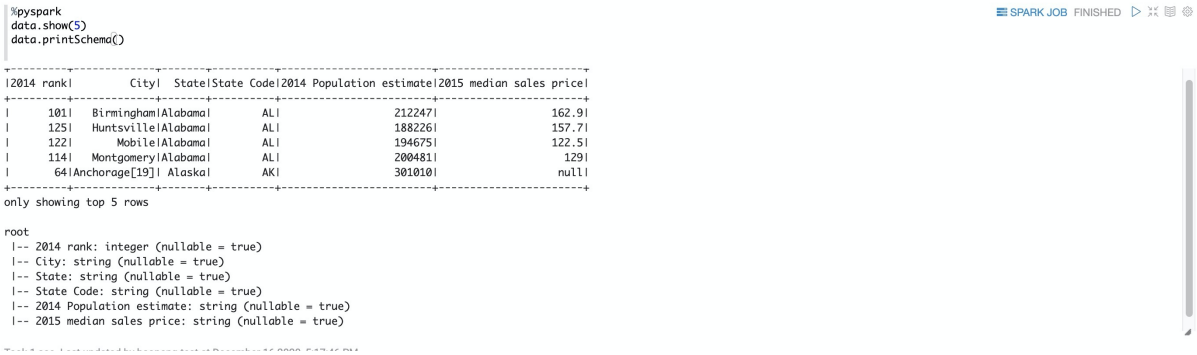
1.加载oss数据，并缓存

```
%pyspark
path="oss://databricks-datasets/samples/data_geo.csv"  # 根据oss路径;
# 加载样本数据并打印;
data = spark.read \
  .option("header", "true") \
  .option("inferSchema", "true") \
  .csv(path) \

data.cache() # 缓存数据以加快重用
```

2.数据展示，打印schema

```
%pyspark
data.show(5)
data.printSchema()
```

数据打印



步骤三：准备ML算法数据

> **说明**
>
> 在监督学习（例如回归算法）中，通常需要定义标签(lable)和一组要素(features)。在此线性回归示例中，标签为2015年中位数销售价格(lable)，特征为2014年人口估计(features)。您使用特征来预测标签（售价)。

1.数据类型转换，删除缺少值的行，然后重命名特征和标签列，并用"_"替换空格

```
%pyspark

from pyspark.sql.functions import col
from pyspark.sql.types import DoubleType
#数据类型转换
datat=data.select(col("2014 rank"),col("city"),col("state"),col("State Code"),col("2014 Pop
ulation estimate"),col("2015 median sales price").cast(DoubleType()))
#删除缺少值的行
datat = datat.dropna()
#重命名特征和标签列，并用"_"替换空格
exprs = [col(column).alias(column.replace(' ', '_')) for column in datat.columns]
```

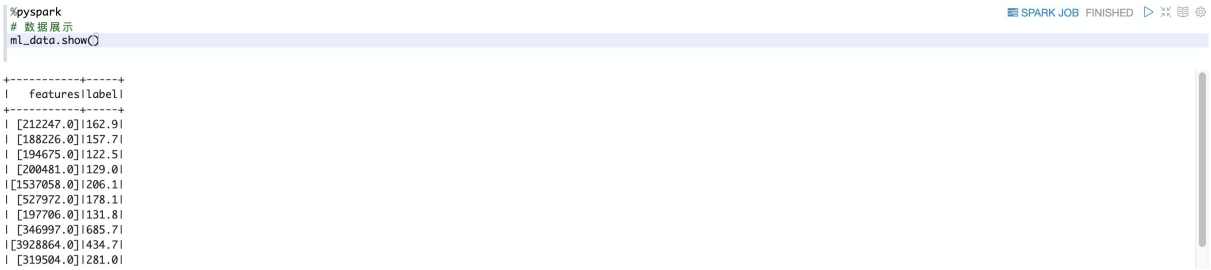
2.为了简化特征的创建，请注册UDF以将特征（2014_Population_estimate）列向量转换为VectorUDT类型并将其应用于该列

```
%pyspark

from pyspark.ml.linalg import Vectors, VectorUDT
# 定义UDF函数，数据向量化处理;
spark.udf.register("oneElementVec", lambda d: Vectors.dense([d]), returnType=VectorUDT())
ml_data = datat.select(*exprs).selectExpr("oneElementVec(2014_Population_estimate) as featu
res", "2015_median_sales_price as label")
```

3.数据展示

```
%pyspark
# 数据展示
ml_data.show()
```

ML算法数据打印



步骤四：建立模型、运行线性回归模

> **说明**
>
> 在本部分中，您将使用不同的正则化参数运行两个不同的线性回归模型，以基于人口（features）确定这两个模型中的任何一个对销售价格（label）的预测程度。

1.建立模型

```
%pyspark

from pyspark.ml.regression import LinearRegression
# 定义LinearRegression算法
lr = LinearRegression()
# 建立模型
modelA = lr.fit(ml_data, {lr.regParam:0.0})
modelB = lr.fit(ml_data, {lr.regParam:100.0})
```

2.使用该模型并作出预测，您可以使用transform()函数进行预测，该函数会添加新的预测列。例如下面的代码采用第一个模型（modelA），并根据特征（features）向您显示标签（原始销售价格）和预测（预测销售价格）

```
%pyspark

# 运行线性回归模型，并展示数据
predictionsA = modelA.transform(ml_data)
predictionsA.show(10)
```

数据打印

```
%pyspark

# 运行线性回归模型，并展示数据
predictionsA = modelA.transform(ml_data)
predictionsA.show(10)
```

SPARK JOB FINISHED

```
+----------+-----+------------------+
|  features|label|        prediction|
+----------+-----+------------------+
| [212247.0]|162.9| 199.31676595846641
| [188226.0]|157.7|198.40882267887193|
| [194675.0]|122.5|198.65258131548592|
| [200481.0]|129.0|198.87203590444247|
|[1537058.0]|206.1|249.39183544694856|
| [527972.0]|178.1|211.25050693302884|
| [197706.0]|131.8| 198.76714674075761
| [346997.0]|685.7| 204.41003255541721
|[3928864.0]|434.7|339.79707185649573|
| [319504.0]|281.0|203.37085497805194|
+----------+-----+------------------+
only showing top 10 rows
```

步骤五：评估线性回归模型

> **说明**
>
> 要评估回归分析，请使用RegressionEvaluator来计算均方根误差 。
>
> 在机器学习中我们用计算测试值和预测值之间出现的误差的均方根的平均值来查看模型的准确性。

```
%pyspark

from pyspark.ml.evaluation import RegressionEvaluator
# 使RegressionEvaluator用来计算均方根误差 。
evaluator = RegressionEvaluator(metricName="rmse")
RMSE = evaluator.evaluate(predictionsA)
print("ModelA: Root Mean Squared Error = " + str(RMSE))
# ModelA: Root Mean Squared Error = 128.602026843

predictionsB = modelB.transform(ml_data)
RMSE = evaluator.evaluate(predictionsB)
print("ModelB: Root Mean Squared Error = " + str(RMSE))
# ModelB: Root Mean Squared Error = 129.496300193
```

RMSE评估结果打印

```
%pyspark

from pyspark.ml.evaluation import RegressionEvaluator
# 使RegressionEvaluator用来计算均方根误差 。
evaluator = RegressionEvaluator(metricName="rmse")
RMSE = evaluator.evaluate(predictionsA)
print("ModelA: Root Mean Squared Error = " + str(RMSE))
# ModelA: Root Mean Squared Error = 128.602026843

predictionsB = modelB.transform(ml_data)
RMSE = evaluator.evaluate(predictionsB)
print("ModelB: Root Mean Squared Error = " + str(RMSE))
# ModelB: Root Mean Squared Error = 129.496300193
```

SPARK JOB FINISHED

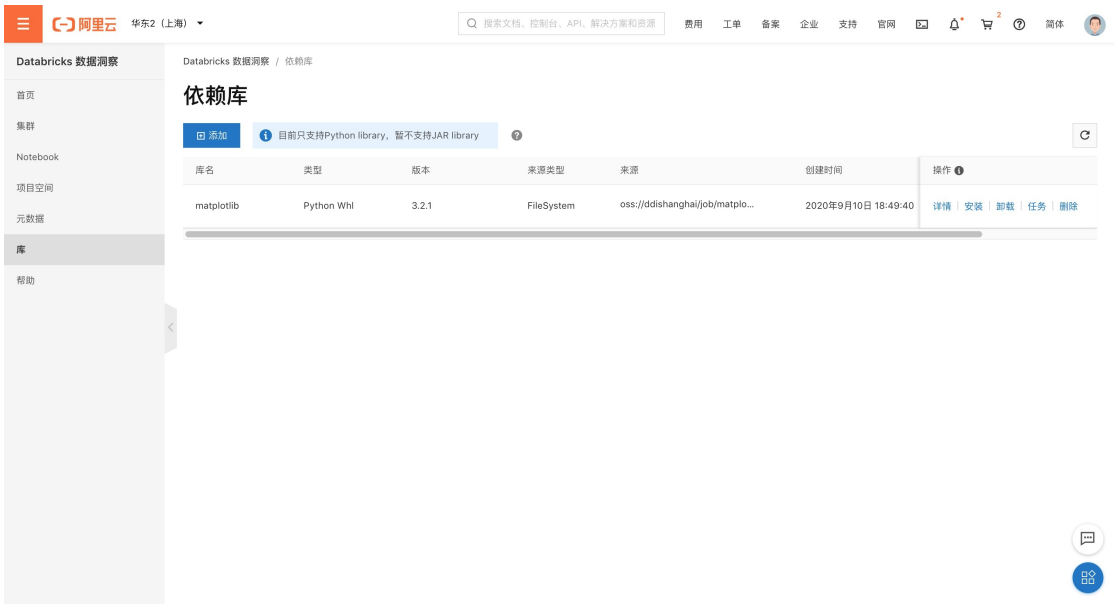
```
ModelA: Root Mean Squared Error = 128.60202684284758
ModelB: Root Mean Squared Error = 129.49630019270606
```

> **说明**
>
> 我们的RMSE值约为128～130。 好的模型的RMSE值应小于180。如果您具有较高的RMSE值，则意味着您可能需要更改功能或调整超参数。

步骤六：可视化线性回归模型

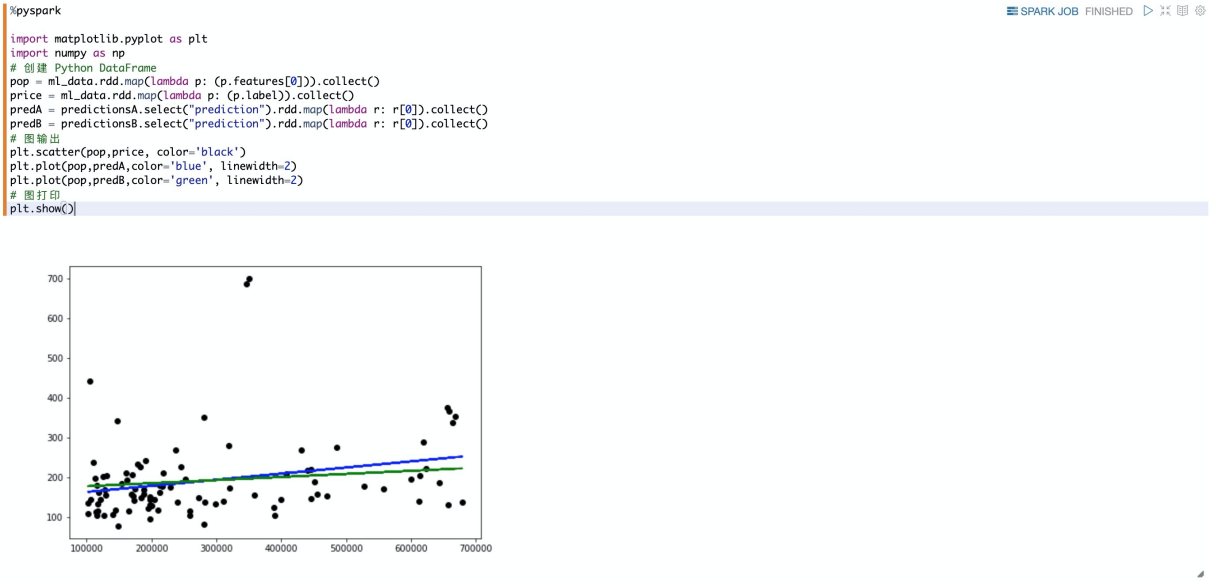
> **说明**
>
> 可视化散点图，与许多机器学习算法一样使用matplotlib创建线性回归图以显示散点图和两个回归模型。

1. 依赖库导入
   i. 当使用pyspark进行开发时候，依赖的库可以通过ddi库能力导入，具体见：https://help.aliyun.com/document_detail/168087.html
   ii. 这里示例导入matplotlib库，并做展示——先download下官方库安装包，在库功能下进行.whl文件的上传，在需要使用库的集群里进行安装



2.创建Python DataFrame 、使用matplotlib做图渲染 、图打印

```
%pyspark
#为展示效果，本实例对样本数据做了部分筛选；
import matplotlib.pyplot as plt
import numpy as np
# 创建 Python DataFrame
pop = ml_data.rdd.map(lambda p: (p.features[0])).collect()
price = ml_data.rdd.map(lambda p: (p.label)).collect()
predA = predictionsA.select("prediction").rdd.map(lambda r: r[0]).collect()
predB = predictionsB.select("prediction").rdd.map(lambda r: r[0]).collect()
# 图渲染
plt.scatter(pop,price, color='black')
plt.plot(pop,predA,color='blue', linewidth=2)
plt.plot(pop,predB,color='green', linewidth=2)
# 图打印
z.show(plt)
```



2.Delta Lake

2.1. Delta Lake 快速开始一

该Notebook展示了如何将JSON数据转换为Delta Lake格式，创建Delta表，在Delta表中Append数据，最后使用Delta Lake元数据命令来显示表的历史记录、格式和详细信息。

前提条件

1. 通过主账号登录阿里云 Databricks控制台。
2. 已创建集群，具体请参见创建集群。
3. 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI支持免密访问OSS路径，结构为：oss://BucketName/Object
>
> - BucketName为您的存储空间名称；
> - Object为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件
>
> ```
> // 从oss地址读取文本文档
> val text = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
> ```

详情可参考参考Databricks官网Blog文章

步骤一：创建集群并通过knox账号访问Notebook

创建集群参考: https://help.aliyun.com/document_detail/167621.html，需注意要设置ram子账号及保存好knox密码，登录WebUI时候需要用到。

步骤二：创建Notebook、导入数据、进行数据分析

示例Note下载：CASE5-Delta快速开始（Dome）.zpln

示例数据下载：events.json

1. **定义Notebook中使用的路径path**

```
%pyspark

# 注意需要将数据文件events.json上传至您的OSS对应bucket下，events.json数据来源Databricks站点的open/close数据
inputPath = "oss://databricks-huhehaote/delta-demo/events.json"
deltaPath = "/delta/events"
database = "Delta_QuickStart_Database"
```

2. 导入数据到Dataframe中 & 打印数据data

```
%pyspark

from pyspark.sql.functions import expr
from pyspark.sql.functions import from_unixtime

# spark.read读取json数据，并将表头time转换为date格式
events = spark.read \
  .option("inferSchema", "true") \
  .json(inputPath) \
  .withColumn("date", expr("time")) \
  .drop("time") \
  .withColumn("date", from_unixtime("date", 'yyyy-MM-dd'))
events.show()
```

3. 将数据使用Delta格式写入

```
%pyspark
events.write.format("delta").mode("overwrite").partitionBy("date").save(deltaPath)
```

4. 再次读取数据查看是否成功保存

```
%pyspark
events_delta = spark.read.format("delta").load(deltaPath)
events_delta.printSchema()
```

5. 重置数据库

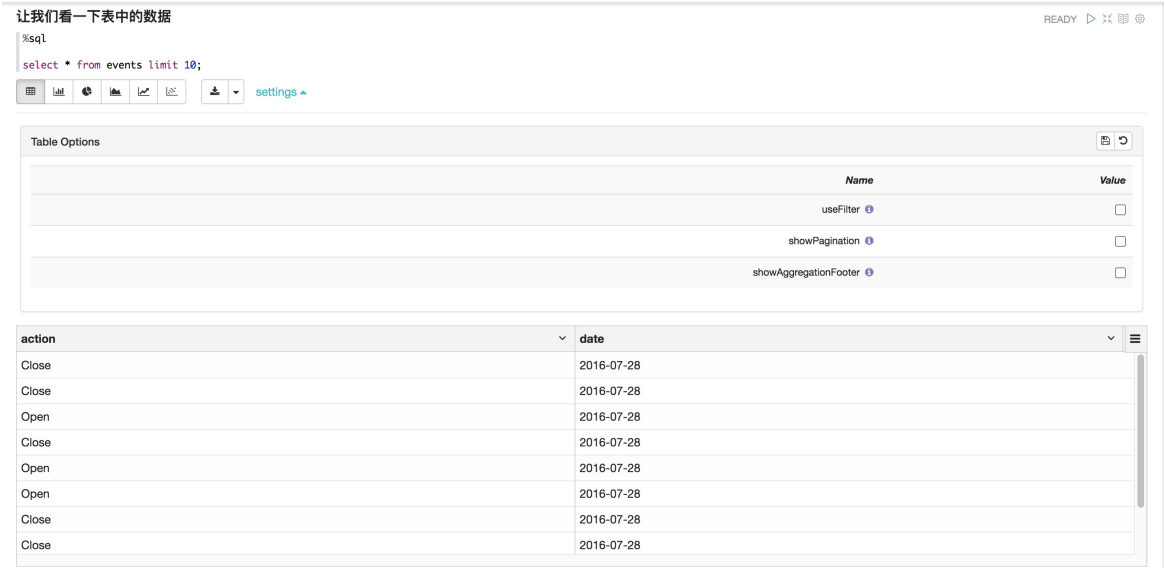
```
%pyspark

spark.sql("DROP DATABASE IF EXISTS {} CASCADE".format(database)) # 注意{}是在pyspark里spark.sql()中使用的变量，参数在.format中指定 （参考：https://stackoverflow.com/questions/44582450/how-to-pass-variables-in-spark-sql-using-python)
spark.sql("CREATE DATABASE IF NOT EXISTS {}".format(database))
```

6. **使用Delta创建表**

```
%pyspark

spark.sql("USE {}".format(database))
spark.sql("CREATE TABLE events USING DELTA LOCATION \"{}\"".format(deltaPath))
```

7. **查看表中的数据**

```
%sql
select * from events limit 10;
```



8. **对数据执行一个简单的count**

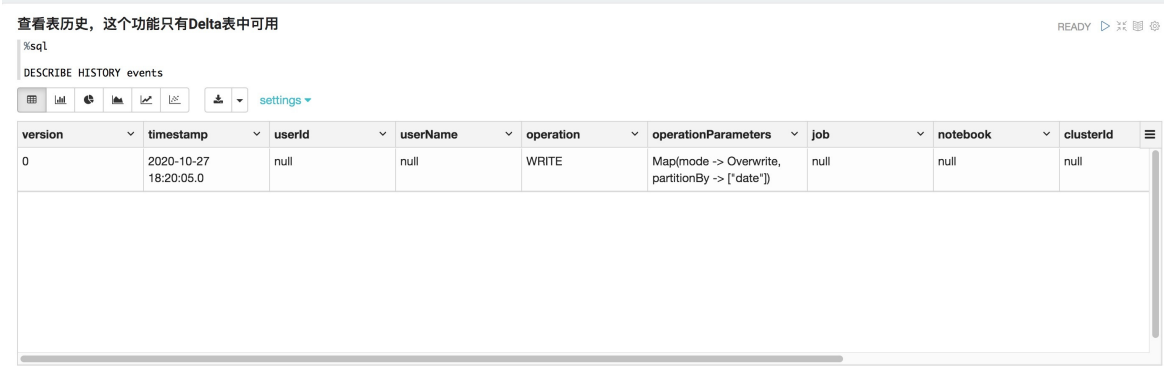
```
%pyspark

events_delta.count()
```

9. **查看events详情**

```
%sql
DESCRIBE DETAIL events;
```

10. **查看表历史，这个功能只有Delta表中可用**

```
%sql

DESCRIBE HISTORY events
```

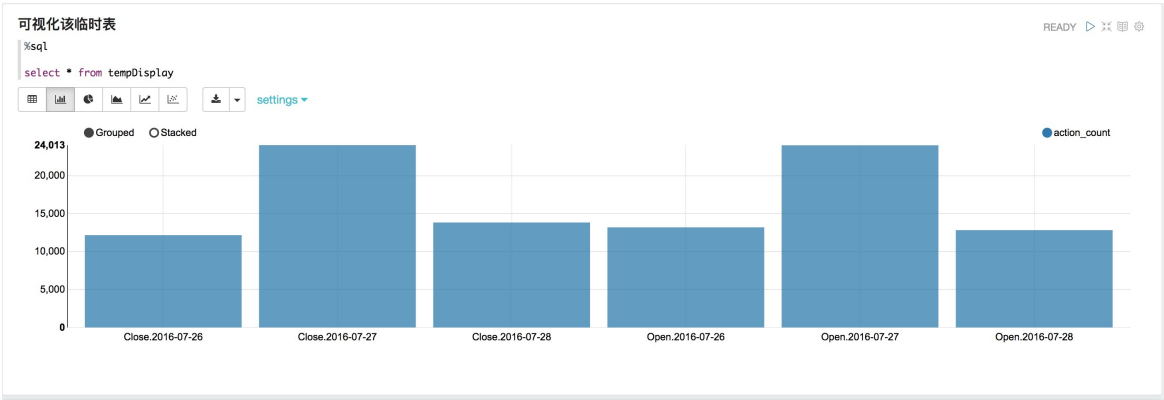


11. **做一个聚合（aggregation）操作并将结果保存在一个临时表中**

```
%pyspark

from pyspark.sql.functions import count
events_delta.groupBy("action","date").agg(count("action").alias("action_count")).orderBy("date", "action").createOrReplaceTempView("tempDisplay")
```

12. **可视化该临时表**

```
%sql

select * from tempDisplay
```



13. **我们可以很方便的使用append模式追加新数据**

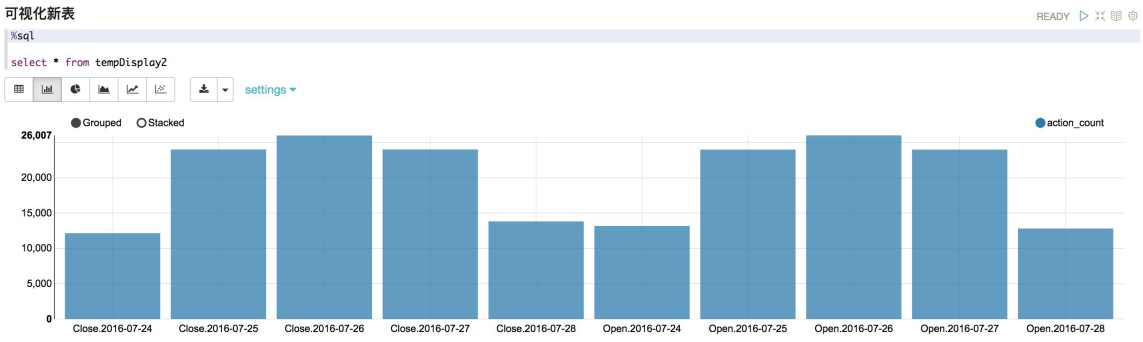
```
%pyspark

# 该操作主要将数据中时间前移2天（172800秒）
historical_events = spark.read \
  .option("inferSchema", "true") \
  .json(inputPath) \
  .withColumn("date", expr("time-172800")) \
  .drop("time") \
  .withColumn("date", from_unixtime("date", 'yyyy-MM-dd'))

historical_events.write.format("delta").mode("append").partitionBy("date").save(deltaPath)

historical_events.show()
```

14. **聚合（aggregate）并查看新数据**

```
%pyspark

events_delta.groupBy("action","date").agg(count("action").alias("action_count")).orderBy("date", "action").createOrReplaceTempView("tempDisplay2")
```

15. **可视化新表**

```
%sql

select * from tempDisplay2
```

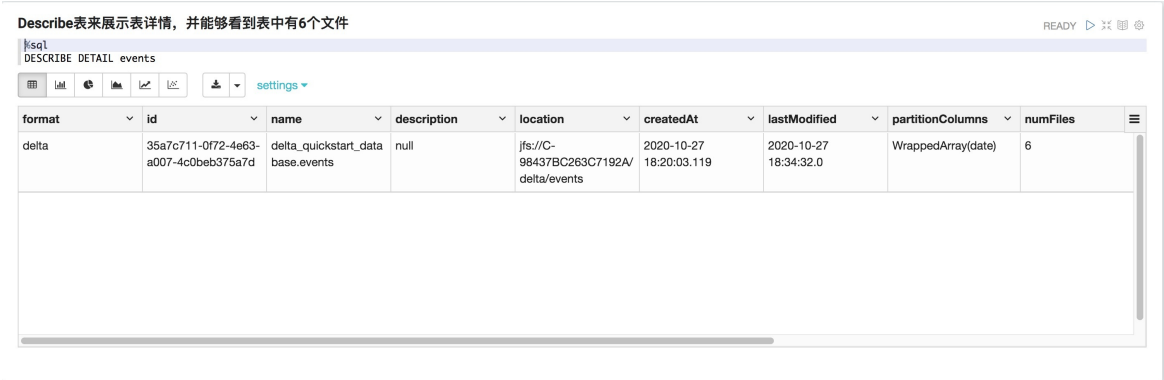


16. **执行count操作查看新append数据行**

```
%pyspark

events_delta.count()
```

17. **Describe表来展示表详情，并能够看到表中有6个文件**

```
%sql
DESCRIBE DETAIL events
```

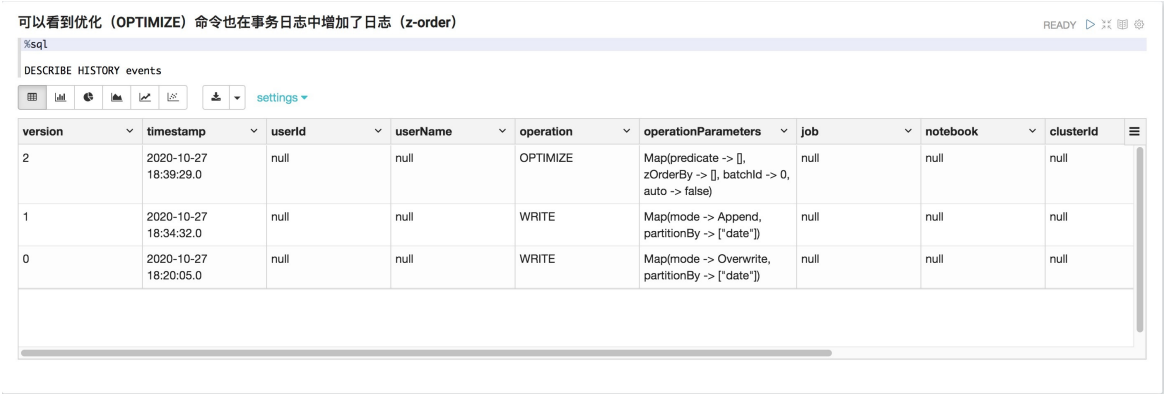


18. **Databricks支持优化（OPTIMIZE）合并文件以提升性能**

```
%pyspark

spark.sql("OPTIMIZE events")
```

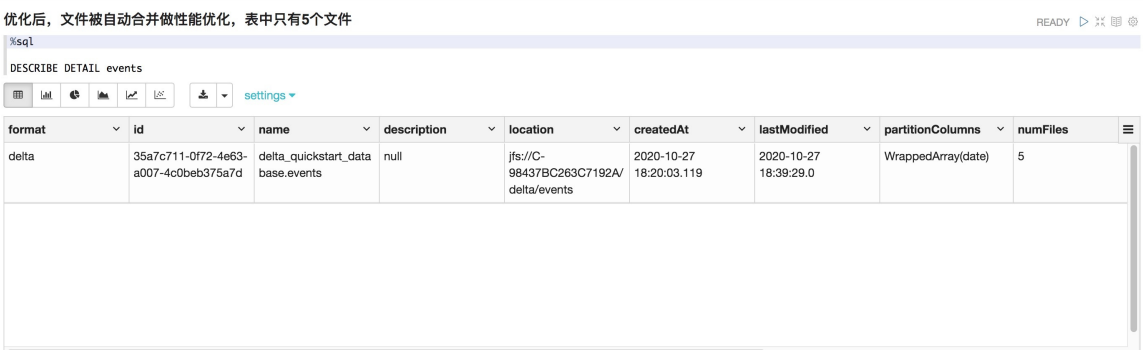
19. **可以看到优化（OPTIMIZE）命令也在事务日志中增加了日志（z-order）**

```
%sql

DESCRIBE HISTORY events
```



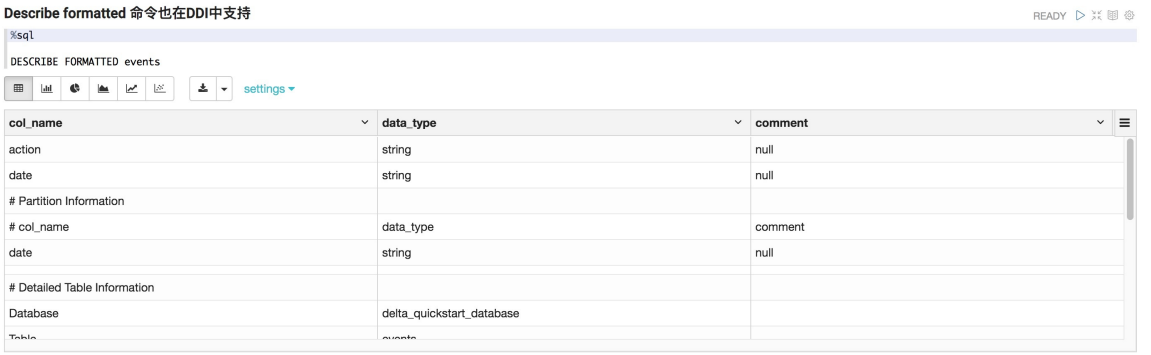
20. **优化后，文件被自动合并做性能优化，表中只有5个文件**

```
%sql

DESCRIBE DETAIL events
```



21. **Describe formatted 命令也在DDI中支持**

```
%sql

DESCRIBE FORMATTED events
```



附录

清理数据库

```
%sql
/*
DROP DATABASE IF EXISTS Delta_QuickStart_Database1 CASCADE;
*/
```

2.2. Delta Lake 快速开始二

本文介绍如何创建数据库、表批读写、表流读写、表删除、更新、合并以及版本控制等功能。

前提条件

- 通过主账号登录阿里云 Databricks控制台。
- 已创建集群，具体请参见创建集群。
- 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI访问OSS路径结构：oss://BucketName/Object
>
> - BucketName为您的存储空间名称。
> - Object为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val dataRDD = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

创建Notebook、导入数据、进行数据分析

示例Note下载：快速入门二Case

示例数据下载：events_data.json

1. **创建数据库**

   i. 不指定路径创建数据库，创建的数据库会存储在当前集群hive路径中

```
%pyspark
# 创建数据库目录，你也可以选择自定义目录
database="db_test"
spark.sql("DROP DATABASE IF EXISTS {} CASCADE".format(database))
spark.sql("CREATE DATABASE {}".format(database))
spark.sql("USE {}".format(database))
```

   ii. 指定路径创建数据库，可以使用OSS路径创建

```
%pyspark
database="db_dome"
#指定路径创建库和表；本掩饰路径为dome路径，您可以使用真实的路径
location='oss://dome-test/case6/';
spark.sql("DROP DATABASE IF EXISTS {} CASCADE".format(database))
spark.sql("CREATE DATABASE {} location 'oss://dome-test/case6/' ".format(database))
spark.sql("USE {}".format(database))
```

> **说明**
>
> 您可以在Databricks数据洞察控制台中的元数据管理中查看创建的数据库和数据表

2. **创建delta表/delta分区表/parquet格式表/元存储表**

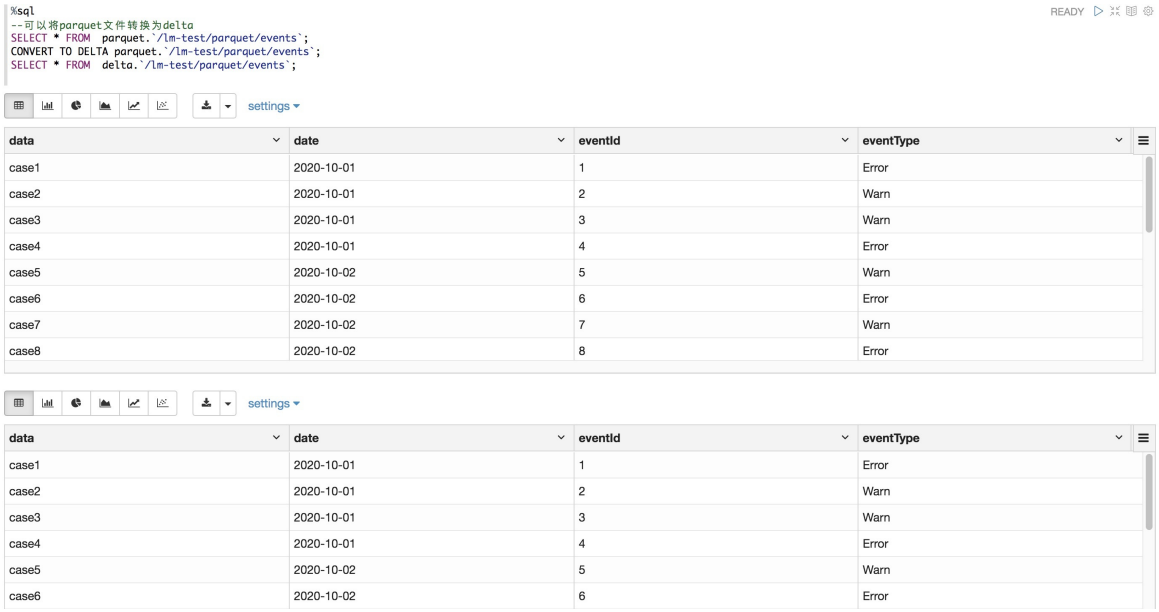
```
%pyspark
from pyspark.sql.functions import expr
from pyspark.sql.functions import from_unixtime, to_date
# 定义路径
inputPath = "/dome-test/events_data.json"
deltaPath = "/dome-test/delta/events"
partitionDeltaPath = "/dome-test/delta/events_partition"
parquetPath = "/dome-test/parquet/events"
jsonPath = "/dome-test/json/events"
events = spark.read \
  .option("inferSchema", "true") \
  .json(inputPath) \
  .withColumn("date", to_date('date', 'yyyy-MM-dd'))
events.show()
events.printSchema()
# 创建delta表/delta分区表/parquet格式表/元存储表
events.write.format("delta").mode("overwrite").save(deltaPath)
events.write.format("delta").mode("overwrite").partitionBy("date").save(partitionDeltaP
ath)
events.write.format("parquet").mode("overwrite").save(parquetPath)
events.write.format("delta").saveAsTable("events")
```

```
inputPath = "/lm-test/events_data.json"
deltaPath = "/lm-test/delta/events"
partitionDeltaPath = "/lm-test/delta/events_partition"
parquetPath = "/lm-test/parquet/events"
jsonPath = "/lm-test/json/events"

events = spark.read \
  .option("inferSchema", "true") \
  .json(inputPath) \
  .withColumn("date", to_date('date', 'yyyy-MM-dd'))
events.show()
events.printSchema()
# 创建delta表/delta分区表/parquet格式表/元存储表
events.write.format("delta").mode("overwrite").save(deltaPath)
events.write.format("delta").mode("overwrite").partitionBy("date").save(partitionDeltaPath)
events.write.format("parquet").mode("overwrite").save(parquetPath)
events.write.format("delta").saveAsTable("events")
```

```
+------+----------+-------+---------+
|  data|      date|eventId|eventType|
+------+----------+-------+---------+
| case1|2020-10-01|      1|    Error|
| case2|2020-10-01|      2|     Warn|
| case3|2020-10-01|      3|     Warn|
| case4|2020-10-01|      4|    Error|
| case5|2020-10-02|      5|     Warn|
| case6|2020-10-02|      6|    Error|
| case7|2020-10-02|      7|     Warn|
| case8|2020-10-02|      8|    Error|
| case9|2020-10-02|      9|    Error|
|case10|2020-10-03|     10|     Warn|
|case11|2020-10-03|     11|    Error|
|case12|2020-10-04|     12|    Error|
|case13|2020-10-04|     13|     Info|
|case14|2020-10-04|     14|     Info|
|case15|2020-10-04|     15|     Info|
```

3. **将parquet文件转换为delta**

```
%sql
--可以将parquet文件转换为delta
SELECT * FROM  parquet.`/dome-test/parquet/events`;
CONVERT TO DELTA parquet.`/dome-test/parquet/events`;
SELECT * FROM  delta.`/dome-test/parquet/events`;
```



```
%sql
--可以将parquet文件转换为delta
SELECT * FROM  parquet.`/lm-test/parquet/events`;
CONVERT TO DELTA parquet.`/lm-test/parquet/events`;
SELECT * FROM  delta.`/lm-test/parquet/events`;
```

READY

settings

data	date	eventId	eventType
case1	2020-10-01	1	Error
case2	2020-10-01	2	Warn
case3	2020-10-01	3	Warn
case4	2020-10-01	4	Error
case5	2020-10-02	5	Warn
case6	2020-10-02	6	Error
case7	2020-10-02	7	Warn
case8	2020-10-02	8	Error

settings

data	date	eventId	eventType
case1	2020-10-01	1	Error
case2	2020-10-01	2	Warn
case3	2020-10-01	3	Warn
case4	2020-10-01	4	Error
case5	2020-10-02	5	Warn
case6	2020-10-02	6	Error

4. **将delta文件还原为parquet文件**

   i. 清空delta数据

```
%sql
-- 可以将delta表还原为parquet表，需要先vacuum操作清空delta数据文件，然后删除_delta_log目录
SELECT * FROM  delta.`/dome-test/parquet/events`;
set spark.databricks.delta.retentionDurationCheck.enabled = false;
vacuum delta.`/dome-test/parquet/events` retain 0 hours
```

   ii. 前往OSS路径中找到_delta_log目录并删除

   iii. 查询转换之后的delta表数据

```
%sql
-- 此操作前要删除表对应的_delta_log目录
SELECT * FROM  delta.`/dome-test/parquet/events`;
```

5. 查询表

```
%sql
select * from events;
```

```
%sql
select * from events;
```

READY

settings

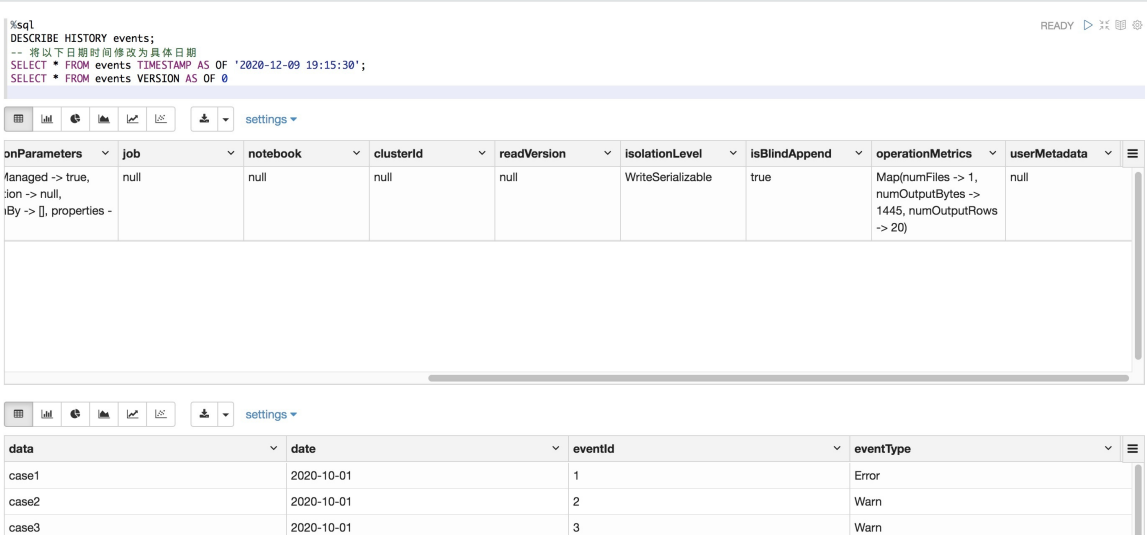
data	date	eventId	eventType
case1	2020-10-01	1	Error
case2	2020-10-01	2	Warn
case3	2020-10-01	3	Warn
case4	2020-10-01	4	Error
case5	2020-10-02	5	Warn
case6	2020-10-02	6	Error
case7	2020-10-02	7	Warn
case8	2020-10-02	8	Error

6. **通过元存储创建表并显示表中内容**

```
%pyspark
# 通过路径在元存储中创建表
spark.sql("CREATE TABLE IF NOT EXISTS events_partition USING DELTA LOCATION '/dome-test
/delta/events_partition'")
# 从元存储和路径中读取表
df1 = spark.table("events")
df2 = spark.read.format("delta").load("/dome-test/delta/events_partition")
df1.show()
df2.show()
```

7. 查询表的旧快照

```
%sql
DESCRIBE HISTORY events;
-- 您需要将以下日期时间修改为具体日期
SELECT * FROM events TIMESTAMP AS OF '2020-12-09 19:15:30';
SELECT * FROM events VERSION AS OF 0
```

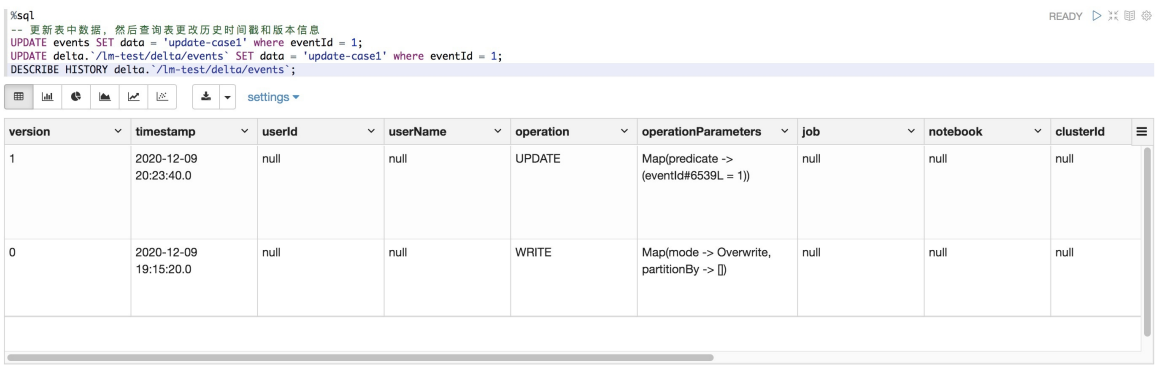


onParameters	job	notebook	clusterId	readVersion	isolationLevel	isBlindAppend	operationMetrics	userMetadata
Managed -> true, tion -> null, By -> [], properties -	null	null	null	null	WriteSerializable	true	Map(numFiles -> 1, numOutputBytes -> 1445, numOutputRows -> 20)	null

data	date	eventId	eventType
case1	2020-10-01	1	Error
case2	2020-10-01	2	Warn
case3	2020-10-01	3	Warn

8. 更新表中数据，然后查询表更改历史时间戳和版本信息

```
%sql
-- 更新表中数据，然后查询表更改历史时间戳和版本信息
UPDATE events SET data = 'update-case1' where eventId = 1;
UPDATE delta.`/dome-test/delta/events` SET data = 'update-case1' where eventId = 1;
DESCRIBE HISTORY delta.`/dome-test/delta/events`;
```



version	timestamp	userId	userName	operation	operationParameters	job	notebook	clusterId
1	2020-12-09 20:23:40.0	null	null	UPDATE	Map(predicate -> (eventId#6539L = 1))	null	null	null
0	2020-12-09 19:15:20.0	null	null	WRITE	Map(mode -> Overwrite, partitionBy -> [])	null	null	null

9. **通过不同时间戳表达式读取历史版本**

```
%pyspark
# 可以读取历史版本数据，time travel，以下时间为对应表的历史时间
d1 = spark.read.format("delta").option("timestampAsOf", "2020-12-09 20:23:40").load("/dome-test/delta/events")
d11 = spark.read.format("delta").load("/dome-test/delta/events@20201209202340000")
d2 = spark.read.format("delta").option("versionAsOf", 0).load("/dome-test/delta/events")
d22 = spark.read.format("delta").load("/dome-test/delta/events@v0")
# 读取最新版本数据或者其它版本数据
latest_version = spark.sql("SELECT * FROM (DESCRIBE HISTORY delta.`/dome-test/delta/events`)").collect()
df = spark.read.format("delta").option("versionAsOf", latest_version[0][0]).load("/dome-test/delta/events")
d1.show()
d11.show()
d2.show()
d22.show()
df.show()
```

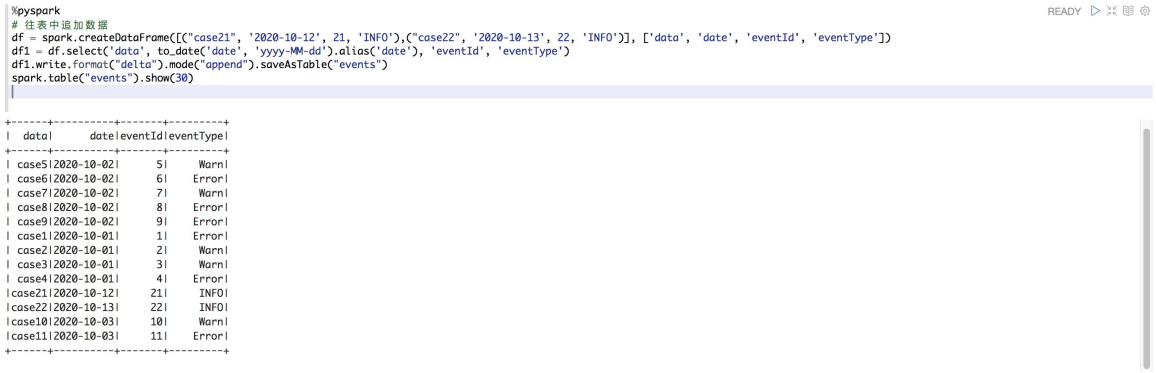
```
%pyspark
# 可以读取历史版本数据，time travel，以下时间为对应表的历史时间
d1 = spark.read.format("delta").option("timestampAsOf", "2020-12-09 20:23:40").load("/lm-test/delta/events")
d11 = spark.read.format("delta").load("/lm-test/delta/events@20201209202340000")
d2 = spark.read.format("delta").option("versionAsOf", 0).load("/lm-test/delta/events")
d22 = spark.read.format("delta").load("/lm-test/delta/events@v0")
# 读取最新版本数据或者其它版本数据
latest_version = spark.sql("SELECT * FROM (DESCRIBE HISTORY delta.`/lm-test/delta/events`)").collect()
df = spark.read.format("delta").option("versionAsOf", latest_version[0][0]).load("/lm-test/delta/events")


d1.show()
d11.show()
d2.show()
d22.show()
df.show()
```

READY

```
+-----------+----------+-------+---------+
|       data|      date|eventId|eventType|
+-----------+----------+-------+---------+
|update-case1|2020-10-01|      1|    Error|
|      case2|2020-10-01|      2|     Warn|
|      case3|2020-10-01|      3|     Warn|
|      case4|2020-10-01|      4|    Error|
|      case5|2020-10-02|      5|     Warn|
|      case6|2020-10-02|      6|    Error|
|      case7|2020-10-02|      7|     Warn|
|      case8|2020-10-02|      8|    Error|
|      case9|2020-10-02|      9|    Error|
|     case10|2020-10-03|     10|     Warn|
|     case11|2020-10-03|     11|    Error|
|     case12|2020-10-04|     12|    Error|
|     case13|2020-10-04|     13|     Info|
|     case14|2020-10-04|     14|     Info|
```

10. **向表中追加数据**

```
%pyspark
# 往表中追加数据
df = spark.createDataFrame([("case21", '2020-10-12', 21, 'INFO'),("case22", '2020-10-13', 22, 'INFO')], ['data', 'date', 'eventId', 'eventType'])
df1 = df.select('data', to_date('date', 'yyyy-MM-dd').alias('date'), 'eventId', 'eventType')
df1.write.format("delta").mode("append").saveAsTable("events")
spark.table("events").show(30)
```



```
%pyspark
# 往表中追加数据
df = spark.createDataFrame([("case21", '2020-10-12', 21, 'INFO'),("case22", '2020-10-13', 22, 'INFO')], ['data', 'date', 'eventId', 'eventType'])
df1 = df.select('data', to_date('date', 'yyyy-MM-dd').alias('date'), 'eventId', 'eventType')
df1.write.format("delta").mode("append").saveAsTable("events")
spark.table("events").show(30)
```

```
+------+----------+-------+---------+
|  data|      date|eventId|eventType|
+------+----------+-------+---------+
| case5|2020-10-02|      5|     Warn|
| case6|2020-10-02|      6|    Error|
| case7|2020-10-02|      7|     Warn|
| case8|2020-10-02|      8|    Error|
| case9|2020-10-02|      9|    Error|
| case1|2020-10-01|      1|    Error|
| case2|2020-10-01|      2|     Warn|
| case3|2020-10-01|      3|     Warn|
| case4|2020-10-01|      4|    Error|
|case21|2020-10-12|     21|     INFO|
|case22|2020-10-13|     22|     INFO|
|case10|2020-10-03|     10|     Warn|
|case11|2020-10-03|     11|    Error|
+------+----------+-------+---------+
```

11. 替换部分数据

```
%pyspark
df = spark.createDataFrame([("case21", '2020-10-12', 21, 'INFO'),("case22", '2020-10-13', 22, 'INFO')], ['data', 'date', 'eventId', 'eventType'])
df1 = df.select('data', to_date('date', 'yyyy-MM-dd').alias('date'), 'eventId', 'eventType')
# 可以替换分区部分数据（replaceWhere中字段必须是分区字段）;此处替换大于等于10-4的数据，并将10-12、10-13号的数据填入
df1.write.format("delta").mode("overwrite").option("replaceWhere", "date >= '2020-10-4'").saveAsTable("events_partition")
spark.table("events_partition").show(50)
```

```
%pyspark
df = spark.createDataFrame([("case21", '2020-10-12', 21, 'INFO'),("case22", '2020-10-13', 22, 'INFO')], ['data', 'date', 'eventId', 'eventType'])
df1 = df.select('data', to_date('date', 'yyyy-MM-dd').alias('date'), 'eventId', 'eventType')
# 可以替换分区部分数据（replaceWhere中字段必须是分区字段）;此处替换大于等于10-4的数据，并将10-12、10-13号的数据填入
df1.write.format("delta").mode("overwrite").option("replaceWhere", "date >= '2020-10-4'").saveAsTable("events_partition")
spark.table("events_partition").show(50)
```

```
+------+----------+-------+---------+
|  data|      date|eventId|eventType|
+------+----------+-------+---------+
| case5|2020-10-02|      5|     Warn|
| case6|2020-10-02|      6|    Error|
| case7|2020-10-02|      7|     Warn|
| case8|2020-10-02|      8|    Error|
| case9|2020-10-02|      9|    Error|
| case1|2020-10-01|      1|    Error|
| case2|2020-10-01|      2|     Warn|
| case3|2020-10-01|      3|     Warn|
| case4|2020-10-01|      4|    Error|
|case21|2020-10-12|     21|     INFO|
|case22|2020-10-13|     22|     INFO|
|case10|2020-10-03|     10|     Warn|
|case11|2020-10-03|     11|    Error|
+------+----------+-------+---------+
```

Took 3 sec. Last updated by dwm at December 15 2020, 7:06:58 PM. (outdated)

12. 添加列和修改列

```
%sql
ALTER TABLE events ADD COLUMNS (name string COMMENT '名称' FIRST);
ALTER TABLE events CHANGE COLUMN name COMMENT '姓名';
```

13. 重写表

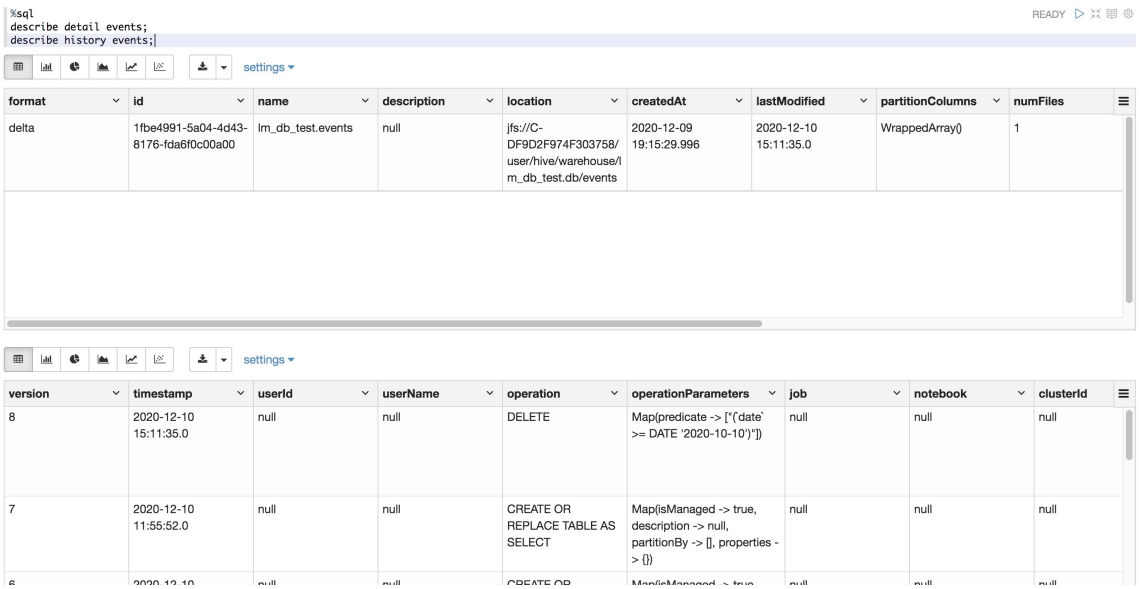
修改表中字段的类型或名称需要重写表，需要加上overwriteSchema选项

```
%pyspark
# 修改表中字段的类型或者名称需要重写表，需要加上overwriteSchema选项
spark.read.table("events").withColumnRenamed("name", "fullname") \
  .write.format("delta").mode("overwrite") \
  .option("overwriteSchema", "true") \
  .saveAsTable("events")
```

```
%pyspark
from pyspark.sql.functions import col
spark.read.table("events").withColumn("fullname2", col("fullname").cast("int")) \
  .write.format("delta").mode("overwrite") \
  .option("overwriteSchema", "true") \
  .saveAsTable("events")
```

14. **展示历史版本表**

```
%sql
describe detail events;
describe history events;
```



```
%sql
describe detail events;
describe history events;
```

READY

settings

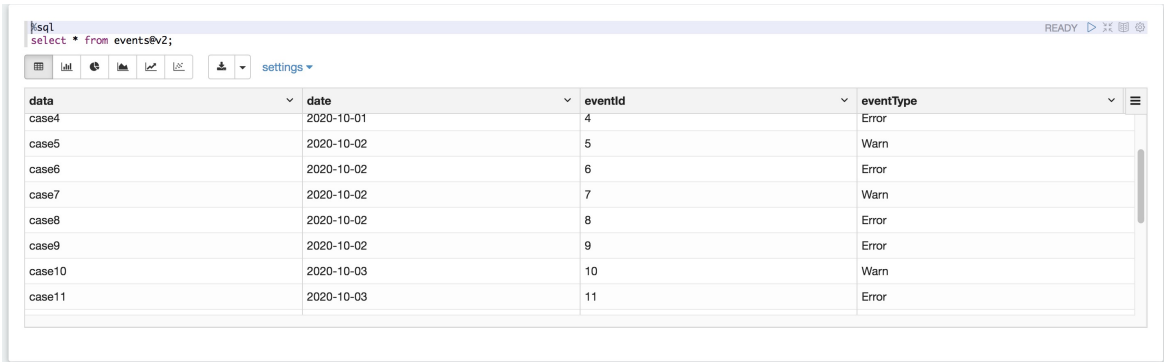
format	id	name	description	location	createdAt	lastModified	partitionColumns	numFiles
delta	1fbe4991-5a04-4d43-8176-fda6f0c00a00	lm_db_test.events	null	jfs://C-DF9D2F974F303758/user/hive/warehouse/lm_db_test.db/events	2020-12-09 19:15:29.996	2020-12-10 15:11:35.0	WrappedArray()	1

settings

version	timestamp	userId	userName	operation	operationParameters	job	notebook	clusterId
8	2020-12-10 15:11:35.0	null	null	DELETE	Map(predicate -> ["(`date` >= DATE '2020-10-10')"])	null	null	null
7	2020-12-10 11:55:52.0	null	null	CREATE OR REPLACE TABLE AS SELECT	Map(isManaged -> true, description -> null, partitionBy -> [], properties -> {})	null	null	null

15. **查询版本为2的表**

```
%sql
select * from events@v2;
```



```
%sql
select * from events@v2;
```

READY

settings

data	date	eventId	eventType
case4	2020-10-01	4	Error
case5	2020-10-02	5	Warn
case6	2020-10-02	6	Error
case7	2020-10-02	7	Warn
case8	2020-10-02	8	Error
case9	2020-10-02	9	Error
case10	2020-10-03	10	Warn
case11	2020-10-03	11	Error

16. **查询日期大于等于10月10号的数据**

```
%sql
delete from events where date >= '2020-10-10';
```

17. **从结构化的输入流中读取数据，经过处理后结构化流输出到delta文件**

```
%pyspark

# 从结构化的输入流中读取数据，经过处理后结构化流输出到delta文件
spark.readStream.format("delta").table("events").groupBy("date").count() \
.writeStream.format("delta").outputMode("complete").option("checkpointLocation", "/dome-test/delta/eventsByDate/_checkpoints/streaming-agg").start("/dome-test/delta/eventsByDate")
```

18. 查询上述文件输出结果

```
%pyspark
# 上述流文件输出完成之后，可以查询生成结果
events = spark.read.option("inferSchema", "true").format("delta").load("/dome-test/delta/eventsByDate")
events.show()
```

19. 更新表

```
%pyspark
from delta.tables import *
from pyspark.sql.functions import *

deltaTable = DeltaTable.forPath(spark, "/dome-test/delta/events/")

deltaTable.delete("date > '2020-10-03'")
deltaTable.toDF().show()
deltaTable.update("eventType = 'Error'", { "eventType": "'error'" })
deltaTable.toDF().show()
```

20. 合并表

```
%pyspark
from delta.tables import *


deltaTable = DeltaTable.forPath(spark, "/dome-test/delta/events/")
df = spark.createDataFrame([("update-case2", '2020-10-12', 2, 'INFO'),("case25", '2020-
10-13', 25, 'INFO')], ['data', 'date', 'eventId', 'eventType'])
updatesDF = df.select('data', to_date('date', 'yyyy-MM-dd').alias('date'), 'eventId', '
eventType')

# 合并表，eventId相等则更新数据，不相等就插入数据
deltaTable.alias("events").merge(
    updatesDF.alias("updates"),
    "events.eventId = updates.eventId") \
  .whenMatchedUpdate(set = { "data" : "updates.data" } ) \
  .whenNotMatchedInsert(values =
    {
      "date": "updates.date",
      "eventId": "updates.eventId",
      "data": "updates.data",
      "eventType": "updates.eventType"
    }
  ).execute()
deltaTable.toDF().show()
```

```
%pyspark
from delta.tables import *


deltaTable = DeltaTable.forPath(spark, "/lm-test/delta/events/")
df = spark.createDataFrame([["update-case2", '2020-10-12', 2, 'INFO'],["case25", '2020-10-13', 25, 'INFO']], ['data', 'date', 'eventId', 'eventType'])
updatesDF = df.select('data', to_date('date', 'yyyy-MM-dd').alias('date'), 'eventId', 'eventType')

# 合并表，eventId相等则更新数据，不相等就插入数据
deltaTable.alias("events").merge(
    updatesDF.alias("updates"),
    "events.eventId = updates.eventId") \
  .whenMatchedUpdate(set = { "data" : "updates.data" } ) \
  .whenNotMatchedInsert(values =
    {
      "date": "updates.date",
      "eventId": "updates.eventId",
      "data": "updates.data",
      "eventType": "updates.eventType"
    }
  ).execute()
deltaTable.toDF().show()
```

READY

```
+-----------+----------+-------+---------+
|       data|      date|eventId|eventType|
+-----------+----------+-------+---------+
|update-case1|2020-10-01|      1|    error|
|update-case2|2020-10-01|      2|     Warn|
|       case3|2020-10-01|      3|     Warn|
|       case4|2020-10-01|      4|    error|
|       case5|2020-10-02|      5|     Warn|
|       case6|2020-10-02|      6|    error|
|       case7|2020-10-02|      7|     Warn|
|       case8|2020-10-02|      8|    error|
|       case9|2020-10-02|      9|    error|
|      case10|2020-10-03|     10|     Warn|
|      case11|2020-10-03|     11|    error|
|      case25|2020-10-13|     25|     null|
+-----------+----------+-------+---------+
```

3.Databricks企业版Spark & Delta引擎

3.1. 使用OPTIMIZE和Z-ORDER优化商品库存查询性能

前提条件

1. 通过主账号登录阿里云 Databricks控制台。
2. 已创建集群，具体请参见创建集群。
3. 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI支持免密访问OSS路径，结构为：oss://BucketName/Object
>
> - **BucketName**为您的存储空间名称；
> - **Object**为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val text = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

如何使用Databricks Delta更新和查询商品库存信息

当执行UPSERT操作时，一般情况下，您需要执行两个不同的任务 - UPdate操作和inSERT操作。为了优化性能，Databricks Delta能够执行“upsert”或MERGE操作，从而简化业务逻辑。

在这个notebook中，我们将会演示2个Use Case：

1. DML: MERGE/DELETE/UPDATE。
2. 如何通过`OPTIMIZE`` and `ZORDER`对查询性能进行优化。

步骤一：创建集群并通过knox账号访问Notebook

创建集群参考: https://help.aliyun.com/document_detail/167621.html，需注意要设置ram子账号及保存好knox密码，登录WebUI时候需要用到。

步骤二：创建Notebook、导入数据、进行数据分析

示例Note下载：CASE6-Delta商品库存案例.zpln。

示例数据下载一：online_retail_mergetable.csv。

示例数据下载二：online_retail.csv。

```
%pyspark

# 将csv文件转化为parquet格式
# 注意文件读取和保存的路径请按照您的oss路径进行配置
spark.read.option("header", "true") \
    .csv("oss://databricks-demo/online_retail.csv") \
    .select("StockCode","Description","Quantity","UnitPrice","Country") \
    .write.format("parquet").mode("overwrite") \
    .save("oss://databricks-demo/parquet_online_retail/inventory")

# 从parquet文件导入DataFrame并查看
df = spark.read.parquet("oss://databricks-demo/parquet_online_retail/inventory")
df.show()
```

Case 1： DML MERGE/DELETE/UPDATE

```
%spark.sql

-- 创建库 DB_Demo_Inventory_OSS
DROP DATABASE if EXISTS DB_Demo_Inventory_OSS CASCADE;

-- 建表
CREATE DATABASE IF NOT EXISTS DB_Demo_Inventory_OSS LOCATION 'oss://databricks-demo/parquet_online_retail/inventory_database';
USE DB_Demo_Inventory_OSS;

-- 从parquet数据建表current_inventory
CREATE TABLE IF NOT EXISTS current_inventory USING PARQUET LOCATION 'oss://databricks-demo/parquet_online_retail/inventory';

-- 查看是否建表成功，简单SELECT语句验证
SHOW TABLES;
SELECT * FROM current_inventory LIMIT 10;
```

```
%spark.sql

-- 创建库 DB_Demo_Inventory_OSS
DROP DATABASE if EXISTS DB_Demo_Inventory_OSS CASCADE;

-- 建表
CREATE DATABASE IF NOT EXISTS DB_Demo_Inventory_OSS LOCATION 'oss://databricks-demo/parquet_online_retail/inventory_database';
USE DB_Demo_Inventory_OSS;

-- 从parquet数据建表current_inventory
CREATE TABLE IF NOT EXISTS current_inventory USING PARQUET LOCATION 'oss://databricks-demo/parquet_online_retail/inventory';

-- 查看是否建表成功，简单SELECT语句验证
SHOW TABLES;
SELECT * FROM current_inventory LIMIT 10;
```

READY

settings

database	tableName	isTemporary
db_demo_inventory_oss	current_inventory	false

settings

StockCode	Description	Quantity	UnitPrice	Country
85123A	WHITE HANGING HEART T-LIGHT HOLDER	6	2.55	United Kingdom
71053	WHITE METAL LANTERN	6	3.39	United Kingdom
84406B	CREAM CUPID HEARTS COAT HANGER	8	2.75	United Kingdom
84029G	KNITTED UNION FLAG HOT WATER BOTTLE	6	3.39	United Kingdom
84029E	RED WOOLLY HOTTIE WHITE HEART.	6	3.39	United Kingdom
22752	SET 7 BABUSHKA NESTING BOXES	2	7.65	United Kingdom
21730	GLASS STAR FROSTED T-LIGHT	6	4.25	United Kingdom

INSERT 或 UPDATE parquet文件的七个步骤

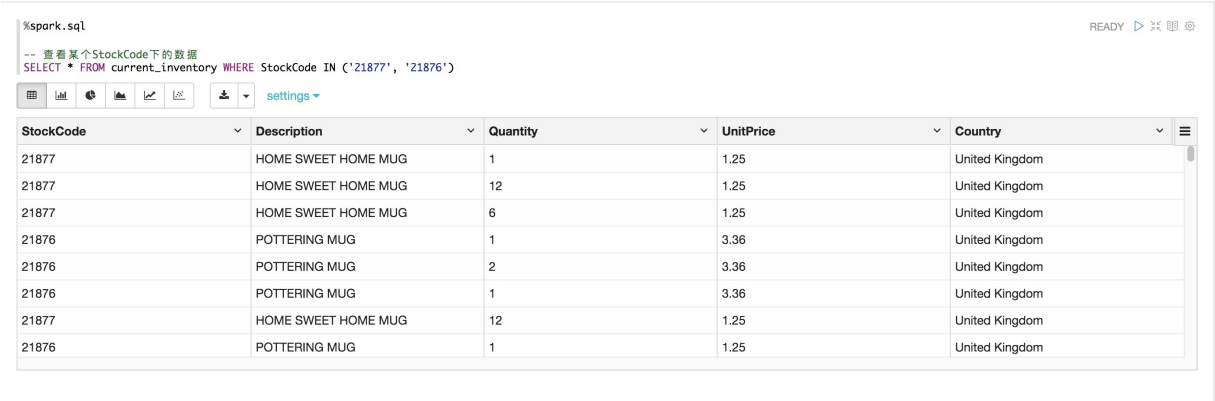
在使用Databricks Delta之前，我们先来看下，在通常情况下如何插入或更新表记录。

1. 确定需要插入的新记录。
2. 确定将会被替换的记录（例如，更新updated）。
3. 确定不会被INSERT或UPDATE操作影响的记录。
4. 基于前面3步的结果，创建一张临时表。
5. 将原表删除（包括所有相关的数据文件）。
6. 对临时表重命名。
7. 删除临时表。

Date	Value
A	1
B	2

查看当前Parquet表的数据

```
%spark.sql

-- 查看某个StockCode下的数据
SELECT * FROM current_inventory WHERE StockCode IN ('21877', '21876')
```



%spark.sql

READY

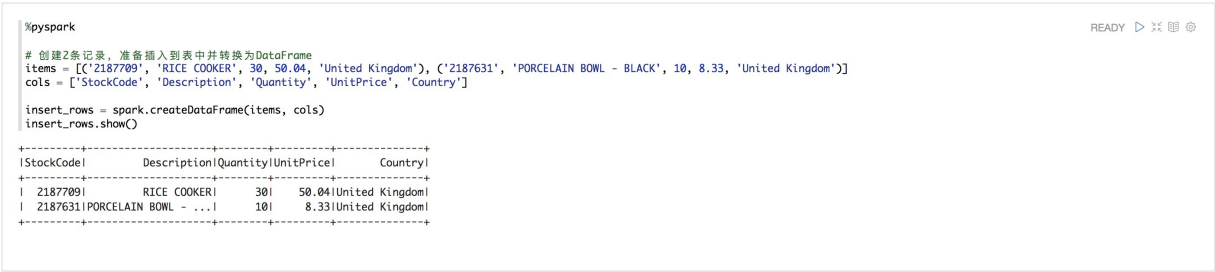
```
-- 查看某个StockCode下的数据
SELECT * FROM current_inventory WHERE StockCode IN ('21877', '21876')
```

settings

StockCode	Description	Quantity	UnitPrice	Country
21877	HOME SWEET HOME MUG	1	1.25	United Kingdom
21877	HOME SWEET HOME MUG	12	1.25	United Kingdom
21877	HOME SWEET HOME MUG	6	1.25	United Kingdom
21876	POTTERING MUG	1	3.36	United Kingdom
21876	POTTERING MUG	2	3.36	United Kingdom
21876	POTTERING MUG	1	3.36	United Kingdom
21877	HOME SWEET HOME MUG	12	1.25	United Kingdom
21876	POTTERING MUG	1	1.25	United Kingdom

步骤1：向Parquet表中插入记录

```
%pyspark

# 创建2条记录，准备插入到表中并转换为DataFrame
items = [('2187709', 'RICE COOKER', 30, 50.04, 'United Kingdom'), ('2187631', 'PORCELAIN BO
WL - BLACK', 10, 8.33, 'United Kingdom')]
cols = ['StockCode', 'Description', 'Quantity', 'UnitPrice', 'Country']

insert_rows = spark.createDataFrame(items, cols)
insert_rows.show()
```



%pyspark

READY

```
# 创建2条记录，准备插入到表中并转换为DataFrame
items = [('2187709', 'RICE COOKER', 30, 50.04, 'United Kingdom'), ('2187631', 'PORCELAIN BOWL - BLACK', 10, 8.33, 'United Kingdom')]
cols = ['StockCode', 'Description', 'Quantity', 'UnitPrice', 'Country']

insert_rows = spark.createDataFrame(items, cols)
insert_rows.show()
```

```
+---------+--------------------+--------+---------+--------------+
|StockCode|         Description|Quantity|UnitPrice|       Country|
+---------+--------------------+--------+---------+--------------+
|  2187709|         RICE COOKER|      30|    50.04|United Kingdom|
|  2187631|PORCELAIN BOWL - ...|      10|     8.33|United Kingdom|
+---------+--------------------+--------+---------+--------------+
```

步骤2：更新Parquet表中的记录

```
%pyspark

# 创建2条记录，准备更新到表中并转换为DataFrame
items = [('21877', 'HOME SWEET HOME MUG', 300, 26.04, 'United Kingdom'), ('21876', 'POTTERI
NG MUG', 1000, 48.33, 'United Kingdom')]
cols = ['StockCode', 'Description', 'Quantity', 'UnitPrice', 'Country']

update_rows = spark.createDataFrame(items, cols)
update_rows.show()
```

```
%pyspark

# 创建2条记录，准备更新到表中并转换为DataFrame
items = [('21877', 'HOME SWEET HOME MUG', 300, 26.04, 'United Kingdom'), ('21876', 'POTTERING MUG', 1000, 48.33, 'United Kingdom')]
cols = ['StockCode', 'Description', 'Quantity', 'UnitPrice', 'Country']

update_rows = spark.createDataFrame(items, cols)
update_rows.show()
```

```
+---------+-------------------+--------+---------+--------------+
|StockCode|        Description|Quantity|UnitPrice|       Country|
+---------+-------------------+--------+---------+--------------+
|    21877|HOME SWEET HOME MUG|     300|    26.04|United Kingdom|
|    21876|      POTTERING MUG|    1000|    48.33|United Kingdom|
+---------+-------------------+--------+---------+--------------+
```

步骤3：找到Parquet表中没有被变更过的记录

```
%pyspark

# 展示不会被以上数据更新的数据
unchanged_rows = spark.sql("select * from current_inventory where StockCode !='21877' and StockCode != '21876'")
unchanged_rows.show()
```

```
%pyspark

# 展示不会被以上数据更新的数据
unchanged_rows = spark.sql("select * from current_inventory where StockCode !='21877' and StockCode != '21876'")
unchanged_rows.show()
```

```
+---------+--------------------+--------+---------+--------------+
|StockCode|         Description|Quantity|UnitPrice|       Country|
+---------+--------------------+--------+---------+--------------+
|   85123A|WHITE HANGING HEA...|       6|     2.55|United Kingdom|
|    71053| WHITE METAL LANTERN|       6|     3.39|United Kingdom|
|   84406B|CREAM CUPID HEART...|       8|     2.75|United Kingdom|
|   84029G|KNITTED UNION FLA...|       6|     3.39|United Kingdom|
|   84029E|RED WOOLLY HOTTIE...|       6|     3.39|United Kingdom|
|    22752|SET 7 BABUSHKA NE...|       2|     7.65|United Kingdom|
|    21730|GLASS STAR FROSTE...|       6|     4.25|United Kingdom|
|    22633|HAND WARMER UNION...|       6|     1.85|United Kingdom|
|    22632|HAND WARMER RED P...|       6|     1.85|United Kingdom|
|    84879|ASSORTED COLOUR B...|      32|     1.69|United Kingdom|
|    22745|POPPY'S PLAYHOUSE...|       6|      2.1|United Kingdom|
|    22748|POPPY'S PLAYHOUSE...|       6|      2.1|United Kingdom|
|    22749|FELTCRAFT PRINCES...|       8|     3.75|United Kingdom|
|    22310|IVORY KNITTED MUG...|       6|     1.65|United Kingdom|
|    84969|BOX OF 6 ASSORTED...|       6|     4.25|United Kingdom|
+---------+--------------------+--------+---------+--------------+
```

步骤4：创建临时表

```
%pyspark

temp_current_inventory = insert_rows.union(update_rows).union(spark.sql("select * from current_inventory").filter("StockCode !='21877' and StockCode != '21876'"))
temp_current_inventory.show()
```

```
%pyspark

temp_current_inventory = insert_rows.union(update_rows).union(spark.sql("select * from current_inventory").filter("StockCode !='21877' and StockCode != '21876'"))
temp_current_inventory.show()
```

```
+---------+--------------------+--------+---------+--------------+
|StockCode|         Description|Quantity|UnitPrice|       Country|
+---------+--------------------+--------+---------+--------------+
|  2187709|         RICE COOKER|      30|    50.04|United Kingdom|
|  2187631|PORCELAIN BOWL - ...|      10|     8.33|United Kingdom|
|    21877| HOME SWEET HOME MUG|     300|    26.04|United Kingdom|
|    21876|       POTTERING MUG|    1000|    48.33|United Kingdom|
|   85123A|WHITE HANGING HEA...|       6|     2.55|United Kingdom|
|    71053| WHITE METAL LANTERN|       6|     3.39|United Kingdom|
|   84406B|CREAM CUPID HEART...|       8|     2.75|United Kingdom|
|   84029G|KNITTED UNION FLA...|       6|     3.39|United Kingdom|
|   84029E|RED WOOLLY HOTTIE...|       6|     3.39|United Kingdom|
|    22752|SET 7 BABUSHKA NE...|       2|     7.65|United Kingdom|
|    21730|GLASS STAR FROSTE...|       6|     4.25|United Kingdom|
|    22633|HAND WARMER UNION...|       6|     1.85|United Kingdom|
|    22632|HAND WARMER RED P...|       6|     1.85|United Kingdom|
|    84879|ASSORTED COLOUR B...|      32|     1.69|United Kingdom|
|    22745|POPPY'S PLAYHOUSE...|       6|      2.1|United Kingdom|
+---------+--------------------+--------+---------+--------------+
```

步骤5：删除原始表

```
%spark.sql

USE DB_Demo_Inventory_OSS;
DROP TABLE current_inventory;
```

步骤6：使用临时表的数据，生产新原始表current_inventory

```
%pyspark

temp_current_inventory.write.format("parquet").mode("overwrite").saveAsTable("current_inventory")
spark.sql("select * from current_inventory").show()
```

步骤7：删除临时表

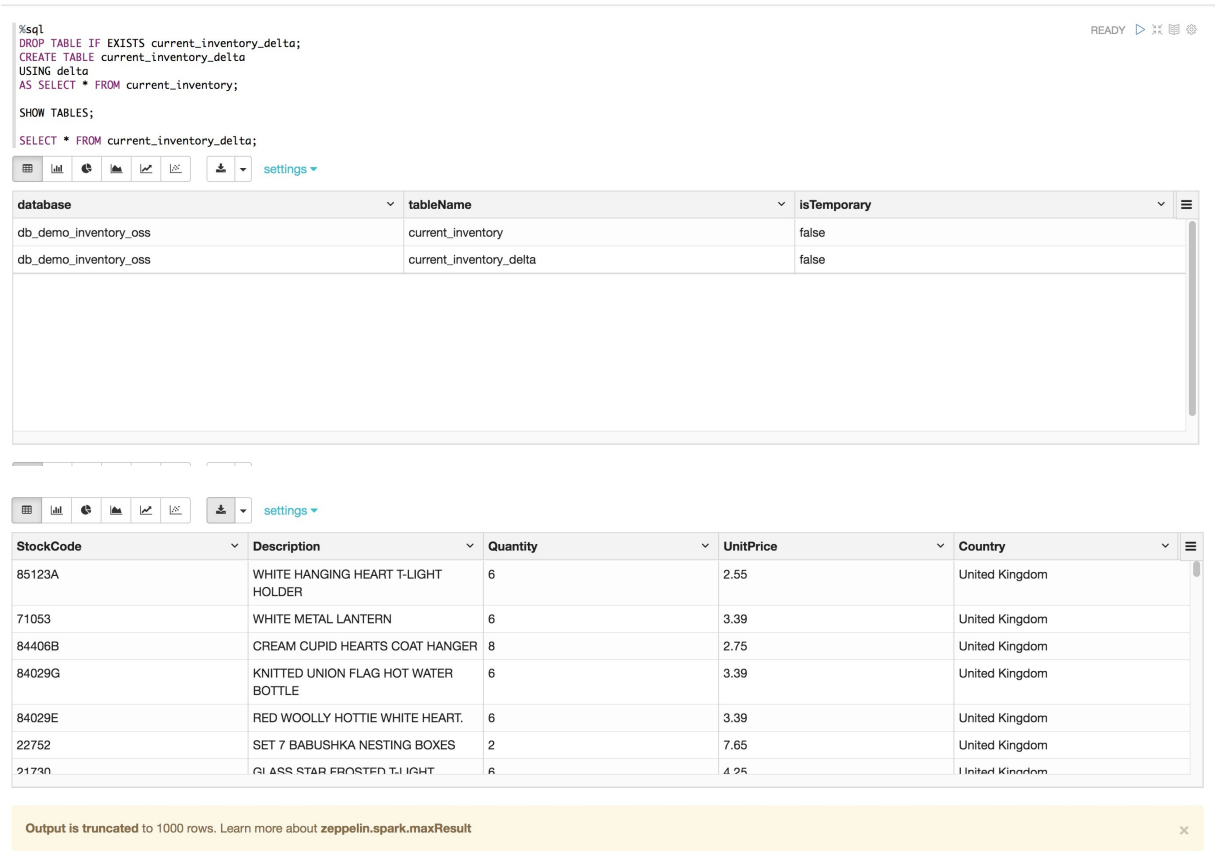
```
%sql
drop table if exists temp_current_inventory;
```

使用Databricks Delta更新表数据仅需2步：

1. 确定需要插入或更新的记录。
2. 使用MERGE。

创建一个delta表

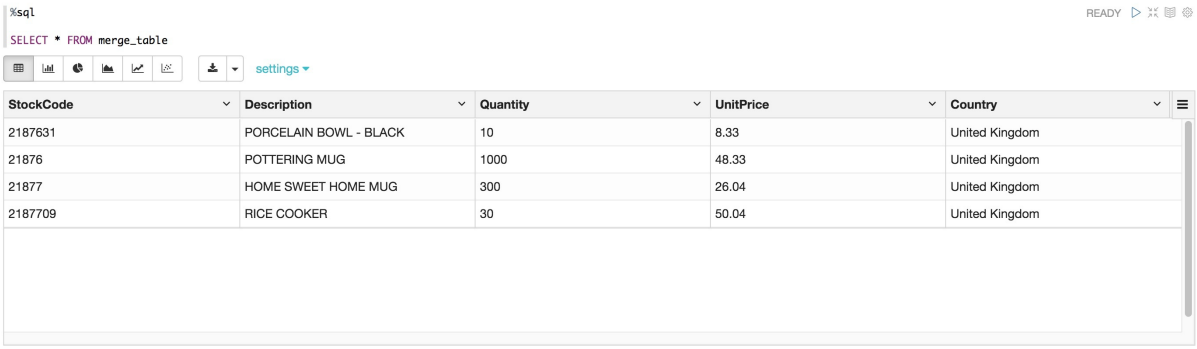
```
%sql
DROP TABLE IF EXISTS current_inventory_delta;
CREATE TABLE current_inventory_delta
USING delta
AS SELECT * FROM current_inventory;

SHOW TABLES;

SELECT * FROM current_inventory_delta;
```



步骤1：将目标数据更新至delta表

```
%pyspark

spark.read.option("header","True").csv("oss://databricks-demo/online_retail_mergetable.csv"
).createOrReplaceTempView("merge_table")
```

```
%sql

SELECT * FROM merge_table
```



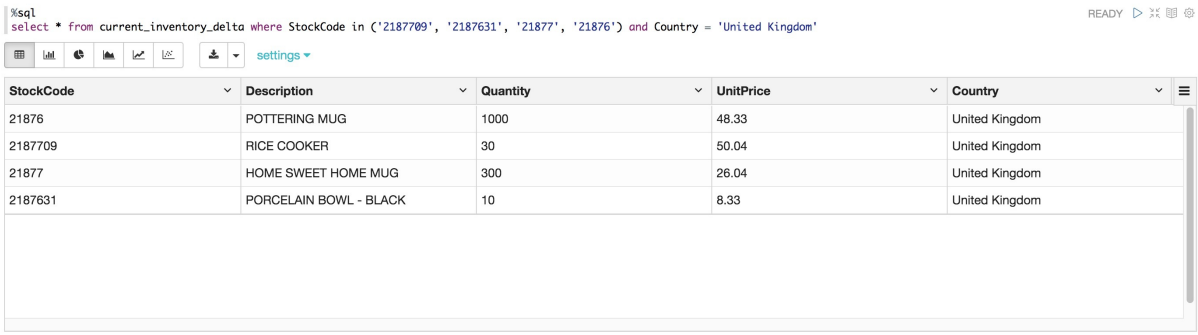
步骤2：使用MERGE插入或更新delta表

```
%sql
MERGE INTO current_inventory_delta as d
USING merge_table as m
on d.StockCode = m.StockCode and d.Country = m.Country
WHEN MATCHED THEN
  UPDATE SET *
WHEN NOT MATCHED
  THEN INSERT *
```

MERGE语句执行成功！

我们可以看到所有数据均被更新，新增数据也插入成功。

```
%sql
select * from current_inventory_delta where StockCode in ('2187709', '2187631', '21877', '21876') and Country = 'United Kingdom'
```



StockCode	Description	Quantity	UnitPrice	Country
21876	POTTERING MUG	1000	48.33	United Kingdom
2187709	RICE COOKER	30	50.04	United Kingdom
21877	HOME SWEET HOME MUG	300	26.04	United Kingdom
2187631	PORCELAIN BOWL - BLACK	10	8.33	United Kingdom

DELETE

同样，我们可以轻松删除Delta表中的记录。

创建redItem

```
%pyspark

redItem = Row({'StockCode':'33REDff', 'Description':'ADDITIONAL RED ITEM', 'Quantity': '8',
'UnitPrice': '3.53', 'Country':'United Kingdom'})
redItemDF = spark.createDataFrame(redItem)
redItemDF.printSchema()
```

分别创建PARQUET表和DELTA表

```
%pyspark

redItemDF.write.format("delta").mode("append").saveAsTable("current_inventory_delta")
redItemDF.write.format("parquet").mode("append").saveAsTable("current_inventory_pq")

spark.sql("""select * from current_inventory_delta where StockCode = '33REDff'""").show()
```

```
%pyspark

redItemDF.write.format("delta").mode("append").saveAsTable("current_inventory_delta")
redItemDF.write.format("parquet").mode("append").saveAsTable("current_inventory_pq")

spark.sql("""select * from current_inventory_delta where StockCode = '33REDff'""").show()
```

```
+---------+--------------------+--------+---------+--------------+
|StockCode|         Description|Quantity|UnitPrice|       Country|
+---------+--------------------+--------+---------+--------------+
|  33REDff|ADDITIONAL RED ITEM|       8|     3.53|United Kingdom|
+---------+--------------------+--------+---------+--------------+
```

删除PARQUET表中的记录将会报错 - 与DELTA的不同之处

```
%sql
delete from current_inventory_pq where StockCode = '33REDff';
```

```
%sql
delete from current_inventory_pq where StockCode = '33REDff';

Error happens in sql: delete from current_inventory_pq where StockCode = '33REDff'
null
set zeppelin.spark.sql.stacktrace = true to see full stacktrace
```

删除DELTA表中的记录 - 成功

```
%sql

delete from current_inventory_delta where StockCode = '33REDff';
select * from current_inventory_delta where StockCode = '33REDff';
```

```
%sql

delete from current_inventory_delta where StockCode = '33REDff';
select * from current_inventory_delta where StockCode = '33REDff';
```

settings

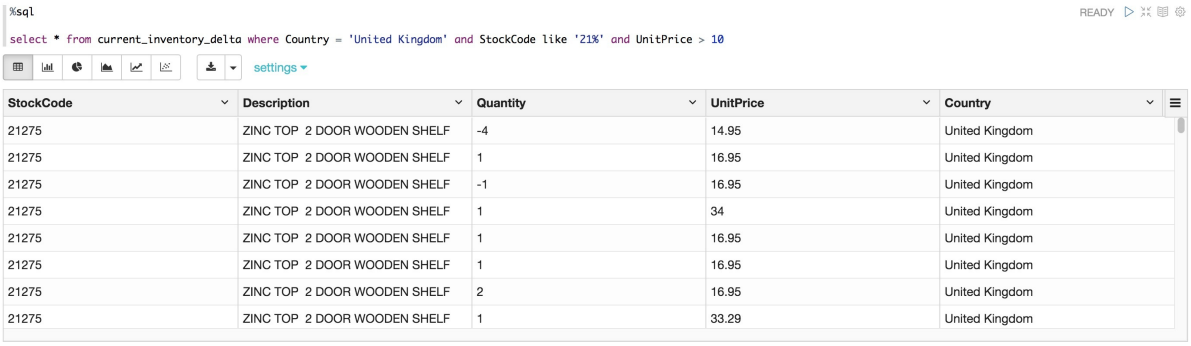
StockCode	Description	Quantity	UnitPrice	Country

Case 2：OPTIMIZE 和 ZORDER

通过OPTIMIZE和ZORDER `ZORDER` 对数据文件进行优化，以提高查询性能。

查询DELTA表

```
%sql

select * from current_inventory_delta where Country = 'United Kingdom' and StockCode like '
21%' and UnitPrice > 10
```

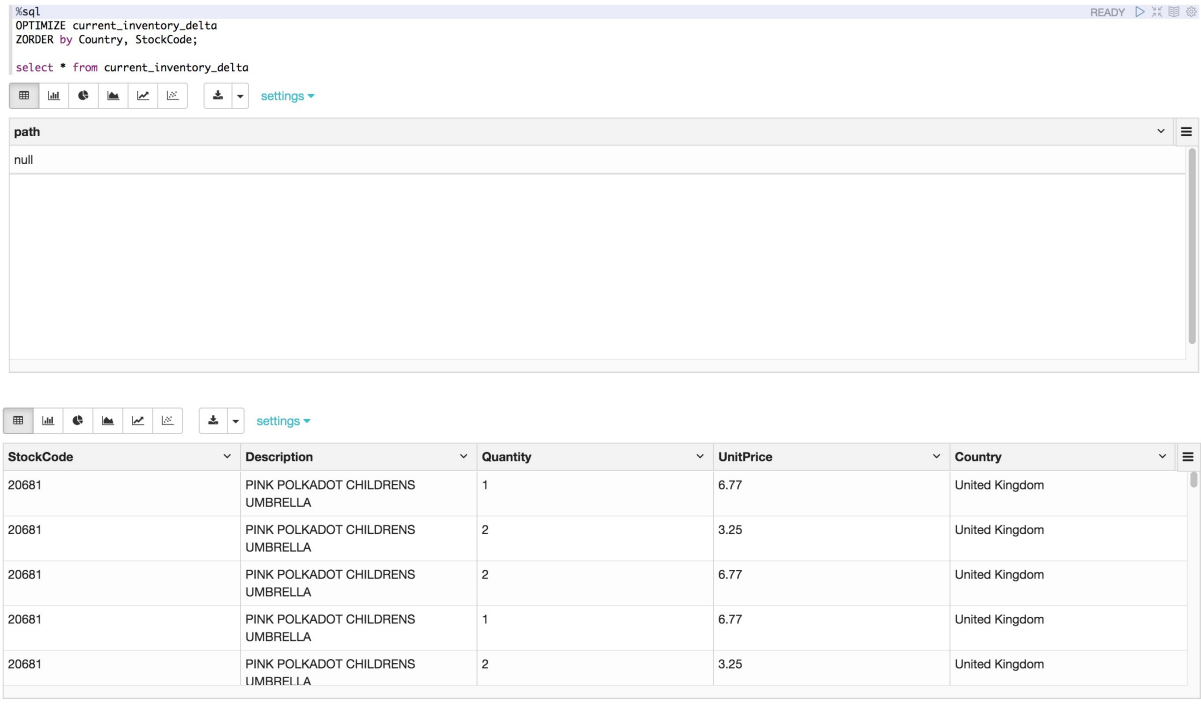


```
%sql
select * from current_inventory_delta where Country = 'United Kingdom' and StockCode like '21%' and UnitPrice > 10
```

settings

StockCode	Description	Quantity	UnitPrice	Country
21275	ZINC TOP 2 DOOR WOODEN SHELF	-4	14.95	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	1	16.95	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	-1	16.95	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	1	34	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	1	16.95	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	1	16.95	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	2	16.95	United Kingdom
21275	ZINC TOP 2 DOOR WOODEN SHELF	1	33.29	United Kingdom

```
%sql
OPTIMIZE current_inventory_delta
ZORDER by Country, StockCode;

select * from current_inventory_delta
```



```
%sql
OPTIMIZE current_inventory_delta
ZORDER by Country, StockCode;

select * from current_inventory_delta
```

settings

path
null

settings

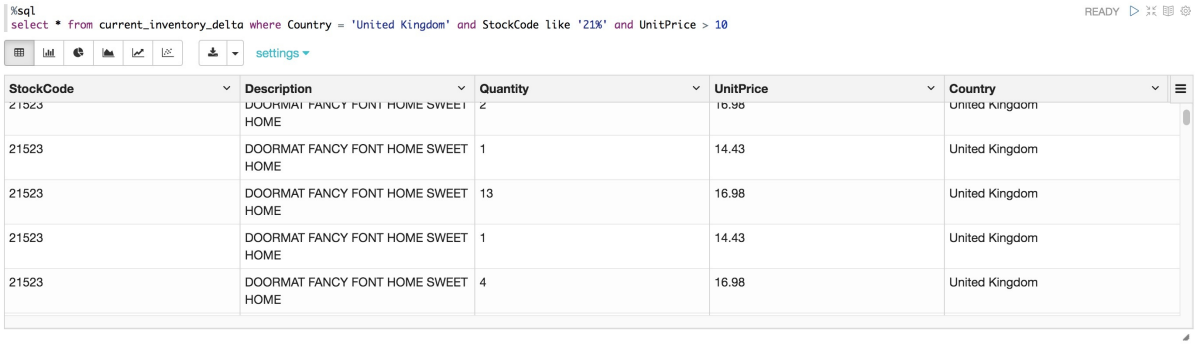
StockCode	Description	Quantity	UnitPrice	Country
20681	PINK POLKADOT CHILDRENS UMBRELLA	1	6.77	United Kingdom
20681	PINK POLKADOT CHILDRENS UMBRELLA	2	3.25	United Kingdom
20681	PINK POLKADOT CHILDRENS UMBRELLA	2	6.77	United Kingdom
20681	PINK POLKADOT CHILDRENS UMBRELLA	1	6.77	United Kingdom
20681	PINK POLKADOT CHILDRENS UMBRELLA	2	3.25	United Kingdom

Output is truncated to 1000 rows. Learn more about **zeppelin.spark.maxResult**

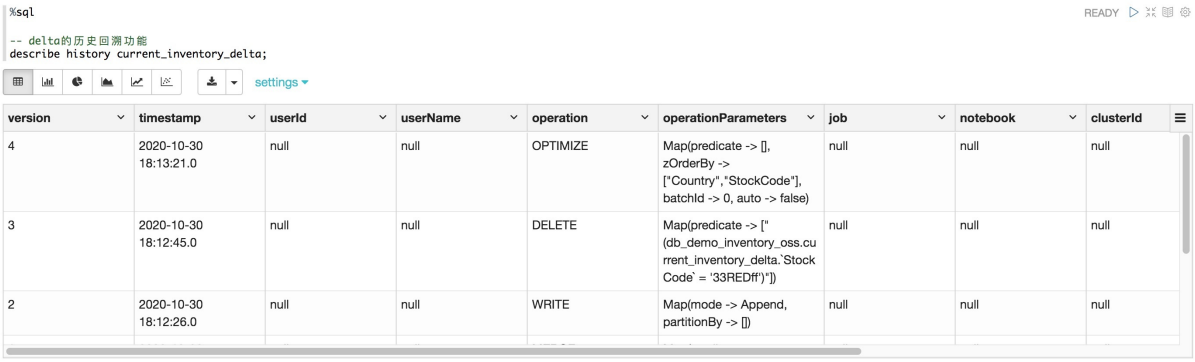
执行相同的查询 - 查询时间更短

实际执行时间与集群ECS规格的选择有关，与标准PARQUET表相比，通常会有 `5-10X` 性能提升，最快会有50X性能提升。

```
%sql
select * from current_inventory_delta where Country = 'United Kingdom' and StockCode like '
21%' and UnitPrice > 10
```




```
%sql

-- delta的历史回溯功能
describe history current_inventory_delta;
```



3.2. 使用Databricks Delta优化Spark作业性能

本文介绍如何使用Databricks Delta进行Spark作业的优化。

前提条件

已创建集群，详情请参见创建集群。

集群应满足以下配置：

区域	详情
地域（Region）	华北2（北京）
集群规模	1个Master节点，5个Worker节点

区域	详情
ECS实例配置	配置如下： • CPU：32核 • 内存：128GiB • ECS规格：ecs.g6.8xlarge • 数据盘配置：ESSD云盘300GB X 4块 • 系统盘配置：ESSD云盘120GB X 1块 **说明** ECS实例会因库存等原因和实际售卖页有出入。此处参数仅供参考，具体请您根据实际情况选择相应的实例规格进行测试。
OSS宽带	10Gbps

背景信息

Dat abricks数据洞察内置了Dat abricks商业版引擎，您可以利用Dat abricks数据洞察创建集群，实现在秒级响应时间内处理PB级别的数据。本文示例制造100亿条数据，利用Dat abricks Delt a的Dat a Skipping和ZOEDER Clust ering特性，对Spark作业进行改造，达到优化性能的目的。Dat abricks Delt a详情请参见Processing Pet abyt es of Dat a in Seconds wit h Dat abricks Delt a。

配置Spark

1. 使用阿里云账号登录Dat abricks数据洞察控制台。
2. 在Dat abricks数据洞察控制台页面，选择所在的地域（Region）。创建的集群将会在对应的地域内，一旦创建后不能修改。
3. 在左侧导航栏中，单击**集群**。
4. 单击待配置集群所在行的**详情**。
5. 在**集群详情**页面，单击上方的**Spark配置**。
6. 配置以下参数。

i. 修改以下配置。

参数	描述
spark.driver.cores	4
spark.driver.memory	8G
spark.executor.memory	23G

ii. 新增以下配置。

a. 在配置区域，单击spark-defaults页签。

b. 单击右侧的自定义配置。

参数	描述
spark.executor.cores	3
spark.executor.instances	22
spark.yarn.executor.memoryOverhead	default

示例

1. 准备数据。
   - 准备测试数据和query脚本。

     在集群中生成数据预计需要5小时，生成测试数据详情请参见Processing Petabytes of Data in Seconds with Databricks Delta。
   - 准备五张表：
     - conn_random：delta格式表
     - conn_random_parquet：parquet格式表
     - conn_optimize：经过OPTIMIZE的表，主要是Compaction
     - conn_zorder_only_ip：ZORDER BY (src_ip, dst_ip)
     - conn_zorder：ZORDER BY (src_ip, src_port, dst_ip, dst_port)
2. 使用OPTIMIZE命令进行优化。

   详细代码如下：

```
import spark.implicits._

val seed = 0
```

```
val numRecords = 10*1000*1000*1000L
val numFiles = 1000*1000

val baseLocation = "oss://mytest/records-10m(1000)3-(1000)2/data/random/"

val dbName = s"mdc_random_$numFiles"
val connRandom = "conn_random"
val connRandomParquet = "conn_random_parquet"
// val connSorted = "conn_sorted"
val connOptimize = "conn_optimize"
val connZorderOnlyIp = "conn_zorder_only_ip"
val connZorder = "conn_zorder"

spark.conf.set("spark.sql.shuffle.partitions", numFiles)
spark.conf.get("spark.sql.shuffle.partitions")

sql(s"drop database if exists $dbName cascade")
sql(s"create database if not exists $dbName")
sql(s"use $dbName")
sql(s"show tables").show(false)

import scala.util.Random

case class ConnRecord(src_ip: String, src_port: Int, dst_ip: String, dst_port: Int)

// 生成数据
def randomIPv4(r: Random) = Seq.fill(4)(r.nextInt(256)).mkString(".")
def randomPort(r: Random) = r.nextInt(65536)

def randomConnRecord(r: Random) = ConnRecord(
  src_ip = randomIPv4(r), src_port = randomPort(r),
  dst_ip = randomIPv4(r), dst_port = randomPort(r))

val df = spark.range(0, numFiles, 1, numFiles).mapPartitions { it =>
  val partitionID = it.toStream.head
  val r = new Random(seed = partitionID)
  Iterator.fill((numRecords / numFiles).toInt)(randomConnRecord(r))
}

// 生成数据表
df.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connRandom)
.saveAsTable(connRandom)

df.write
.mode("overwrite")
.format("parquet")
.option("path", baseLocation + connRandomParquet)
.saveAsTable(connRandomParquet)

spark.read.table(connRandom)
.write
```

```
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connOptimize)
.saveAsTable(connOptimize)

spark.read.table(connRandom)
.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connZorderOnlyIp)
.saveAsTable(connZorderOnlyIp)

spark.read.table(connRandom)
.write
.mode("overwrite")
.format("delta")
.option("path", baseLocation + connZorder)
.saveAsTable(connZorder)

spark.conf.set("spark.databricks.io.skipping.mdc.addNoise", "false")

// OPTIMIZE优化命令
sql(s"OPTIMIZE '${baseLocation + connOptimize}'")
sql(s"OPTIMIZE '${baseLocation + connZorderOnlyIp}' ZORDER BY (src_ip, dst_ip)")
sql(s"OPTIMIZE '${baseLocation + connZorder}' ZORDER BY (src_ip, src_port, dst_ip, dst_
port)")
```

3. 验证Spark SQL。

```
select count(*) from conn_random where src_ip like '157%' and dst_ip like '216.%';

select count(*) from conn_random_parquet where src_ip like '157%' and dst_ip like '216.
%';

select count(*) from conn_optimize where src_ip like '157%' and dst_ip like '216.%';

select count(*) from conn_zorder_only_ip where src_ip like '157%' and dst_ip like '216.
%';

select count(*) from conn_zorder where src_ip like '157%' and dst_ip like '216.%';
```

测试结论

本示例各表情况如下。

表名称	时间（s）
conn_random_parquet	2504
conn_random	2324

表名称	时间（s）
conn_optimize	112
conn_zorder	65
conn_zorder_only_ip	46

> 说明
>
> 通过以上示例，可以发现：
>
> - 经过OPTIMIZE的表，文件大小会在1G左右，而且进行了delta元数据的优化，提高了data-skipping的效率，在性能上提升约20倍（2504/112=22X）。
> - Zorder使得data-skipping的优化效果进一步深化，性能提升约40倍（2504/65=38X）。
> - 当Zorder列是查询列时，优化效果会更加明显，实验显示性能提升约50倍（2504/46=54X）。

问题反馈

您在使用阿里云Databricks数据洞察过程中有任何疑问，欢迎用钉钉扫描下面的二维码加入钉钉群进行反馈。

3.3. 基于TPC-DS测试Databricks引擎性能

本文介绍如何使用TPC-DS进行测试DDI引擎性能，给出推荐的实践步骤。

前提条件

1. 通过主账号登录阿里云 Databricks控制台，当前 TPC-DS 测试流程已在 Databricks Runtime 9.1 以上版本做验证，请选择DBR9及以上版本产品。
2. 为保证测试效果，推荐最小测试数据量应为1T，故需创建特定规模的集群，创建步骤请参见创建集群，推荐集群规模如下：

ECS实例类型	vCore	vMem	系统盘/数据盘	实例个数
Master	16 cores	64GB	200GB/500GB	1
Worker	32 cores	128GB	200GB/500GB	6

元数据选择类型建议使用**数据湖元数据**，该方式采用阿里云数据湖构建（Data Lake Formation）作为统一元数据存储，采用服务化高可用的元数据，实现了多引擎统一元数据存储。

3. 已使用OSS管理控制台创建**非系统目录存储空间**，详情请参见创建存储空间。

> **警告**
>
> 首次使用DDI产品创建的Bucket为系统目录Bucket，**不建议存放数据**，您需要再创建一个Bucket来读写数据。

> **说明**
>
> DDI支持免密访问OSS路径，结构为：oss://BucketName/Object
>
> - **BucketName**为您的存储空间名称；
> - **Object**为上传到OSS上的文件的访问路径。
>
> 例：读取在存储空间名称为databricks-demo-hangzhou文件路径为demo/The_Sorrows_of_Young_Werther.txt的文件

```
// 从oss地址读取文本文档
val text = sc.textFile("oss://databricks-demo-hangzhou/demo/The_Sorrows_of_Young_Werther.txt")
```

TPC-DS项目简介

本项目基于社区的TPC-DS项目，该项目包含生成测试集数据、运行实例SQL、输出各阶段执行时间指标等功能，以此检验引擎的性能。本文的TPC-DS实现将生成数据、运行测试以及迭代循环等功能整合，统一使用一套参数，用户可以专注于测试数据与DDI引擎性能本身，屏蔽底层较繁琐的细节。

参数列表

可执行参数	参数定义	推荐值
--scaleFactor	Int型参数，测试集数据规模，单位为GB	推荐最小数据量为1T，参数为1000
--location	String类型参数，数据集与运算结果产生的OSS目录	避免系统目录bucket

可执行参数	参数定义	推荐值
--format	String类型参数，Spark数据的存储类型	数据湖元数据，参数为delta
--overwrite	Boolean型参数，是否覆盖生成数据集	推荐为true，也可根据步骤判断
--numPartitions	Int型参数，dsdgen生成数据的分区数，即输入任务数	根据数据量决定，1T数据推荐为100
--numberOfIteration	Int型参数，测试集运算执行次数	单独生成数据集时，参数为0 运行测试数据集时，参数为大于0
--generateData	Boolean型参数，是否生成数据集	单独生成数据集时，参数为true 运行测试数据集时，参数为false

其余参数较为固定，可参考以下推荐用例。

TPC-DS项目最佳实践

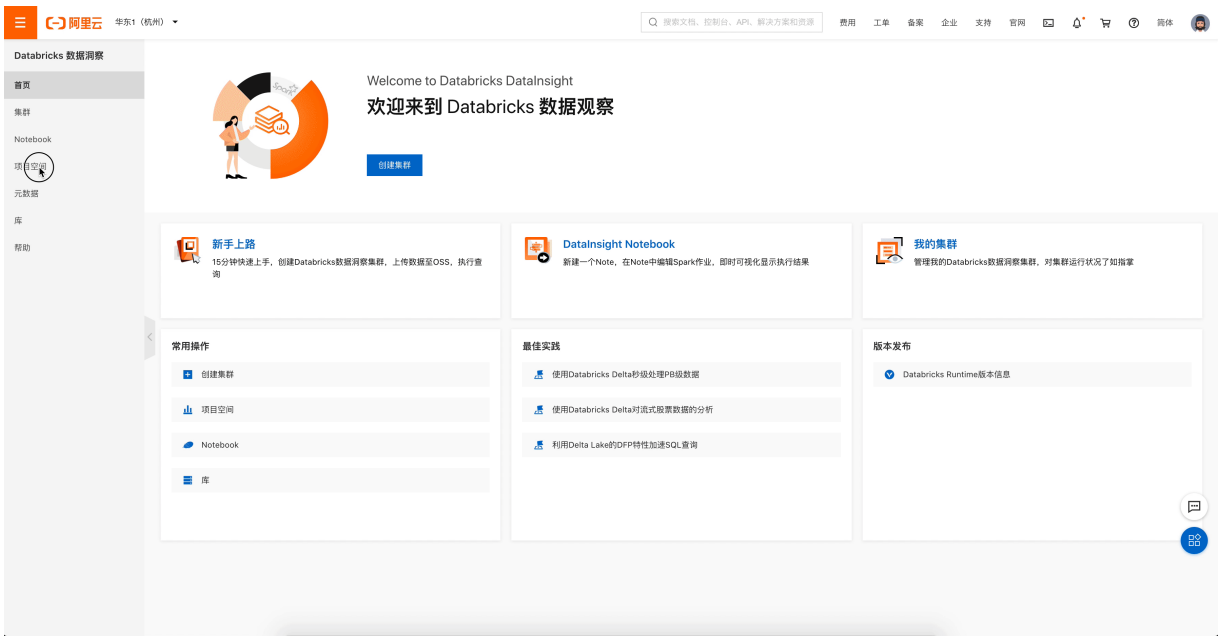
步骤一：TPC-DS工程下载

- 示例工程下载：spark-sql-perf-assembly-0.5.1-SNAPSHOT.jar
- 在OSS已创建的bucket中创建一个目录，存放上述下载的jar包。

步骤二：创建TPC-DS测试项目

> 说明
>
> 1. 本文的TPC-DS的实现基于TPC-DS的基准测试，并不能与已发布的TPC-DS基准测试结果相比较，本文中的测试并不符合TPC-DS的基准测试的所有要求。

在Dat abricks数据洞察的项目空间中创建一个新项目，操作如下：



步骤三： 生成测试数据集脚本

- spark资源可以根据测试数据量与集群规模配置的具体情况，进行逐一调整。
- scale_factor参数控制整个测试集的数据量（如scale_factor=1000 即1T的数据量）。
- **以下的oss路径要替换为用户自己设置的路径**，分别用于读取jar包和存放生成的测试数据集与性能测试结果。

执行脚本如下:

```
--class com.databricks.spark.sql.perf.tpcds.GenTPCDSData
--deploy-mode cluster
--name generate_dataset
--queue default
--master yarn
--conf spark.yarn.submit.waitAppCompletion=true
--conf spark.driver.cores=4
--conf spark.driver.memory=12G
--conf spark.executor.cores=7
--conf spark.executor.memory=20G
--conf spark.executor.instances=30
--conf spark.executor.memoryOverhead=2048
--conf spark.default.parallelism=600
--conf spark.shuffle.service.enabled=true
--conf spark.sql.autoBroadcastJoinThreshold=-1
oss://tpc-ds-test/tpcds-jar/spark-sql-perf-assembly-0.5.1-SNAPSHOT.jar
--scaleFactor 1000
--location oss://tpc-ds-test
--format delta
--overwrite true
--numPartitions 100
--numberOfIteration 0
--generateData true
```

生成的数据集会存储在location参数所指定路径的data目录下（如上面的例子，数据集在：oss://tpc-ds-test/data）

步骤四：运行TPC-DS测试集

运行测试集的方法和生成数据的操作一致，仅需要修改下面两个参数：

- numberOfIteration：更改为1，代表测试集将被运行1次，产生1次迭代结果，也可自行选择迭代执行次数。
- generateData：更改为false，代表只执行测试集运算，不进行重复测试集数据生成。

执行脚本如下：

```
--class com.databricks.spark.sql.perf.tpcds.GenTPCDSData
--deploy-mode cluster
--name run_dataset
--queue default
--master yarn
--conf spark.yarn.submit.waitAppCompletion=true
--conf spark.driver.cores=4
--conf spark.driver.memory=12G
--conf spark.executor.cores=7
--conf spark.executor.memory=20G
--conf spark.executor.instances=30
--conf spark.executor.memoryOverhead=2048
--conf spark.default.parallelism=600
--conf spark.shuffle.service.enabled=true
--conf spark.sql.autoBroadcastJoinThreshold=-1
oss://tpc-ds-test/tpcds-jar/spark-sql-perf-assembly-0.5.1-SNAPSHOT.jar
--scaleFactor 1000
--location oss://tpc-ds-test
--format delta
--overwrite true
--numPartitions 100
--numberOfIteration 1
--generateData false
```

运行完成后结果会存储在location参数所指定路径的result目录下（如上面的例子，结果在：oss://tpc-ds-test/result）

步骤五：结果分析

最终在oss://tpc-ds-test/result/timestamp=${TimeStamp}的目录下，产生一个json类型的结果文件，实例如下图(数值仅供参考)：

```
    "results": [{
                "name": "q1-v2.4",
                "mode": "collect",
                "parameters": {},
                "joinTypes": [],
                "tables": ["customer_total_return", "store", "customer"],
                "parsingTime": 0.014937,
                "analysisTime": 0.00281,
                "optimizationTime": 140.684418,
                "planningTime": 29.981971,
                "executionTime": 33900.334225,
                "breakDown": [],
                "queryExecution": "== Parsed Logical Plan ==\nCTE [customer_total_return]\n:  +- 'SubqueryAlias customer_tot
            }, {
                "name": "q2-v2.4",
                "mode": "collect",
                "parameters": {},
                "joinTypes": [],
                "tables": ["wswscs", "date_dim", "wswscs", "date_dim"],
                "parsingTime": 0.002332,
                "analysisTime": 0.00121,
                "optimizationTime": 102.955206,
                "planningTime": 30.438411,
                "executionTime": 16664.121023,
                "breakDown": [],
                "queryExecution": "== Parsed Logical Plan ==\nCTE [wscs, wswscs]\n:  :- 'SubqueryAlias wscs\n:  :  +- 'Proje
            }, {
                "name": "q3-v2.4",
                "mode": "collect",
                "parameters": {},
                "joinTypes": [],
                "tables": ["date_dim", "store_sales", "item"],
                "parsingTime": 0.001825,
                "analysisTime": 0.001356,
                "optimizationTime": 66.223683,
                "planningTime": 10.013753,
                "executionTime": 4742.557425,
                "breakDown": [],
```


可以看到针对每个SQL测试都会产生相应的指标结果项（毫秒单位），可以着重关注于executionTime，该指标区分度较大，主要体现DDI引擎性能上的优势。