Name: Shaktim Nepal

Class: BDE59

Assignment: Spark Project – 1 (Iphone Sales Analysis)

Spark Project -1

# Iphone Sales Analysis

## 

## Problem Statement

An iphone store wants to analyse the daily sales, Store has an application running which collects the data from user activities and pushes the sales data into partitioned hive table in parquet format. Store also has a metadata table in the hive which stores the product info.

As a data expert you are supposed to work with multiple teams specially 2 teams one which is collecting the data from user activity(software engineers) and another which wants to analyse the collected data(Business data analyst or data scientist). So basically you are supposed to expose(write) to utilities.

Data Collection

Data Layer

Data Analysis

## Data collector utilities for software engineers

1. Sales Data collector utility
   1. This utility will consume the sales data from a text file with header and will publish the output into a hive partitioned table in parquet format. You can assume **sale\_date** as a partitioned column.
   2. So write a python function which will take the spark session and local file path as inputs and will return the name of the partitioned hive table.
      1. Assumptions
         1. Text file has metadata inform of file header.
         2. Data is separated by ‘|’

1. Product Data collector utility
   1. *Note - First convert text data into a parquet data for product*
   2. This utility will consume the data from a parquet file and will publish into a non partitioned hive table.

## Data preparation utilities for BDA or Data Scientist

1. Write an utility which will consume the data from product and sales tables. And will publish the following output in another hive table.
2. Expected output-
   1. Reports the **buyers** who have bought *S8* but not *iPhone*. Note that *S8* and *iPhone* are products present in the Product table

# Metadata

## Table: Product

+--------------+---------+

| Column Name | Type |

+--------------+---------+

| product\_id | int |

| product\_name | varchar |

| unit\_price | int |

+--------------+---------+

1. Product\_id is the primary key of this table.
2. Each row of this table indicates the name and the price of each product.

## Table: Sales

+-------------+---------+

| Column Name | Type |

+-------------+---------+

| seller\_id | int |

| product\_id | int |

| buyer\_id | int |

| sale\_date | date |

| quantity | int |

| price | int |

+-------------+---------+

1. This table has no primary key, it can have repeated rows.
2. product\_id is a foreign key to the Product table.
3. buyer\_id is never NULL.
4. sale\_date is never NULL.
5. Each row of this table contains some information about one sale.

# Sample Data

## **Input:**

Product table:

+------------+--------------+------------+

| product\_id | product\_name | unit\_price |

+------------+--------------+------------+

| 1 | S8 | 1000 |

| 2 | G4 | 800 |

| 3 | iPhone | 1400 |

+------------+--------------+------------+

Sales table:

+-----------+------------+----------+------------+----------+-------+

| seller\_id | product\_id | buyer\_id | sale\_date | quantity | price |

+-----------+------------+----------+------------+----------+-------+

| 1 | 1 | 1 | 2019-01-21 | 2 | 2000 |

| 1 | 2 | 2 | 2019-02-17 | 1 | 800 |

| 2 | 1 | 3 | 2019-06-02 | 1 | 800 |

| 3 | 3 | 3 | 2019-05-13 | 2 | 2800 |

+-----------+------------+----------+------------+----------+-------+

## **Output: (Hive Table with single column)**

+-------------+

| buyer\_id |

+-------------+

| 1 |

+-------------+

**Explanation:** The buyer with id 1 bought an S8 but did not buy an iPhone. The buyer with id 3 bought both.

# Submissions

1. Submit your pySpark solution in a doc with the following utilities.
   1. **sales\_data\_collector\_api**(spark, text\_file\_path)

Solution:

from pyspark.sql import SparkSession  
  
# Initialize Spark session  
spark = SparkSession.builder.appName("Sales Data Collector").enableHiveSupport().getOrCreate()  
  
  
def sales\_data\_collector\_api(spark, text\_file\_path):  
 # Load the sales data from the given text file with '|' delimiter  
 sales\_df = spark.read.option("header", True).option("delimiter", "|").csv(text\_file\_path)  
 spark.conf.set("hive.exec.dynamic.partition", "true")  
 spark.conf.set("hive.exec.dynamic.partition.mode", "nonstrict")  
  
 # Write the data to a Hive table with partitioning on 'sale\_date'  
 sales\_df.write.mode("overwrite").partitionBy("sale\_date").format("parquet").saveAsTable("iphone\_sales\_analysis1.sales\_hive\_table")  
  
 return "iphone\_sales\_analysis1.sales\_hive\_table"  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 filepath = 'file:///home/takeo/data/iphone\_sales\_analysis\_project/sales\_data'  
 sales\_data\_collector\_api(spark, filepath)

* 1. **product\_data\_collector\_api**(spark, parquet\_file\_path)

Solution:

# Initialize Spark session with Hive support  
spark = SparkSession.builder \  
 .appName("Product Data Collector") \  
 .enableHiveSupport() \  
 .getOrCreate()  
  
def product\_data\_collector\_api(spark, text\_file\_path, parquet\_file\_path):  
 # Step 1: Load the CSV data from the text file with '|' delimiter  
 product\_df = spark.read.option("header", True).option("delimiter", "|").csv(text\_file\_path)  
  
 # Step 2: Write the DataFrame to Parquet format  
 product\_df.write.mode("overwrite").parquet(parquet\_file\_path)  
  
 # Step 3: Read the Parquet data back into a DataFrame  
 parquet\_df = spark.read.parquet(parquet\_file\_path)  
  
 # Step 4: Write the data to a Hive table from the Parquet DataFrame  
 parquet\_df.write.mode("overwrite").saveAsTable("iphone\_sales\_analysis1.product\_hive\_table")  
  
 return "iphone\_sales\_analysis1.product\_hive\_table"  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Define file paths  
 csv\_filepath = 'file:///home/takeo/data/iphone\_sales\_analysis\_project/product\_data/product\_data\_csv'  
 parquet\_filepath = 'file:///home/takeo/data/iphone\_sales\_analysis\_project/product\_data/product\_data\_parquet'  
  
 # Call the product data collector function  
 product\_data\_collector\_api(spark, csv\_filepath, parquet\_filepath)

* 1. **data\_preparation\_api**(spark, product\_hive\_table, sales\_hive\_table, target\_hive\_table)

Solution:

from pyspark.sql import SparkSession  
from pyspark.sql.functions import col  
  
def data\_preparation\_api(spark, product\_hive\_table, sales\_hive\_table, target\_hive\_table):  
 # Load product and sales data from Hive tables  
 product\_df = spark.table(product\_hive\_table).alias("products")  
 sales\_df = spark.table(sales\_hive\_table).alias("sales")  
  
 # Filter product IDs for 'S8' and 'iPhone'  
 s8\_id = product\_df.filter(col("product\_name") == "S8").select("product\_id").first()[0]  
 iphone\_id = product\_df.filter(col("product\_name") == "iPhone").select("product\_id").first()[0]  
  
 # Find buyers who bought S8 but not iPhone  
 buyers\_s8 = sales\_df.filter(col("product\_id") == s8\_id).select("buyer\_id").distinct()  
 buyers\_iphone = sales\_df.filter(col("product\_id") == iphone\_id).select("buyer\_id").distinct()  
  
 # Perform anti-join to get buyers who bought S8 but not iPhone  
 buyers\_only\_s8 = buyers\_s8.join(buyers\_iphone, "buyer\_id", "left\_anti")  
  
 # Save the result to the target Hive table  
 buyers\_only\_s8.write.mode("overwrite").saveAsTable(target\_hive\_table)  
  
 return target\_hive\_table  
  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 # Initialize Spark session with Hive support  
 spark = SparkSession.builder.appName("Data Preparation").enableHiveSupport().getOrCreate()  
 # Define the Hive table names  
 product\_table = "iphone\_sales\_analysis1.product\_hive\_table"  
 sales\_table = "iphone\_sales\_analysis1.sales\_hive\_table"  
 target\_table = "iphone\_sales\_analysis1.target\_hive\_table"  
  
 # Call the data preparation function  
 data\_preparation\_api(spark, product\_table, sales\_table, target\_table)