

Assignment 2 - DSECLZG522 – BDS

Group No: 07

1.) Business Understanding

An e-Commerce company in Europe is analysing the online retail sales data of 2010 to formulate the sales strategy for 2011. The name of the sales data file is "Assignment-2 2023 BDS DATA SET online_retail_data.csv". This file contains 525461 records. The schema of this data set is (Record No, Invoice No, StockCode, Description, Quantity, InvoiceDate, Price, Customer ID, Country). Some of the fields in some records may be blank. The analysis should be carried out using Hadoop eco system products after storing the data on HDFS. The results of the analysis along with the code and queries developed should be submitted.

1.1. Problem statement:

We have used using Hadoop Eco system product for the storage and querying the sales data.

1.2. The dataset, download dataset from Campus.

Assignment-2 2023 BDS DATA SET online_retail_data.csv

This data set contains all the transactions occurred between 01/12/2010 and 09/12/2011 for a UK-based online retail non-store. Most of the customers are whole sale customers

1.3. Dataset Info

Attribute Information:

InvoiceNo: Invoice number. Nominal, a 6-digit integral number uniquely assigned to each transaction. If this code starts with letter 'c', it indicates a cancellation.

StockCode: Product (item) code. Nominal, a 5-digit integral number uniquely assigned to each distinct product.

Description: Product (item) name. Nominal.

Quantity: The quantities of each product (item) per transaction. Numeric.

InvoiceDate: Invoice Date and time. Numeric, the day and time when each transaction was generated.

Price: Unit price. Numeric, Product price per unit in sterling.

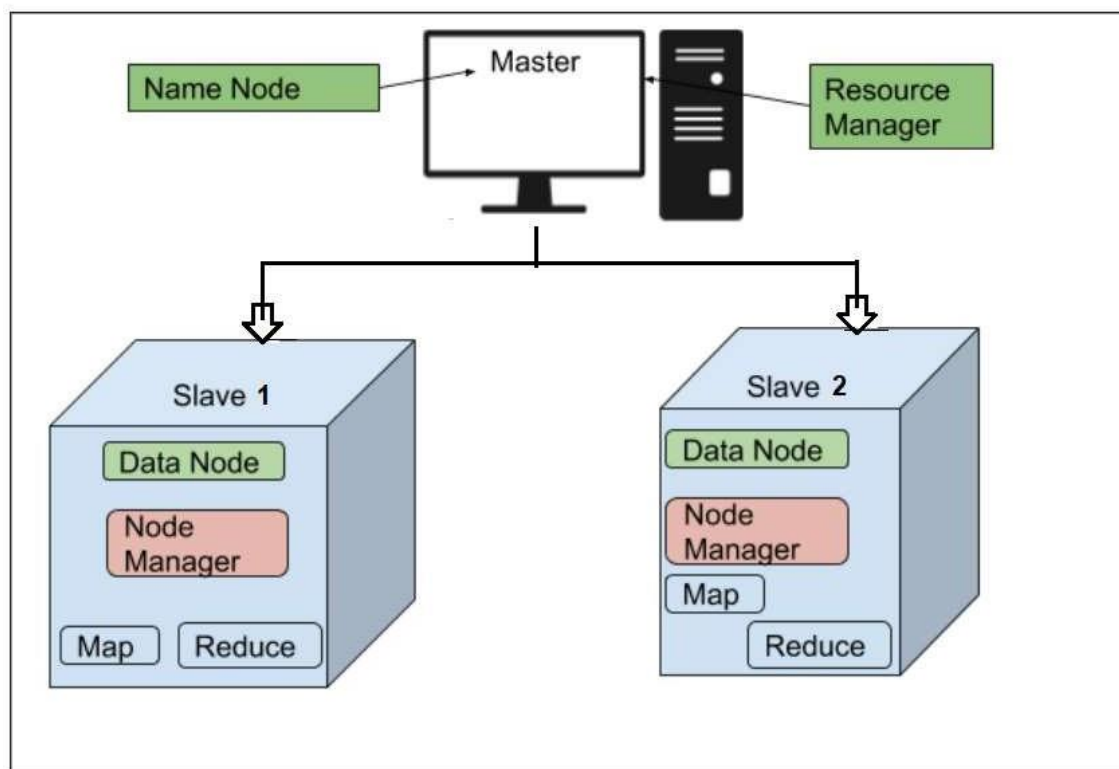
CustomerID: Customer number. Nominal, a 5-digit integral number uniquely assigned to each customer.

Country: Country name. Nominal, the name of the country where each customer resides.

1.4. Hadoop environment setup

Apache Hadoop - Apache Hadoop software library is a framework that allows for the distributed processing of large data sets across clusters of computers using simple programming models

HADOOP CLUSTER ARCHITECTURE



It provides a software framework for distributed storage and the processing of big data using the MapReduce. It was originally design for computer clusters ("is a set of tightly connected computers that work together, so they can be viewed as a single system") commodity hardware, it has also found use on cluster of higher-end hardware.

The base Apache Hadoop framework is composed of the following modules:

- Hadoop YARN — Apache Hadoop YARN is the resource management and job scheduling technology in the open source Hadoop distributed processing framework.
- Hadoop MapReduce — MapReduce is a framework using which we can write applications to process huge amounts of data, in parallel, on large clusters of commodity hardware in a reliable manner
- Apache Hive -- Hive is an open source data warehouse software, similar to SQL, for reading, writing and managing large data sets that are stored in Apache Hadoop files.
- Hadoop Common — Hadoop Common refers to the collection of common utilities and libraries that support other Hadoop modules. contains libraries and utilities needed by other Hadoop modules;
- Hadoop Distributed File System (HDFS) — HDFS is a distributed file system that handles large data sets running on commodity hardware. It is used to scale a single Apache Hadoop cluster to hundreds (and even thousands) of nodes.

Software configured for executing this assignment

Ubuntu 18.04 installed on a VM.

Hadoop Multi-Node Cluster

- **Java 8;**
- **SSH;**
- **PDSH;**

Hadoop Echo system installation

- **hadoop-3.3.4.tar.gz**
- **apache-hive-3.1.2-bin.tar.gz**

2. Data Acquisition – Storage and Metadata

For the problem identified, we have to download the dataset from campus and upload into HDFS

2.1. Create Hive Directories in HDFS

Create two separate directories to store data in the HDFS layer:

- The temporary, *tmp* directory is going to store the intermediate results of Hive processes.
- The *warehouse* directory is going to store the [Hive related tables](#).

2.2. Create tmp Directory

Create a *tmp* directory within the HDFS storage layer. This directory is going to store the intermediary data Hive sends to the HDFS:

```
hdfs dfs -mkdir /tmp
```

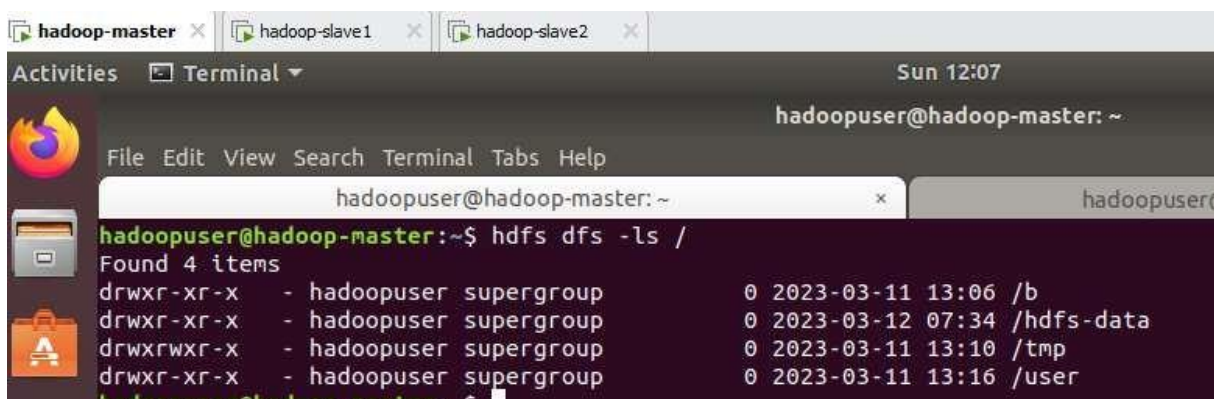
Add write and execute permissions to tmp group members:

```
hdfs dfs -chmod g+w /tmp
```

Check if the permissions were added correctly:

```
hdfs dfs -ls /
```

The output confirms that users now have write and execute permissions.



```
hadoopuser@hadoop-master: ~
hadoopuser@hadoop-master: ~$ hdfs dfs -ls /
Found 4 items
drwxr-xr-x - hadoopuser supergroup          0 2023-03-11 13:06 /b
drwxr-xr-x - hadoopuser supergroup          0 2023-03-12 07:34 /hdfs-data
drwxrwxr-x - hadoopuser supergroup          0 2023-03-11 13:10 /tmp
drwxr-xr-x - hadoopuser supergroup          0 2023-03-11 13:16 /user
```

2.3. Create warehouse Directory

Create the *warehouse* directory within the */user/hive/* parent directory:

```
hdfs dfs -mkdir -p /user/hive/warehouse
```

Add **write** and **execute** permissions to *warehouse* group members:

```
hdfs dfs -chmod g+w /user/hive/warehouse
```

Check if the permissions were added correctly:

```
hdfs dfs -ls /user/hive
```

The output confirms that users now have write and execute permissions.

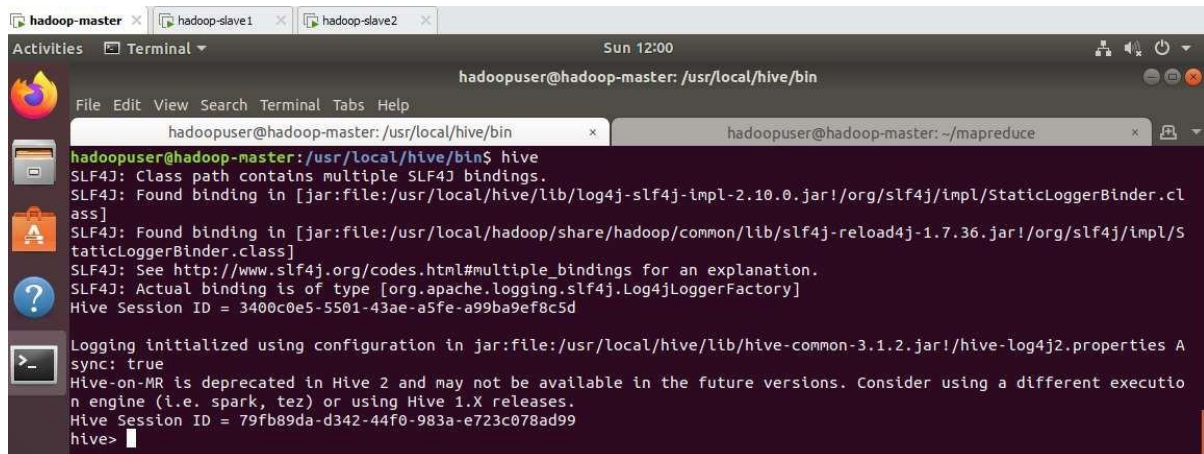
2.4. Launch Hive Client Shell on Ubuntu

Start the Hive command-line interface using the following commands:

```
cd $HIVE_HOME/bin
```

```
hive
```

Now we are able to issue SQL-like commands and directly interact with HDFS.



2.5. Create table in Hive Client Shell on Ubuntu

The statement used to create a table in Hive is Create Table.

Sr.No	Field Name	Data Type
1	RecordNo	Bigint
2	Invoice	VARCHAR(64)
3	StockCode	VARCHAR(64),
4	Description	VARCHAR(512)
5	Quantity	Bigint
6	InvoiceDate	VARCHAR(512)
7	Price	Double
8	Customer_ID	Bigint
9	Country	VARCHAR(64)

The following data is a Comment, Row formatted fields such as Field terminator, Description double quotes.

```
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'  
WITH SERDEPROPERTIES (  
  "separatorChar" = ",",  
  "quoteChar" = "\""
```

The following query creates a table named **online_retail_data** using the above data.

```
hive> DROP TABLE IF EXISTS online_retail_data;  
hive> create table online_retail_data  
RecordNo bigint,  
Invoice VARCHAR(64),
```

```

StockCode VARCHAR(64),
Description VARCHAR(512),
Quantity bigint,
InvoiceDate VARCHAR(512),
Price double,
Customer_ID bigint,
Country VARCHAR(64)
)
ROW FORMAT SERDE 'org.apache.hadoop.hive.serde2.OpenCSVSerde'
WITH SERDEPROPERTIES (
  "separatorChar" = ",",
  "quoteChar" = "\"";

```

If you add the option IF NOT EXISTS, Hive ignores the statement in case the table already exists.

2.6. Load data into table in Hive Client Shell on Ubuntu

In Hive, we can insert data using the LOAD DATA statement.

While inserting data into Hive, it is better to use LOAD DATA to store bulk records. There are two ways to load data: one is from local file system and second is from Hadoop file system.

The following query loads the given text into the table.

```

hive> load data local inpath '/home/hadoopuser/data/online_retail_data.csv' into TABLE
online_retail_data;

```

3. Data Analytics

We need to perform analysis of the data to find out the following 7 parameters for finalizing the sales strategy of the company for year 2011:

1. Total number of unique customers in the "given country"

Solution:

Query Criteria: Given Country is "United Kingdom"

Hive Query: select country,count(distinct customer_id) from online_retail_data where country='United Kingdom' group by country;

Result:

```

hive> select country,count(distinct customer_id) from online_retail_data where country='United Kingdom' group by country;
Query ID = hadoopuser_20230312045746_70ca2410-4d36-4213-b791-97ab924980d4
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>

```

```

In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 04:57:52,063 Stage-1 map = 0%, reduce = 0%
2023-03-12 04:58:52,585 Stage-1 map = 0%, reduce = 0%
2023-03-12 04:59:26,915 Stage-1 map = 100%, reduce = 0%
2023-03-12 04:59:27,919 Stage-1 map = 100%, reduce = 100%
Ended Job = job_local1285148571_0002
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 197617328 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
United Kingdom 4036
Time taken: 101.749 seconds, Fetched: 1 row(s)

```

2. Country from which the maximum revenue was collected from sales in the month of March 2010.

Solution:

Query Criteria: Invoice Date is March 2010

Hive Query:

```

Select country,sum(revenue) revenue from (select country, price*quantity revenue from
online_retail_data where
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-
yyyy'))),'yyyyMM')='201003' and country='United Kingdom') aa group by country;

```

Result:

```

hive> Select country,sum(revenue) revenue from (select country, price*quantity revenue fro
m online_retail_data where date_format(coalesce(from_unixtime(unix_timestamp(invoicedat
e , 'MM/dd/yyyy')), from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')), from_unixtim
e(unix_timestamp(invoicedate , 'dd-MM-yyyy')),from_unixtime(unix_timestamp(invoicedate , '
MM-dd-yyyy'))),'yyyyMM')='201003' and country='United Kingdom') aa group by country;
Query ID = hadoopuser_20230312050716_b24404be-a4a1-4671-8897-00a1c0da3d87
Total jobs = 1
Launching Job 1 out of 1
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>

```

```

In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 05:07:36,850 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:08:38,033 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:09:38,491 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:09:43,986 Stage-1 map = 100%, reduce = 0%
2023-03-12 05:09:44,992 Stage-1 map = 100%, reduce = 100%
Ended Job = job_local917476511_0001
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 98804568 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
United Kingdom 640495.531999983
Time taken: 148.463 seconds, Fetched: 1 row(s)

```

3. Month of 2010 in which maximum number of items were sold.

Solution:

Query Criteria: Invoice Date is year 2010

Hive Query:

```

select aa. monthOfYear, sum(aa.Quantity) itemSold from (select
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyyMM') monthOfYear,
Quantity Quantity from online_retail_data where
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyy')='2010' ) aa group
by aa.monthOfYear order by itemSold desc limit 1;

```

Result:

```

MapReduce Jobs Launched: hive> select aa. monthOfYear, sum(aa.Quantity) itemSold from
(select date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')), fr
om_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')), from_unixtime(unix_timestamp(in
voicedate , 'dd-MM-yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyy
MM') monthOfYear, Quantity Quantity from online_retail_data where date_format(coalesce(fr
om_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')), from_unixtime(unix_timestamp(in
voicedate , 'dd/MM/yyyy')), from_unixtime(unix_timestamp(invoicedate , 'dd-MM-yyyy')),from
_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyy')='2010' ) aa group by aa.mont
hOfYear order by itemSold desc limit 1;

```



```

Query ID = hadoopuser_20230312051259_8899eaa8-cedc-4a49-b9b4-fadea5aa7c97
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 05:13:03,250 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:14:03,554 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:14:57,660 Stage-1 map = 100%, reduce = 0%
2023-03-12 05:14:58,663 Stage-1 map = 100%, reduce = 100%
Ended Job = job_local1806004784_0002
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 05:15:00,939 Stage-2 map = 100%, reduce = 100%
Ended Job = job_local1132033867_0003
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 197609136 HDFS Write: 0 SUCCESS
Stage-Stage-2: HDFS Read: 197609136 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
201009 574791.0
Time taken: 121.377 seconds, Fetched: 1 row(s)

```

4. In the month of January 2010, find the country in which maximum number of items were sold

Solution:

Query Criteria: Invoice Date is January 2010

Hive Query:

```

select aa.Country, aa.Quantity from (select Country, sum(Quantity) Quantity from
online_retail_data where
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-
yyyy'))),'yyyyMM')='201001' group by Country) aa order by Quantity desc limit 1;

```

Result:

```

hive> select aa.Country, aa.Quantity from (select Country, sum(Quantity) Quantity from onlin
e_retail_data where date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/
dd/yyyy')), from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')), from_unixtime(unix_t
imestamp(invoicedate , 'dd-MM-yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-
yyyy'))),'yyyyMM')='201001' group by Country) aa order by Quantity desc limit 1;
Query ID = hadoopuser_20230312052458_d39748e4-9ac0-4fff-966b-2ef511d61aed
Total jobs = 2
Launching Job 1 out of 2
Number of reduce tasks not specified. Estimated from input data size: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 05:25:03,028 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:26:03,283 Stage-1 map = 0%, reduce = 0%
2023-03-12 05:26:33,286 Stage-1 map = 100%, reduce = 100%
Ended Job = job_local497737127_0004
Launching Job 2 out of 2
Number of reduce tasks determined at compile time: 1
In order to change the average load for a reducer (in bytes):
  set hive.exec.reducers.bytes.per.reducer=<number>
In order to limit the maximum number of reducers:
  set hive.exec.reducers.max=<number>
In order to set a constant number of reducers:
  set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 05:26:38,232 Stage-2 map = 100%, reduce = 0%
2023-03-12 05:26:39,235 Stage-2 map = 100%, reduce = 100%
Ended Job = job_local1403861080_0005
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 296413704 HDFS Write: 0 SUCCESS
Stage-Stage-2: HDFS Read: 296413704 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
United Kingdom 269953.0
Time taken: 100.71 seconds, Fetched: 1 row(s)

```

5. The StockCode of the item with the highest number of sales in the "given country" in the year 2010

Solution:

Query criteria:

Given Country is "United Kingdom" and Invoice Date is 2010

Hive Query:

```
select StockCode, Quantity from (select StockCode, sum(Quantity) Quantity from
online_retail_data where
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyy')='2010'
and Country ='United Kingdom' group by StockCode) aa order by Quantity desc limit 1;
```

Result:

```
hive> select StockCode, Quantity from (select StockCode, sum(Quantity) Quantity from onlin
e_retail_data where date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/
dd/yyyy')), from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')), from_unixtime(unix_t
imestamp(invoicedate , 'dd-MM-yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-
yyyy'))),'yyyy')='2010' and Country = 'United Kingdom' group by StockCode) aa order by Qu
antity desc limit 1;
```

Query ID = hadoopuser_20230312075709_769eeafe-4e76-4b20-a1d2-40b165a1d14a

Total jobs = 2

Launching Job 1 out of 2

Number of reduce tasks not specified. Estimated from input data size: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2023-03-12 07:57:55,745 Stage-1 map = 0%, reduce = 0%

2023-03-12 07:58:56,318 Stage-1 map = 0%, reduce = 0%

2023-03-12 07:59:37,598 Stage-1 map = 100%, reduce = 0%

2023-03-12 07:59:38,654 Stage-1 map = 100%, reduce = 100%

Ended Job = job_local2046938602_0001

Launching Job 2 out of 2

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>

Job running in-process (local Hadoop)

2023-03-12 07:59:40,819 Stage-2 map = 0%, reduce = 0%

2023-03-12 07:59:41,839 Stage-2 map = 100%, reduce = 100%

Ended Job = job_local10808891_0002

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 98804568 HDFS Write: 0 SUCCESS

Stage-Stage-2: HDFS Read: 98804568 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

84077 49019.0

Time taken: 151.727 seconds, Fetched: 1 row(s)

6. StockCode of the item for which the maximum revenue was received by sales in the month of December 2010.

Solution:

Query criteria: Invoice Date is December 2010

Hive Query:

```
select StockCode, sum(revenue) revenue from (select StockCode, (Price*Quantity) revenue
from online_retail_data where
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-
yyyy'))),'yyyyMM')='201012') aa group by StockCode order by revenue desc limit 1;
```

Result:

MapReduce Jobs Launched:

```
<pre>hive>; select StockCode, sum(revenue) revenue from (select StockCode, (Price*Qua
ntity) revenue from online_retail_data where date_format(coalesce(from_unixtime(unix_time
stamp(invoicedate , &apos;MM/dd/yyyy&apos;)), from_unixtime(unix_timestamp(invoicedate
, &apos;dd/MM/yyyy&apos;)), from_unixtime(unix_timestamp(invoicedate , &apos;dd-MM-yy
yy&apos;)),from_unixtime(unix_timestamp(invoicedate , &apos;MM-dd-yyyy&apos;))),&apos;
yyyyMM&apos;)=&apos;201012&apos;) aa group by StockCode order by revenue desc limit
1;
```

Query ID = hadoopuser_20230312080953_8023670f-80ea-44a9-bc0e-cf6d8b0f7164

Total jobs = 2

Launching Job 1 out of 2

Number of reduce tasks not specified. Estimated from input data size: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>;

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>;

In order to set a constant number of reducers:

set mapreduce.job.reduces=<number>;

Job running in-process (local Hadoop)

2023-03-12 08:09:56,259 Stage-1 map = 0%, reduce = 0%

2023-03-12 08:10:56,659 Stage-1 map = 0%, reduce = 0%

2023-03-12 08:11:56,885 Stage-1 map = 0%, reduce = 0%

2023-03-12 08:11:57,890 Stage-1 map = 100%, reduce = 100%

Ended Job = job_local671913460_0003

Launching Job 2 out of 2

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

set hive.exec.reducers.bytes.per.reducer=<number>;

In order to limit the maximum number of reducers:

set hive.exec.reducers.max=<number>;

In order to set a constant number of reducers:
set mapreduce.job.reduces=<number>
Job running in-process (local Hadoop)
2023-03-12 08:11:59,767 Stage-2 map = 100%, reduce = 100%
Ended Job = job_local1050237045_0004
MapReduce Jobs Launched:
Stage-Stage-1: HDFS Read: 197609136 HDFS Write: 0 SUCCESS
Stage-Stage-2: HDFS Read: 197609136 HDFS Write: 0 SUCCESS
Total MapReduce CPU Time Spent: 0 msec
OK
22423 5565.71
Time taken: 126.755 seconds, Fetched: 1 row(s)

7. The country in which minimum number of sales happened in 2010.

Solution:

Query Criteria: Invoice Date is 2010

Hive Query:

```
select aa.Country, aa.Quantity from (select Country, sum(Quantity) Quantity from  
online_retail_data where  
date_format(coalesce(from_unixtime(unix_timestamp(invoicedate , 'MM/dd/yyyy')),  
from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),  
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-  
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyy')='2010'  
group by Country) aa order by Quantity asc limit 1;
```

Result:

```
hive> select aa.Country, aa.Quantity from (select Country, sum(Quantity) Quantity from
online_retail_data where date_format(coalesce(from_unixtime(unix_timestamp(invoicedate ,
'MM/dd/yyyy')), from_unixtime(unix_timestamp(invoicedate , 'dd/MM/yyyy')),
from_unixtime(unix_timestamp(invoicedate , 'dd-MM-
yyyy')),from_unixtime(unix_timestamp(invoicedate , 'MM-dd-yyyy'))),'yyyy')='2010' group by
Country) aa order by Quantity asc limit 1;
```

Query ID = hadoopuser_20230312082100_eed79d30-cbb4-4e41-a0a2-dc2bdacc2c83

Total jobs = 2

Launching Job 1 out of 2

Number of reduce tasks not specified. Estimated from input data size: 1

In order to change the average load for a reducer (in bytes):

```
set hive.exec.reducers.bytes.per.reducer=<number>
```

In order to limit the maximum number of reducers:

```
set hive.exec.reducers.max=<number>
```

In order to set a constant number of reducers:

```
set mapreduce.job.reduces=<number>
```

Job running in-process (local Hadoop)

2023-03-12 08:21:18,056 Stage-1 map = 0%, reduce = 0%

2023-03-12 08:22:18,190 Stage-1 map = 0%, reduce = 0%

2023-03-12 08:23:12,400 Stage-1 map = 100%, reduce = 0%

2023-03-12 08:23:13,468 Stage-1 map = 100%, reduce = 100%

Ended Job = job_local1905248091_0001

Launching Job 2 out of 2

Number of reduce tasks determined at compile time: 1

In order to change the average load for a reducer (in bytes):

```
set hive.exec.reducers.bytes.per.reducer=<number>
```

In order to limit the maximum number of reducers:

```
set hive.exec.reducers.max=<number>
```

In order to set a constant number of reducers:

```
set mapreduce.job.reduces=<number>
```

Job running in-process (local Hadoop)

2023-03-12 08:23:20,983 Stage-2 map = 0%, reduce = 0%

2023-03-12 08:23:24,706 Stage-2 map = 100%, reduce = 100%

Ended Job = job_local1019689135_0002

MapReduce Jobs Launched:

Stage-Stage-1: HDFS Read: 98804568 HDFS Write: 0 SUCCESS

Stage-Stage-2: HDFS Read: 98804568 HDFS Write: 0 SUCCESS

Total MapReduce CPU Time Spent: 0 msec

OK

Lebanon 71.0

Time taken: 144.569 seconds, Fetched: 1 row(s)

8. Map Reducer Implementation for the query 1:

Mapper.py source code below

```
#!/usr/bin/python3

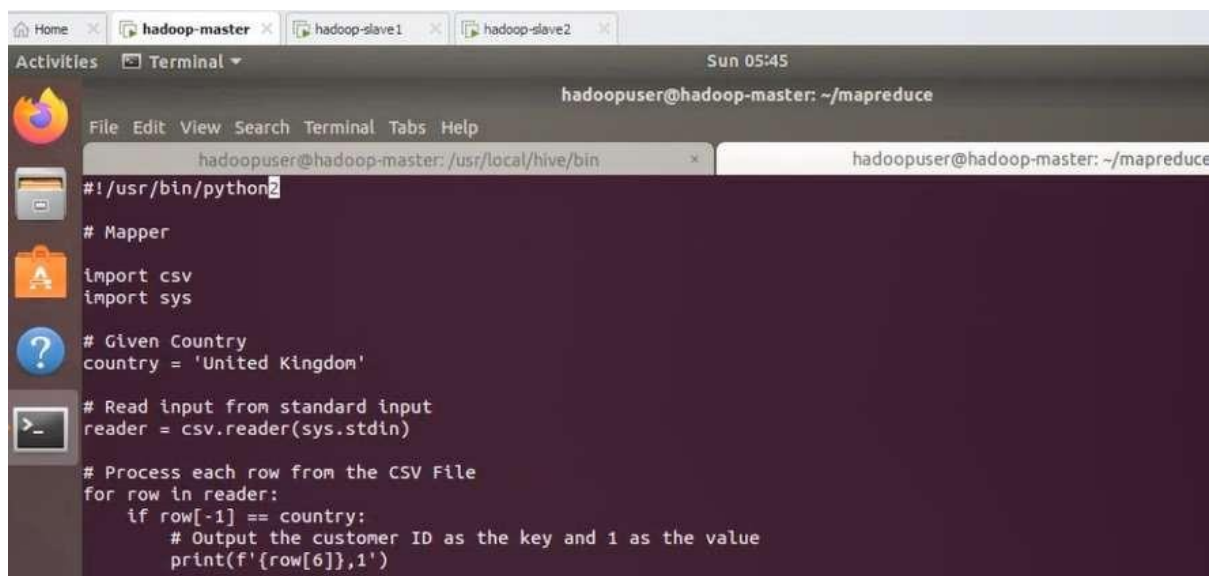
# Mapper

import csv
import sys

# Given Country
country = 'United Kingdom'

# Read input from standard input
reader = csv.reader(sys.stdin)

# Process each row from the CSV File
for row in reader:
    if row[-1] == country:
        # Output the customer ID as the key and 1 as the value
        print(f'{row[6]},1')
```

A screenshot of a terminal window on a Linux system. The window title is "Sun 05:45" and the prompt is "hadoopuser@hadoop-master: ~/mapreduce". The terminal shows the same Mapper.py source code as the previous block. The code is being displayed in a terminal window with a dark background and light text. The window has a standard Linux desktop environment with a taskbar at the top showing icons for Home, Activities, and Terminal. The terminal window is titled "hadoop-master" and "hadoop-slave1", "hadoop-slave2". The code is being displayed in a terminal window with a dark background and light text. The window has a standard Linux desktop environment with a taskbar at the top showing icons for Home, Activities, and Terminal. The terminal window is titled "hadoop-master" and "hadoop-slave1", "hadoop-slave2".

```
hadoopuser@hadoop-master: ~/mapreduce
#!/usr/bin/python3
# Mapper
import csv
import sys
# Given Country
country = 'United Kingdom'
# Read input from standard input
reader = csv.reader(sys.stdin)
# Process each row from the CSV File
for row in reader:
    if row[-1] == country:
        # Output the customer ID as the key and 1 as the value
        print(f'{row[6]},1')
```

Reducer.py source code below

```
#!/usr/bin/python3

# Reducer

import sys

# Read the desired country from command-line arguments
country = 'United Kingdom'

# To store the unique customer ids
```

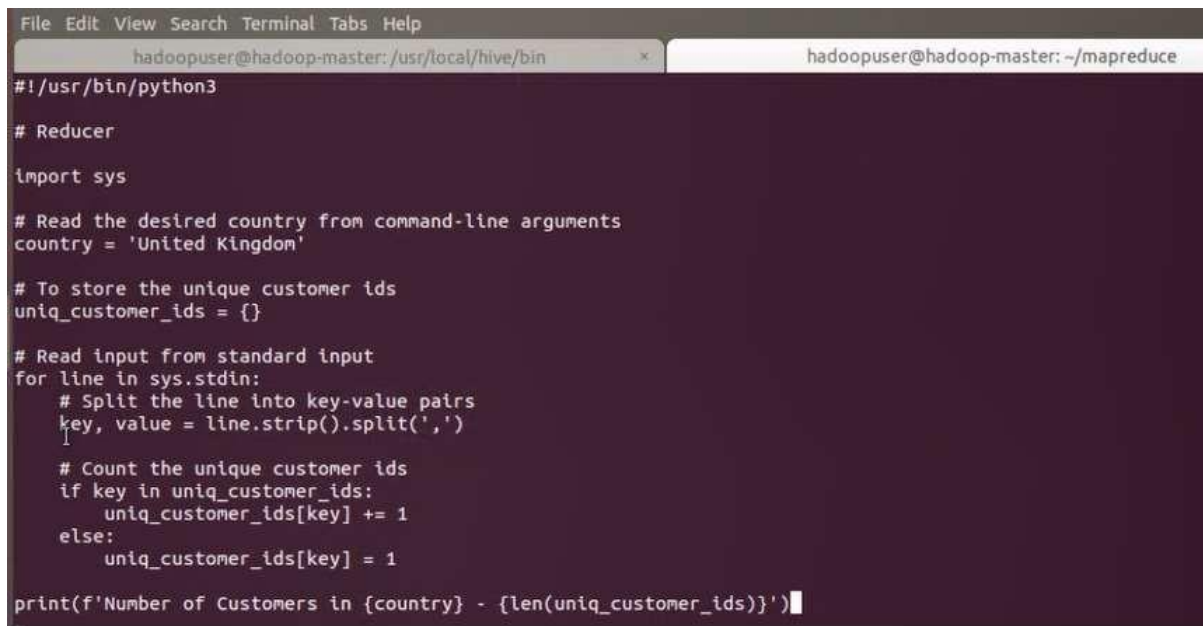
```

uniq_customer_ids = {}

# Read input from standard input
for line in sys.stdin:
    # Split the line into key-value pairs
    key, value = line.strip().split(',')

    # Count the unique customer ids
    if key in uniq_customer_ids:
        uniq_customer_ids[key] += 1
    else:
        uniq_customer_ids[key] = 1

```



```

File Edit View Search Terminal Tabs Help
hadoopuser@hadoop-master: /usr/local/hive/bin x hadoopuser@hadoop-master: ~/mapreduce
#!/usr/bin/python3
# Reducer
import sys

# Read the desired country from command-line arguments
country = 'United Kingdom'

# To store the unique customer ids
uniq_customer_ids = {}

# Read input from standard input
for line in sys.stdin:
    # Split the line into key-value pairs
    key, value = line.strip().split(',')

    # Count the unique customer ids
    if key in uniq_customer_ids:
        uniq_customer_ids[key] += 1
    else:
        uniq_customer_ids[key] = 1

print(f'Number of Customers in {country} - {len(uniq_customer_ids)}')

```

Command to execute the python file of Mapper and reducer

```

hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar -files
"/home/hadoopuser/mapreduce/mapper.py,/home/hadoopuser/mapreduce/reducer.py
" -input /hdfs-data/online_retail/online_retail_data.csv -output /hdfs-data/result -
mapper "/usr/bin/python3 mapper.py" -reducer "/usr/bin/python3 reducer.py"

```

Logs captured in the below screen-print:


```
hadoopuser@hadoop-master: /usr/local/hive/bin x hadoopuser@hadoop-master: ~/mapreduce x
hadoopuser@hadoop-master:~/mapreduce$ hadoop fs -rm /hdfs-data/online_retail/online_retail_data.csv
Deleted /hdfs-data/online_retail/online_retail_data.csv
hadoopuser@hadoop-master:~/mapreduce$ hadoop fs -put online_retail_data.csv /hdfs-data/online_retail
hadoopuser@hadoop-master:~/mapreduce$ hadoop jar /usr/local/hadoop/share/hadoop/tools/lib/hadoop-streaming-3.3.4.jar -f
iles "/home/hadoopuser/mapreduce/mapper.py,/home/hadoopuser/mapreduce/reducer.py" -input /hdfs-data/online_retail/online
_retail_data.csv -output /hdfs-data/result -mapper "/usr/bin/python3 mapper.py" -reducer "/usr/bin/python3 reducer.py"
2023-03-12 07:08:59,548 INFO impl.MetricsConfig: Loaded properties from hadoop-metrics2.properties
2023-03-12 07:09:04,303 INFO impl.MetricsSystemImpl: Scheduled Metric snapshot period at 10 second(s).
2023-03-12 07:09:04,307 INFO impl.MetricsSystemImpl: JobTracker metrics system started
2023-03-12 07:09:04,687 WARN impl.MetricsSystemImpl: JobTracker metrics system already initialized!
2023-03-12 07:09:08,783 INFO mapred.FileInputFormat: Total input files to process : 1
2023-03-12 07:09:09,013 INFO mapreduce.JobSubmitter: number of splits:1
2023-03-12 07:09:10,690 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_local1898775024_0001
2023-03-12 07:09:10,691 INFO mapreduce.JobSubmitter: Executing with tokens: []
2023-03-12 07:09:14,670 INFO mapred.LocalDistributedCacheManager: Localized file:/home/hadoopuser/mapreduce/mapper.py a
s file:/tmp/hadoop-hadoopuser/mapred/local/job_local1898775024_0001_d25fb3f1-3b30-40b1-a353-fb0688cc6f9a/mapper.py
2023-03-12 07:09:14,962 INFO mapred.LocalDistributedCacheManager: Localized file:/home/hadoopuser/mapreduce/reducer.py
as file:/tmp/hadoop-hadoopuser/mapred/local/job_local1898775024_0001_53ba16c0-ca35-4c33-8c76-afc4e4be4d653/reducer.py
2023-03-12 07:09:16,509 INFO mapreduce.Job: The url to track the job: http://localhost:8080/
2023-03-12 07:09:16,622 INFO mapred.LocalJobRunner: OutputCommitter set in config null
2023-03-12 07:09:16,877 INFO mapred.LocalJobRunner: OutputCommitter is org.apache.hadoop.mapred.FileOutputCommitter
2023-03-12 07:09:16,953 INFO output.FileOutputCommitter: File Output Committer Algorithm version is 2
2023-03-12 07:09:16,960 INFO output.FileOutputCommitter: FileOutputCommitter skip cleanup _temporary folders under outp
ut directory:false, ignore cleanup failures: false
2023-03-12 07:09:17,618 INFO mapred.LocalJobRunner: Waiting for map tasks
2023-03-12 07:09:17,701 INFO mapred.LocalJobRunner: Starting task: attempt_local1898775024_0001_m_000000_0
2023-03-12 07:09:17,819 INFO mapreduce.Job: Job job_local1898775024_0001 running in uber mode : false
```

```
hadoopuser@hadoop-master: /usr/local/hive/bin x hadoopuser@hadoop-master: ~/mapreduce x
Spilled Records=963254
Shuffled Maps =1
Failed Shuffles=0
Merged Map outputs=1
GC time elapsed (ms)=824
Total committed heap usage (bytes)=475004928
Shuffle Errors
BAD_ID=0
CONNECTION=0
IO_ERROR=0
WRONG_LENGTH=0
WRONG_MAP=0
WRONG_REDUCE=0
File Input Format Counters
Bytes Read=45309689
File Output Format Counters
Bytes Written=46
2023-03-12 07:35:07,193 INFO streaming.StreamJob: Output directory: /hdfs-data/result
hadoopuser@hadoop-master:~/mapreduce$ hadoop fs -cat /hdfs-data/result/part-00000
Number of Customers in United Kingdom - 4036
```

Number of customers in United Kingdom = 4036

/home/hadoopuser/mapreduce/				
Name	Size	Changed	Rights	Owner
..		12/03/2023 06:14:37 PM	rw-r--r--	hadoo...
mapper.py	1 KB	12/03/2023 08:04:06 PM	rw-rw-r--	hadoo...
online_retail_data.csv	44,248 KB	12/03/2023 07:32:52 PM	rw-rw-r--	hadoo...
reducer.py	1 KB	12/03/2023 06:17:58 PM	rw-rw-r--	hadoo...

9. Conclusion

Configuration of Hadoop Cluster:

Mater Node	1
Slave (Worker Node)	2
Block Size	DEFAULT- 128 MB
Replication Factor	2-configured in hdfs-site.xml attached

We have configured a Hadoop cluster with 1 Master Node, 2 Slave (worker) nodes, Block Size-Default (128 MB) and Replication Factor-2 (configured in hdfs-site.xml attached).

CSV data is stored in HDFS (Hadoop Distributed File System) cluster.

Hive Tables-Hive tables from open-source Hadoop framework was configured in Master node.

The data is read from HDFS and stored in HIVE Tables using Hive Query Language

Metadata - The data is retained as given and no modifications were done

Used:

We have used

- Hadoop Framework
- Hadoop HDFS
- Hive
- HiveQL
- Ubuntu (Linux based Kernal)
- MapReduce – When we run HQL query in Hive Tables, the query is converted into Map Reduce jobs (when there is computation involved in the query)

Data Analysis

The analysis was done and the following results were obtained using HQL queries

1. Total number of unique customers in the given country (United Kingdom)?
Result – **4036**
2. Country which collected maximum revenue by sales in Mar 2010?
Result – Country is United Kingdom and Revenue is – **640495**
3. In which month in the year 2010, maximum number of items were sold?
Result – **201009** **574791.0**
4. Country in which maximum number of items sold in Jan, 2010?
Result – **United Kingdom** – 269953)
5. StockCode of item with highest number of sales in 2010?
Result – StockCode is **84077 (Country – United Kingdom)**
6. StockCode for item which received maximum revenue by sales in Dec 2010?
Result – StockCode is **22423 (Revenue is 5565.71)**

7. Country in which maximum number of sales happened in 2010?
Result – **Lebanon (no of sales – 71)**

Implementation considerations:

- We have the required dataset, we have displayed the output of First 10 rows of the database.
- we have successfully accessed and extract data from Hive **default** database and got the required output by using appropriate queries and its result in section 3
- We have learnt how to query a Hive default database on Hadoop environment.
- We have Implemented Map-Reduce using Python code.