**Team Members**

Ayrin George

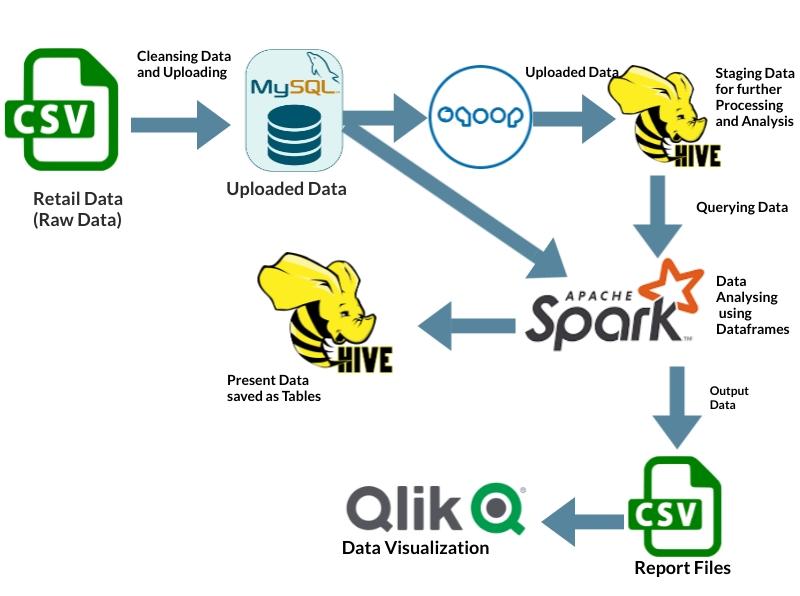
Aakash Bhatt

Manan Sharma

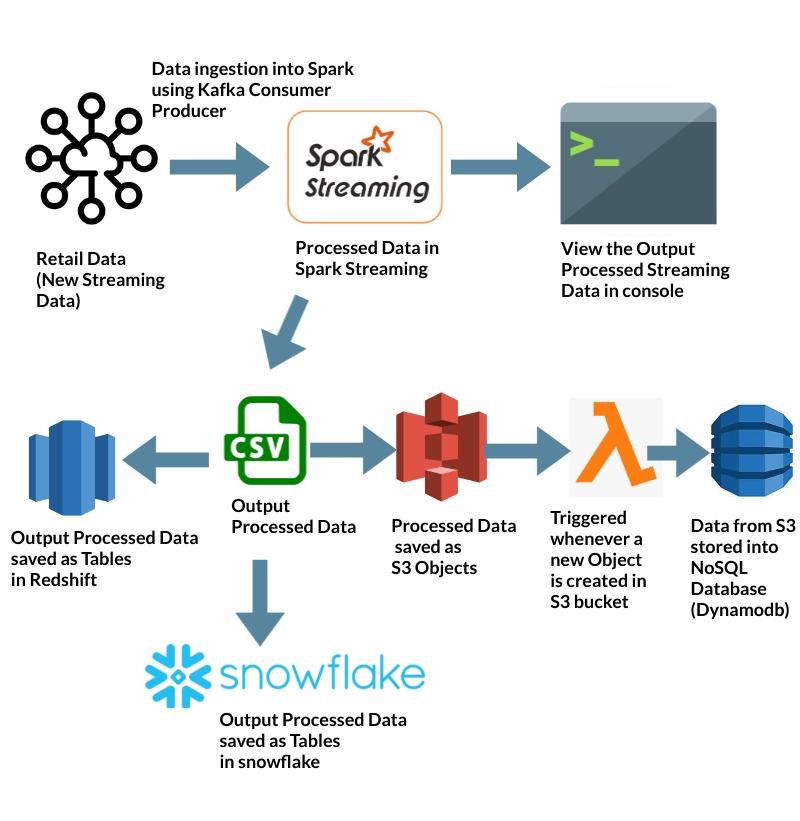
**Retail Case Study**

**Project Approach**

**Workflow 1**

****

**Workflow 2**

****

**Project Overview**

**Step 1:Downloading Retail Dataset**

* <https://github.com/akgeoinsys/retail>

**Step 2: Loading Data into MySQL**

* Creating MySQL database(retaildb)
* Creating tables and loading the csv files into it while also cleaning the data

**Step 3: Loading the Data into Hive from Mysql using Sqoop**

* Using Sqoop import to directly import into hive or to hdfs and then creating an external table.

**Step 4: Loading the Data into Spark and performing Transformation**

* Importing from either MySQL or Hive into Spark DataFrame.
* Applying transformations
* Using various optimisation techniques to store the result set.

**Step 5: Kafka and Spark Structured Streaming**

* Using Kafka as the producer and Spark as the consumer for streaming data.

**Step 6: Spark integration with various technologies**

* Integrating Spark with AWS S3, AWS Redshift and Snowflake.

**Step 7 :AWS Lambda**

* AWS Lambda function for s3 file watcher to trigger data load and process

**Step 8:Visualization**

* Using Visualization software(QlikSense) for the business queries

**Phase 1**

**Preparing And Importing Data**

1. **Cleansing and loading Data to MySQL**

* **Creating Database**

Create database retaildb;

* **Creating Tables**

**#creating table part**

Create table part (

partkey int PRIMARY KEY,

name Varchar(40),

mfgr varchar(40),

brand varchar(40),

type varchar(40),

size int,

container varchar(40),

retailprice float(11,2),

comment varchar(152) ) ;

**#creating table lineitem**

Create table lineitem(

orderkey int references orders(orderkey),

partkey int references partsupp(partkey),

suppkey int references partsupp(suppkey),

linenumber int,quantity float(11,2),

extendedprice float(11,2),

discount float(11,2),

tax float(11,2),

returnflag varchar(1),

linestatus varchar(1),

shipdate DATE,

commitdate DATE,

receiptdate DATE,

shipinstruct varchar(25),

shipmode varchar(10),

comment varchar(44));

**#creating table region**

Create table region(

region\_key int PRIMARY KEY,

name Varchar(40),

Comment Varchar(152));

**#creating table supplier**

Create table Supplier(

suppkey int PRIMARY KEY,

name Varchar(40),

Address varchar(160),

Nationkey int references nation(nationkey),

Phone int,

Acctbal float(11,2),

Comment varchar(180));

**#creating table partsupply**

Create table partsupply(

partkey int references part(partkey),

suppkey int references supplier(suppkey),

availqty int,

supplycost float(12,2),

comment varchar(199));

**#creating table customer**

Create table customer(

custkey int(38) primary key,

name varchar(25),

address varchar(40),

nationkey int(38) references nation(nationkey),

Phone varchar(15),

Acctbal float(12,2),

Mktsegment varchar(10),

Comment varchar(112));

**#creating table nation**

Create table nation(

Nationkey int primary key,

Name varchar(50) references region(region\_key),

Regionkey int,

Comment varchar(120));

**#creating table orders**

Create table orders(

Orderkey int primary key,

custkey int references customer(custkey),

Orderstatus varchar(1),

Totalprice float(12,2),

Order\_date date,

Orderpriority varchar(15),

Clerk varchar(15),

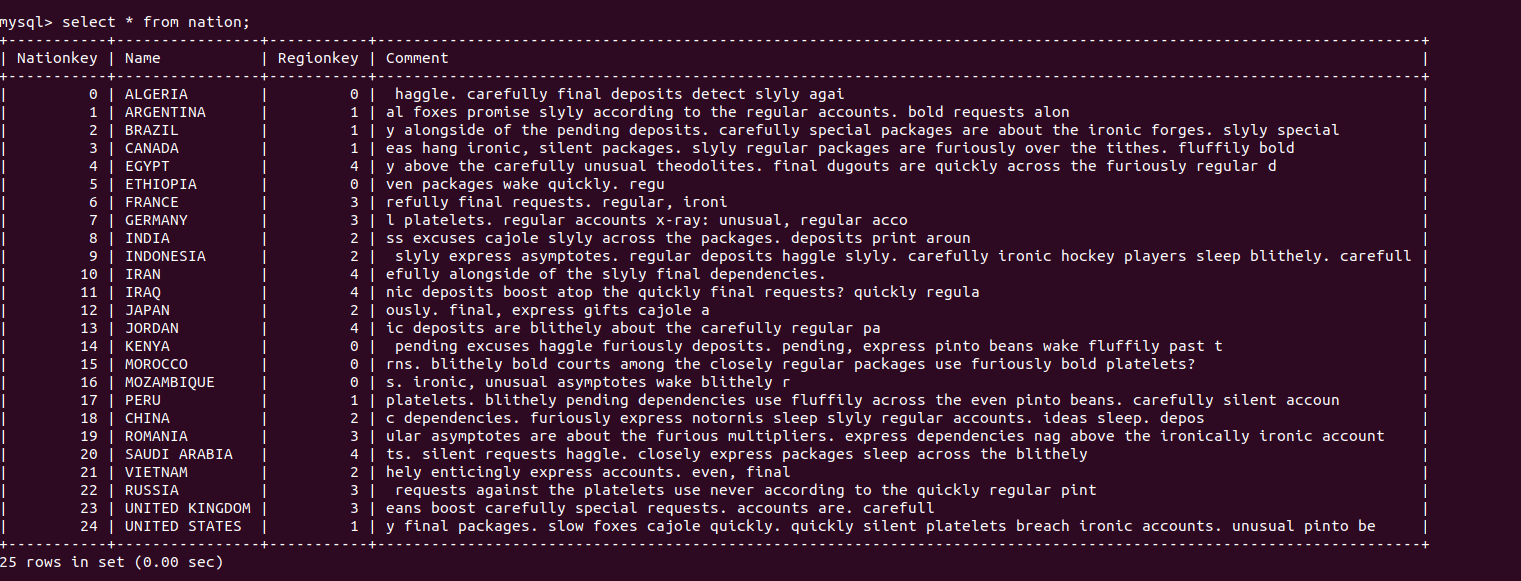
Shippriority int,

Comment varchar(79));

* **Loading Data into MySQL Table**

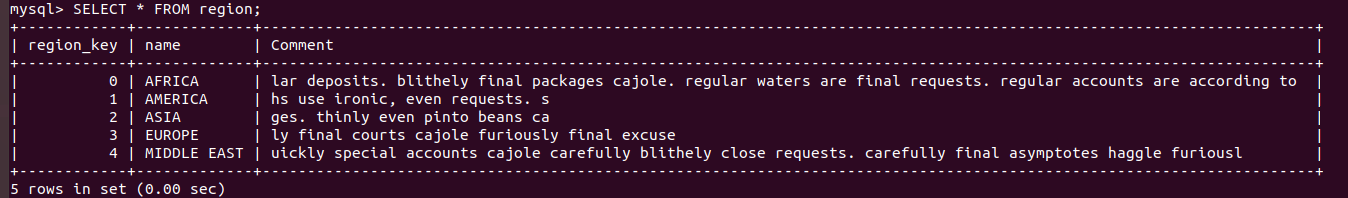
**#loading data into table nation**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/nation\_0\_0\_0.csv' INTO TABLE nation FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'



**#loading data into table region**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/region\_0\_0\_0.csv' INTO TABLE region FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'



**#loading data into table part**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/part\_0\_0\_0.csv' INTO TABLE part FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table supplier**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/supplier\_0\_0\_0.csv' INTO TABLE supplier FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table customer**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/customer\_0\_0\_0.csv' INTO TABLE ]customer FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table partsupp**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/partsupp\_0\_1\_0.csv' INTO TABLE partsupp FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table partsupp**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/partsupp\_0\_2\_0.csv' INTO TABLE partsupp FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table partsupp**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/partsupp\_0\_3\_0.csv' INTO TABLE partsupp FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table orders**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/orders\_0\_0\_0.csv' INTO TABLE orders FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table orders**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/orders\_0\_1\_0.csv' INTO TABLE orders FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table orders**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/orders\_0\_1\_1.csv' INTO TABLE orders FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table orders**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/orders\_0\_3\_0.csv' INTO TABLE orders FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table orders**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/orders\_0\_3\_1.csv' INTO TABLE orders FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_0\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_0\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_1\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_1\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_2\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_2\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_2\_2.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_3\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_3\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_3\_2.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_4\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_4\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_5\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_5\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_6\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_6\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_7\_0.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**#loading data into table lineitem**

LOAD DATA LOCAL INFILE’'/home/ak/Downloads/drive-download-20201230T045705Z-001/lineitem\_0\_7\_1.csv' INTO TABLE lineitem FIELDS TERMINATED BY ',' ENCLOSED BY '"' LINES TERMINATED BY '\n'

**2.Loading the Data into Hive from Mysql using Sqoop**

**A.Creating Database in Hive**

Create database retaildb;

Use retaildb;

**B.Loading Data from mysql**

# Creating Hive tables - Managed and External:

**Managed Tables External Tables**

**============ =============**

1)region 1)Customer.

2)part 2)Orders

3)party supply 3)Line Items

4)Nation

5)Supplier

**Managed Tables**

**#creating managed table-region**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword --table region -create-hive-table -hive-table retaildb.region -hive-import --fields-terminated-by ',' -m 1

**#creating managed table-partsupply**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table partsupply -create-hive-table -hive-table retaildb.partsupply -hive-import --fields-terminated-by ',' --split-by suppkey -m 3

**#creating managed table-part**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table part -create-hive-table -hive-table retaildb.part -hive-import --fields-terminated-by ',' --split-by suppkey -m 3

**#incremental load**

sqoop-import --connect jdbc:mysql://localhost/retaildb --username hiveuser -password hivepassword -m 1 --table orders --append --hive-database retaildb --hive-table --check-column order\_date --last-value ‘1998-08-02’ --incremental append --fields-terminated-by '|' --target-dir /project/orders;

**External Table:**

**# creating external table-nation**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table nation --fields-terminated-by '|' --target-dir /project/nation --append -- -m 1;

CREATE EXTERNAL TABLE IF NOT EXISTS nationext (

nationkey int,name string,address string,comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location '/project/nation';



**# creating external table-lineitem with mapper 3**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table lineitem --fields-terminated-by '|' --target-dir /project/lineitem --append --split-by orderkey -m 3;

CREATE EXTERNAL TABLE IF NOT EXISTS lineitemext (

orderkey int ,partkey int ,suppkey int ,linenumber int,quantity double,extendedprice double,discount double,tax double,returnflag string,linestatus string,shipdate string,commitdate string, receiptdate string,shipinstruct string,shipmode string,comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location '/project/lineitem';

**# creating external table-supplier**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table supplier --fields-terminated-by '|' --target-dir /project/supplier --append --split-by suppkey -m 3;

CREATE EXTERNAL TABLE IF NOT EXISTS supplierext (

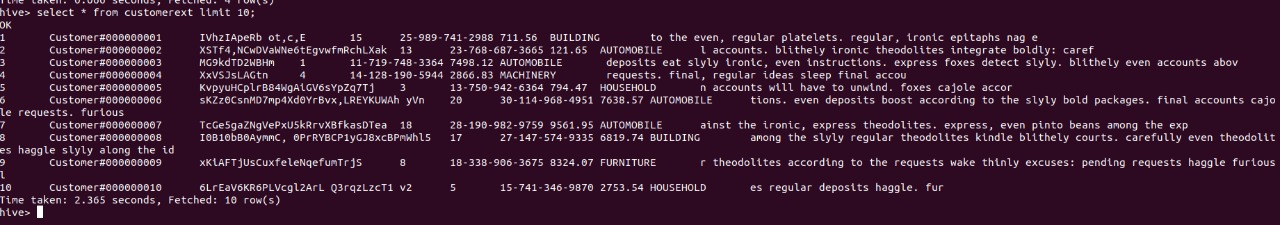
suppkey int,name string,address string,nationkey int,phone int,acctbal double,comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location '/project/supplier';

**#creating external table-customer**

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table customer --fields-terminated-by ‘|’ --target-dir /project/customer --append --split-by custkey -m 3;

CREATE EXTERNAL TABLE IF NOT EXISTS customerext (

custkey int, name string,address string,nationkey int,phone string ,acctbal double,mktsegment string,Comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location '/project/customer’;



**Part of Sqoop Shell Script**

!#/bin/bash

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword --table region -create-hive-table -hive-table retaildb.region -hive-import --fields-terminated-by ',' -m 1;

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword --table part -create-hive-table -hive-table retaildb.part -hive-import --fields-terminated-by ',' -m 1;

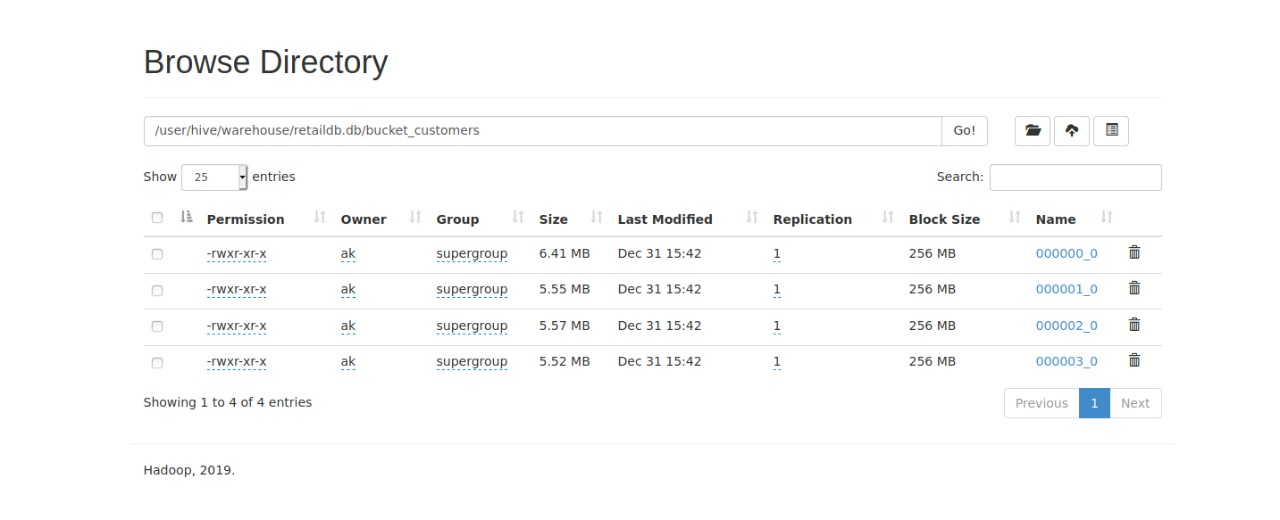
**#to run in terminal**

Bash project.sh

**Bucketing**

create table bucket\_customers(custkey int, name string,address string,nationkey int,phone string ,acctbal double,mktsegment string,Comment string) clustered by (nationkey) into 4 buckets ;

insert overwrite table bucket\_customers select \* from customerext;



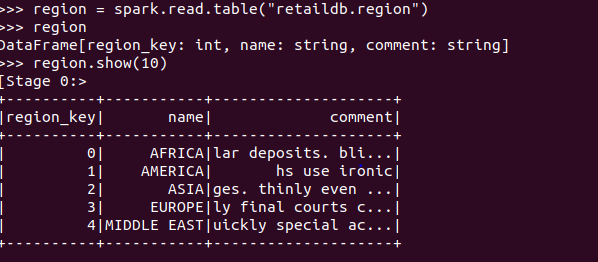
**3.Loading MySQL and Hive tables into Pyspark**

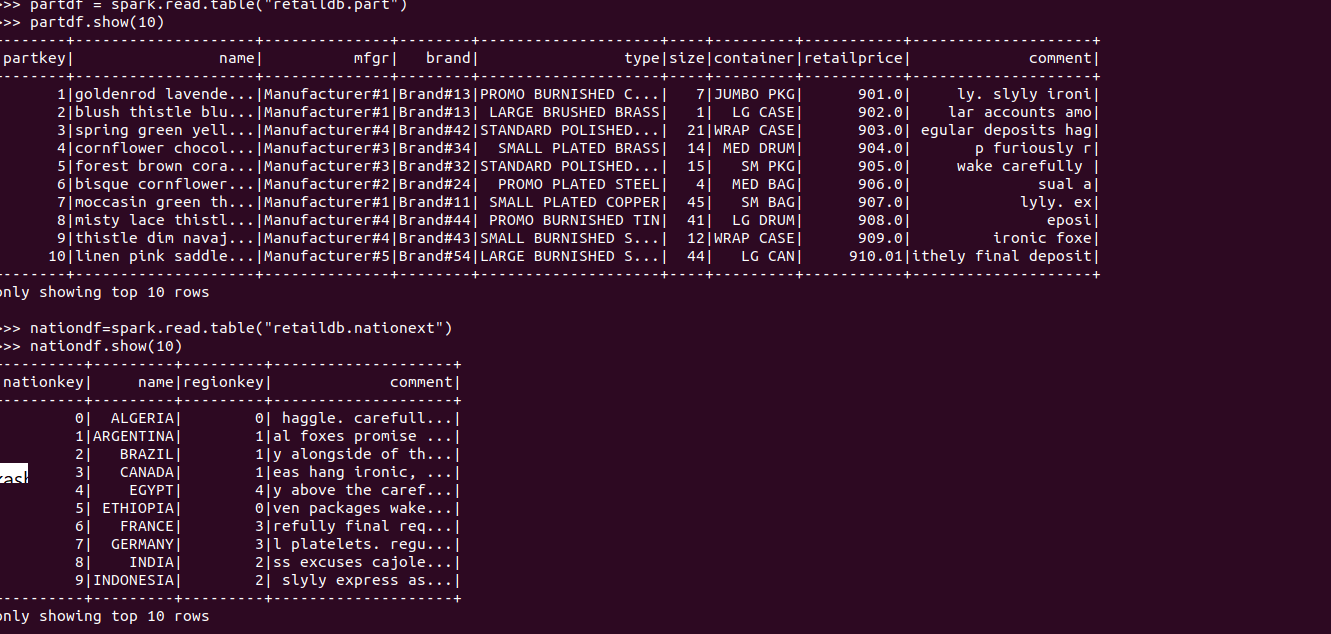
* **Creating Pyspark DataFrame from Hive tables**

regiondf = spark.read.table("retaildb.region")

partdf = spark.read.table("retaildb.part")

nationdf=spark.read.table("retaildb.nationext")

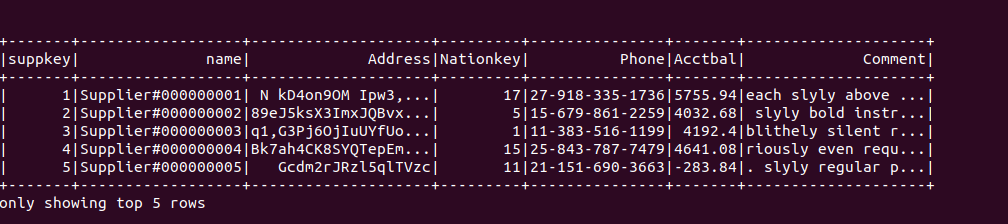




* **Creating Pyspark DataFrame From MySQL Tables**

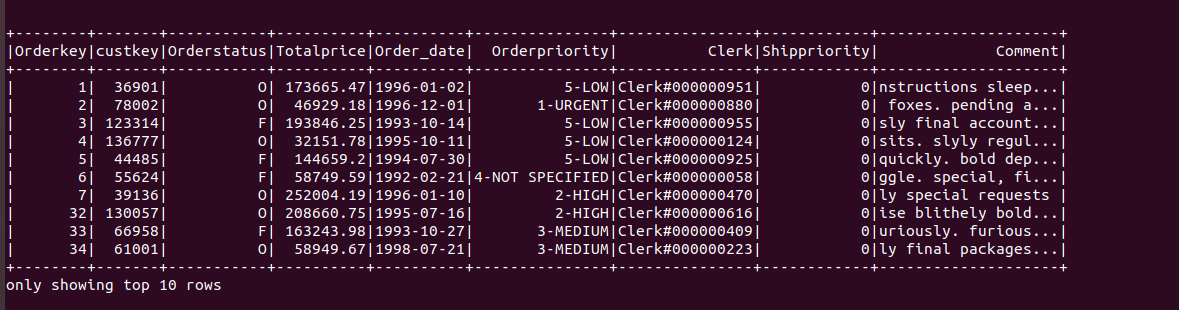
**#creating dataframe from supplier table**

supplierdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "supplier").option("lowerBound", "1").option("upperBound", "10000").option("numPartitions", " 5").option("partitionColumn","suppkey").option("user", "hiveuser").option("password", "hivepassword").load()



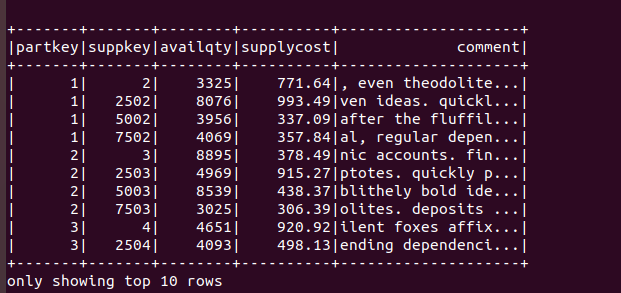
**#creating dataframe from order table**

ordersdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "orders").option("lowerBound", "1").option("upperBound", "1500000").option("numPartitions", " 10").option("partitionColumn","orderkey").option("user", "hiveuser").option("password", "hivepassword").load()



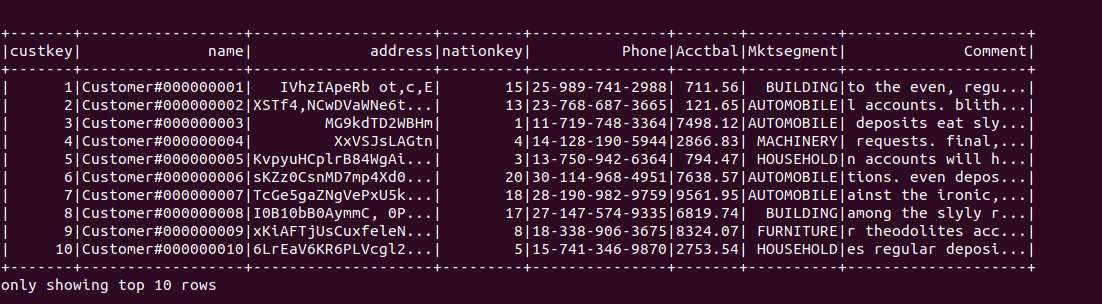
**#creating dataframe from partsupplytable**

partsupplydf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "partsupply").option("lowerBound", "1").option("upperBound", "800000").option("numPartitions", “8").option("partitionColumn","partkey").option("user", "hiveuser").option("password", "hivepassword").load()



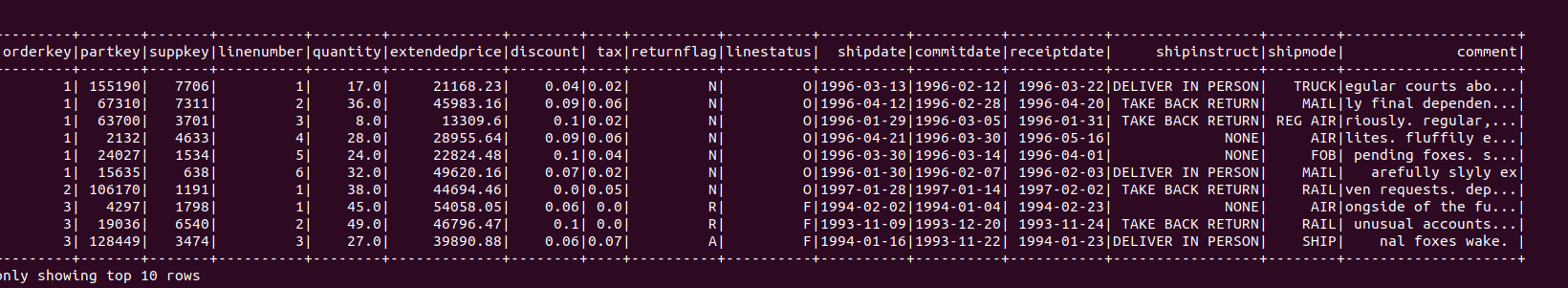
**#creating dataframe from customer table**

customerdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "customer").option("lowerBound", "1").option("upperBound", "150000").option("numPartitions", " 10").option("partitionColumn","custkey").option("user", "hiveuser").option("password", "hivepassword").load()



**#creating dataframe from lineitem table**

lineitemdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "lineitem").option("lowerBound", "1").option("upperBound", "6001215").option("numPartitions", " 20").option("partitionColumn","orderkey").option("user", "hiveuser").option("password", "hivepassword").load()



**4)Records Count Validation in Mysql,Hive and Spark**

| **Table** | **MySql** | **Spark** | **Hive** |
| --- | --- | --- | --- |
| **Region** | **5** | **5** | **5** |
| **Nation** | **25** | **25** | **25** |
| **Supplier** | **10000** | **10000** | **10000** |
| **Customer** | **150000** | **150000** | **150000** |
| **PartSupply** | **800000** | **800000** | **800000** |
| **Part** | **200000** | **200000** | **200000** |
| **Order** | **1500000** | **1500000** | **1500000** |
| **Lineitem** | **6001215** | **6001215** | **6001215** |

**5)Observation and Challenges**

* **Data Load Time**

| **Table** | **Sqoop** | **Pyspark** |
| --- | --- | --- |
| **Line Item** | **26sec** | **4sec** |
| **orders** | **14sec** | **2sec** |

* **Data format used was CSV. Delimited by comma and line separator ‘\n’. But certain fields also contained the commas within the strings.Hence it caused data mismatch while loading.To overcome this challenge we used DELIMITED FIELDS TERMINATED BY '|' while creating an external table on top of the data which was in HDFS.**
* **We can observe from point 4, the data has an enormous number of records. During partitioning and bucketing the number of records had to**

**be limited due to system limitations.**

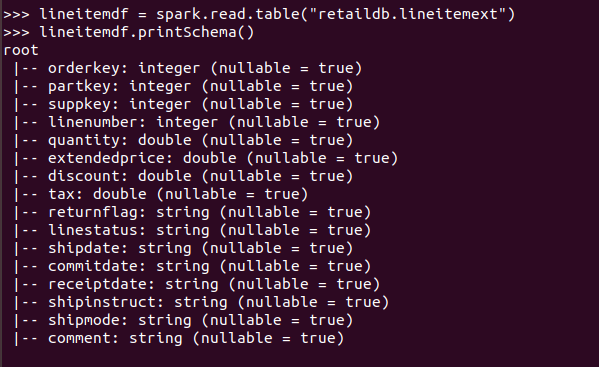
**Phase 2**

**Transforming Data**

**1.Type Casting Where Required**

**#before type casting**

lineitemdf = spark.read.table("retaildb.lineitemext")



from pyspark.sql.types import \*

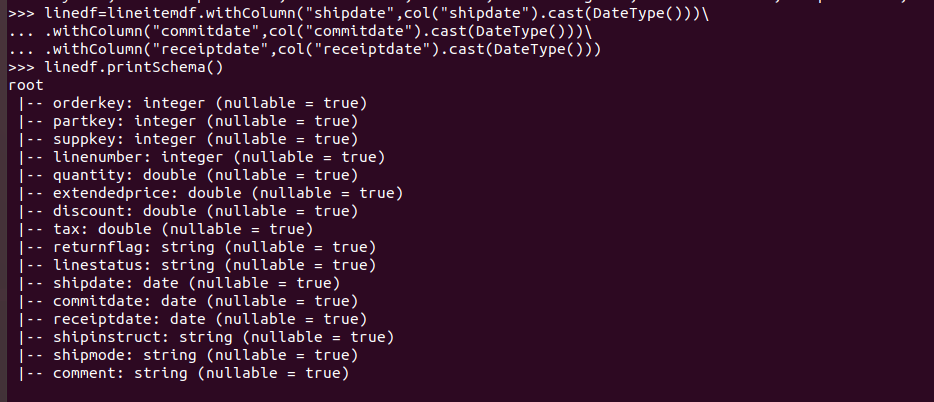
from pyspark.sql.functions import \*

linedf=lineitemdf.withColumn("shipdate",col("shipdate").cast(DateType()))\

.withColumn("commitdate",col("commitdate").cast(DateType()))\

.withColumn("receiptdate",col("receiptdate").cast(DateType()))

**#after Type Casting**



**2.Renaming Column**

customerdf =regiondf.withColumnRenamed("name","customer\_name")

**3.Applying Join**

**Joins are one of the fundamental operation when developing a spark job.**

**#Types of Join**

* **Outer Join**

Outer a.k.a full, fullouter join returns all rows from both Spark DataFrame/Datasets, where join expression doesn’t match it returns null on respective record columns.

* **Inner Join**

Spark Inner join is the default join and it’s mostly used. It is used to join two DataFrames/Datasets on key columns, and where keys don’t match the rows get dropped from both datasets

* **Left Outer Join**

Spark Left a.k.a Left Outer join returns all rows from the left DataFrame/Dataset regardless of match found on the right dataset when join expression doesn’t match, it assigns null for that record and drops records from right where match not found.

**Example:**

**partsupplydf=partsupplydf.groupBy("partkey").agg({'availqty':'sum'}).join(partdf,on=['partkey'],how='left').select(col('partkey'),col('sum(availqty)'),col('name'))**

* **Right Outer Join**

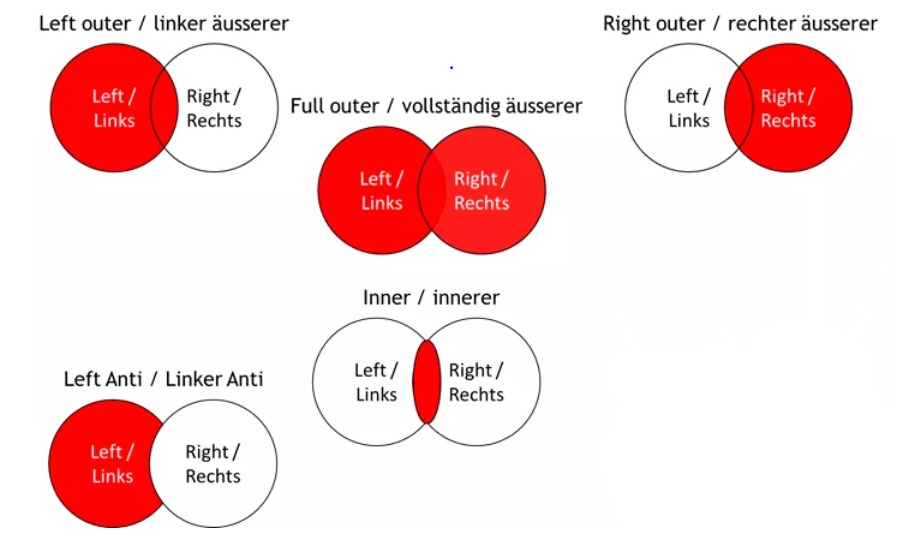
Spark Right a.k.a Right Outer join is opposite of left join, here it returns all rows from the right DataFrame/Dataset regardless of math found on the left dataset, when join expression doesn’t match, it assigns null for that record and drops records from left where match not found.

* **Left Semi Join**

Spark Left Semi join is similar to inner join difference being leftsemi join returns all columns from the left DataFrame/Dataset and ignores all columns from the right dataset. In other words, this join returns columns from the only left dataset for the records match in the right dataset on join expression, records not matched on join expression are ignored from both left and right datasets.

* **Left Anti Join**

Left Anti join does the exact opposite of the Spark leftsemi join, leftanti join returns only columns from the left DataFrame/Dataset for non-matched records.

****

**#Join Optimisations**

**SORT MERGE JOIN**

* **Sort-Merge join is composed of 2 steps. The first step is to sort the datasets and the second operation is to merge the sorted data in the partition by iterating over the elements and according to the join key join the rows having the same value.**
* **From spark 2.3 Merge-Sort join is the default join algorithm in spark. However, this can be turned down by using the internal parameter ‘spark.sql.join.preferSortMergeJoin’ which by default is true.**

**Broadcast Join**

* **Broadcast joins are the one which yield the maximum performance in spark. However, it is relevant only for little datasets. In broadcast join, the smaller table will be broadcasted to all worker nodes. Thus, when working with one large table and another smaller table always makes sure to broadcast the smaller table. We can hint spark to broadcast a table.**
* **Recently Spark has increased the maximum size for the broadcast table from 2GB to 8GB. Thus, it is not possible to broadcast tables which are greater than 8GB**
* **Spark also internally maintains a threshold of the table size to automatically apply broadcast joins. The threshold can be configured using “spark.sql.autoBroadcastJoinThreshold” which is by default 10mb.**

**Shuffle Hash join**

* **Shuffle Hash join works based on the concept of map reduce. Map through the data frames and use the values of the join column as output key. Shuffles the data frames based on the output keys and join the data frames in the reduce phase as the rows from the different data frame with the same keys will ended up in the same machine.**

**4.Applying Filter**

filterdf=supplierdf.filter(supplierdf["acctbal"]>0)

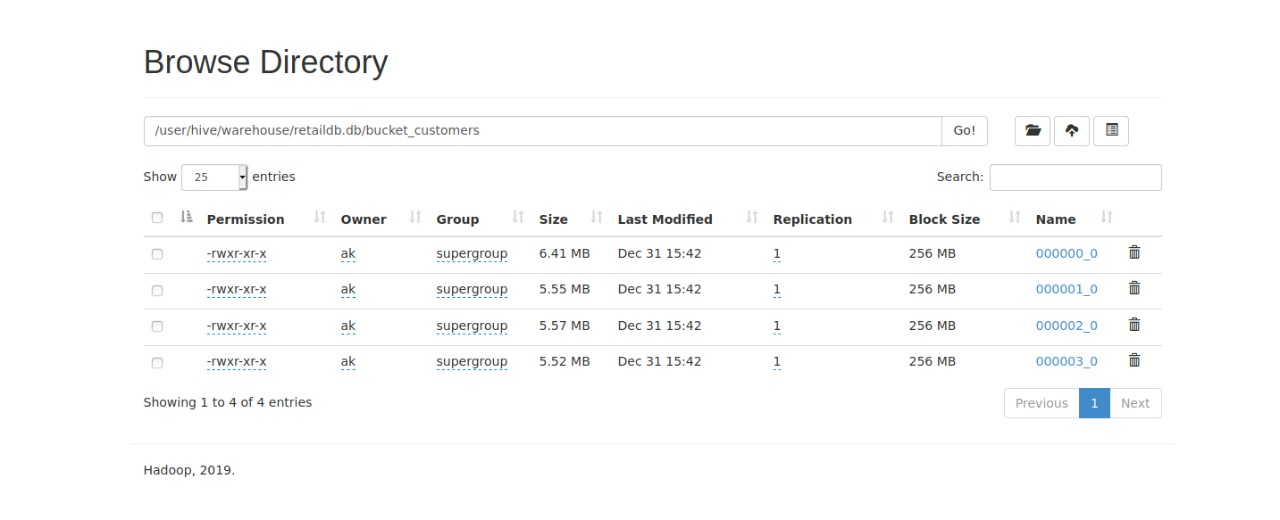
**Phase 3**

**Bucketing and Partitioning**

**Bucketing using Hive**

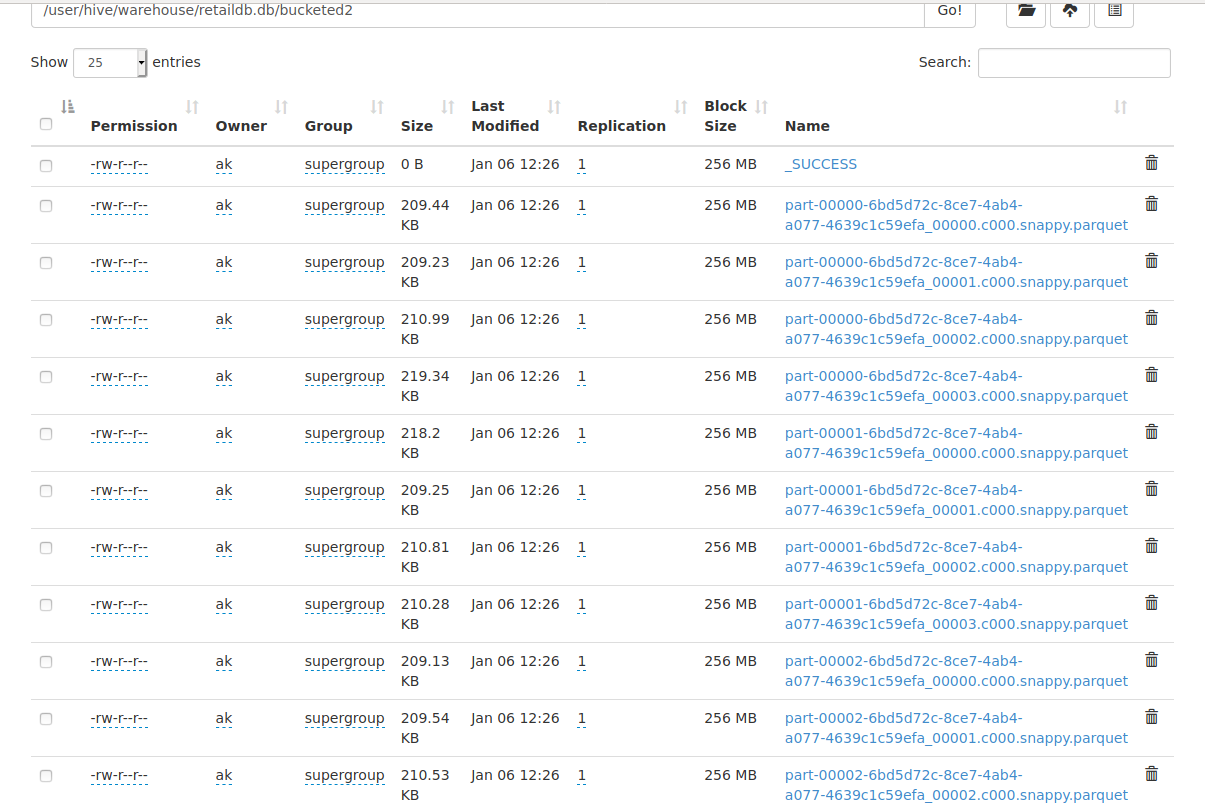
create table bucket\_customers(custkey int, name string,address string,nationkey int,phone string ,acctbal double,mktsegment string,Comment string) clustered by (nationkey) into 4 buckets ;

insert overwrite table bucket\_customers select \* from customerext;



**Bucketing using Spark**

partsupplydf.write.bucketBy(4,’partkey’).saveAsTable(‘bucketed’,format=’parquet’)



**Partitioning using spark**

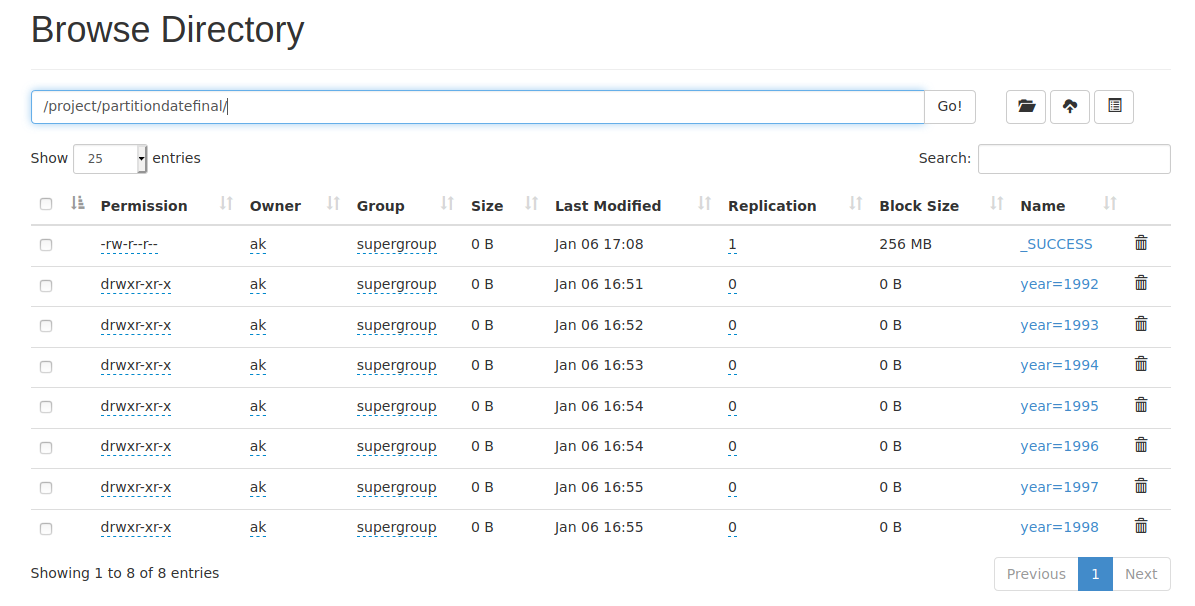
ordersdf.withColumn("year",year(col("Order\_date"))).withColumn("month",month(col("order\_date"))).withColumn("day",dayofmonth(col("order\_date"))).partitionBy("year","month","day").write.csv("/project/partitiondate")

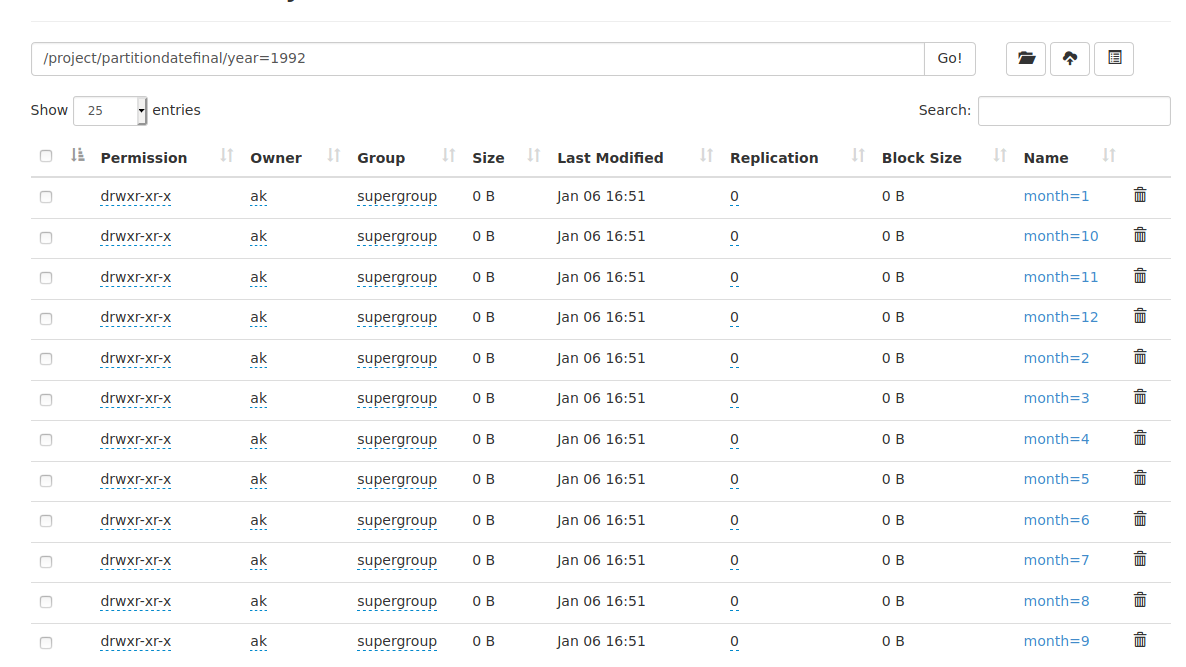
part1= ordersdf.withColumn("year",year(col("Order\_date")))

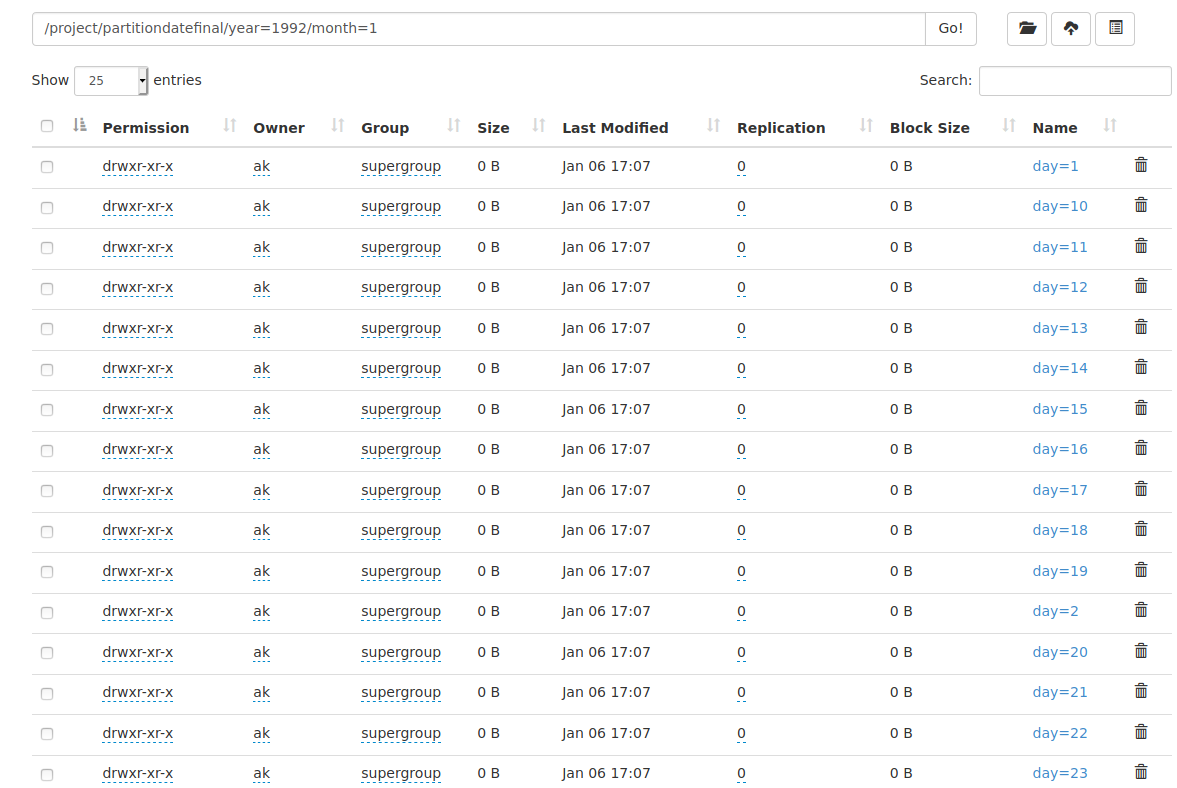
part2=part1.withColumn("month",month(col("order\_date")))

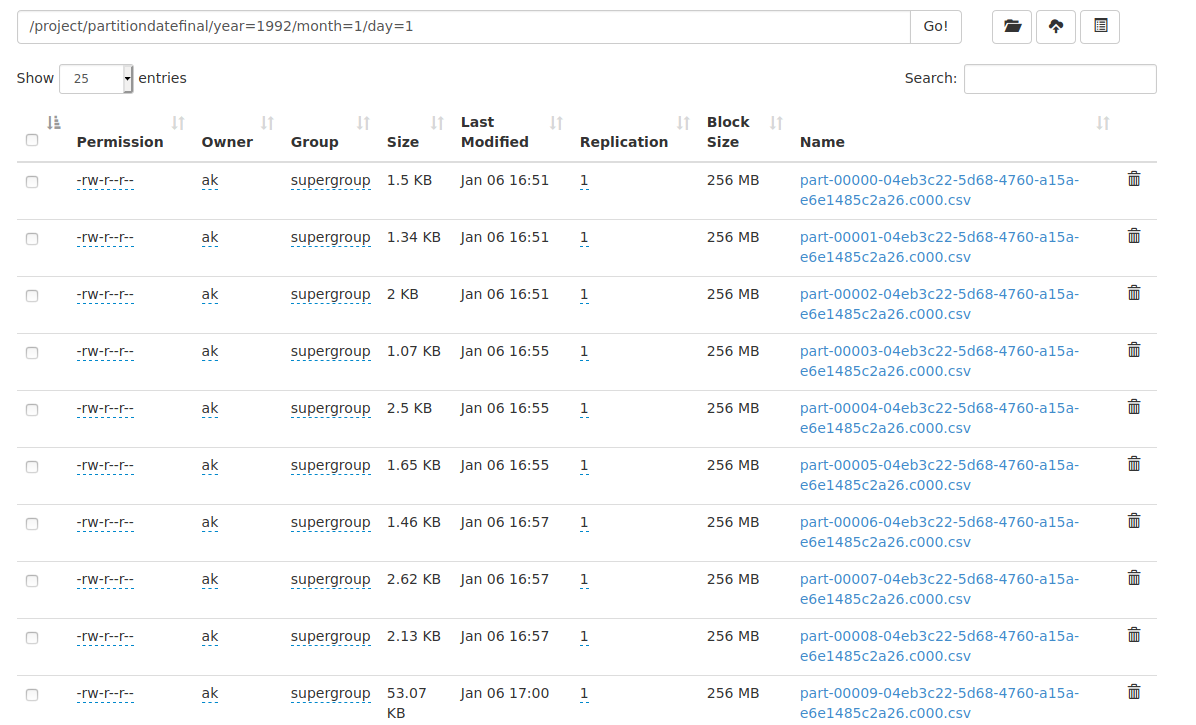
part3=part2.withColumn("day",dayofmonth(col("order\_date")))

part3.write.partitionBy("year","month","day").csv("/project/partitiondate3"







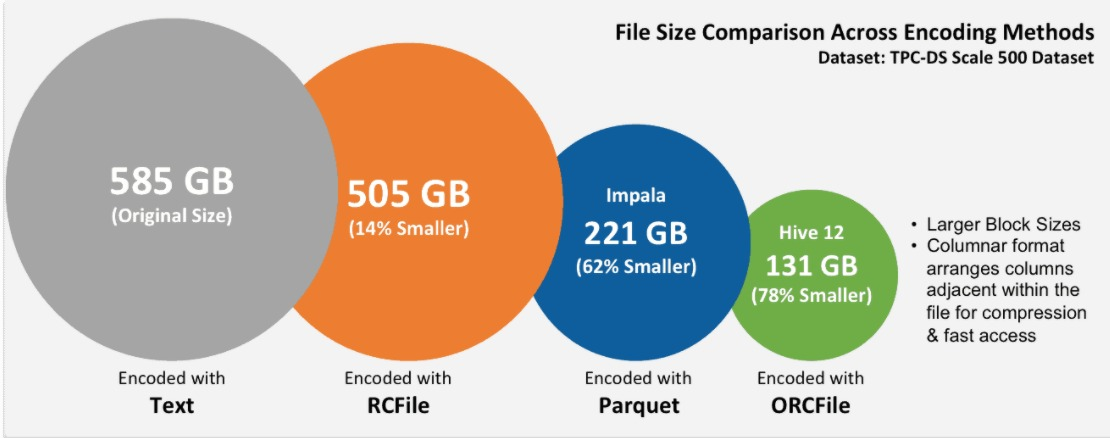


**File Format:**

**#Different Type of File Formats**

* **CSV**
* **Parquet**
* **Json**
* **Avro**
* **Orc**

**#Comparison of Different file Formats**

****

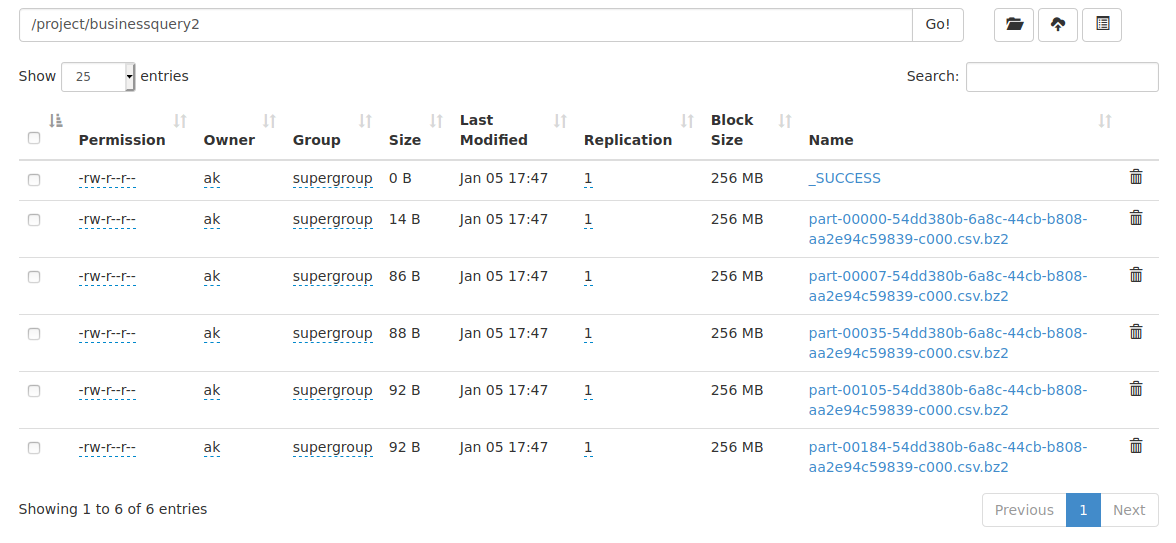
**File Compression**

**#Different Types of Compression Codecs.**

| **Compression Format** | **File extension** | **Compression Level** | **Speed** | **CPU Usage** | **Is Splitable** |
| --- | --- | --- | --- | --- | --- |
| **gzip** | **.gzip** | **Medium** | **Medium** | **Medium** | **No** |
| **bzip2** | **.bz2** | **High** | **Slow** | **High** | **No** |
| **lzo** | **.lzo** | **Average** | **Fast** | **Low** | **yes** |
| **snappy** | **.snappy** | **Average** | **Fast** | **Low** | **yes** |

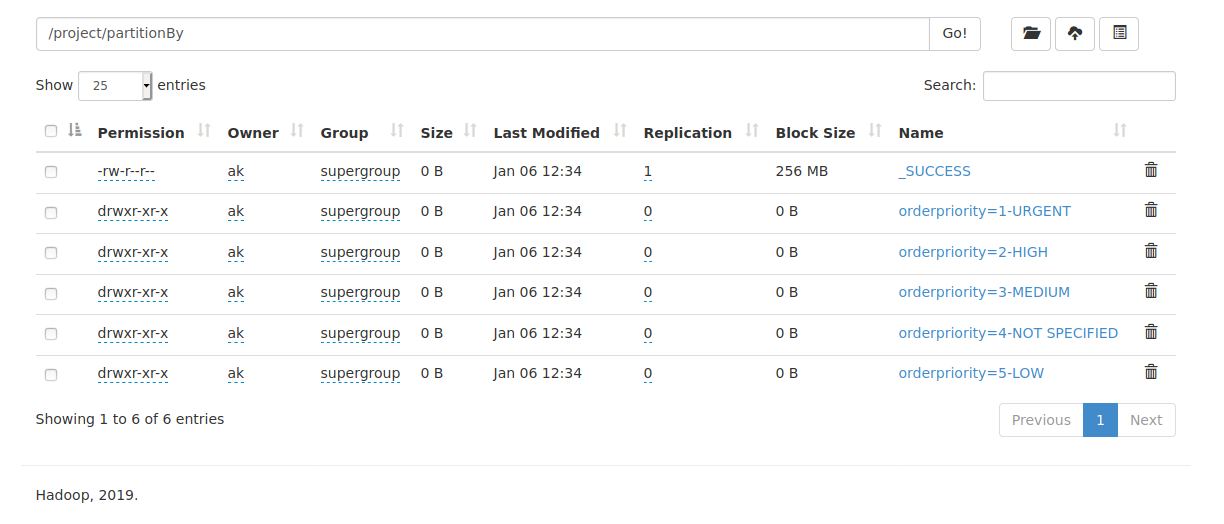
**#compression in bzip2**

solndffinal.write.option(“codec”,”bzip2”).parquet('/project/businessquery')



**#compression in gzip**

orderdf=order.write.partitionBy("orderpriority").option('codec','gzip').csv('/project/partitionBy')



**Implementing ACID properties using Delta Lake and Table**

**Delta Lake**

Delta Lake is an open-source storage layer that brings ACID transactions to Apache Spark and big data workloads.

**Features**

* ACID Transactions
* Scalable Metadata Handling
* Time Travel (data versioning)
* DML operations and many more….

**Application**

pyspark --packages io.delta:delta-core\_2.11:0.4.0

from delta.tables import \*

from pyspark.sql.functions import \*

**1.Creating Delta Table**

delta=spark.read.csv("file:///home/ak/Desktop/Project/customer.csv")

delta.write.format("delta").save("file:///home/ak/data/delta")

deltaTable=DeltaTable.forPath(spark,"file:///home/ak/projectglad/delta/")

**2.To save delta table as a dataframe**

delf=deltaTable.toDF()

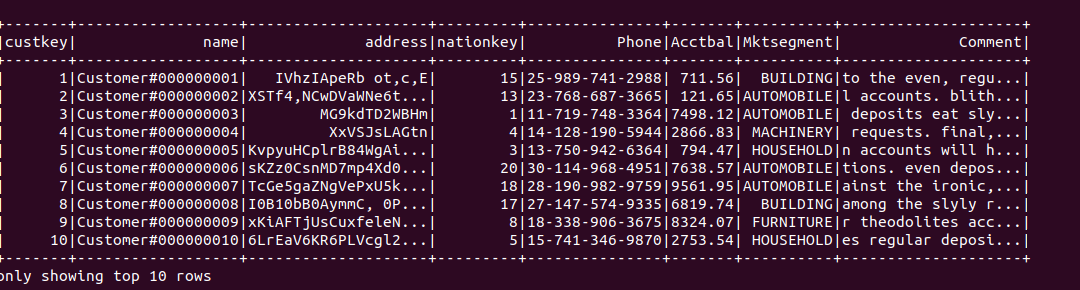
delf.show()

**3.Update using Delta Tables**

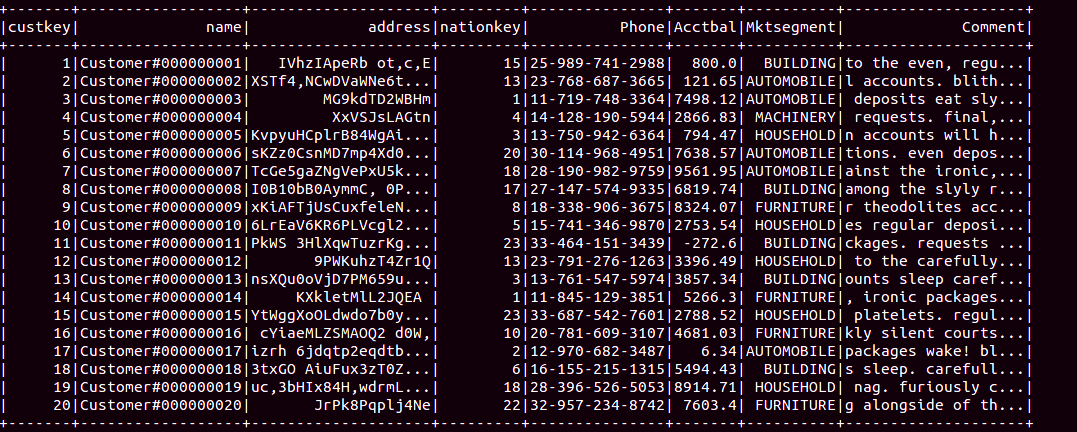
deltaTable.update("acctbal=’711.56’", {"acctbal": ‘800.0’})

delf.show()

#Before Updating Delta Table



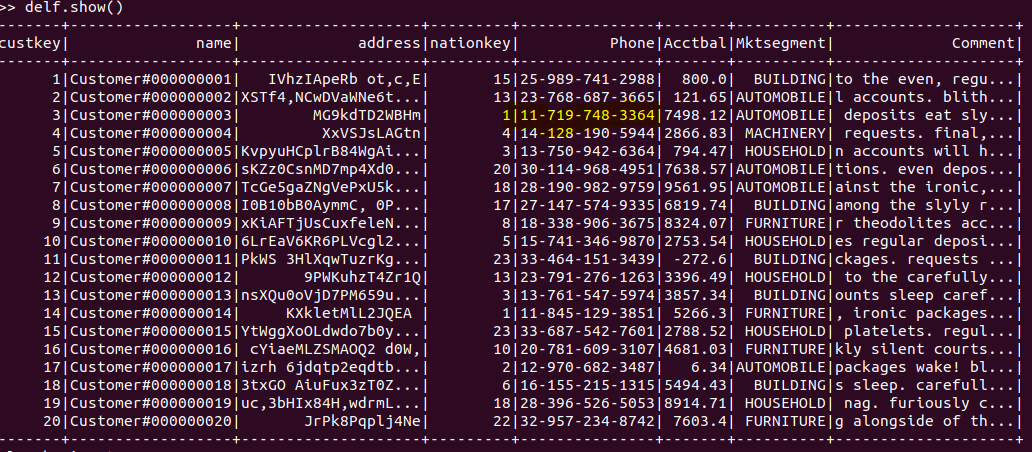
#After Updating



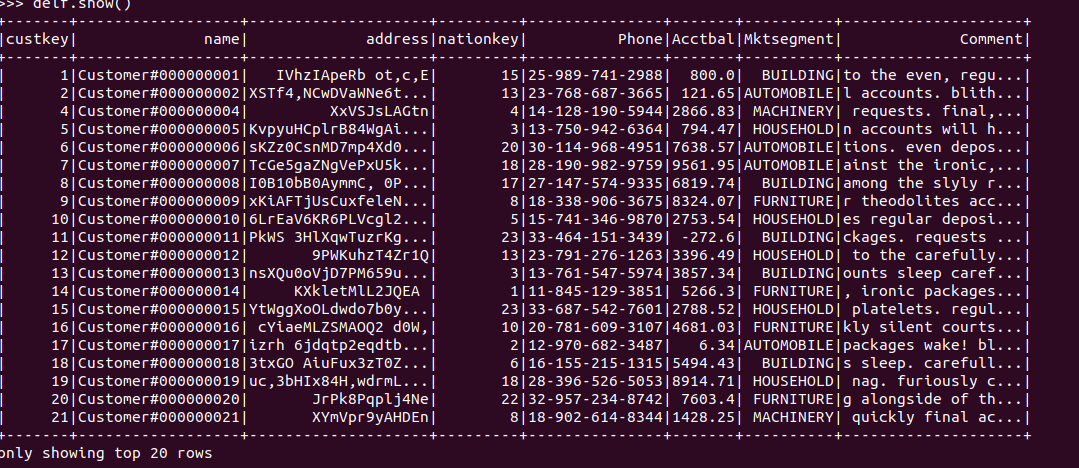
**4.Delete using Delta Tables**

deltaTable.delete(“Phone=’11-719-748-3364’”)

#Before Deleting



#After Deleting



**Phase 4**

**Kafka and Spark Structured Streaming Integration**

**Kafka Streaming**

Apache Kafka is a distributed commit log service and is widely used for data ingestion.It is similar to the concept to publishing and subscribing to messages.

It aslo offers scalability, performance, reliability, and flexibility

**Spark Structured Streaming**

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine. You can express your streaming computation the same way you would express a batch computation on static data.

**Why use Spark Structured Streaming instead of Spark Streaming**

* Spark streaming uses DStream API while structured streaming uses Dataframe API.
* The distribution of the data into micro batches(RDD) takes a lot of time on the macro level and hence leads to a considerable increase in the latency. While the Spark Structured Streaming uses Dataframe and the incoming data is directly inserted into an unbounded table.
* The libraries available for transformations on RDD are few in number and complex to understand meanwhile Dataframes have a huge amount libraries available which are easy to understand.
* The internal optimisation techniques used for Dataframes is way better than those used for RDDs.
* There is no provision for late data in spark streaming but it is provided in Structured Streaming (Watermarking).

**Application**

cd kafka-2.11-2.4.1./bin

**#### To Start zookeeper daemons**

--------------------------------------------

zookeeper-server-start.sh config/zookeeper.properties

**###To start the kafka server (in new terminal tab, run this)**

-------------------------------------------------

kafka-server-start.sh config/server.properties

**###To create the kafka topic(new tab)**

------------------------------------------------

kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 1 --partitions 1 --topic retaildb



**###To list the topics**

----------------------------------------------------------

kafka-topics.sh --list --zookeeper localhost:2181

**### To read kafka topic in pyspark**

-----------------------------------------------------

cat /home/ak/nation\_0\_0\_0.csv | /home/ak/kafka\_2.11-2.4.1/bin/kafka-console-producer.sh --broker-list localhost:9092 --topic retaildb

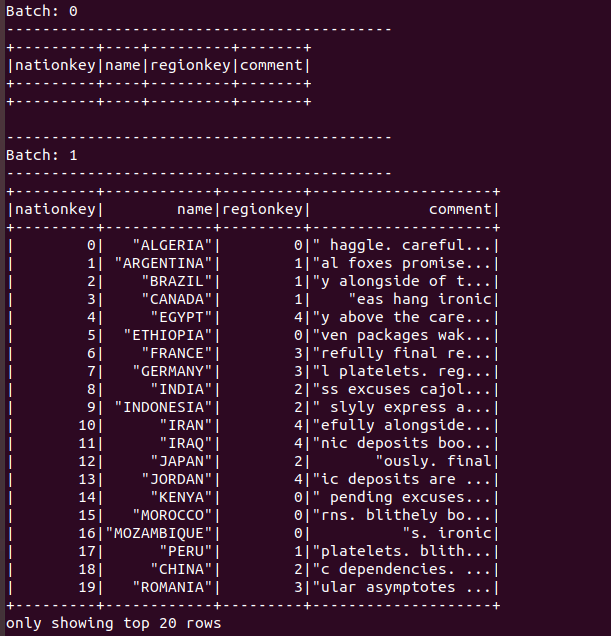
**Without Watermarking/Windowing**

pyspark --packages org.apache.spark:spark-sql-kafka-0-10\_2.11:2.4.5

line = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("subscribe","retaildb").load().selectExpr("CAST(value AS STRING)")

words =line.select(split(line.value,",")[0].alias("nationkey"),split(line.value,",")[1].alias("name"),split(line.value,",")[2].alias("regionkey"),split(line.value,",")[3].alias("comment"))

query =words.writeStream.outputMode("append").format("console")



**With Watermarking and Windowing**

line=spark.readStream.format("kafka").option('schema','nationSchema').option("kafka.bootstrap.servers","localhost:9092").option("subscribe","retaildb").option("includeTimestamp","true")load().selectExpr("CAST(value AS STRING)",”timestamp”)

from pyspark.sql.functions import window

words=line.select(split(line.value,",")[0].alias("nationkey"),split(line.value,",")[1].alias("name"),split(line.value,",")[2].alias("regionkey"),"timestamp")

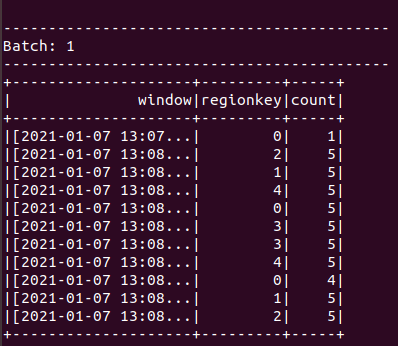
**// with watermark**

**// Group the data by window and regionkey and compute the count of each group**

WwindowedCounts = words.withWatermark("timestamp","10 seconds").groupBy(window("timestamp","10 seconds","5 seconds"), "regionkey" ).count().orderBy("window")

**// Start running the query that prints the windowed regionkey counts to the console**

query1 = WwindowedCounts.writeStream.outputMode("complete").format("console").start()



**Observation and Challenges**

* Generally Kafka Dataframe consists of key, value, topic, partition, offset,timestamp, timestamp type. Which had to be splitted into required format.
* Kafka dataframe value had to be separated using the regex pattern to avoid the column mismatch.

**Phase 5**

**Big Data and AWS Integration**

**1.AWS S3-Spark Integration**

**Reading from S3**

**1.Importing jars and packages**

pyspark --packages com.amazonaws:aws-java-sdk:1.7.4,org.apache.hadoop:hadoop-aws:2.7.3

**2.Defining the accesskey and secret key in /spark-2.4.5-bin-hadoop2.7/conf/spark-defaults.conf**

spark.hadoop.fs.s3a.access.key AKIAYLYEKLCYKPHKB2FX

Spark.hadoop.fs.s3a.secret.key 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

spark.hadoop.fs.s3a.impl org.apache.hadoop.fs.s3a.S3AFileSystem

park.hadoop.fs.s3n.access.key AKIAYLYEKLCYKPHKB2FX

spark.hadoop.fs.s3n.secret.key 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

spark.hadoop.fs.s3n.impl org.apache.hadoop.fs.s3n.S3AFileSystem

**3. Set the following hadoop configurations from Pyspark**

sc.\_jsc.hadoopConfiguration().set("fs.s3a.awsAccessKeyId","AKIAYLYEKLCYF7JKWGNA") sc.\_jsc.hadoopConfiguration().set("fs.s3a.awsSecretAccessKey","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")

**4.Reading from s3a**

df=spark.read.csv(“s3a://lti871/ayrin/supplier\_0\_0\_0.csv)

From pyspark.sql.types import \*

nationSchema = StructType([ \

StructField('nationkey', IntegerType(), True), \

StructField('name', StringType(), True),\

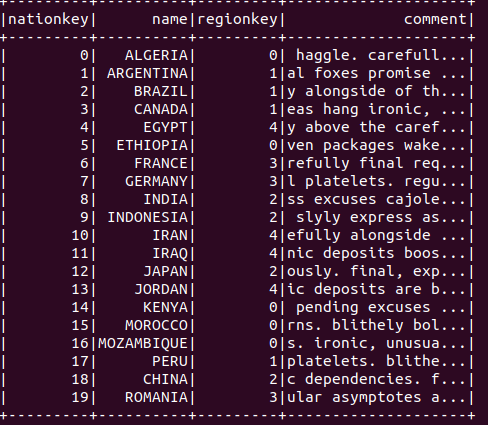
StructField('regionkey', IntegerType(), True),\

StructField('comment', StringType(), True) \

])

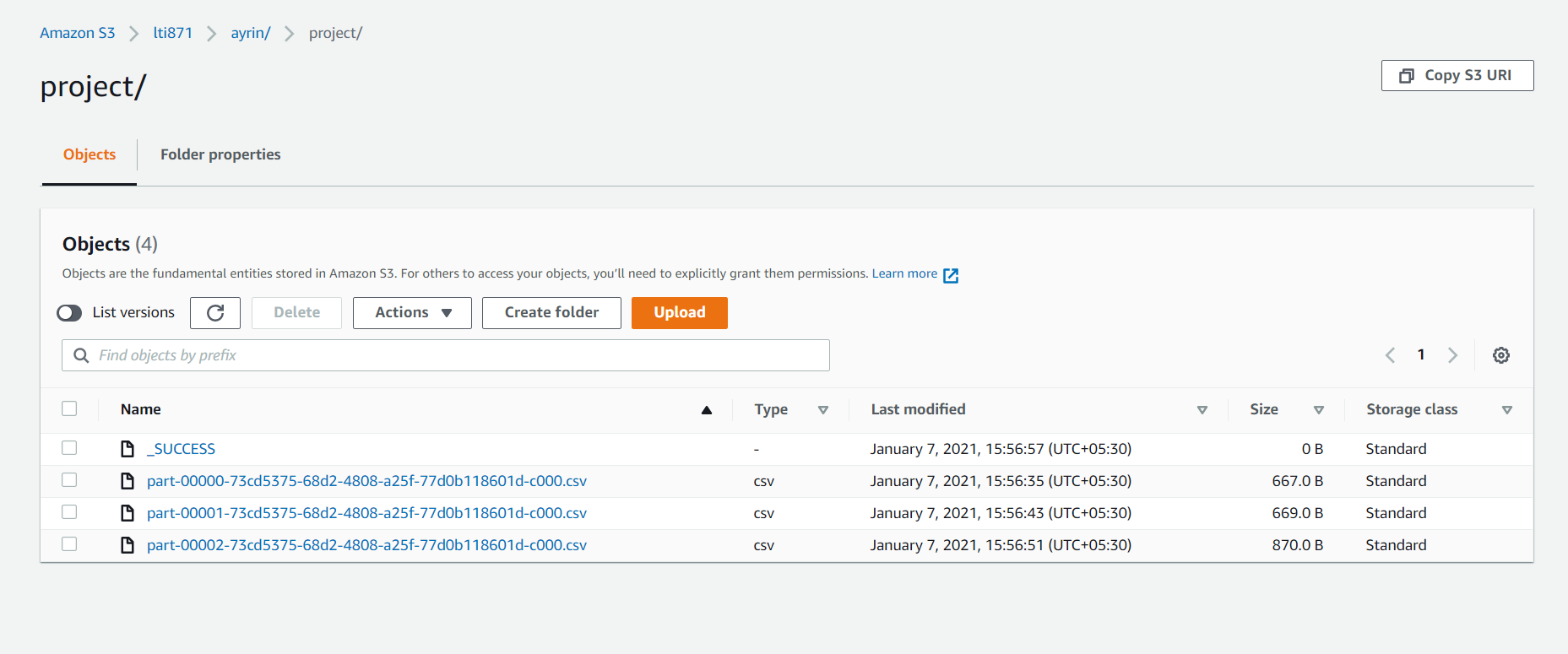
df=df.withColumn("\_c0",col("\_c0").cast(IntegerType())).withColumn("\_c2",col("\_c2").cast(IntegerType()))

df2=spark.createDataFrame(df.collect(),nationSchema)



**Writing to s3**

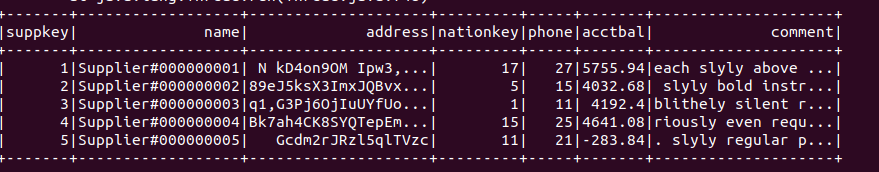
df2.write.csv("s3a://lti871/ayrin/project")



**Creating External table on top of S3**

spark.sql(“CREATE EXTERNAL TABLE IF NOT EXISTS suppliers3 (suppkey int,name string,address string,nationkey int,phone int,acctbal double,comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location 's3a://lti871/ayrin/supplier'”)

spark.sql(“SELECT \* FROM suppliers3 limit 5”)



**2.Redshift-Spark Integration**

**1.Importing the jars and packages in pyspark**

pyspark --packages com.databricks:spark-redshift\_2.10:0.5.0 --jars / home/ak/redshift-jdbc42-1.2.1.1001.jar,/home/ak/.ivy2/jars/com.amazonaws\_aws-java-sdk-1.7.4.jar,/home/ak/.ivy2/jars/org.apache.hadoop\_hadoop-aws-2.7.3.jar

**2.Reading in Spark from Redshift**

df=spark.read.options(url='jdbc:redshift://ltiredshift.cs17hobzoa0z.us-west-1.redshift.amazonaws.com:5439/dev',driver = 'com.amazon.redshift.jdbc42.Driver', user = 'awsuser', password = 'Pa55word', dbtable = 'lti\_schema.supplier').format('jdbc').load()

**3.Snowflake - Spark Integration**

**Snowflake**

**Features**

* Hybrid Architecture
* Columnar Storage
* Secure Platform

**1.Using Snowflake in CLI**

snowsql -c manan

use database lti\_db;

use schema ltischema;

**2.Creating Stage**

CREATE STAGE LTISCHEMA.partstage;

**3.Using SnowSQL**

**-----put data from local into stage**

Put file://C:\USERS\91836\Desktop\part-m\*.csv @partstage auto\_compress=false;

**4.Create file format to load the data from stage to table**

CREATE FILE FORMAT LTI\_DB.LTISCHEMA.partfileformat TYPE = 'CSV' COMPRESSION = 'AUTO' FIELD\_DELIMITER = '|' RECORD\_DELIMITER = '\n' SKIP\_HEADER = 1 ;

**5.Creating Table in Snowflake**

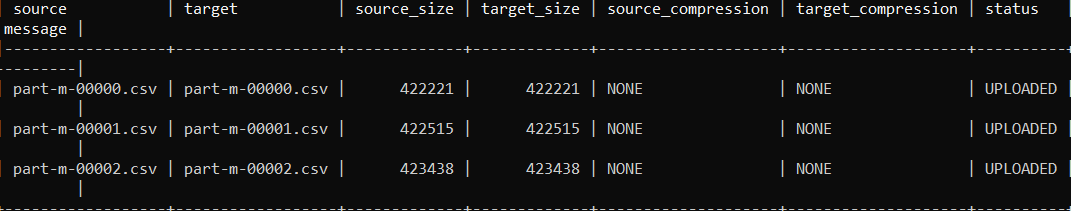
Create table Supplier(suppkey number PRIMARY KEY,name Varchar(40),Address varchar(160),Nationkey number ,Phone number,Acctbal number(11,2),Comment varchar(180));

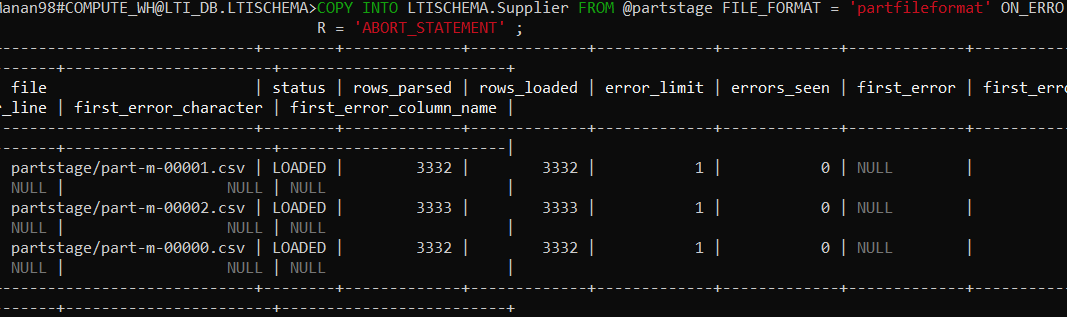
**6.Loading data from stage to table**

COPY INTO LTI\_DB.LTISCHEMA.PART FROM @partstage FILE\_FORMAT = 'LTI\_DB.partfileformat' ON\_ERROR = 'ABORT\_STATEMENT' ;

List @partstage;

Put file://C:\USERS\91836\Desktop\part-m\*.csv @partstage auto\_compress=false;





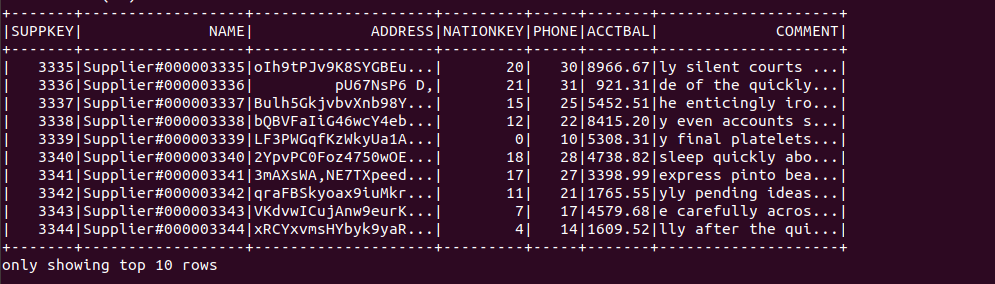
**7.Importing jars and packages in Pyspark**

pyspark --packages net.snowflake:spark-snowflake\_2.11:2.5.1-spark\_2.3

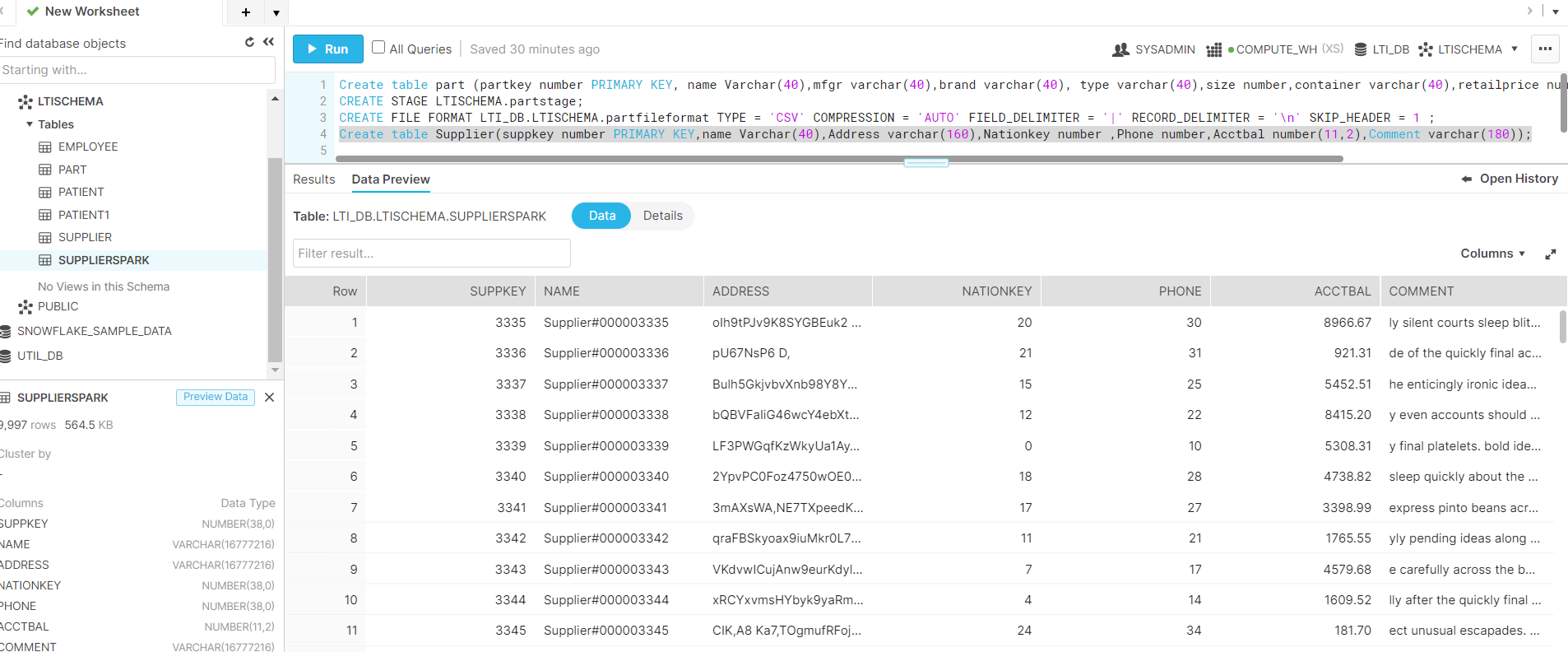
**8.To write into snowflake**

sfOptions = {"sfURL" : "upa33798.us-east-1.snowflakecomputing.com","sfUser" : "Manan98","sfPassword" : "\*\*\*\*\*\*\*\*\*\*","sfDatabase" : "LTI\_DB","sfSchema" : "LTISCHEMA","sfWarehouse" : "COMPUTE\_WH"}

df=spark.read.format("net.snowflake.spark.snowflake").options(\*\*sfOptions).option("dbtable","supplier").load()



df.write.format("net.snowflake.spark.snowflake").options(\*\*sfOptions).option("dbtable", "supplierspark").save()



**4.Aws Lambda function to write the processed spark dataframe output file stored in s3 bucket into Dynamodb table**

**AWS Lambda**

AWS Lambda is a serverless compute service that lets you run code without provisioning or managing servers, creating workload-aware cluster scaling logic, maintaining event integrations, or managing runtimes. With Lambda, you can run code for virtually any type of application or backend service - all with zero administration.

**Application**

import boto3

from datetime import datetime

print('Loading function')

s3 = boto3.client('s3')

def lambda\_handler(event, context):

#print("Received event: " + json.dumps(event, indent=2))

# Get the object from the event and show its content type

bucket = event['Records'][0]['s3']['bucket']['name']

key = urllib.parse.unquote\_plus(event['Records'][0]['s3']['object']['key'], encoding='utf-8')

try:

response = s3.get\_object(Bucket=bucket, Key=key)

data = response['Body'].read().decode("utf-8").strip()

print(data)

lines = data.split('\n')

row=[]

for i in range(len(lines)):

z = lines[i].split("|")

row.append(z)

print(row)

dynamo\_db = boto3.resource('dynamodb')

dynamoTable = dynamo\_db.Table('retail')

for i in range(len(y)):

dynamoTable.put\_item(Item = {

'Suppkey':int(row[i][0]),

'Name':int(row[i][1]),

‘Address’:str(row[i][2]),

‘NationKey’:int(row[i][3]),

‘Phone’:str(row[i][4]),

‘AcctBal’:int(row[i][5])

})

return response['ContentType']

except Exception as e:

print(e)

print('Error getting object {} from bucket {}. Make sure they exist and your bucket is in the same region as this function.'.format(key, bucket))

raise e

In the event section type the below code

Event: Amazon S3 put

{

"Records": [

{

"eventVersion": "2.0",

"eventSource": "aws:s3",

"awsRegion": "us-east-1",

"eventTime": "1970-01-01T00:00:00.000Z",

"eventName": "ObjectCreated:Put",

"userIdentity": {

"principalId": "EXAMPLE"

},

"requestParameters": {

"sourceIPAddress": "127.0.0.1"

},

"responseElements": {

"x-amz-request-id": "EXAMPLE123456789",

"x-amz-id-2": "EXAMPLE123/5678abcdefghijklambdaisawesome/mnopqrstuvwxyzABCDEFGH"

},

"s3": {

"s3SchemaVersion": "1.0",

"configurationId": "testConfigRule",

"bucket": {

"name": "lti871",

"ownerIdentity": {

"principalId": "EXAMPLE"

},

"arn": "arn:aws:s3:::example-bucket"

},

"object": {

"key": "ayrin/suppleir.csv",

"size": 1024,

"eTag": "0123456789abcdef0123456789abcdef",

"sequencer": "0A1B2C3D4E5F678901"

}

}

}

]

}

**5.Deplying Spark Jobs on EMR cluster**

**1.Deploying in Client mode**

Spark-submit --master spark://<IP to master node of EMR>:7077 --deploy-mode client --executor-memory client --executor-memory 20G --total-executor-cores 100 /path/to/spark.py

**2.Deploying in yarn cluster**

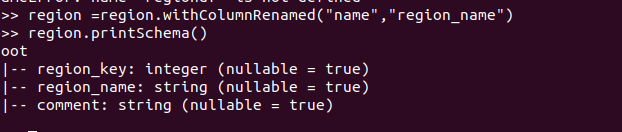
Spark-submit --master yarn --deploy-mode cluster --executor-memory 20G --num-executors 50 /path/to/project.py

**Phase 6**

**Business Queries**

**Q1. Replace column name ‘name’ with ‘region\_name’ for regiondf.**

regiondf =regiondf.withColumnRenamed("name","region\_name")

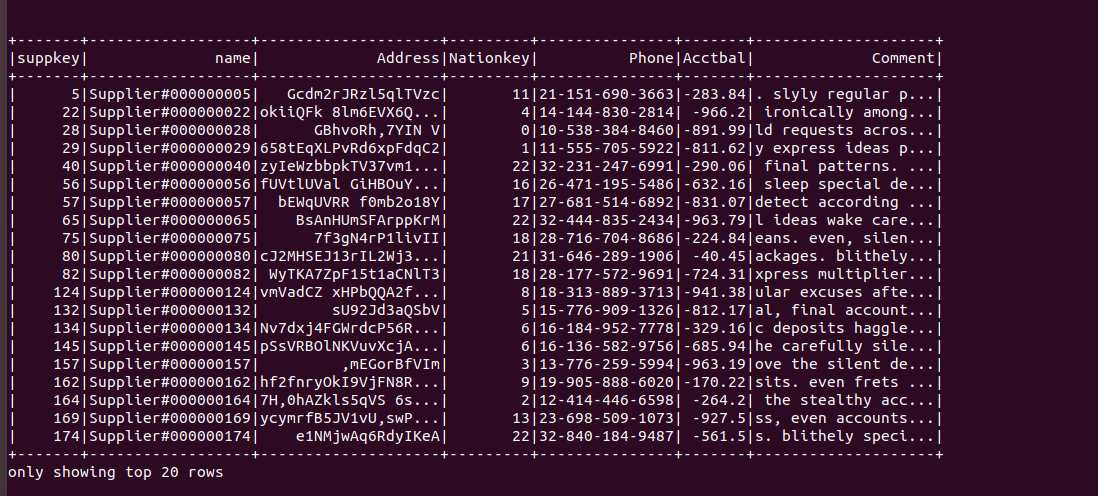


**Q2)The Account Team wants to know the records of Suppliers With Negative Account Balance**.

supplierdf.filter(supplierdf["acctbal"]<0).show()

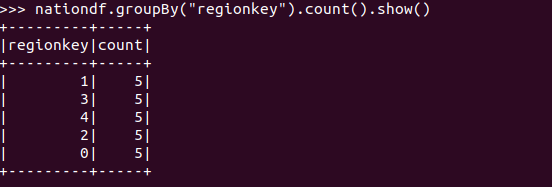
supplierdf.where(col("acctbal")<0).show()

spark.sql("SELECT \* FROM supp where acctbal <0").show()



**Q3) The CEO wants to know Count of Nations in a specific Region where our Retail Company is Operating.**

nationdf.groupBy("regionkey").count().show()

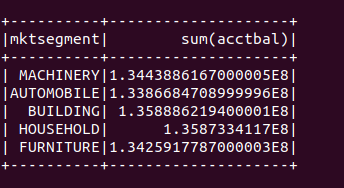


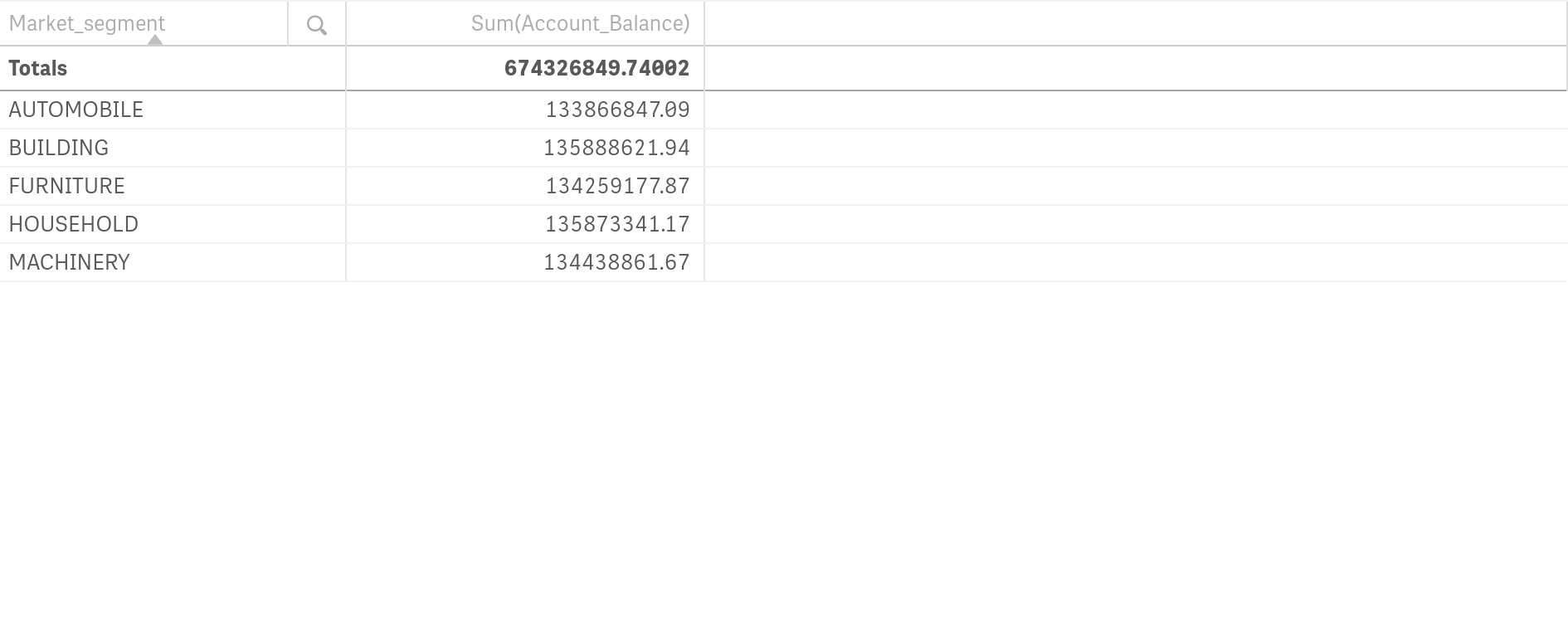
**Q4)Calculate the account balance for each market segment.**

Select sum(acctbal),mktSegment from customerext groupby mktsegment;

customerdfdf.groupBy("mktsegment").count().show()

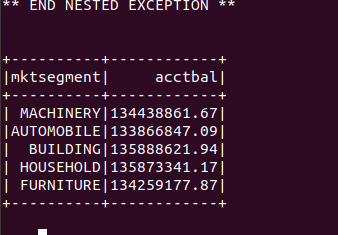
customerdf.groupBy("mktsegment").agg({"acctbal":”sum"})





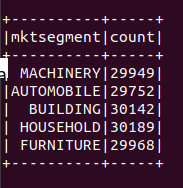
custdf =df.withColumnRenamed("sum(acctbal)","acctbal")

df=custdf.withColumn('acctbal',custdf.acctbal.cast(DecimalType(38,2)))



**5)The Retail Company wants to know the Count of customers in each market segment for audit.**

customerdf.groupBy("mktsegment").agg(f.count('custkey').alias("count")).show()



**6) Business Question**

**The Pricing Summary Report Query provides a summary pricing report for all line items shipped as of a given date. The date is within 60-120 days of the greatest ship date contained in the database.The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of line items in each group is included**

from pyspark.sql.functions import col,lit

From pyspark.sql.functions import udf

From pyspark.sql.functions

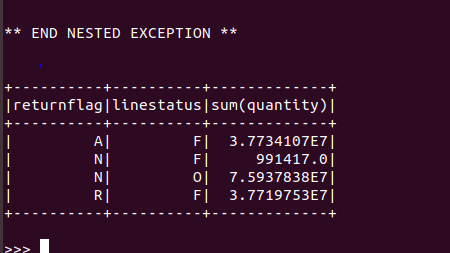
solndf=lineitemdf.where(col("shipdate") < '1998-10-12')

Decrease = udf(lambda a,b: a\*(1-b))

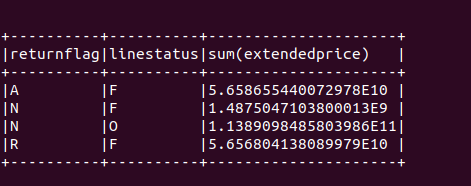
solndf=solndf.groupBy("returnflag","linestatus").agg(F.sum("quantity"),F.sum("extendedprice"),F.sum(decrease("extendedprice","discount")),F.sum(increase(decrease("extendedprice","discount"),"tax")),F.avg("quantity"),F.avg("extendedprice"),F.avg("discount"),F.count("quantity")).sort("returnflag","linestatus")

FINAL SOLUTION FOR THE BUSINESS QUESTION

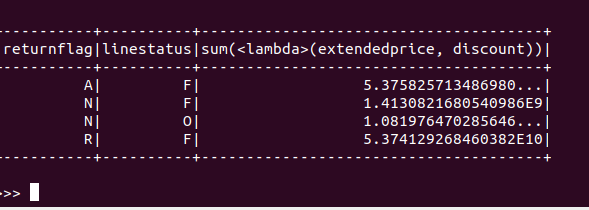
soln1=solndf.groupBy("returnflag","linestatus").agg(F.sum("quantity")).sort("returnflag","linestatus")



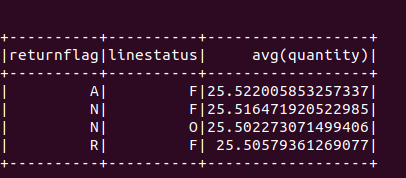
soln2=solndf.groupBy("returnflag","linestatus").agg(F.sum("extendedprice")).sort("returnflag","linestatus")



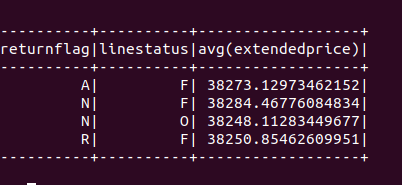
decrease = udf(lambda a,b: a\*(1-b),DoubleType())

soln3=solndf.groupBy("returnflag","linestatus").agg(F.sum(decrease("extendedprice","discount"))).sort("returnflag","linestatus")

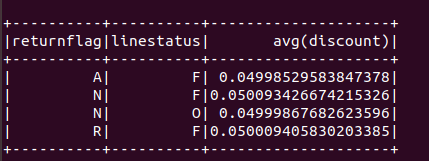
soln5=solndf.groupBy("returnflag","linestatus").agg(F.avg("quantity")).sort("returnflag","linestatus")



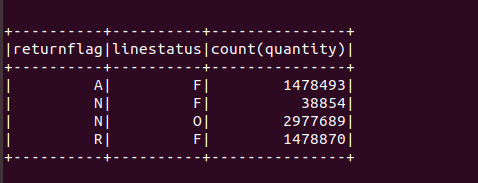
soln6=solndf.groupBy("returnflag","linestatus").agg(F.avg("extendedprice")).sort("returnflag","linestatus")



soln7=solndf.groupBy("returnflag","linestatus").agg(F.avg("discount")).sort("returnflag","linestatus")



soln8=solndf.groupBy("returnflag","linestatus").agg(F.count("quantity")).sort("returnflag","linestatus")



df = df1.join(df2, on=['id'], how='inner')

solndf1=soln1.join(soln2,on=['returnflag','linestatus'],how='inner')

solndf2=solndf1.join(soln3,on=['returnflag',’linestatus'],how='inner')

solndf3=solndf2.join(soln5,on=['returnflag',’linestatus'],how='inner')

solndf4=solndf3.join(soln6,on=['returnflag',’linestatus'],how='inner')

solndf5=solndf4.join(soln7,on=['returnflag',’linestatus'],how='inner')

solndf6=solndf5.join(soln8,on=['returnflag',’linestatus'],how='inner')

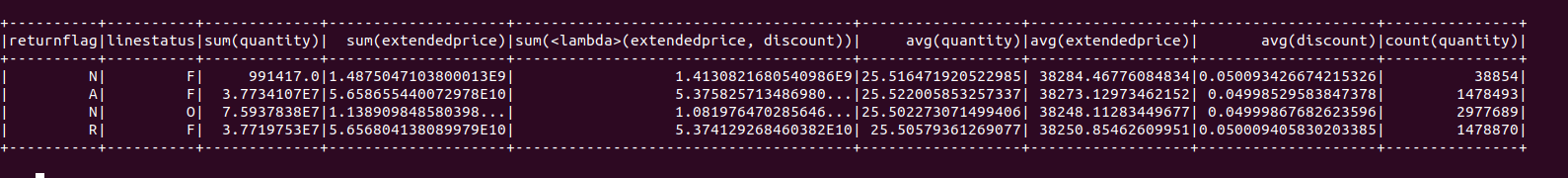
solndf7=solndf6.join(soln3,on=['returnflag','linestatus'],how='inner')

solndf8=solndf7.join(soln5,on=['returnflag','linestatus'],how='inner')

df=custdf.withColumn('acctbal',custdf.acctbal.cast(DecimalType(38,2)))

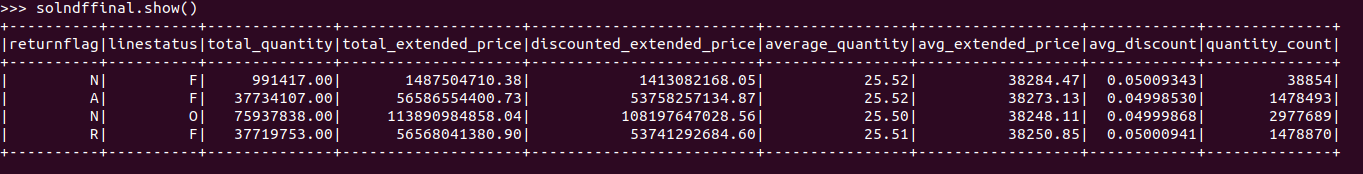
from pyspark.sql.types import \*

solndffinal = solndf.withColumn('total\_quantity',solndf.total\_quantity.cast(DecimalType(38,2))).withColumn('total\_extented\_price',solndf.total\_extended\_price.cast(DecimalType(38,2))).withColumn('discounted\_extented\_price',solndf.discounted\_extended\_price.cast(DecimalType(38,2))).withColumn('average\_quantity',solndf.average\_quantity.cast(DecimalType(38,2))).withColumn('avg\_extended\_price',solndf.avg\_extended\_price.cast(DecimalType(38,2))).withColumn('avg\_discount',solndf.avg\_discount.cast(DecimalType(38,3)))

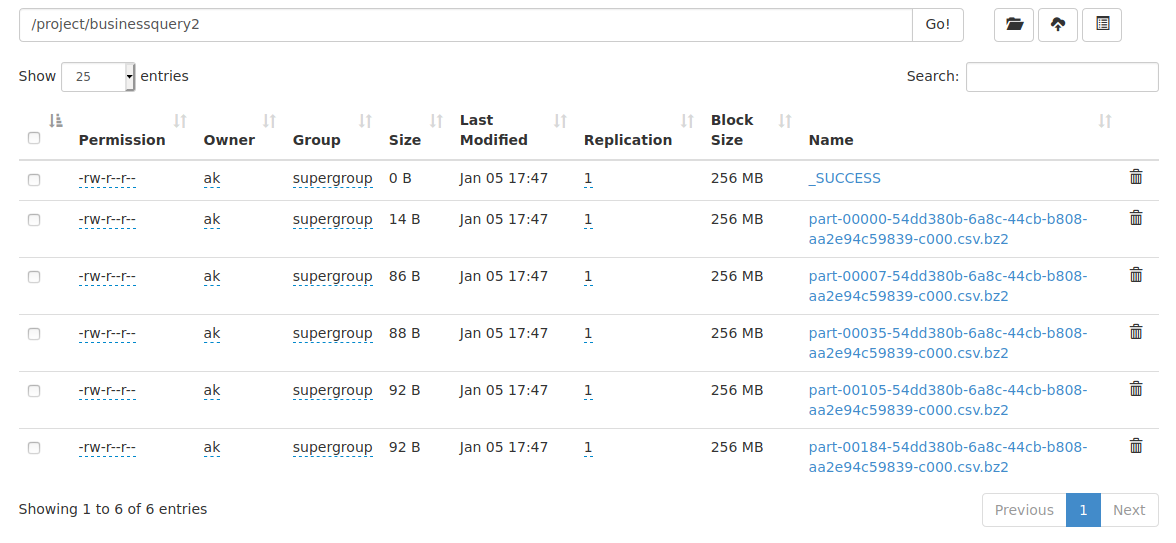


from pyspark.sql.types import StringType

>>> solndf=solndf6.select(col('returnflag'),col('linestatus'),col('sum(quantity)').alias('total\_quantity'),col('sum(extendedprice)').alias('total\_extended\_price'),col('sum(<lambda>(extendedprice, discount))').alias('discounted\_extended\_price'),col('avg(quantity)').alias('average\_quantity'),col('avg(extendedprice)').alias('avg\_extended\_price'),col('avg(discount)').alias('avg\_discount'),col('count(quantity)').alias('quantity\_count'))



solndffinal.write.option(“codec”,”bzip2”).parquet('/project/businessquery')

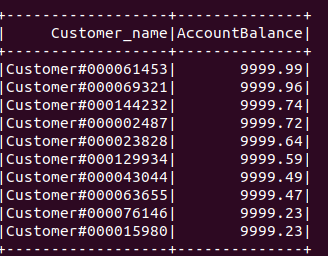




**Q7.Find the Customers Having top 10 Acct Balances.**

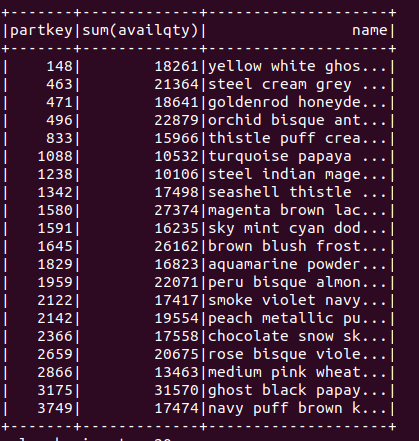
customer=customerdf.sort('acctbal',ascending=False).select(col('name').alias('Customer\_name'),col('acctbal').alias('AccountBalance'))

customer.show(10)



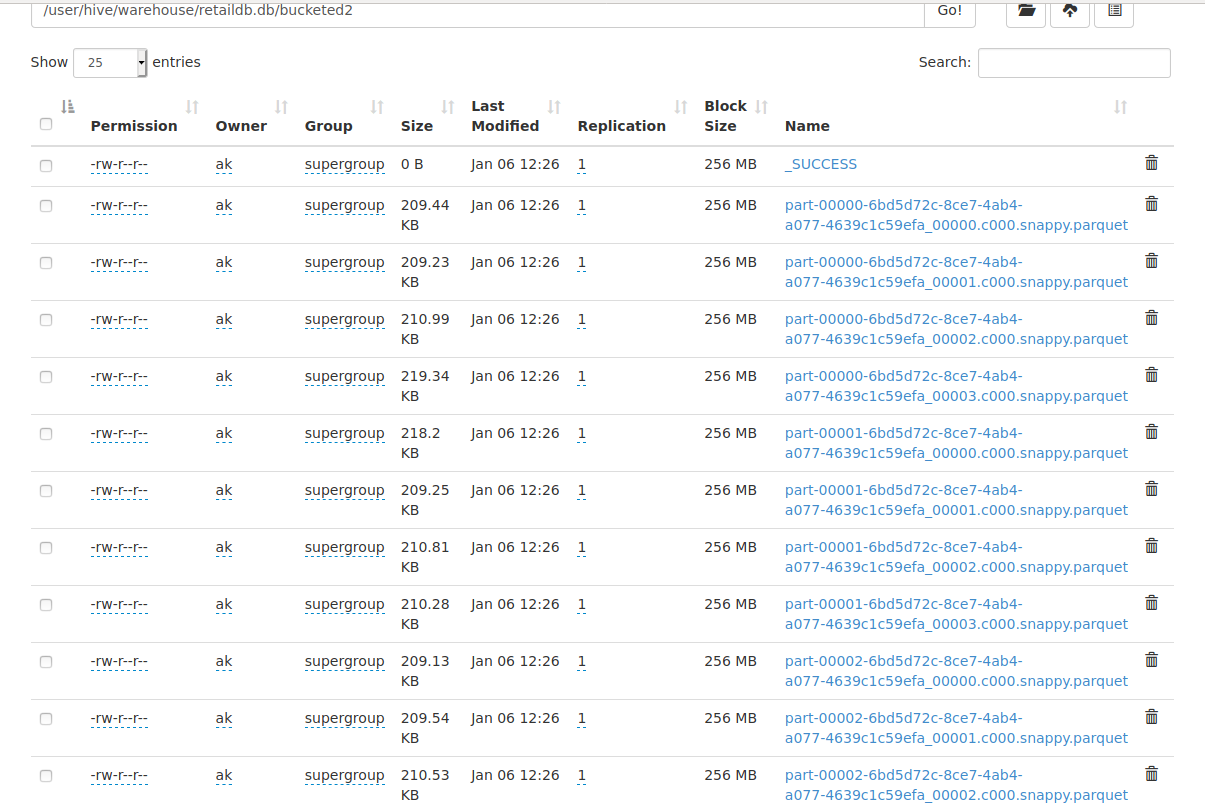
**Q8.The Goods Management Team wants to know the total quantity available per part and their names so that if stock is less they can order that Particular part.**

soln=partsupplydf.groupBy("partkey").agg({'availqty':'sum'}).join(partdf,on=['partkey'],how='left').select(col('partkey'),col('sum(availqty)'),col('name'))



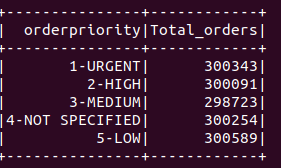
#using bucketby

soln.write.bucketBy(4,’partkey’).saveAsTable(‘bucketed’,format=’parquet’)

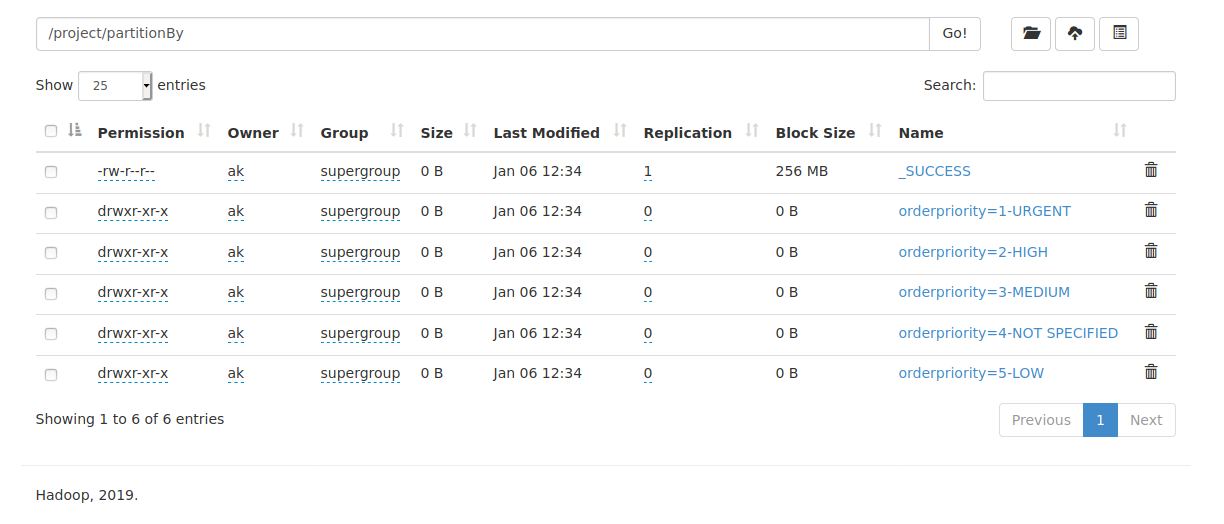


**Q9.The Delivery team wants to know the Count of Orders coming under different types of Order\_Priority so that they can deliver the orders on priority basis.**

order=ordersdf.groupBy("orderpriority").agg(F.count("orderkey").alias('Total\_orders')).sort("orderpriority")

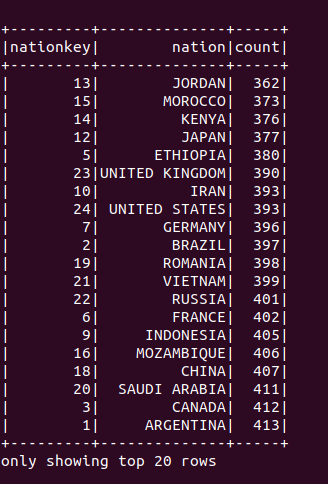


order.write.partitionBy("orderpriority").option('codec','gzip').csv('/project/partitionBy')



**Q10. Calculate the number of suppliers per nation.**

ans= supplierdf.groupBy("nationkey").agg({"suppkey":"count"}).join(nationdf,on=["nationkey"],how="left").select(col("nationkey"),col("name").alias("nation"),col("count(suppkey)").alias("count").orderBy("count",desc=1)

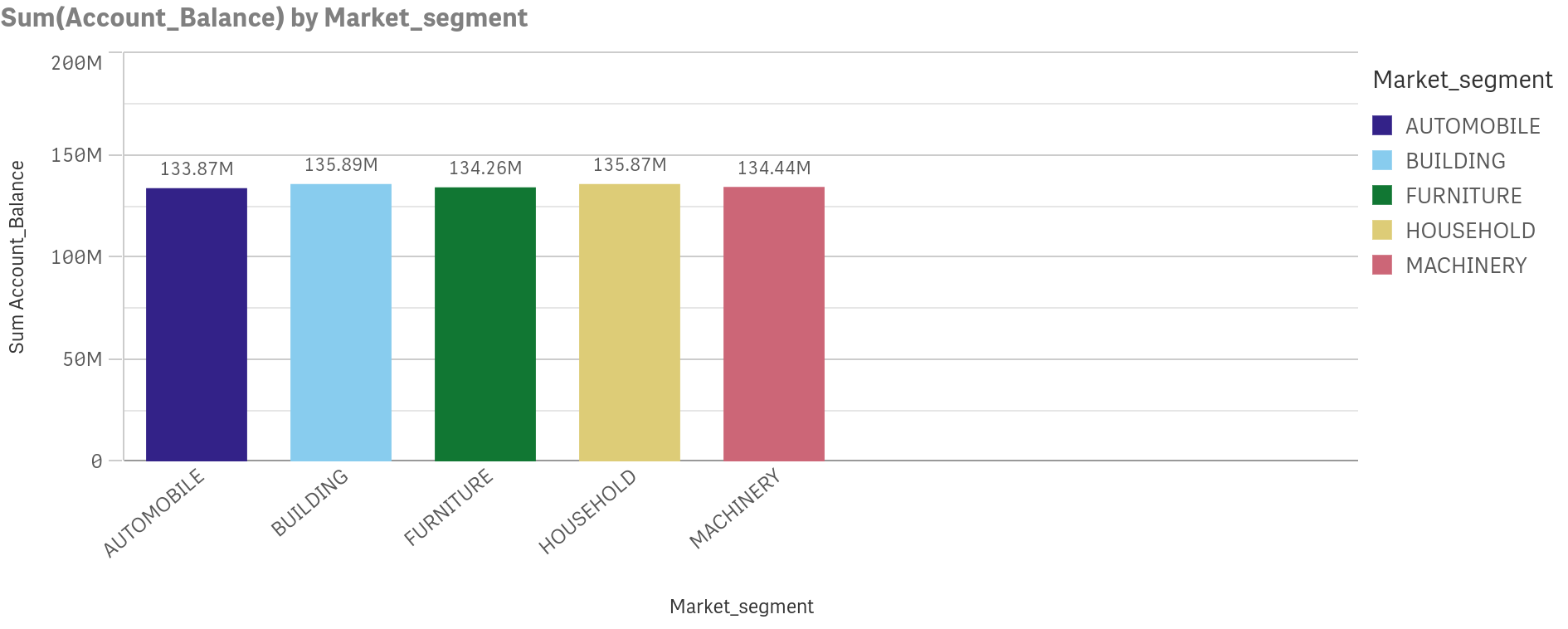


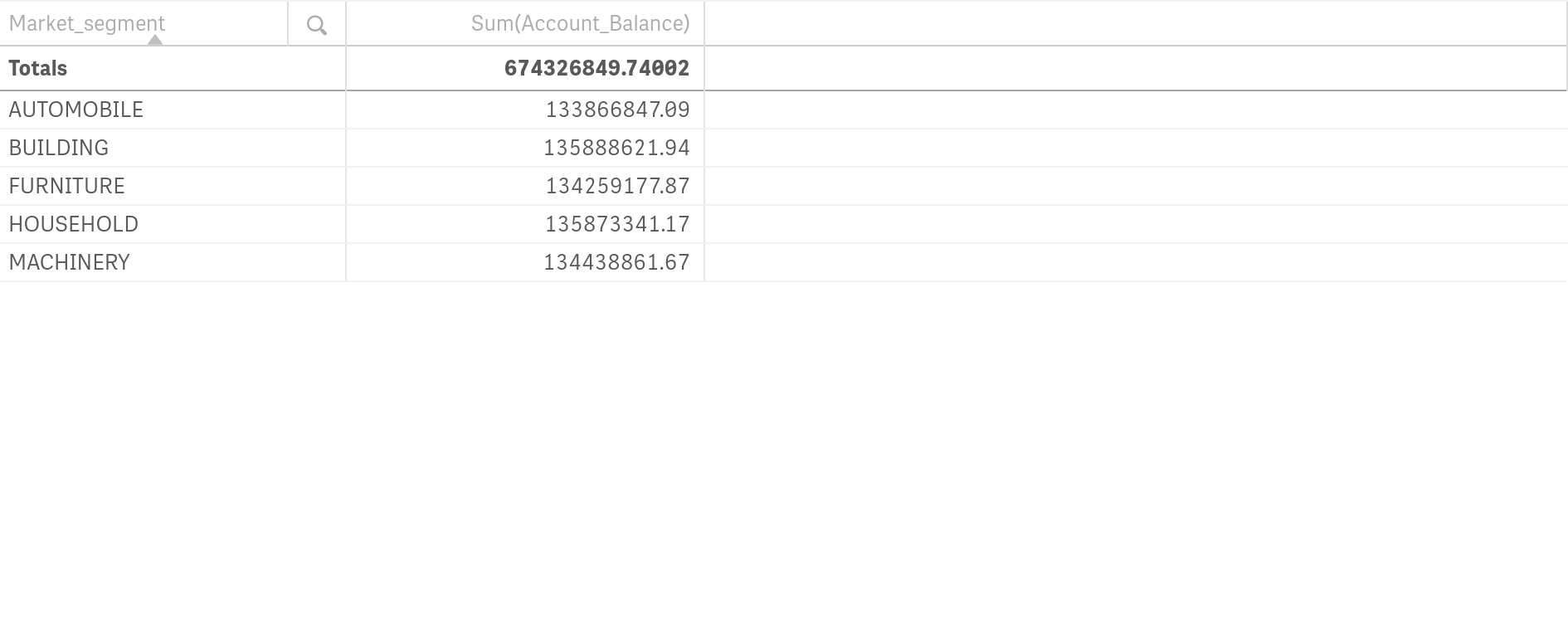
**Phase 7**

**Data Analysis**

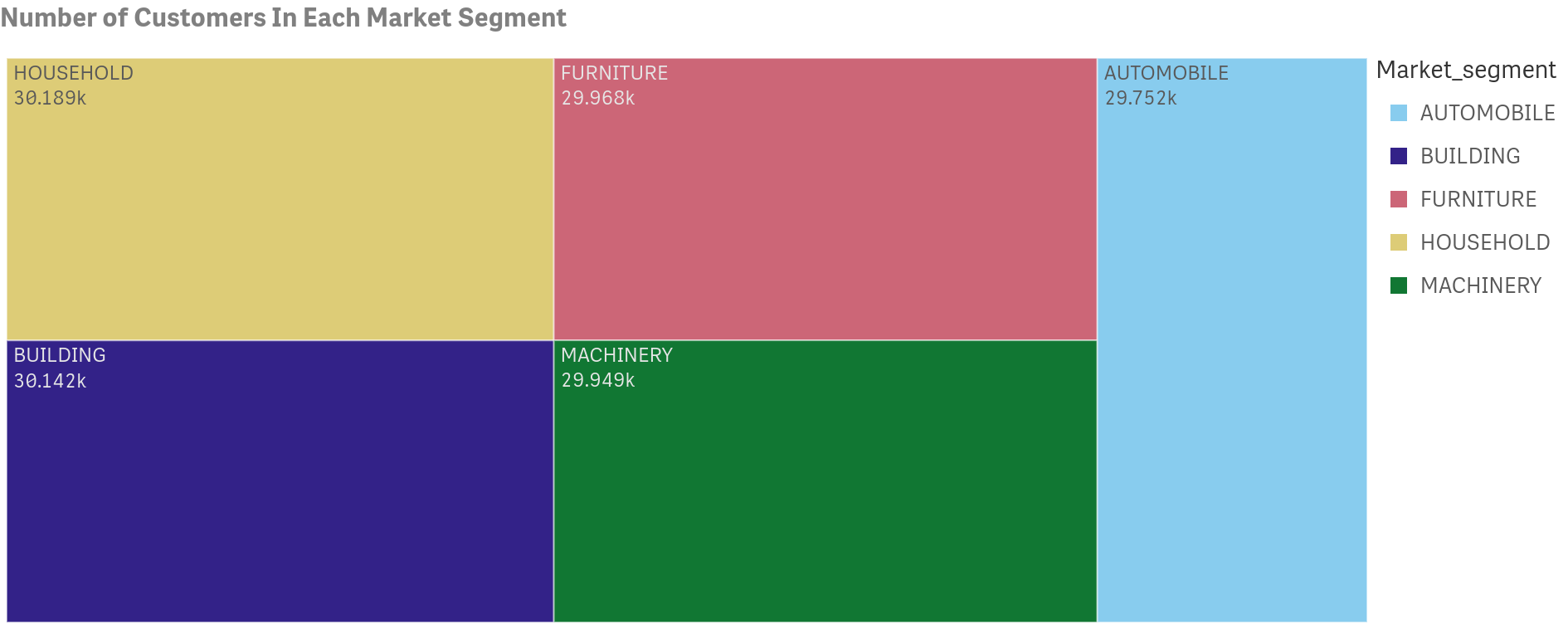
**Visualization:**

**Scenario 1: Calculate the Account Balance for each Market Segment.**

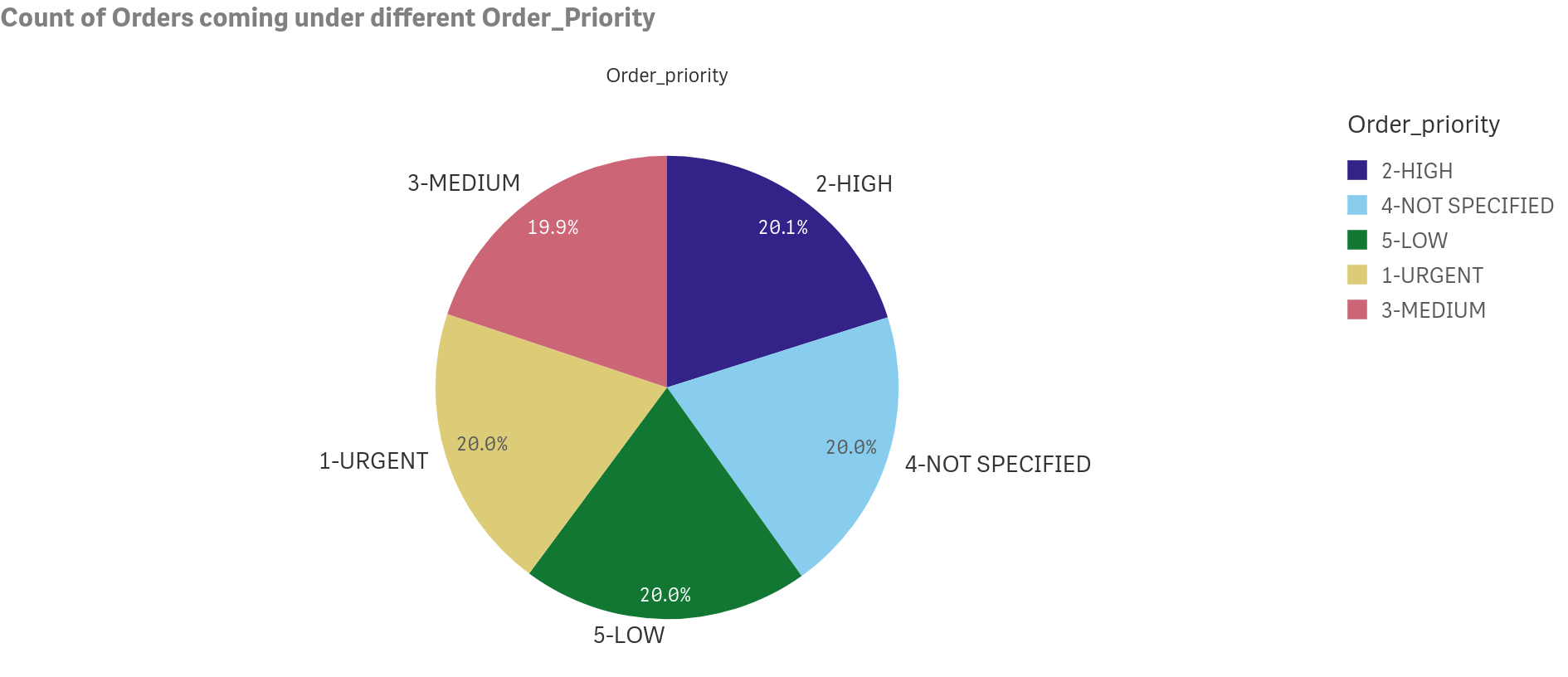




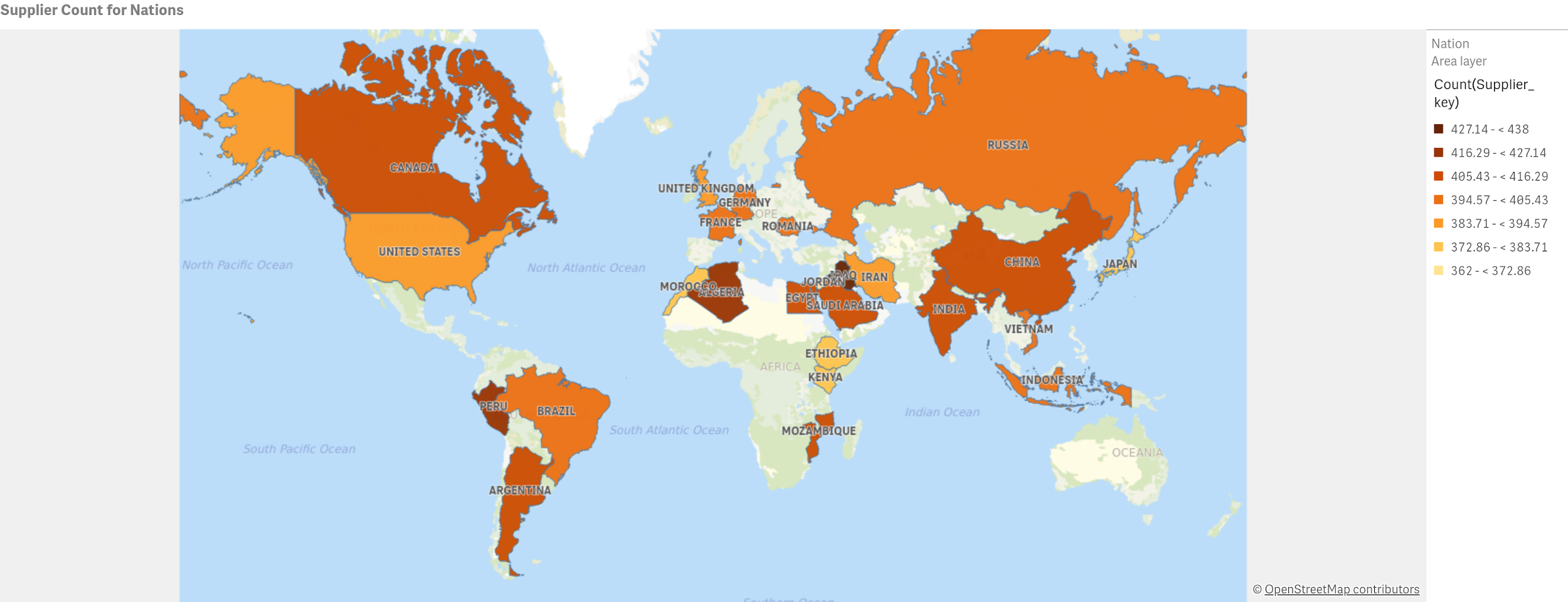
**Scenario 2: Retail Company wants to know the Number of customers present in Each Market Segment.**

****

**Scenario-3 :The Delivery team wants to know the Count of Orders coming under different types of Order\_Priority so that they can deliver the orders on priority basis.**

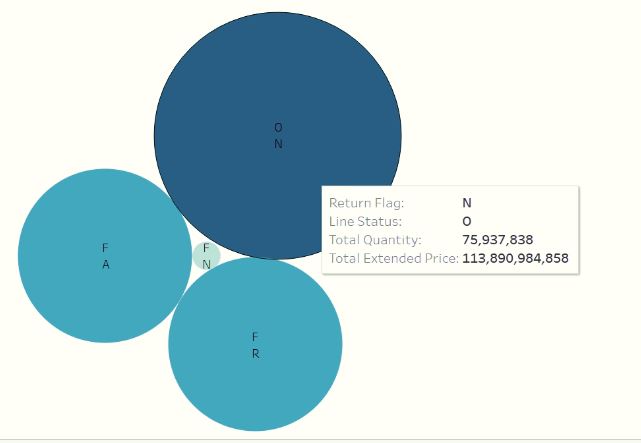


**Scenario 4:Display the number of suppliers per nation.**

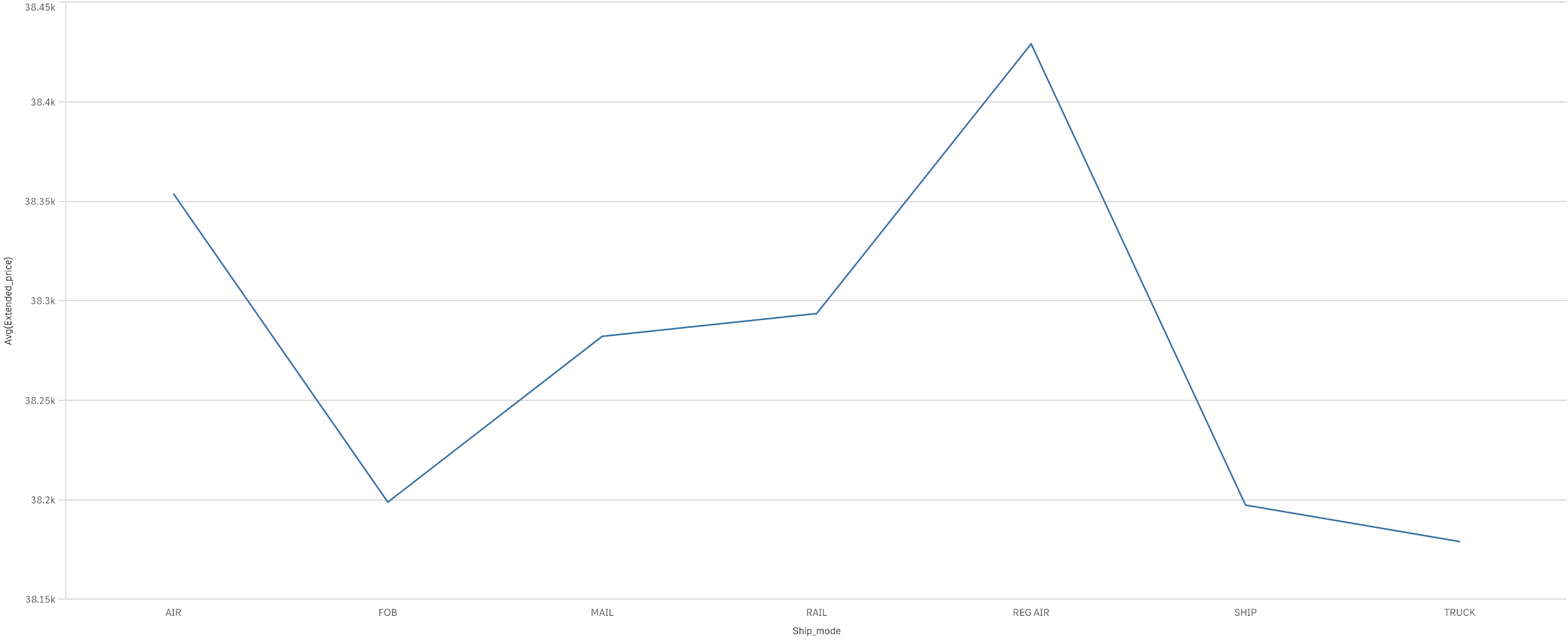


**Scenario 5: The Retail Company Wants to know the number of orders shipped in a calendar year.**

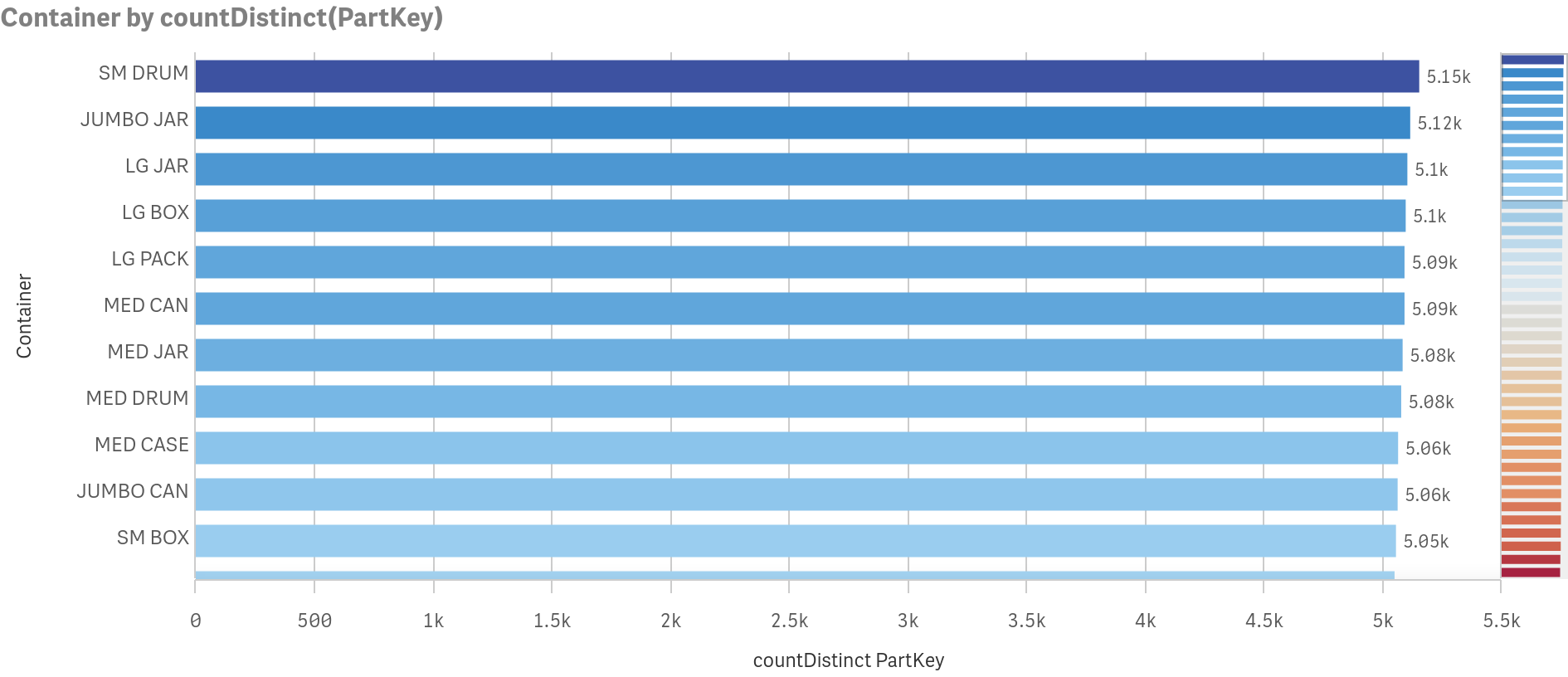
**Scenario 6: The Pricing Summary Report Query provides a summary pricing report for all line items shipped as of a given date. The date is within 60-120 days of the greatest ship date contained in the database.The query lists totals for extended price, discounted extended price, discounted extended price plus tax, average quantity, average extended price, and average discount. These aggregates are grouped by RETURNFLAG and LINESTATUS, and listed in ascending order of RETURNFLAG and LINESTATUS. A count of the number of line items in each group is included .**

****

**Scenario 7: The Shipping department at the retail company wants to know the breakdown of services of the different modes of shipment. Help him analyze the required data.**

****

**Scenario 8: The Shipping Company wants to know the most used Container for shipping the parts.**

****