**Snowflake command**

CREATE FILE FORMAT "LTI\_DB"."LTISCHEMA".FMT\_CSV\_GZIP

TYPE = 'CSV'

COMPRESSION = 'GZIP' FIELD\_DELIMITER = ','

RECORD\_DELIMITER = '\n' SKIP\_HEADER = 0

FIELD\_OPTIONALLY\_ENCLOSED\_BY = '\042' TRIM\_SPACE = TRUE

ERROR\_ON\_COLUMN\_COUNT\_MISMATCH = TRUE

ESCAPE = '\134' ESCAPE\_UNENCLOSED\_FIELD = '\134'

DATE\_FORMAT = 'AUTO' TIMESTAMP\_FORMAT = 'AUTO'

NULL\_IF = ('\\N');

create stage st\_unload3 file\_format =FMT\_CSV\_GZIP;

copy into @st\_unload3/lineitem from SNOWFLAKE\_SAMPLE\_DATA.TPCH\_SF1.lineitem;

**Mysql Commands**

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) ) ;

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));

Create table region(

region\_key int PRIMARY KEY,

name Varchar(40),

Comment Varchar(152));

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));

Create table partsupply(

partkey int references part(partkey),

suppkey int references supplier(suppkey),

availqty int,

supplycost float(12,2),

comment varchar(199));

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));

Create table nation(

Nationkey int primary key,

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

Regionkey int,

Comment varchar(120));

Create table orders(

Orderkey int primary key,

custkey int references customer(custkey),

Orderstatus varchar(1),

Totalprice float(12,2),

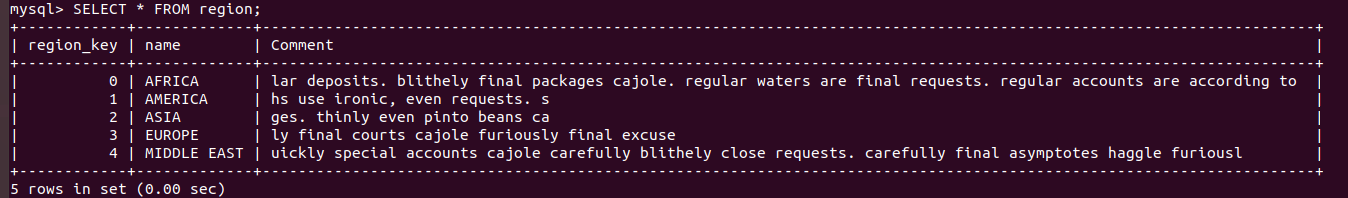
Order\_date date,

Orderpriority varchar(15),

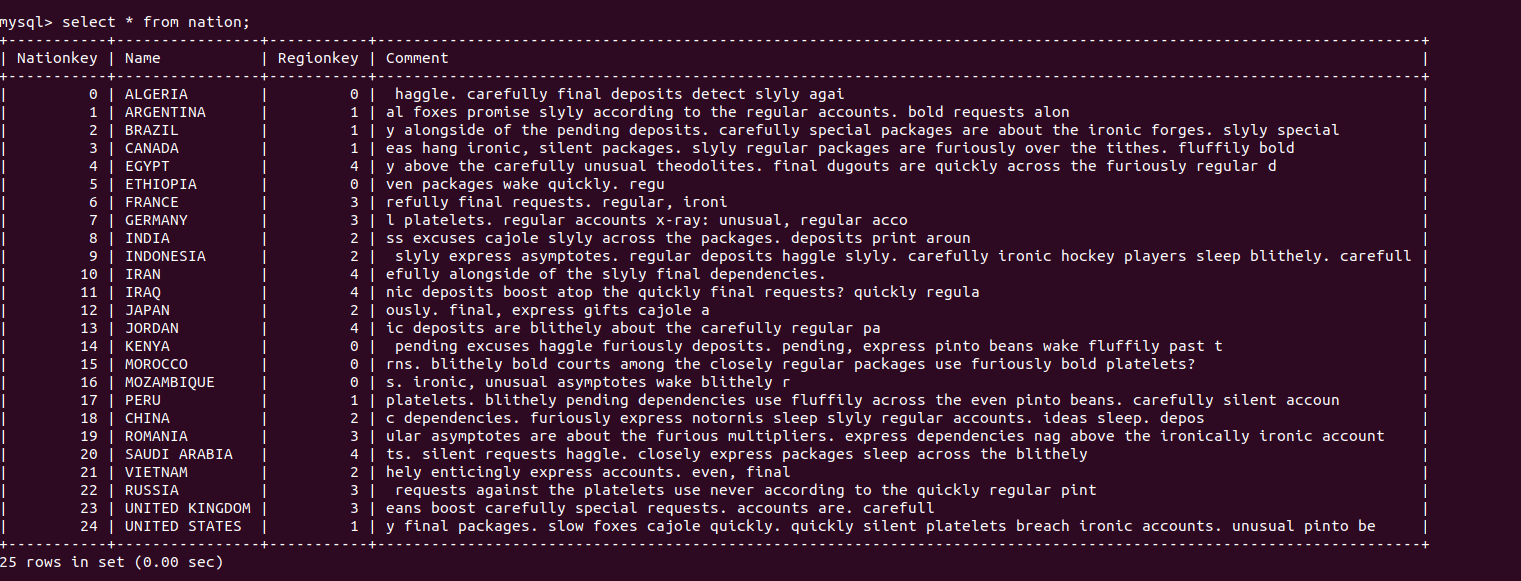
Clerk varchar(15),

Shippriority int,

Comment varchar(79));



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';



Warning when loaded data into part table

**SQOOP IMPORT**

**Managed Tables**

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

**1)region**

**2)part**

**3)partsupply**

**External Tables**

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

**1)Customer.**

**2)Orders**

**3)Line Items**

**4)Nation**

**5)Supplier**

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;

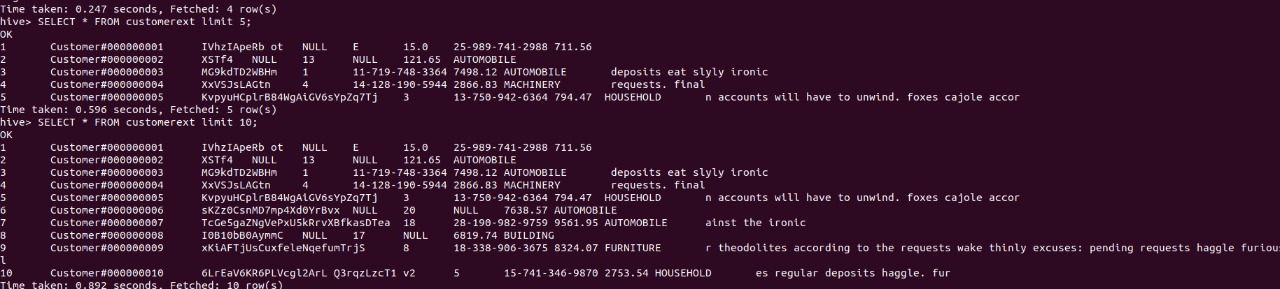
sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword --table region -create-hive-table -hive-table retaildb.region -hive-import --fields-terminated-by ',' -m 1;

#incremental load

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

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;

sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table nation -m 1 ;



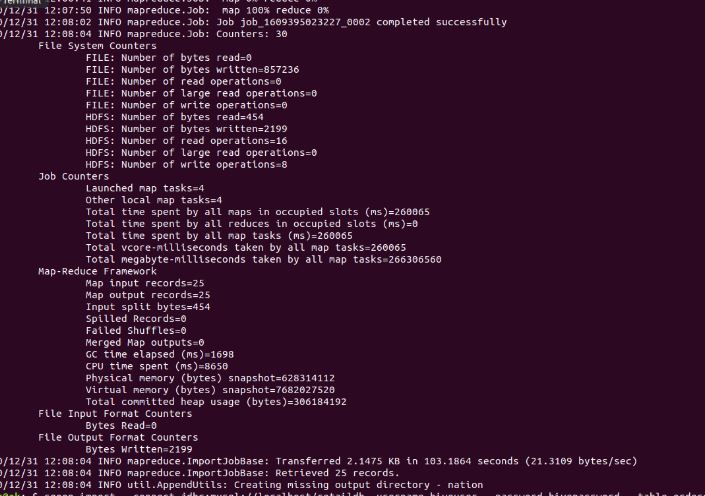
CREATE EXTERNAL TABLE IF NOT EXISTS customerext (

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’;

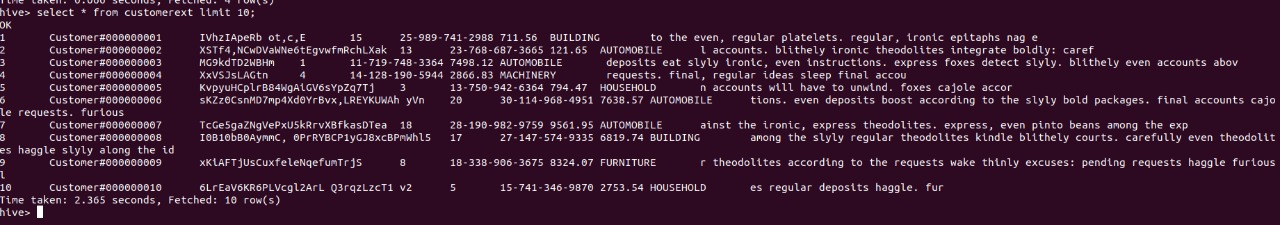
sqoop-import --connect jdbc:mysql://localhost/retaildb -username hiveuser -password hivepassword -table customer --fields-terminated-by ‘|’ --target-dir /project/customer --append --split-by custkey -m 3;

**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';

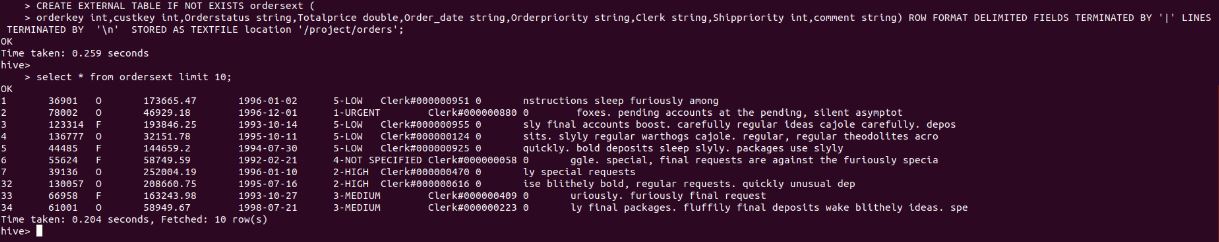
;



CREATE EXTERNAL TABLE IF NOT EXISTS ordersext (

orderkey int,custkey int,Orderstatus string,Totalprice double,Order\_date string,Orderpriority string,Clerk string,Shippriority int,comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location '/project/orders';

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

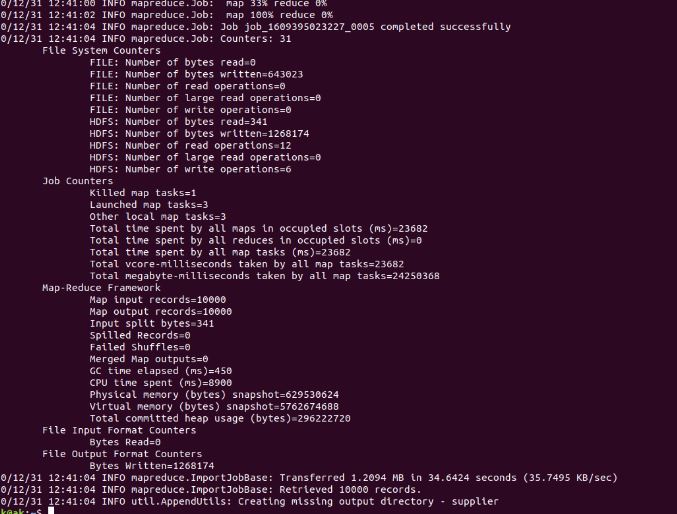
CREATE EXTERNAL TABLE IF NOT EXISTS lineitemext (

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';

CREATE EXTERNAL TABLE IF NOT EXISTS supplierext (

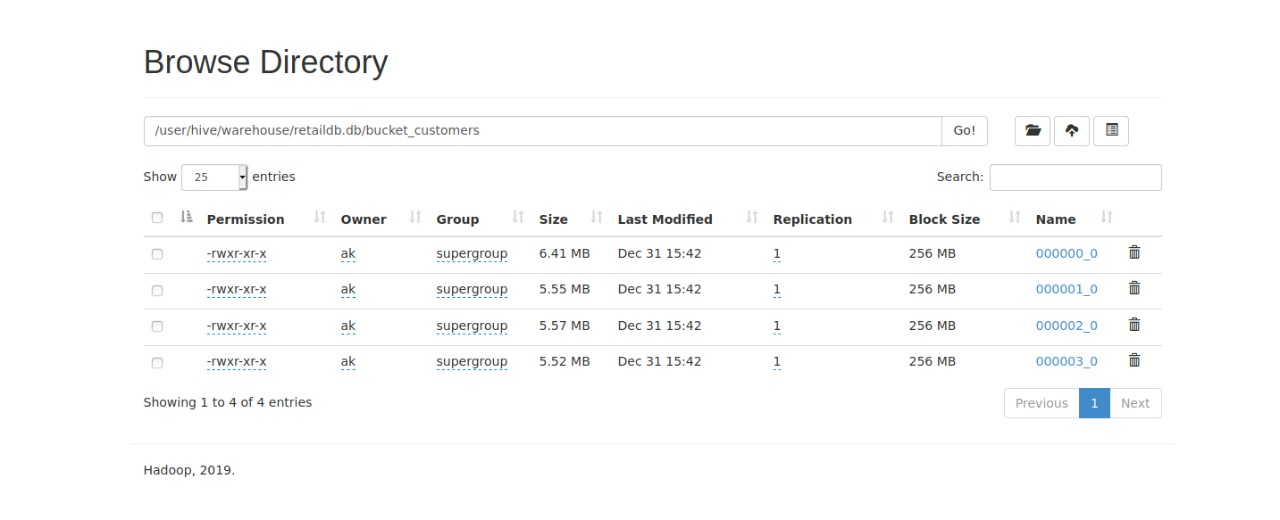
suppkey int,name string,address string,nationkey int,phone int,acctbal double,comment string) ROW FORMAT DELIMITED FIELDS TERMINATED BY '|' LINES TERMINATED BY '\n' STORED AS TEXTFILE location '/project/supplier';

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 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 ;

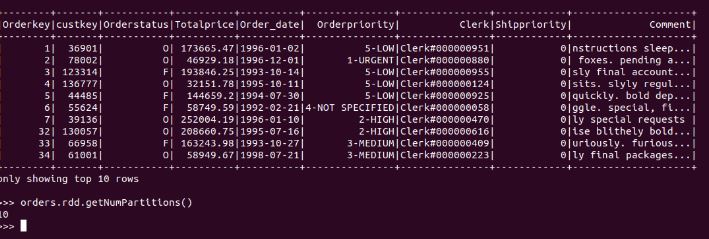
insert overwrite table bucket\_customers select \* from customerext;



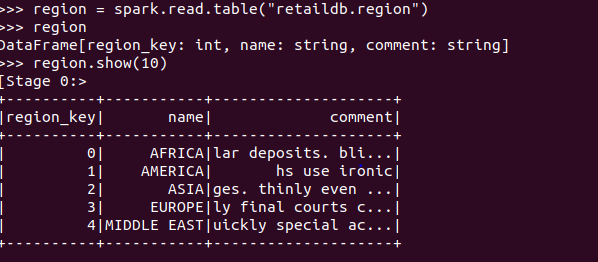
**===============SPARK===================**

---Loading Data Into Data Frames----

orders = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "orders").option("lowerBound", "0").option("upperBound", "1500000").option("numPartitions", " 10").option("partitionColumn","orderkey").option("user", "hiveuser").option("password", "hivepassword").load()

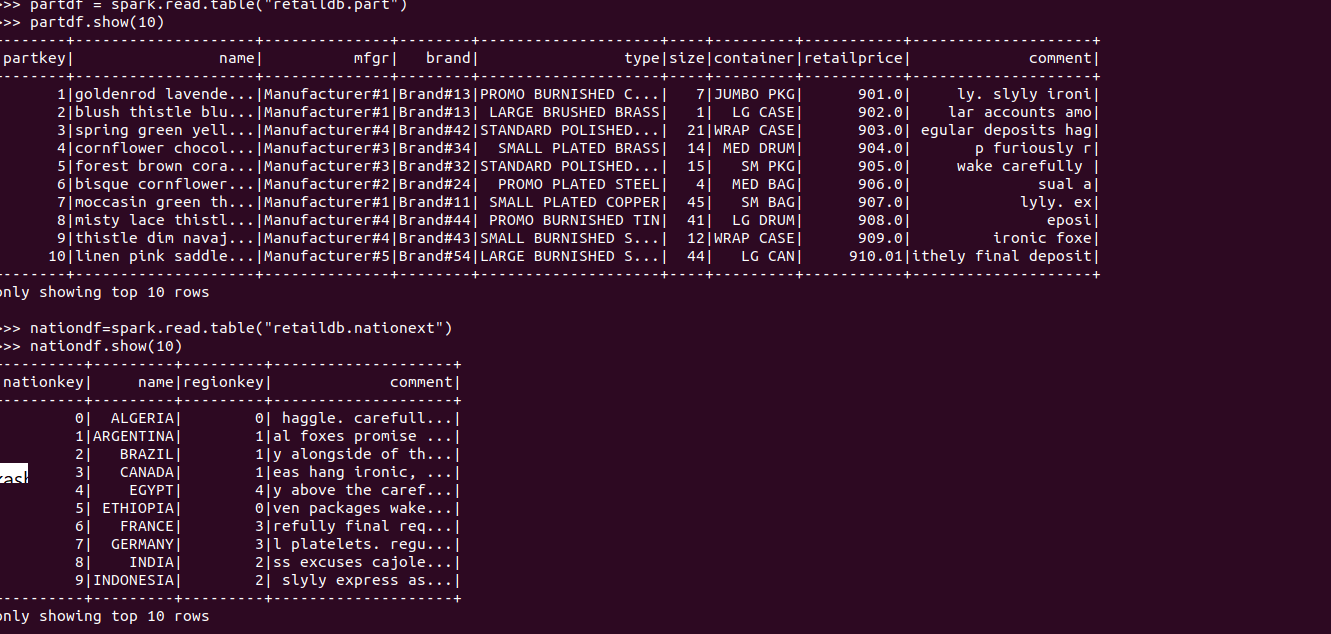


regiondf = spark.read.table("retaildb.region")

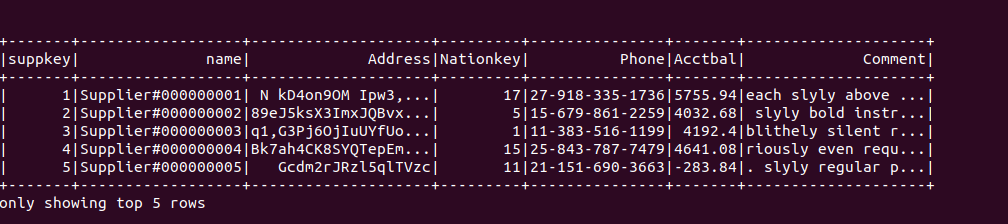


partdf = spark.read.table("retaildb.part")

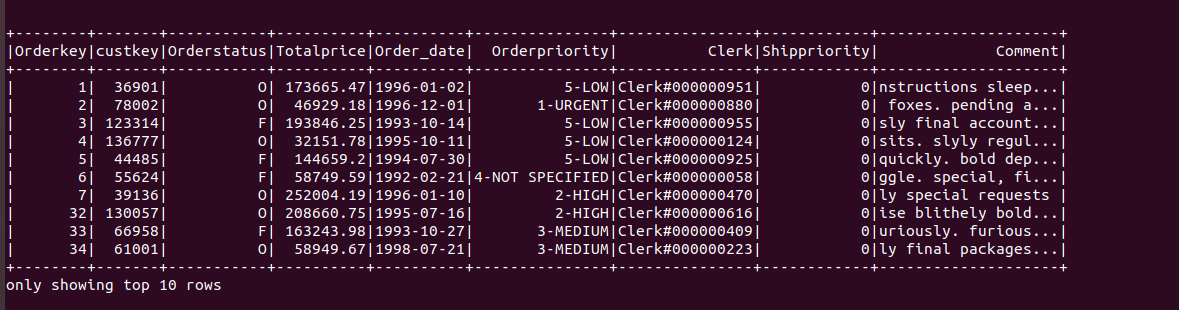
nationdf=spark.read.table("retaildb.nationext")



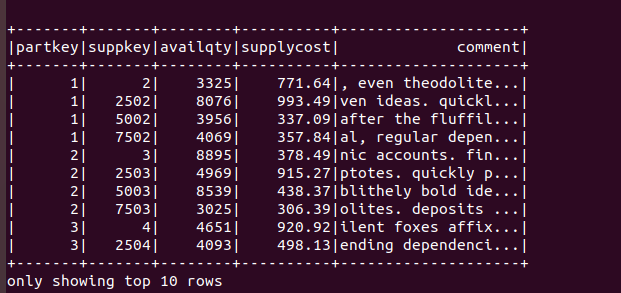
supplierdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "supplier").option("lowerBound", "0").option("upperBound", "10000").option("numPartitions", " 5").option("partitionColumn","suppkey").option("user", "hiveuser").option("password", "hivepassword").load()



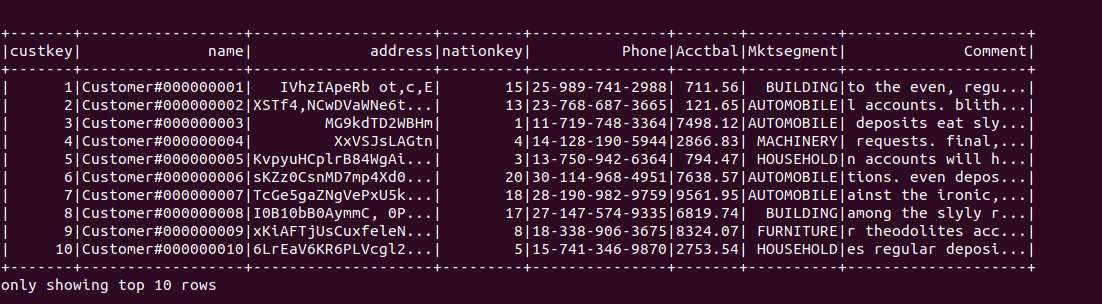
ordersdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "orders").option("lowerBound", "0").option("upperBound", "1500000").option("numPartitions", " 10").option("partitionColumn","orderkey").option("user", "hiveuser").option("password", "hivepassword").load()



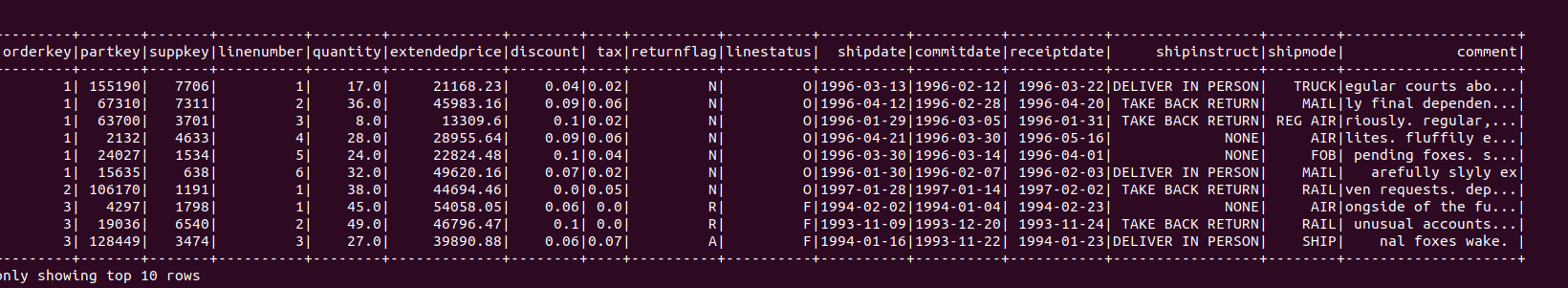
partsupplydf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "partsupply").option("lowerBound", "0").option("upperBound", "800000").option("numPartitions", “8").option("partitionColumn","partkey").option("user", "hiveuser").option("password", "hivepassword").load()



customerdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "customer").option("lowerBound", "0").option("upperBound", "150000").option("numPartitions", " 10").option("partitionColumn","custkey").option("user", "hiveuser").option("password", "hivepassword").load()

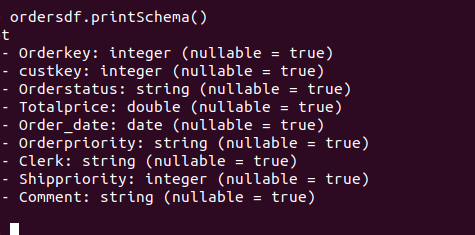


lineitemdf = spark.read.format("jdbc").option("url", "jdbc:mysql://localhost:3306/retaildb").option("driver", "com.mysql.jdbc.Driver").option("dbtable", "lineitem").option("lowerBound", "0").option("upperBound", "6001215").option("numPartitions", " 20").option("partitionColumn","orderkey").option("user", "hiveuser").option("password", "hivepassword").load()



====================================

**Original**

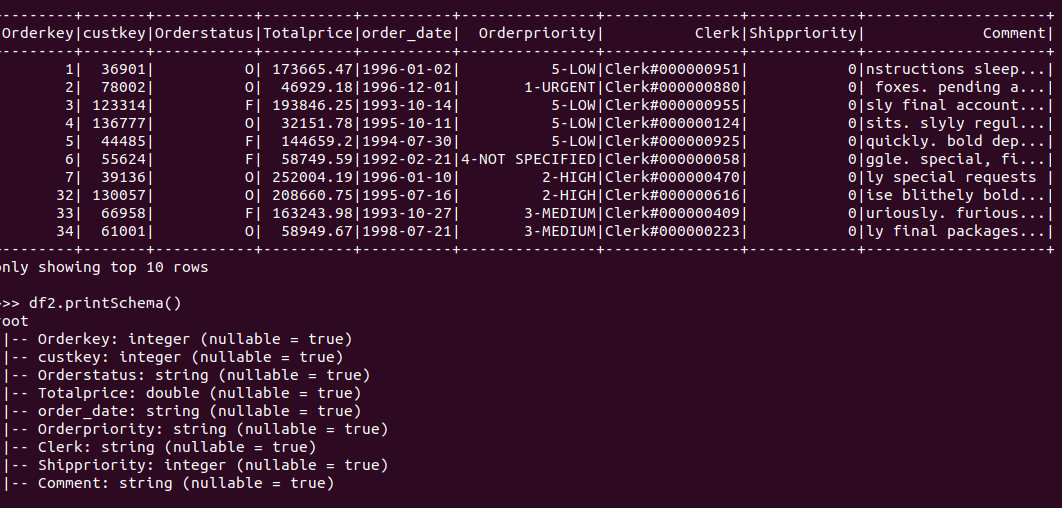


from pyspark.sql.functions import col

from pyspark.sql.types import StringType

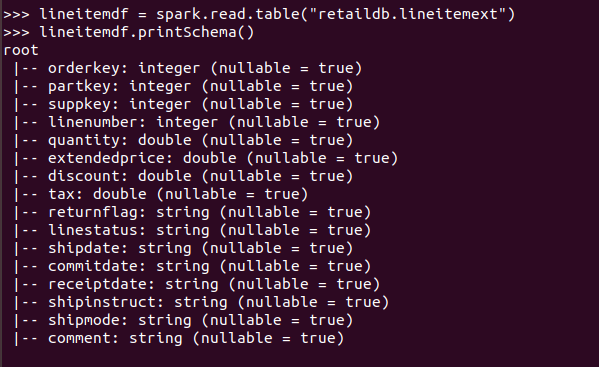
df2=ordersdf.withColumn("order\_date",col("order\_date").cast(StringType())

**After Type Casting**



**Type Casting for Line Item**

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

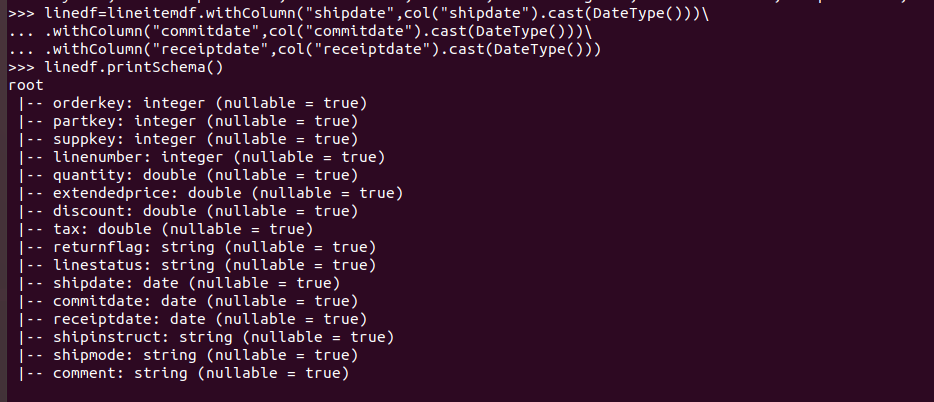


from pyspark.sql.types import \*

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

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

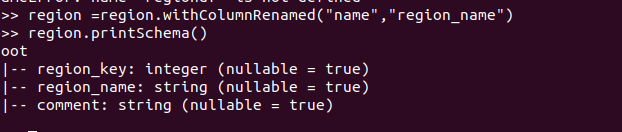
.withColumn("receiptdate",col("receiptdate").cast(DateType()))



TRANSFORMATIONS

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

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

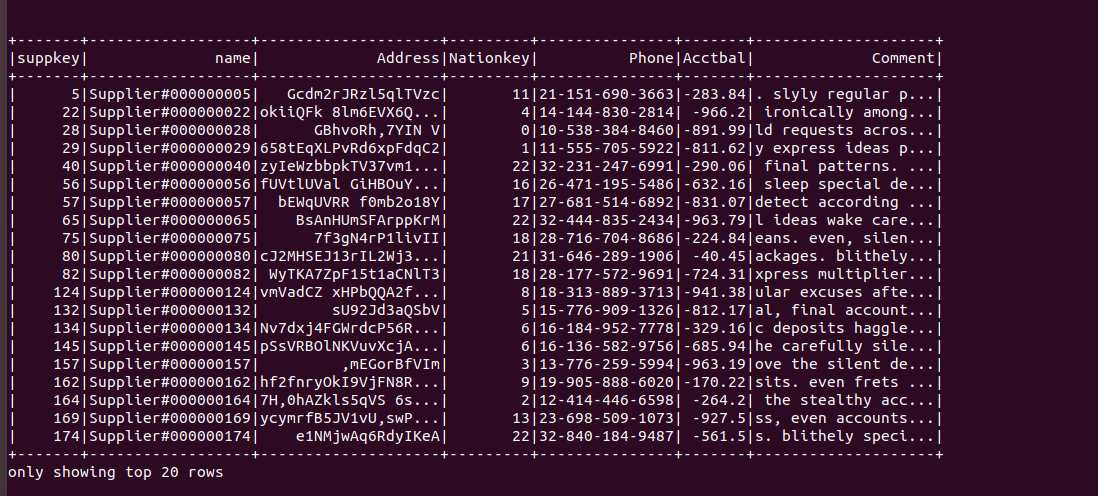


**Q2)The Account Team wants to know the records of Suppliers With Negative Account Balance.**

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

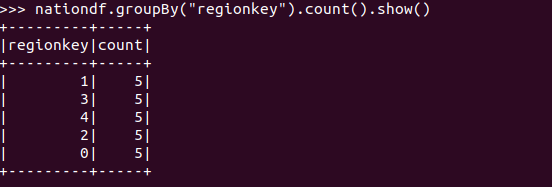
supplierdf.where(col("acctbal")<0).show()

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()**

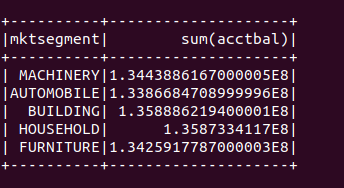


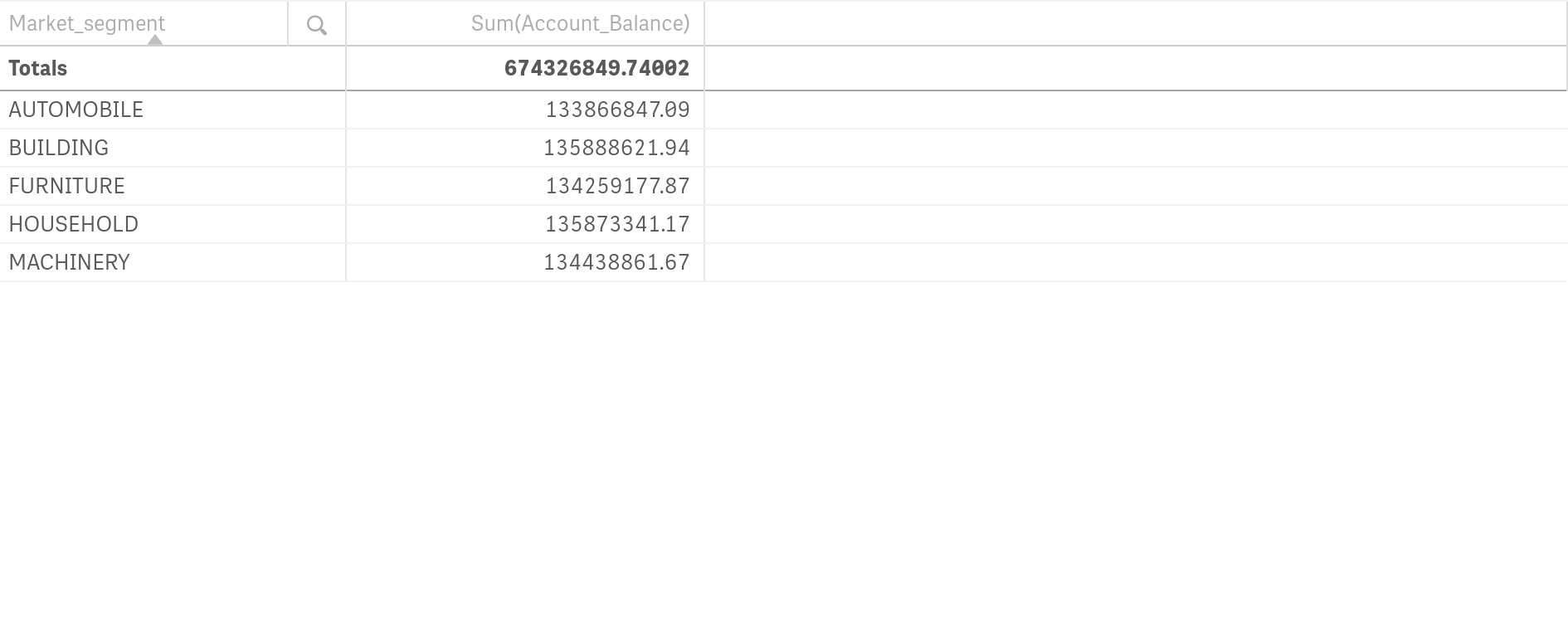
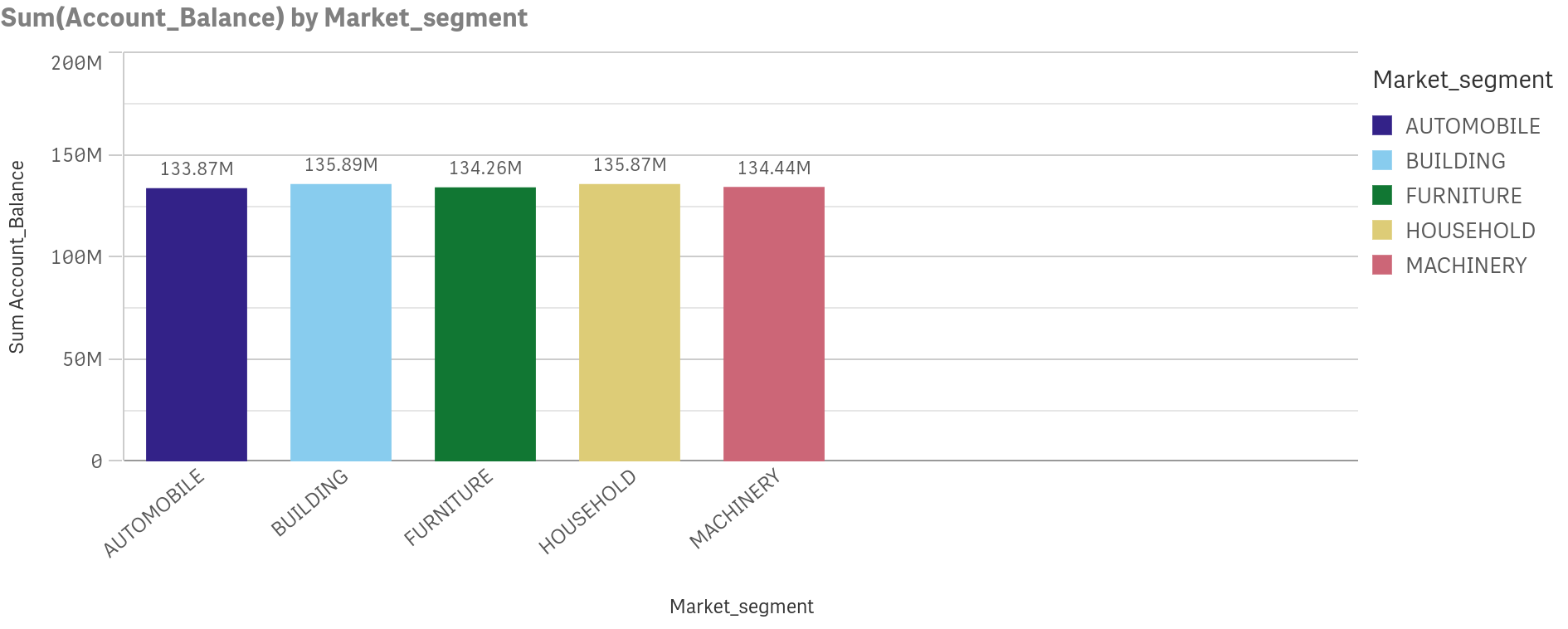
**Q4)Calculate the account balance for each market segment.**

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

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

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

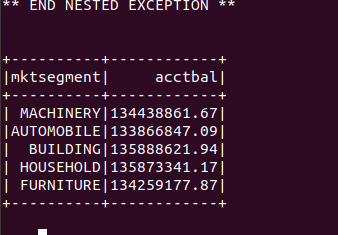




1.3443886167000005E8

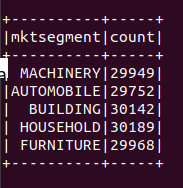
custdf =df.withColumnRenamed("sum(acctbal)","acctbal")

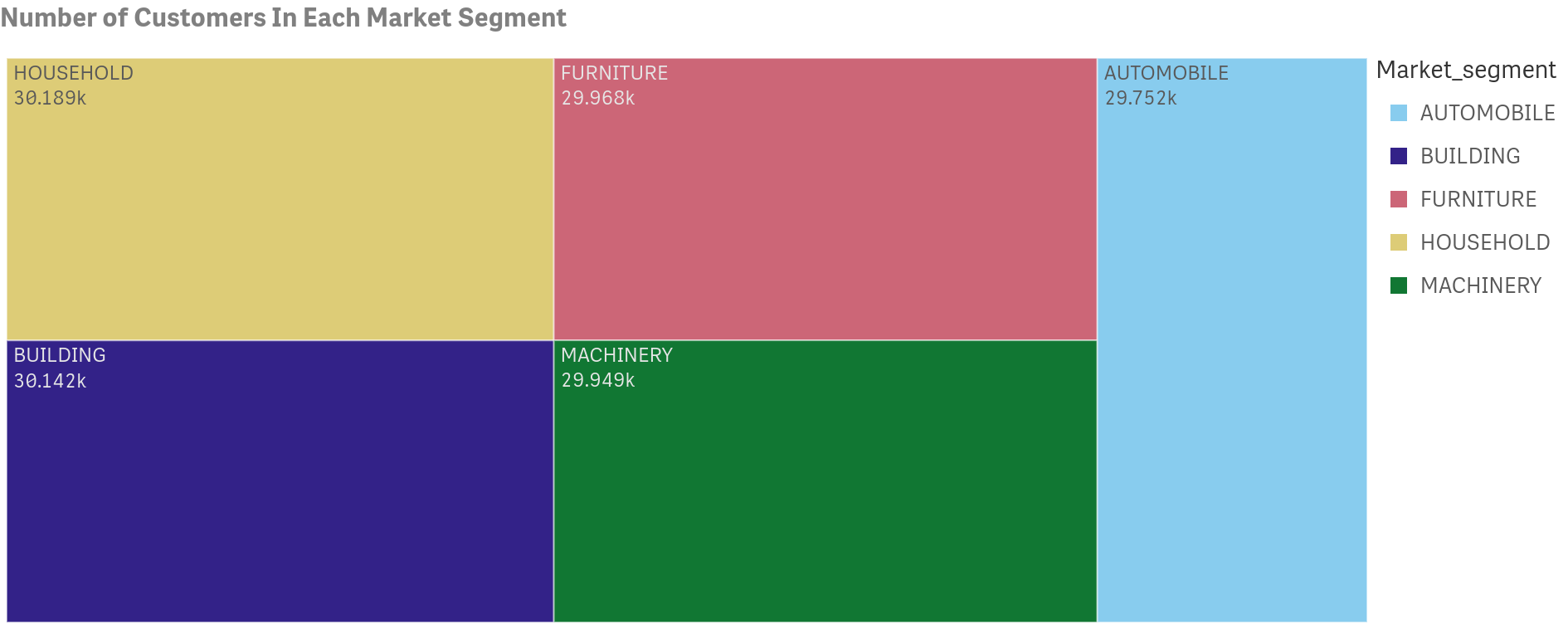
df=custdf.withColumn('acctbal',custdf.acctbal.cast(DecimalType(38,2)))



**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

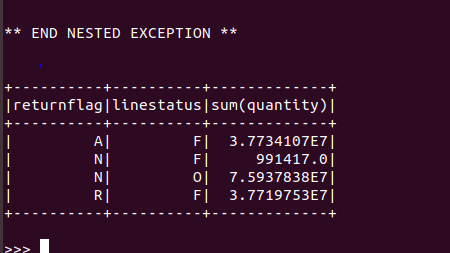
solndf=lineitemdf.where(col("shipdate") < '1998-10-12')

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

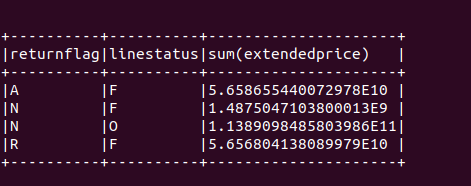
solndf=solndf.groupBy("returnflag","linestatus").agg(F.sum("quantity"),F.sum("extendedprice"),F.sum(decrease("extendedprice","discount")),F.sum(increase(decrease("extendedprice","discount"),"tax")),F.avg("quantity"),F.avg("extendedprice"),F.avg("discount"),F.count("quantity")).sort("returnflag","linestatus")

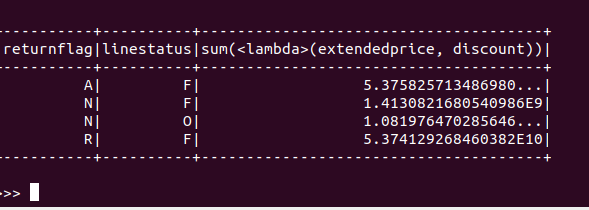
**FINAL SOLUTION FOR THE BUSINESS QUESTION**

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

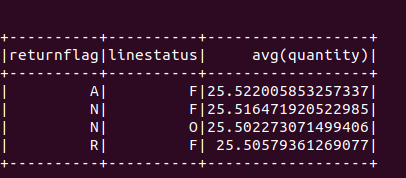


soln2=solndf.groupBy("returnflag","linestatus").agg(F.sum("extendedprice")).sort("returnflag","linestatus")

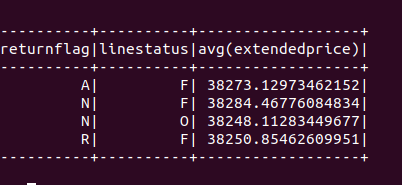


soln3=solndf.groupBy("returnflag","linestatus").agg(F.sum(decrease("extendedprice","discount"))).sort("returnflag","linestatus")

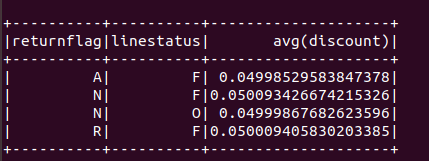
soln5=solndf.groupBy("returnflag","linestatus").agg(F.avg("quantity")).sort("returnflag","linestatus")



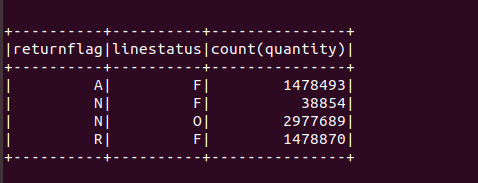
soln6=solndf.groupBy("returnflag","linestatus").agg(F.avg("extendedprice")).sort("returnflag","linestatus")



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



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



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

df = df1.join(df2, (df1.x1 == df2.x1) & (df1.x2 == df2.x2))

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')

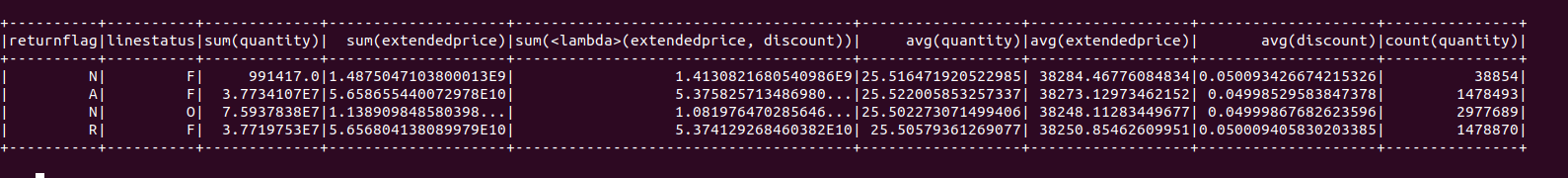
solndf2=solndf1.join(soln3,on=['returnflag','linestatus'],how='inner')

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

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

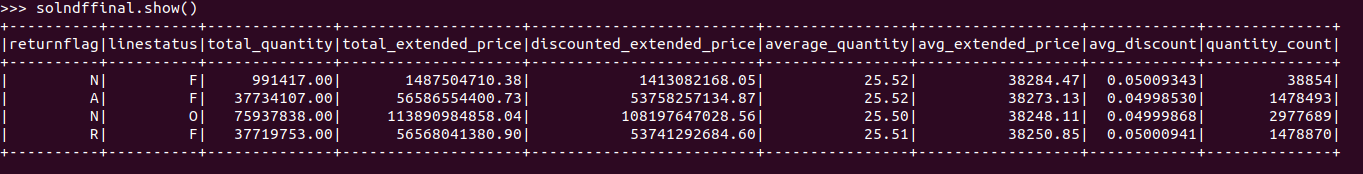
from pyspark.sql.types import \*

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)))

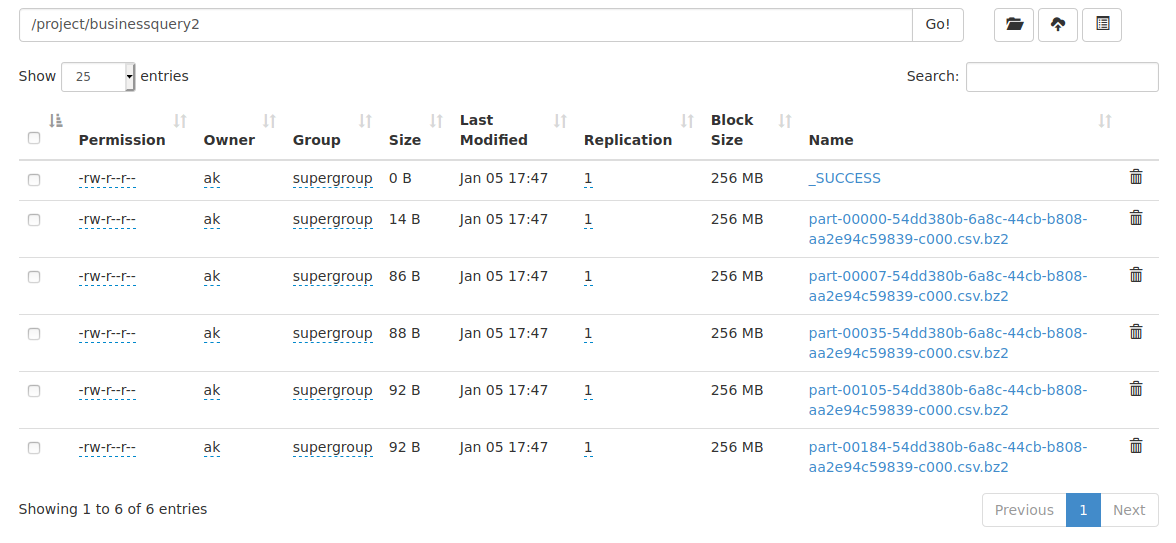


from pyspark.sql.types import StringType

>>> 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')

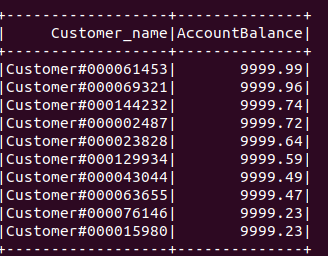




Q7.Find the Customers Having top 10 Acct Balances.

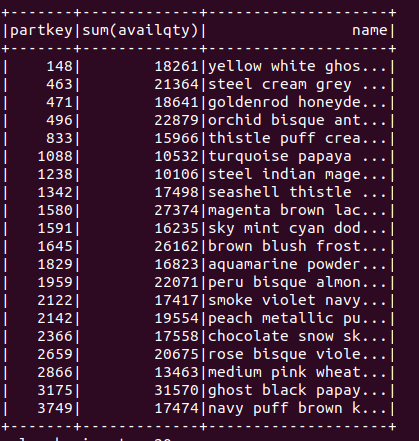
customer=customerdf.sort('acctbal',ascending=False).select(col('name').alias('Customer\_name'),col('acctbal').alias('AccountBalance'))

customer.show(10)



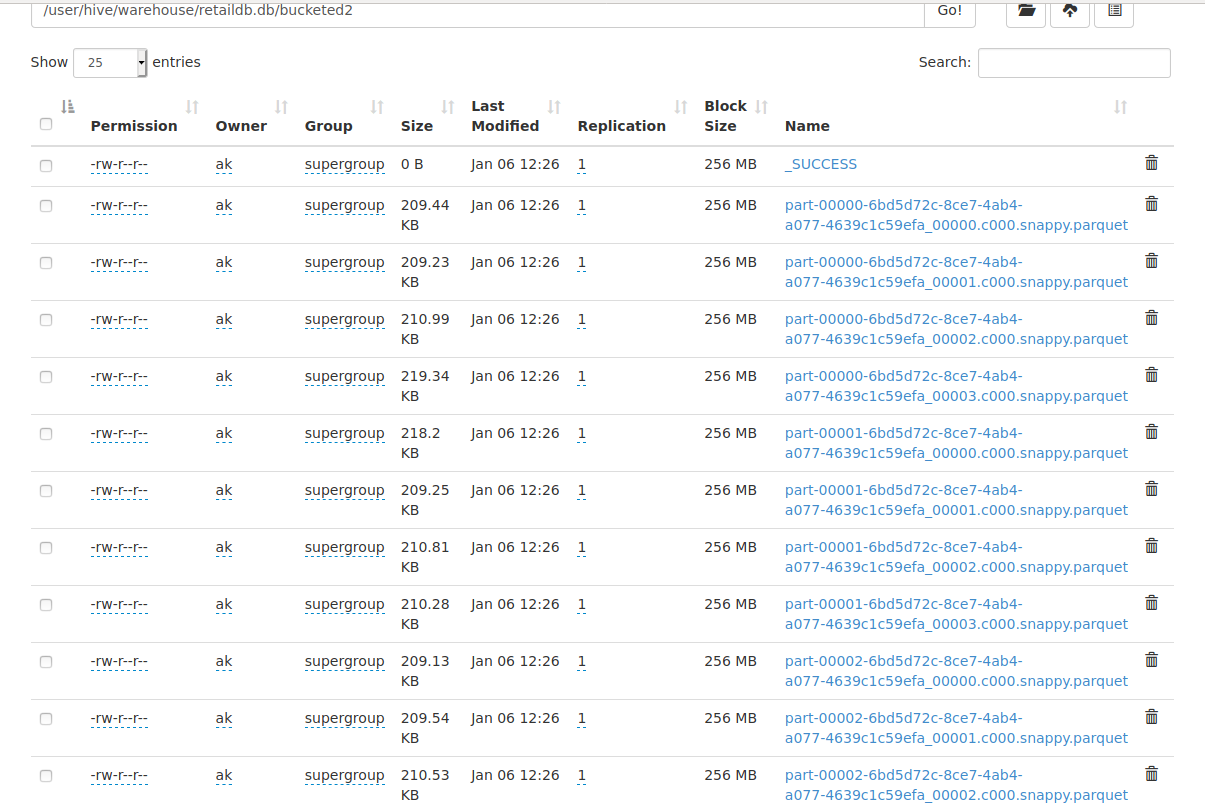
Q8.The Goods Management Team wants to know the total quantity available per part and there names so that if stock is less they can order that Particular part.

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



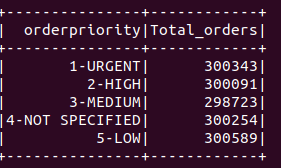
#using bucketby

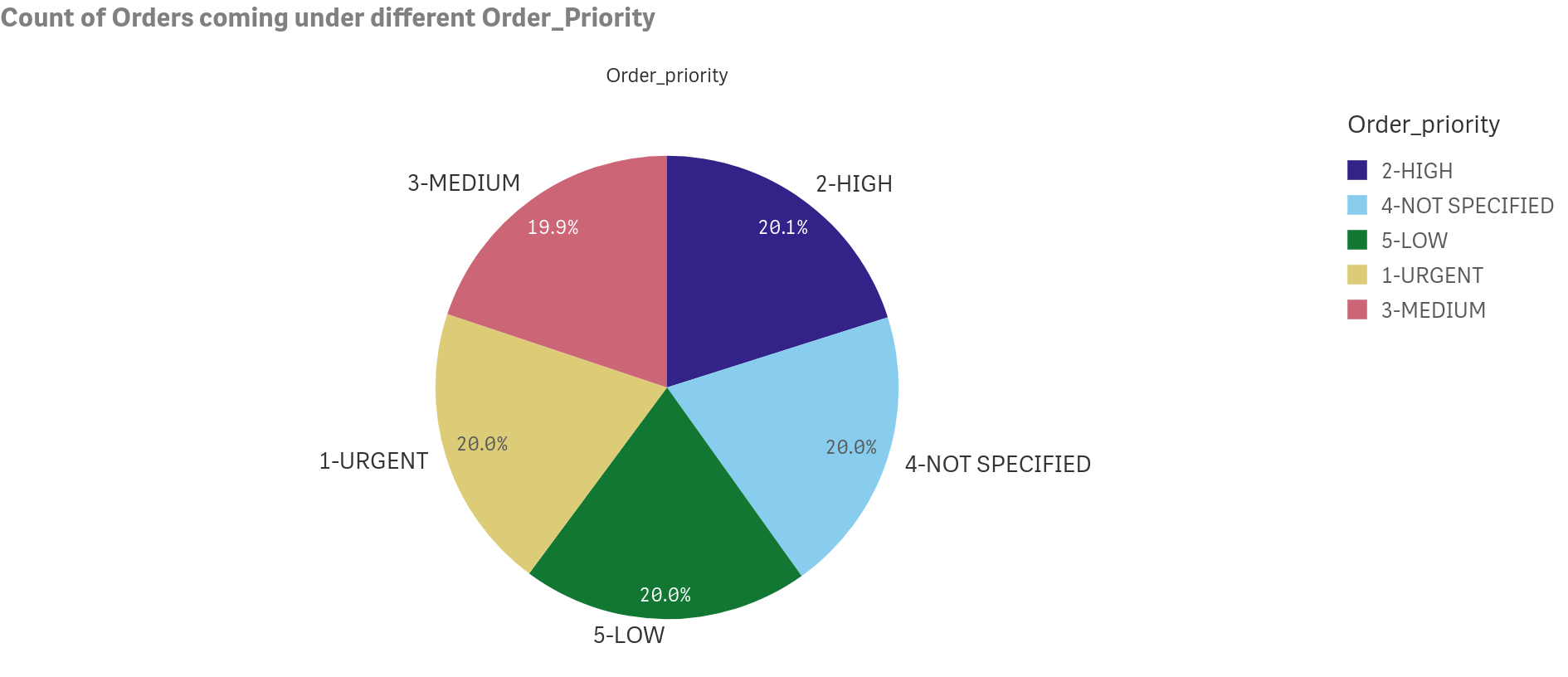
soln.write.bucketBy(4,’partkey’).saveAsTable(‘bucketed’,format=’parquet’)



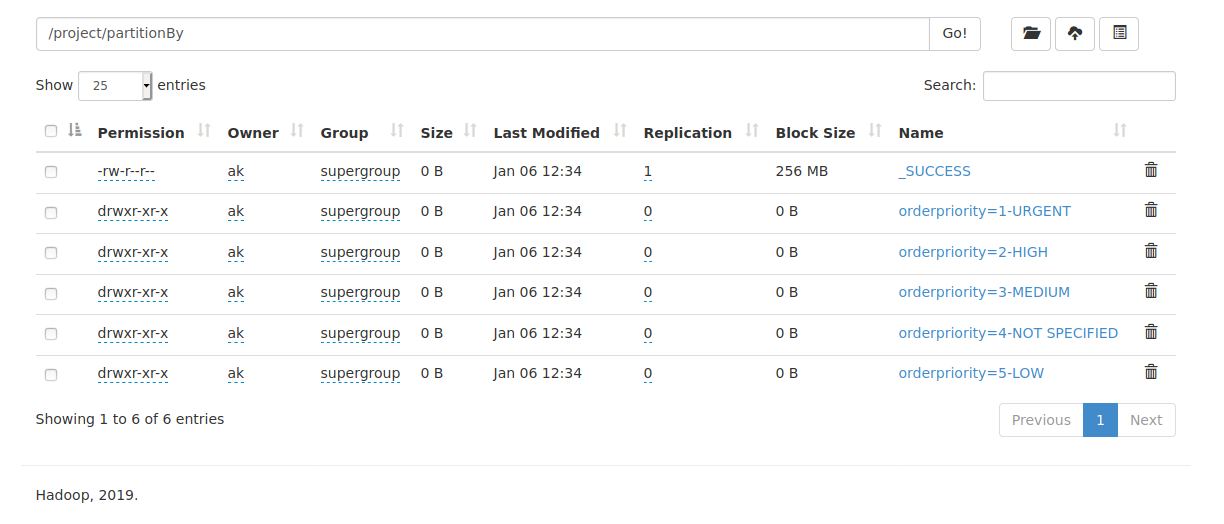
Q9.The Delivery team wants to know the Count of Orders coming under different types of Order\_Priority so that they can deliver the orders on priority basis.

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



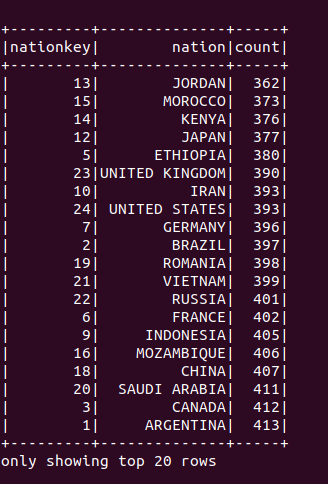


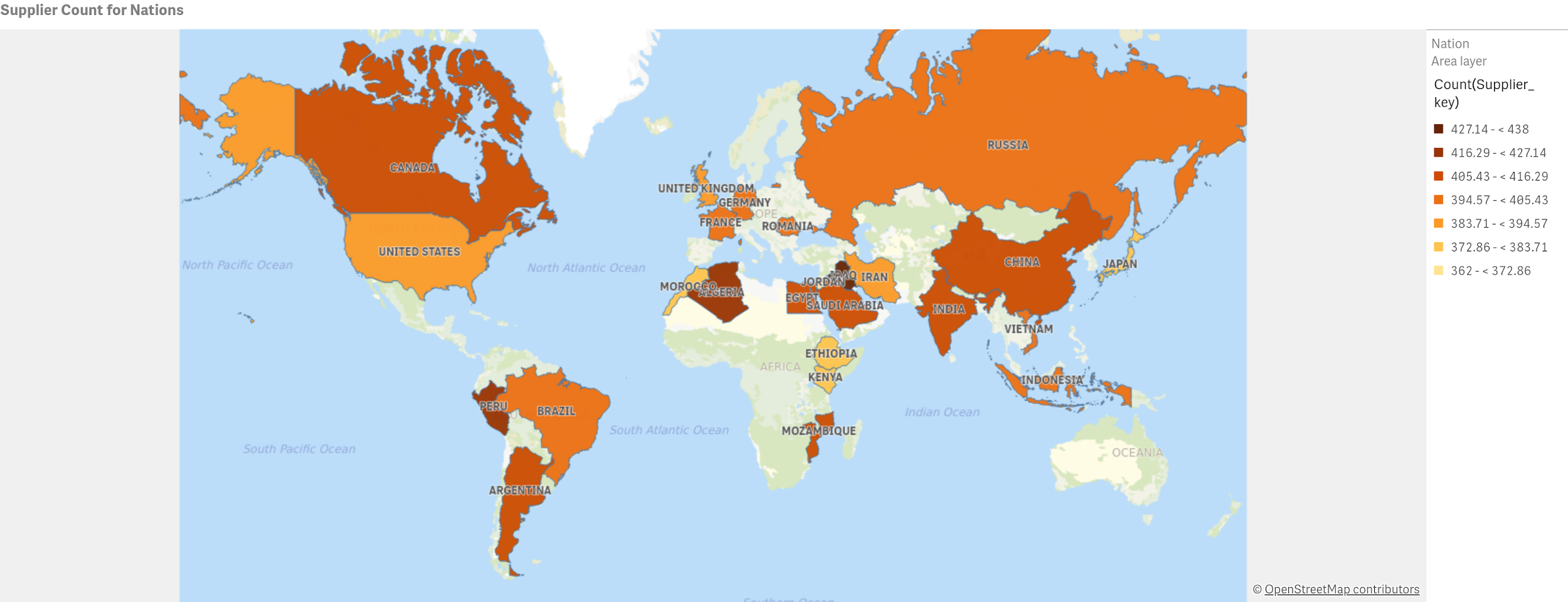
order.write.partitionBy("orderpriority").option('codec','gzip').csv('/project/partitionBy')



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)







==============================================================

**DELTA TABLE**

* All DataFrame columns must exist in the target table. If there are columns in the DataFrame not present in the table, an exception is raised. Columns present in the table but not in the DataFrame are set to null.
* DataFrame column data types must match the column data types in the target table. If they don’t match, an exception is raised.
* DataFrame column names cannot differ only by case. This means that you cannot have columns such as “Foo” and “foo” defined in the same table. While you can use Spark in case sensitive or insensitive (default) mode, Parquet is case sensitive when storing and returning column information. Delta Lake is case-preserving but insensitive when storing the schema and has this restriction to avoid potential mistakes, data corruption, or loss issues.

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

>>> from delta.tables import \*

>>> from pyspark.sql.functions import \*

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

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

spark.sql("select \* from delta").show(10)

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

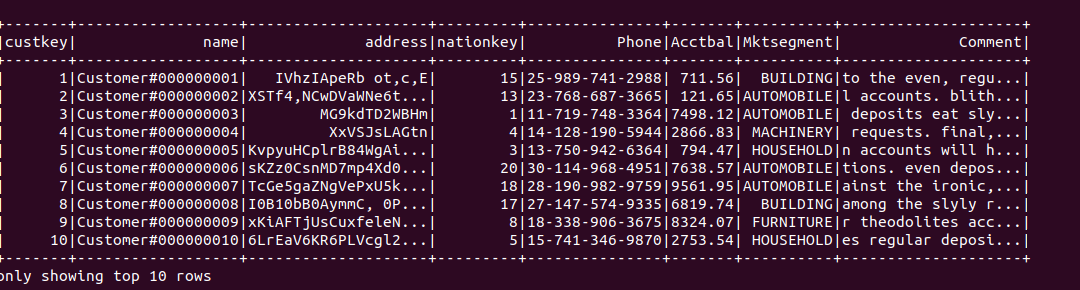
delf=deltaTable.toDF()//to save delta table as a dataframe

delf.show()

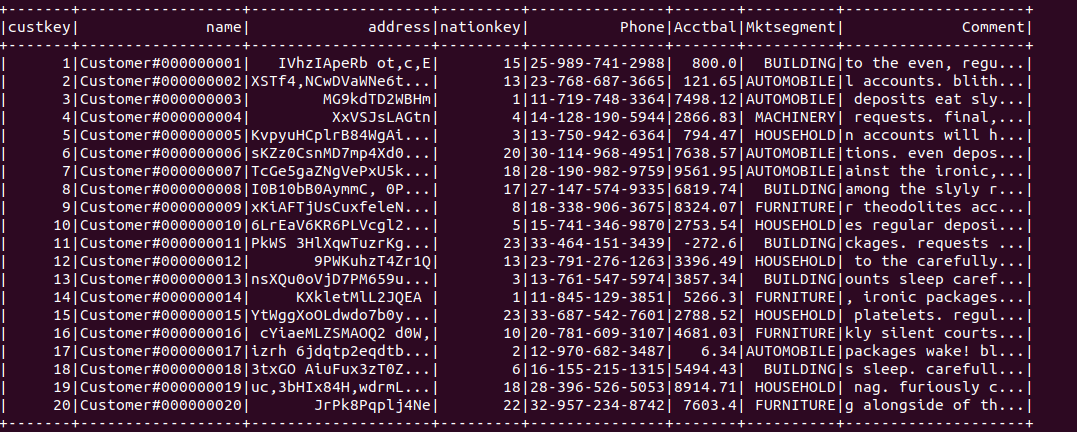
**Delta Update**

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

#Before Updating Delta Table



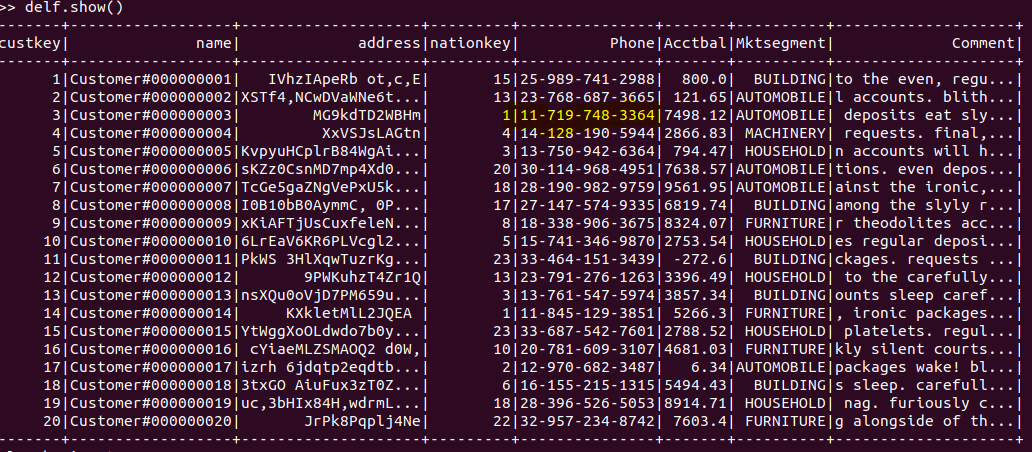
#After Updating



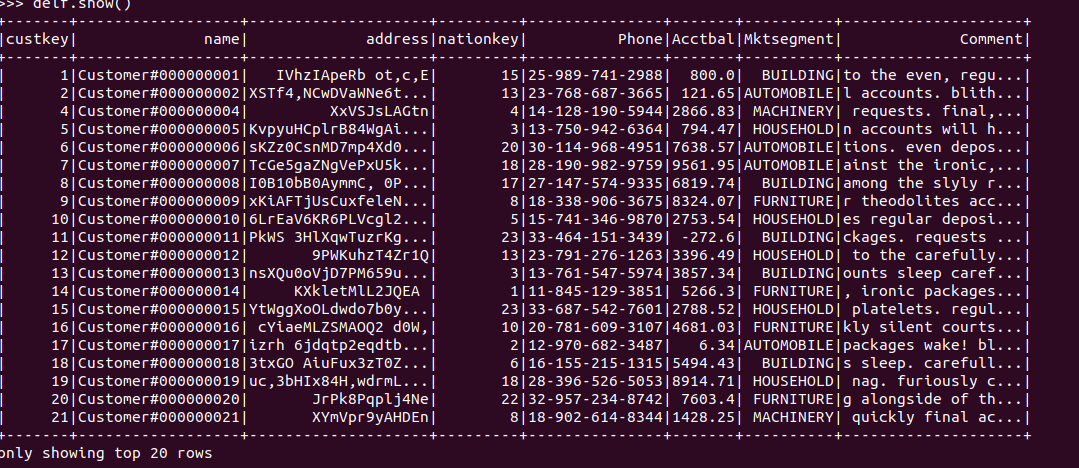
**Delete**

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

#Before Deleting



#After Deleting



**Hierarchical partitioning**

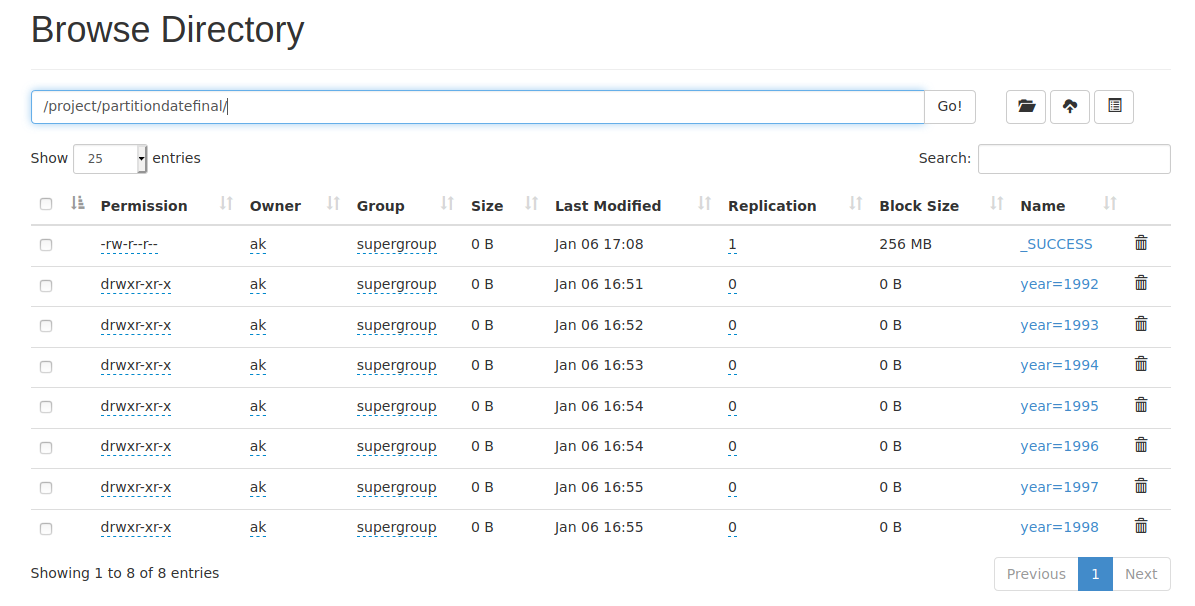
ordersdf.withColumn("year",year(col("Order\_date"))).withColumn("month",month(col("order\_date"))).withColumn("day",dayofmonth(col("order\_date"))).partitionBy("year","month","day").write.csv("/project/partitiondate")

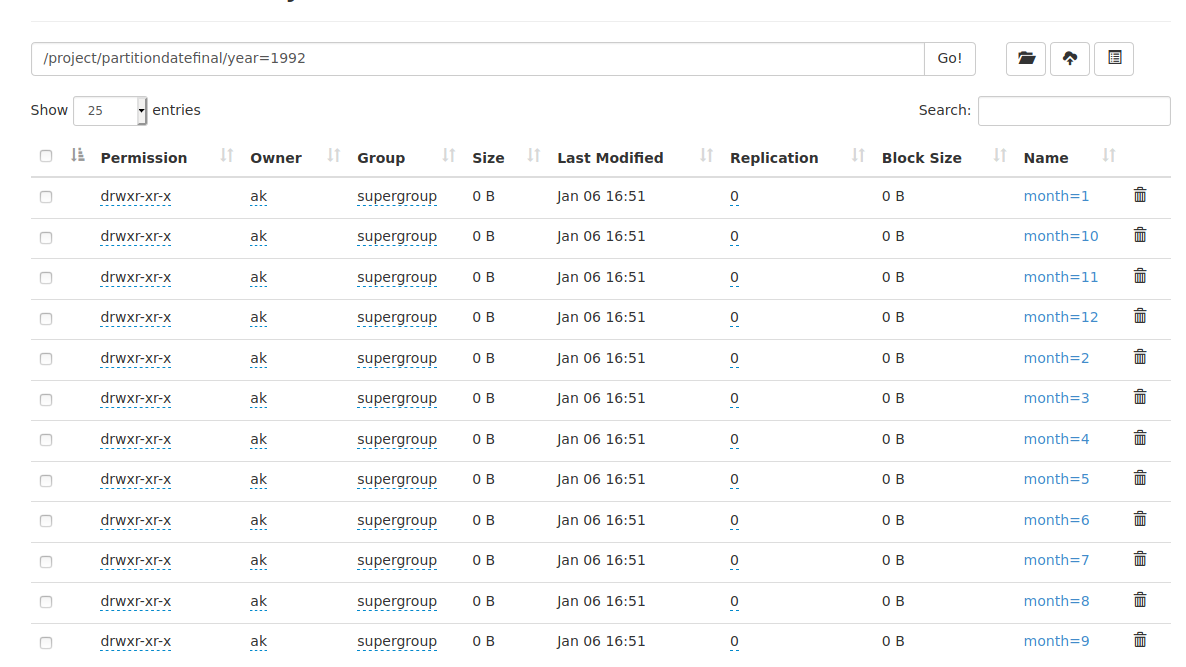
part1= ordersdf.withColumn("year",year(col("Order\_date")))

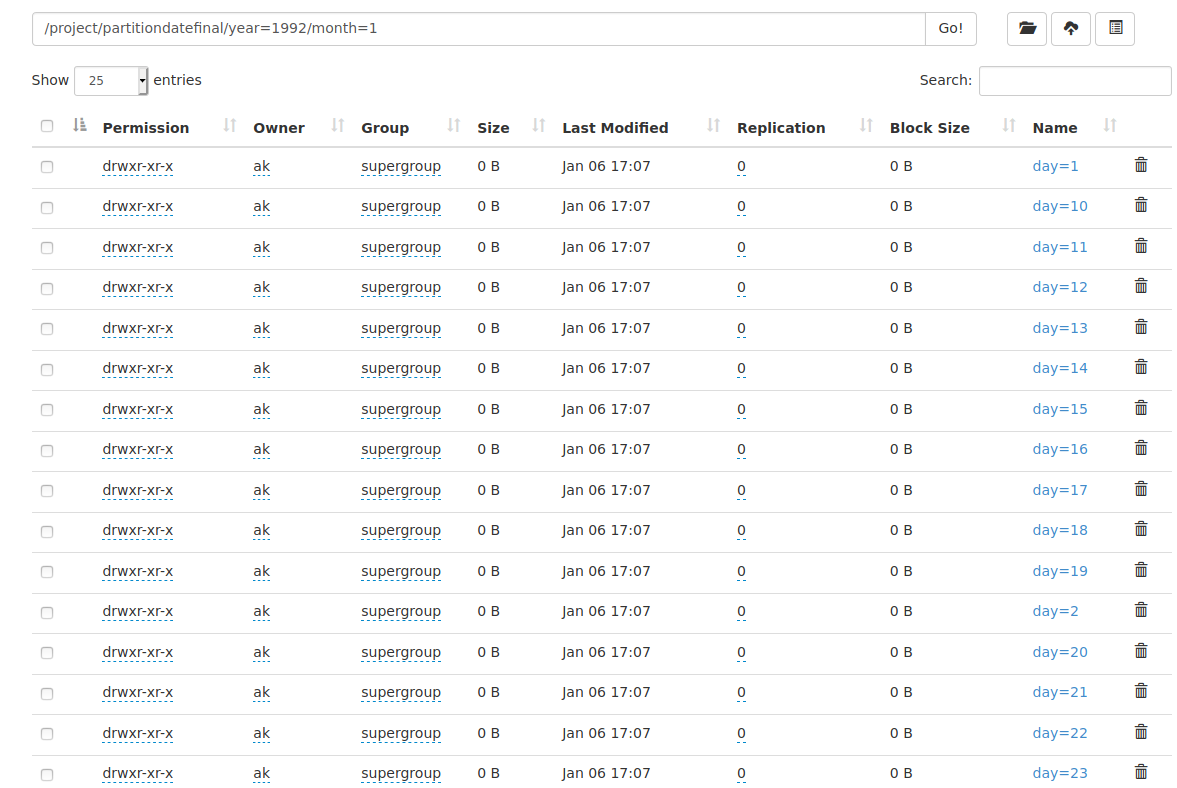
part2=part1.withColumn("month",month(col("order\_date")))

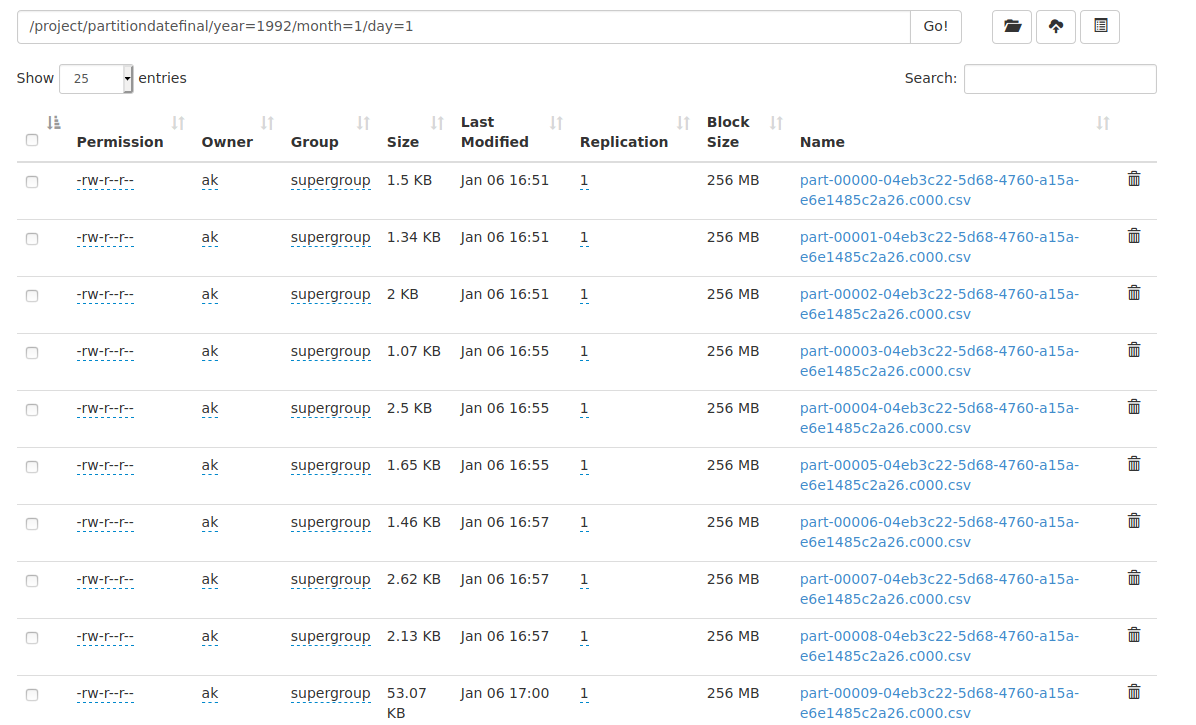
part3=part2.withColumn("day",dayofmonth(col("order\_date")))

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









Kafka-Spark Structured Streaming Integration

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

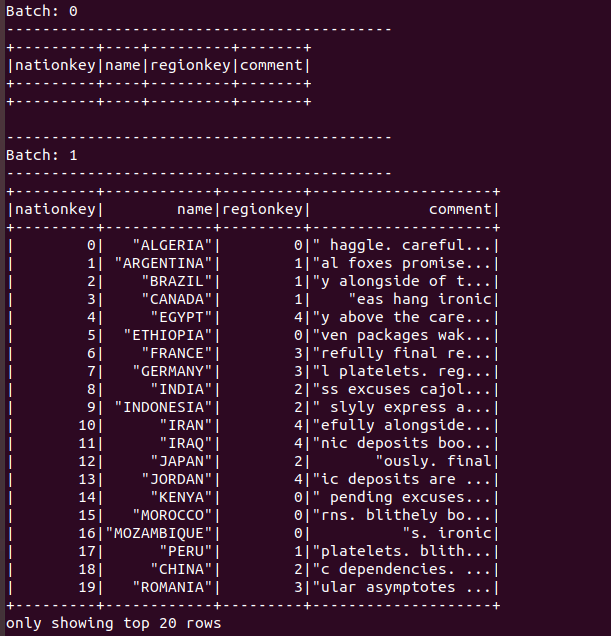
**Without Watermarking/Windowing**

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")

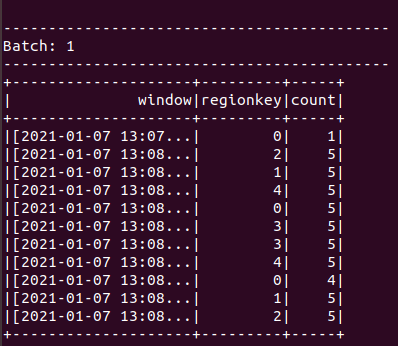
**// with watermark**

**// 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()



AWS S3-Spark Integration

**Reading From S3**

Have to mention the keys in the spark-default.conf file

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

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

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),\

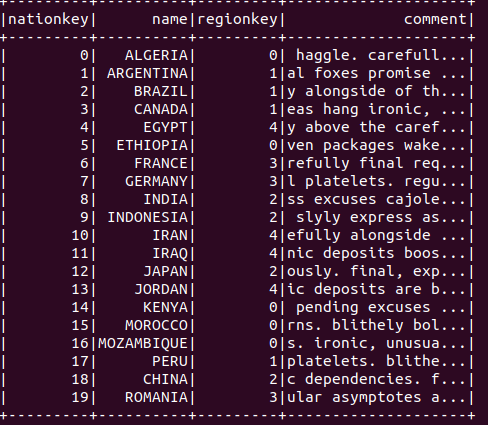
StructField('regionkey', IntegerType(), True),\

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

])

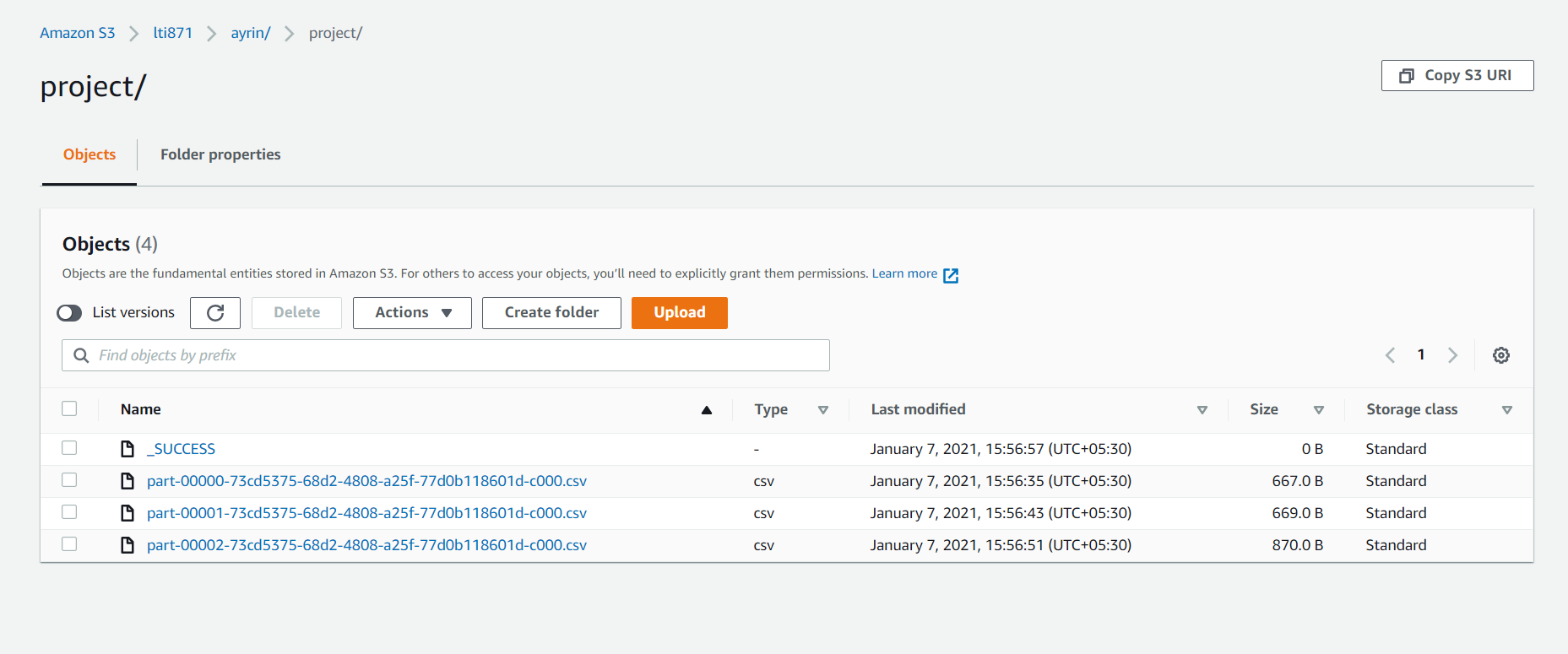
df=df.withColumn("\_c0",col("\_c0").cast(IntegerType())).withColumn("\_c2",col("\_c2").cast(IntegerType()))

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



**Writing to s3**

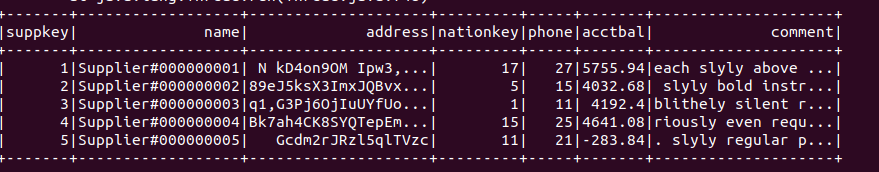
df2.write.csv("s3a://lti871/ayrin/project")



**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”)



**Snowflake - Spark**

snowsql -c manan

use database lti\_db;

use schema ltischema;

--creating stage

CREATE STAGE LTISCHEMA.partstage;

Using SnowSQL

-----put data from local into stage

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

----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 ;

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));

--------------loading data from stage to table ⇒

--successful copy command move the data from stage to table storage

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

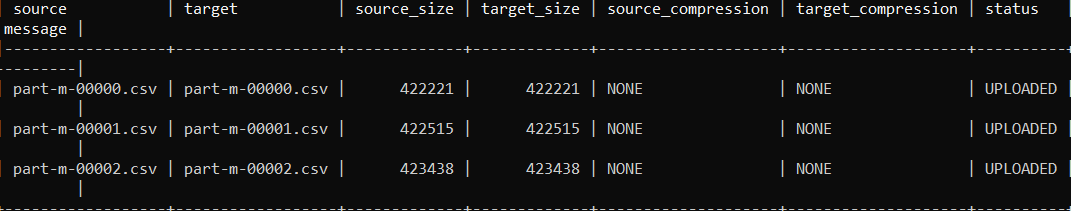
----------------purge = true moves the data from staging to table

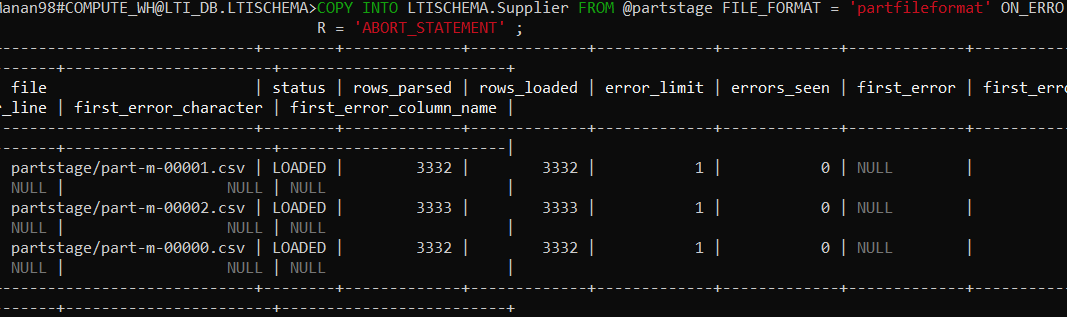
-------------loading same stage file to same table is not possible

------------------to list the data in the stages ---------------------------------------------

List @partstage;

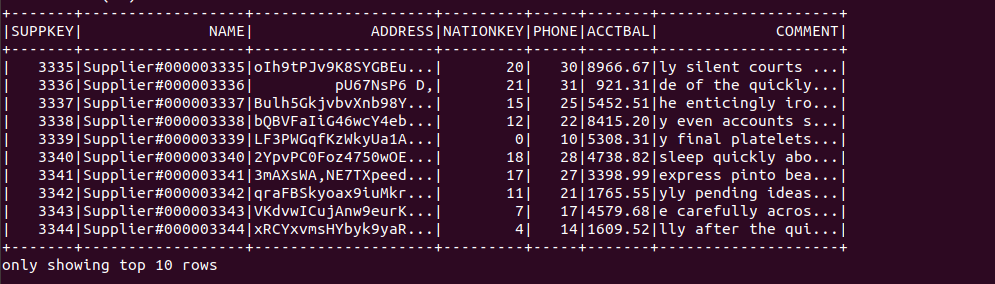
Put file://C:\USERS\91836\Desktop\part-m\*.csv @partstage auto\_compress=false;





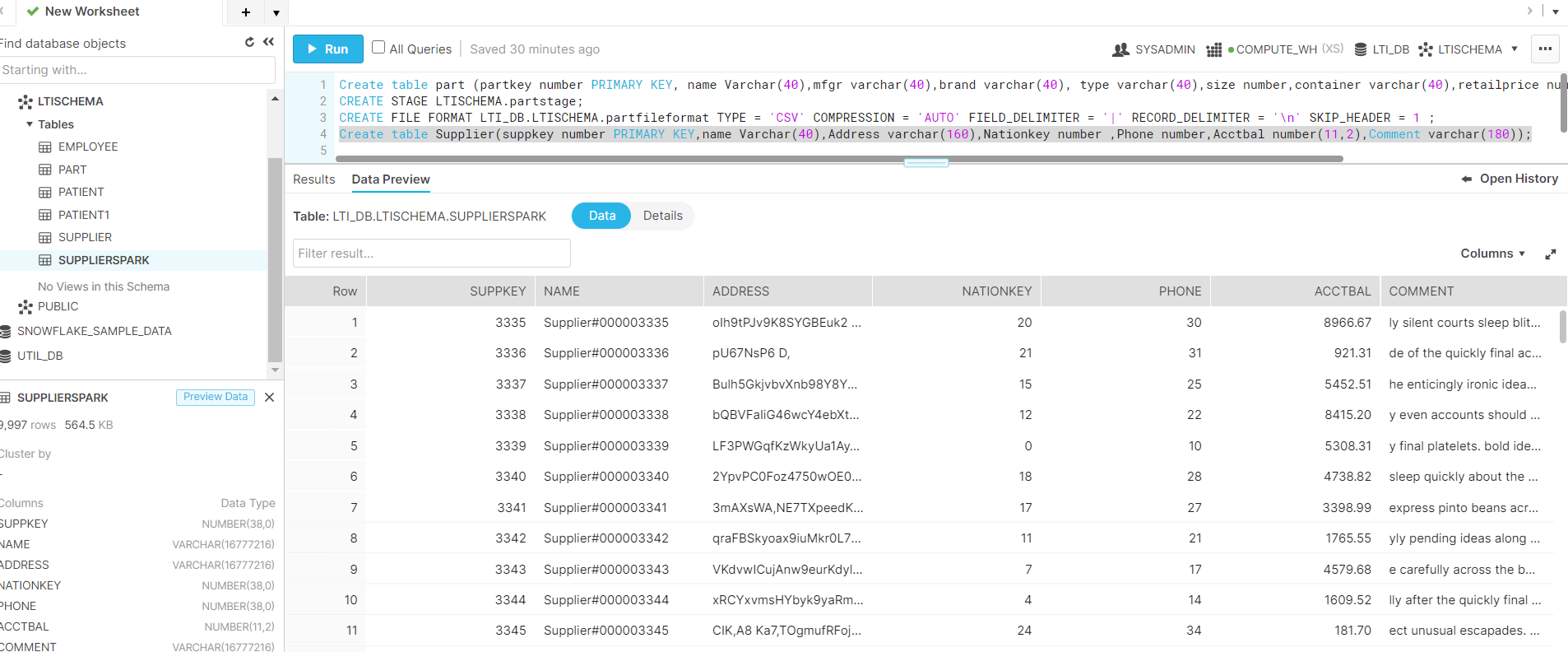
pyspark --packages net.snowflake:spark-snowflake\_2.11:2.5.1-spark\_2.3

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



sfOptions = {"sfURL" : "upa33798.us-east-1.snowflakecomputing.com","sfUser" : "Manan98","sfPassword" : "Deepak@69","sfDatabase" : "LTI\_DB","sfSchema" : "LTISCHEMA","sfWarehouse" : "COMPUTE\_WH"}

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



**Redshift Spark**

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

pyspark --jars /home/ak/.ivy2/cache/com.databricks/spark-redshift\_2.10/jars/spark-redshift\_2.10-0.5.0.jar,/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

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.patient').format('jdbc').load()

Spark Job

On yarn mode

spark-submit --master yarn --deploy-mode cluster --class org.apache.spark.examples.SparkPi /usr/lib/spark/examples/jars/spark-examples.jar

On Standalone