MINI PROJECT

- Source Tables setup:
- Refer the data model attached in this zip file. Create a new database in MySQL instance called “sales” and create 3 tables as mentioned in the data model.
- The names of the new tables are as follows
- Product
- Department
- Sales

Creating sales database

```
create database sales;
```

Creating table Product

```
create table Product(
Product varchar(100),
Department varchar(50)
);
```

Creating table Sales

```
create table Sales(
Product varchar(100),
Area varchar(20),
Employee_Id int,
Percentage_Sale decimal (10,3),
Profit_Or_Loss decimal (10,3)
);
```

Creating table Department

```
create table Department(
Employee_Id int,
```

Department varchar(50)

);

- Data Load into MySQL
- Import the 3 CSV files to the above created 3 tables. Ensure that all data loaded by checking the count of records and random values check.

Loading data into Department table

load data infile 'home/cloudera/Downloads/Department.csv'

into table Department

fields terminated by ','

enclosed by ' " '

lines terminated by '\r\n'

ignore 1 lines;

Loading data into Sales table

load data infile 'home/cloudera/Downloads/sale.csv'

into table Sales

fields terminated by ','

enclosed by ' " '

lines terminated by '\r\n'

ignore 1 lines;

Loading data into Product table

load data infile 'home/cloudera/Downloads/Product.csv'

into table Product

fields terminated by ','

enclosed by ' " '

lines terminated by '\r\n'

ignore 1 lines;

- Create a new database in Hive called ‘sales’.

create database sales;

- Data Ingestion

- Import all the tables in HIVE using Sqoop. Don’t create the table manually in Hive. Let Sqoop create the equivalent tables in Hive (under the new database sales) and load the data into those tables. Create one script for each table.

Importing Product table

```
sqoop import

--connect jdbc:mysql://localhost/sales

--username root

--password cloudera

--table Product

--hive-import

--create-hive-table

--hive-table sales.product

--target-dir assignment2

--m1
```

Importing Department table

```
sqoop import

--connect jdbc:mysql://localhost/sales

--username root

--password cloudera

--table Department

--hive-import

--create-hive-table

--hive-table sales.department

--target-dir assignment3
```

--m1

Importing Sales table

sqoop import

--connect jdbc:mysql://localhost/sales

--username root

--password cloudera

--table Sales

--hive-import

--create-hive-table

--hive-table sales.sales

--target-dir assignment4

--m1

- Now, do the above thing(4a) using Spark. If the target table exists, your script need to drop the table and recreate it with the respective source table data. If your script got executed multiple times also, it should not give any error, each time it should drop the target table and load the data from source table. Create one script for each table. It is also called Truncate reload method.

Importing Product table into Hive using Spark

val result = sqlContext.read.format("jdbc")

.option("url", "jdbc:mysql://localhost/sales")

.option("dbtable", "Product")

.option("user", "root")

.option("password", "cloudera")

.load()

.write()

.mode("overwrite")

.saveAsTable("sales.product")

Importing Department table into Hive using Spark

```
val result = sqlContext.read.format("jdbc")
.option("url", "jdbc:mysql://localhost/sales")
.option("dbtable", "Department")
.option("user", "root")
.option("password", "cloudera")
.load()
.write()
.mode("overwrite")
.saveAsTable("sales.department")
```

Importing Sales table into Hive using Spark

```
val result = sqlContext.read.format("jdbc")
.option("url", "jdbc:mysql://localhost/sales")
.option("dbtable", "Sales")
.option("user", "root")
.option("password", "cloudera")
.load()
.write()
.mode("overwrite")
.saveAsTable("sales.sales")
```

- Create 3 new external tables in Hive with the following structure.

Table Name	Columns	Partition Column
Department_Part	Employee ID	Department
Product_Part	Product	Department
Sales_part	Product, Employee_ID, Percentage_Sale, Profit_Or_Loss	Area

- Develop Hive Scripts, to load the above new partitioned tables from the respective Hive tables. Make sure that your script completely overwrites the existing data in the target table. Load method is Truncate reload.

Source Table	Target Table
Department	Department_part
Product	Product_part
Sales	Sales_part

Creating external table department_part

```
create external table department_part(

Employee_Id

)

partitioned by (Department String)

row format delimited fields terminated by ','

lines trminated by '\n';

set hive.exec.dynamic partition.mode = nonstrict;

insert overwrite table department_part

partition (Department)

select Employee_Id, Department from department;
```

Creating external table product_part

```
create external table product_part(

Product

)

partitioned by (Department String)

row format delimited fields terminated by ','

lines terminated by '\n';

set hive.exec.dynamic partition.mode = nonstrict;
```

insert overwrite table product_part

partition (Department)

select Product, Department from department;

Creating external table sales_part

create external table sales_part(

Product,

Employee_Id,

Percentage_Sale,

Profit_Or_Loss

)

partitioned by (Area String)

row format delimited fields terminated by ','

lines terminated by '\n';

set hive.exec.dynamic partition.mode = nonstrict;

insert overwrite table sales_part

partition (Area)

select Product, Employee_Id,Percentage_Sale,Profit_Or_Loss from sales;

- Create a new table called Sales_Emp_bucket with the below structure. Create buckets on Employee_ID column.

Sales_Emp_bucket
Product
Area
Employee_ID
Percentage_Sale
Profit_Or_Loss

- Develop one hive script that loads the above new table Sales_Emp_bucket from the Sales table.

```
create table sale_emp_bucket(
Product String,
Area String,
Employee_Id int,
Percentage_Sale decimal (10,3),
Profit_Or_Loss decimal (10,3)
)
clustered by (Employee_Id) into 4 buckets
row format delimited fields terminated by ','
lines terminated by '\n';
set hive.enforce.bucketing = true
insert overwrite table sales_emp_bucket
select Product,Area,Employee_Id,Percentage_Sale,Profit_Or_Loss from sales;
```

- Develop Spark Scala script for the following.
- Map all the employees with the product they sale using Department and Product Hive tables. The result of this script should create a new internal Hive table called Sales.Emp_Product with the following structure and load the data. If the target table exist, your script should drop the table and create a new table and load the data.

Sales.Emp_Product
Employee_ID
Product
Department

```
val dep = hctx.table ("sales.department")
val prd = hctx.table ("sales.product")
val dep_prd = dep.join (prd, "Department")
```

dep_prd.write.mode ("overwrite").saveAsTable ("sales.emp_product")

- Create a new hive managed table with the following structure using Hive DDL statements.

Sales.Sales_Profit
Product
Area
Employee ID
Percentage_Sale
Profit_Or_Loss
Profit_Flag

- Develop a Spark script to load the above table from Sales.Sales table with the following mapping. Load method is truncate reload.

Source Table	Source Column	Target Table	Target Column	Transformation
Sales.Sales	Product	Sales.Sales_Profit	Product	as is
Sales.Sales	Area	Sales.Sales_Profit	Area	as is
Sales.Sales	Employee ID	Sales.Sales_Profit	Employee ID	as is
Sales.Sales	Percentage_Sale	Sales.Sales_Profit	Percentage_Sale	If Sales.Sales.Percentage_sale column is Null/Blank then load 0, else Sales.Sales.Percentage_sale
Sales.Sales	Profit_Or_Loss	Sales.Sales_Profit	Profit_Or_Loss	as is
		Sales.Sales_Profit	Profit_Flag	If Sales.Sales.Profit_Or_Loss is positive, then 'Y' else 'N'

create table sales_profit(

Product varchar (100),

Area varchar (20),

```
Employee_Id int,

Percentage_Sale decimal (10,3),

Profit_Or_Loss decimal (10,3),

profit_flag int

)

row format delimited fields terminated by ','

lines terminated by '\n';
```

```
val sales = hctx.table ("sales.Sales")

val df2 = sales.withColumn ("Percentage_Sale", regexp_replace(col("Percentage_Sale") , "null", "0.00" ))

val calculate = udf( ( Profr_Loss: Double) => {

if (Profit_Or_Loss > 0) 1

else

0

} )

val df3 = df2.withColumn ("Profit_Flag", clculate ($"Profit_Or_Loss"))

df3.write.mode ("overwrite").saveAsTable ("sales.sales_profit")
```

- Transformation – 3
- Create a new Hive table as follows.

Sales.Sales_Area_Productivity
Area
Product_Count

- Develop Spark script to load the above hive table from Sales.Sales table. It should store the count of all products sold in that area. Load method is truncate reload.
- Also write the output to a new HDFS file called /User/YourName/Sales_Area_Productivity.csv. Load method is truncate reload. Fields comma separated.

```
create table sales_area_productivity (

Area varchar (20),

Product_count int

)

row format delimited fields terminated by ','

lines terminated by '\n';
```

```
val df1 = hctx.table ("sales.sales")

val df2 = df1.groupBy ("Area").count

val df3 = df2.withColumnRenamed ("count", "product_count")

df3.write.mode ("overwrite").saveAsTable ("sales.sales_area_productivity")

df3.rdd.saveAsTextFile ("hdfs:///home/cloudera/Desktop/sales1")
```

- Create a new Hive table as follows.

Sales.Percentage_Sale
Employee_Id
Total_Percentage_Sale

- Develop Spark script to load the above hive table from Sales.Sales table. It should store the total percentage sale of every employee. Load method is truncate reload.

- Also write the output to a new HDFS file called /User/YourName/Sales_Percentage_Sale.csv. Load method is truncate reload. Fields comma separated.

```
create table Percentage_Sale (

Employee_Id int,

Total_Percentage_Sale decimal (10,3)

)
```

```
row format delimited fields terminated by ','

lines terminated by '\n';


val df1 = hctx.table ("sales.Sales")

val df2 = df1.groupBy ("Employee_Id").agg (sum ("Percentage_Sale") )

val df3 = df2.withColumnRenamed ("sum (Percentage_Sale)", "Total_Percentage_Sale ")

df3.rdd.saveAsTextFile ("hdfs:///home/cloudera/Desktop/total_perc")

df3.write.mode ("overwrite").saveAsTable ("sales.total_percentage_sale")
```

- Transformation – 5
- Create a new Hive table as follows.

Sales.Total_Profit_Loss

Employee_Id

Total_Profit_Or_Loss

- Develop Spark script to load the above hive table from Sales.Sales table. It should store the total profit/loss for every employee. Load method is truncate reload.
- Also write the output to a new HDFS file called /User/YourName/Sales_Total_Profit_Loss.csv. Load method is truncate reload. Fields comma separated.

```
create table Total_Profit_Loss (

Employee_Id int,

Total_Profit_Or_Loss decimal (10,3)

)

row format delimited fields terminated by ','

lines terminated by '\n';


val df1 = hctx.table ("sales.Sales")

val df2 = df1.groupBy ("Employee_Id").agg (sum ("Profit_Or_Loss ") )
val df3 = df2.withColumnRenamed ("sum (Profit_Or_Loss )", "Total_Profit_Or_Loss ")

df3.rdd.saveAsTextFile ("hdfs:///home/cloudera/Desktop/total_profit_loss")

df3.write.mode ("overwrite").saveAsTable ("sales.total_profit_loss")
```

- Transformation – 7
- Develop Spark script to find the top three employees who sold maximum products across all departments.
- Write the output to a new HDFS file called /User/YourName/Top_Employess.csv. Load method is truncate reload. Fields comma separated.

```
val df1 = hctx.table ("sales.Department")

val df2 = df1.groupBy ("Employee_Id").count()

import org.apache.spark.sql.functions._

val df3 = df2.withColumnRenamed ("count", "max_pro_sold")

val df4 = df3.sort (desc ("max_pro_sold") )

df4.limit(3).rdd.saveAsTextFile ("top_emp")
```

- Transformation – 9
- Develop Spark script to find the top three products which fetched max cumulative profit and max cumulative loss.
- Write the output to a new HDFS file called /User/YourName/Max_Profit_Loss.csv. Load method is truncate reload. Fields comma separated.

```
val df1 = hctx.table ("sales.Sales")

val calc1 = udf (

(Profit_Or_Loss: Double) => {

if (Profit_Or_Loss > 0)  Profit_Or_Loss

else 0.00

} )

val calc2 = udf (

(Profit_Or_Loss: Double) => {
```

```
if (Profit_Or_Loss > 0)  0.00
else Profit_Or_Loss
} )

val df2 = df1.withColumnRenamed ("Profit" , calc1 ($"Profit_Or_Loss ") )
val df3 = df2.withColumnRenamed ("Loss" , calc2 ($"Profit_Or_Loss ") )
val df4 = df3.drop ("Profit_Or_Loss ")
val df5 = df4.groupBy ("Product").agg (sum ("Profit") , sum ("Loss") )
val df6 = df5.withColumnRenamed ("sum(Profit)", "max_profit")
val7 = df6.withColumnRenamed ("sum(Loss)", "max_loss")
val df8 = df7.sort ($"max_profit".desc)
df8.limit(3).rdd.saveAsTextFile ("pro_with_max_profit")
val df9 = df7.sort ($"max_loss".asc)
df8.limit(3).rdd.saveAsTextFile ("pro_with_max_loss")
```

- Transformation – 10
- Develop Spark script to find the department with maximum number of products.

Write the output to a new HDFS file called /User/YourName/Max_Products.csv. Load method is truncate reload. Fields comma separated

```
val df1 = hctx.table ("sales.Product")
val df2 = df1.groupBy ("Department").count
val df3 = df2.sort (desc ("count") )
df2.sort (desc ("count") ).limit(1).rdd.saveAsTextFile ("dep_with_max_no_of_products")
```

- Transformation – 11
- Develop Spark script to find the employees who sold the products for more than one department.
- Write the output to a new HDFS file called /User/YourName/Sold_For_Many_Dept.csv. Load method is truncate reload. Fields comma separated.

```
val df1 = hctx.table ("sales.Department")
val df2 = df1.distinct.groupBy ("Employee_Id").count

df2.write.mode ("overwrite").saveAsTable ("sales.emp_sold_to_more_than_one_dep")

val df = hctx.sql ("select  * from sales.emp_sold_to_more_than_one_dep where   count >1")

df.rdd.saveAsTextFile ("emp_sold_to_more_than_one_dep_updated1")
```