E-commerce Data Warehouse in HIVE using AWS

This big data project will look at Hive's capabilities to run analytical queries on massive datasets. Using only sales and Customer demographics data from the Adventure works dataset to perform analysis and answer the following questions:

- To find the upper and lower discount limits offered for any product
- Sales contributions by customer
- To Understand customer persona purchasing pattern based on gender, education and yearly income
- To find the sales contribution by customers on the overall year to date sales belong to categorized by same gender, yearly income.
- To identify the top performing territory based on sales
- To find the territory-wise sales and their adherence to the defined sales quota.

Aim

To perform Hive analytics on Sales and Customer Demographics data using big data tools such as Sqoop, Spark, and HDFS.

Data Description

Adventure Works is a free sample database of retail sales data. In this project, we will be only using Customer test, Individual test, Credit card, Sales order details, Store, Sales territory, Salesperson, Sales order header, Special offer tables from this database.

Tech Stack

➔ Language: SQL, Scala

➔ Services: AWS EC2, Docker, MySQL, Sqoop, Hive, HDFS, Spark

Approach

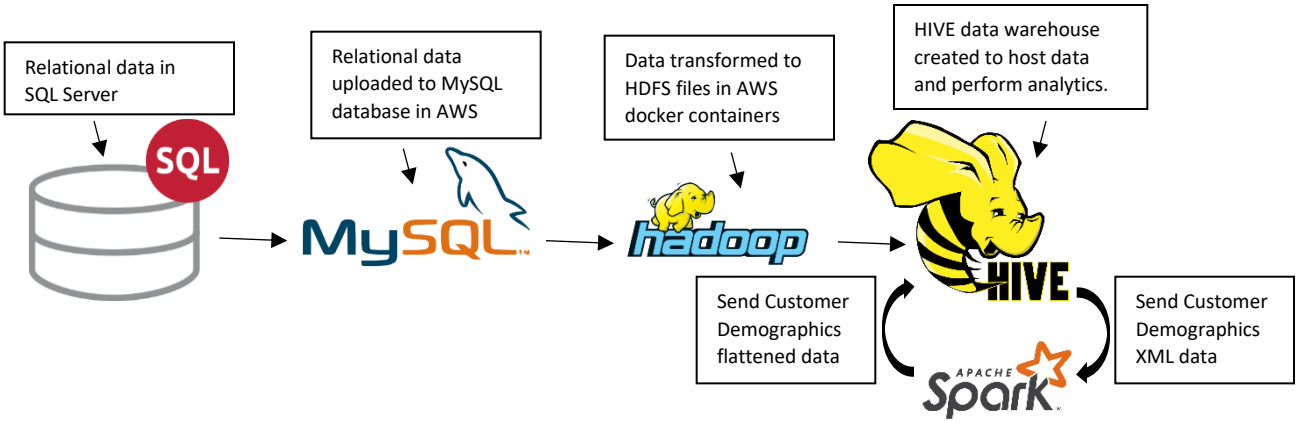
- Create an AWS EC2 instance and launch it.
- Create docker images using docker-compose file on EC2 machine via SSH.
- Create tables in MySQL.
- Load data from MySQL into HDFS storage using Sqoop commands.
- Move data from HDFS to Hive.
- Using Scala programming language, extract Customer demographics information from data and store it as parquet files.
- Move parquet files from Spark to Hive.
- Create tables in Hive and load data from Parquet files into tables.
- Perform Hive analytics on Sales and Customer demographics data.

Team

Student Name: Srividya Katam
Number of Students in Team: One

Git Repository: srividyakatam/ECommerce-Datawarehouse-Implementation-using-HIVE (github.com)

Architecture

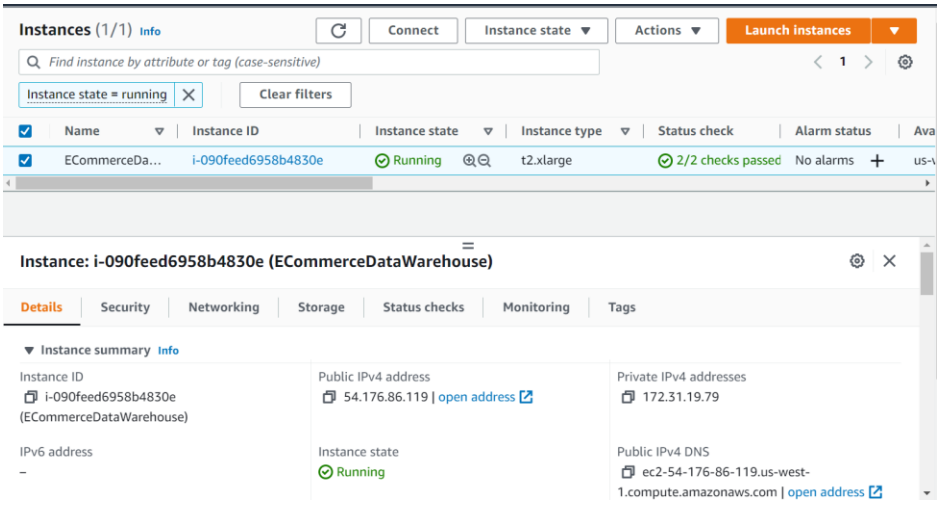


Implementation Details

- Create an AWS EC2 instance and launch it.
- Create docker images using docker-compose file on EC2 machine via SSH.
- Create tables in MySQL.
- Load data from MySQL into HDFS storage using Sqoop commands.
- Move data from HDFS to Hive.
- Using Scala programming language, extract Customer demographics information from data and store it as parquet files.
- Move parquet files from Spark to Hive.
- Create tables in Hive and load data from Parquet files into tables.
- Perform Hive analytics on Sales and Customer demographics data.

AWS Instance:

Create t2.2xlarge ec2 instance with Amazon Linux 2 AMI (HVM) and 96GB storage.
Created pem file for security keys.
Added security group to allow all inbound traffic for easy evaluation of project.



Docker containers:

Using following commands, install docker in the EC2 instance.
Updated docker-compose.yml with container names, user name and passwords for each container.

```
sudo yum update -y

sudo yum install docker

sudo curl -L "https://github.com/docker/compose/releases/download/1.29.1/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

sudo chmod +x /usr/local/bin/docker-compose

sudo gpasswd -a $USER docker

newgrp docker
```

Attached docker set up files in “Installation” folder.

Data copy from SQL to MySQL:

1. Created data copy files to insert relational data into MySQL database in AWS. Attached SQL files in “Code” folder.

Creating HDFS files from MySQL:

1. Created Sqoop jobs to read data from MySQL and create HDFS files. Attached Sqoop jobs. Refer to “Sqoop-import.txt” in “Code” folder

Installing Hive and Spark Dependencies:

1. Copy hive-site.xml file copied to Spark container conf folder.
2. Download Postgresql JAR from official website and copy to EC2 cluster and then to spark container jars folder.

Create Hive tables:

1. Using the HDFS files created, create and load tables in Hive. Refer to “04_Hive_tables_creation(cust,sales,stores).hql” in “Code” folder.

Using Spark to extract Customer Demographics XML data:

1. Extract customer demographics XML data and flatten using Scala program in Spark. Refer to “05_customer_demographic.scala” file in “code” folder.

Copy data from Spark container to Hive container:

1. Using file copy commands, copy data from Spark container to Hive container. Refer to “06_File_copy_commands (from spark container to hive container).txt” in “code” folder.

Creating Customer Demographics table from parquet file:

1. From the file copied in above step, create new table customer demographics with flattened data. Refer to “07_customer_demograhics_creation.hql.txt” in “code” folder.

Hive analytics queries

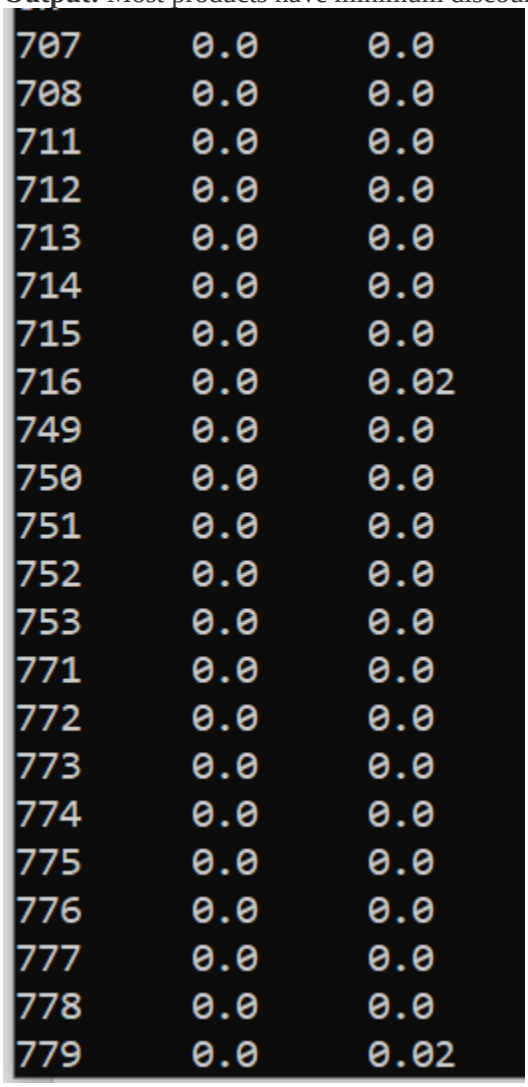
Performed Hive analytics to answer questions discussed above.

- To find the upper and lower discount limits offered for any product

Query:

```
select productid, min(discountpct) as min_discount, max(discountpct) as max_discount

from sales_order_details

group by productid
```

Output: Most products have minimum discount of zero and maximum discount of 2%.

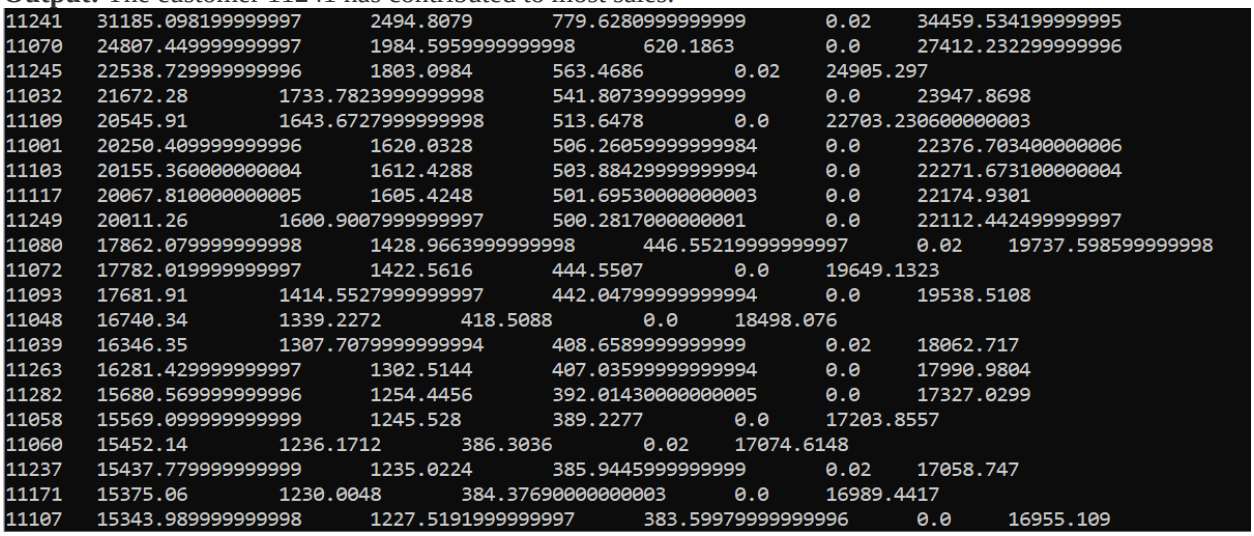


707	0.0	0.0
708	0.0	0.0
711	0.0	0.0
712	0.0	0.0
713	0.0	0.0
714	0.0	0.0
715	0.0	0.0
716	0.0	0.02
749	0.0	0.0
750	0.0	0.0
751	0.0	0.0
752	0.0	0.0
753	0.0	0.0
771	0.0	0.0
772	0.0	0.0
773	0.0	0.0
774	0.0	0.0
775	0.0	0.0
776	0.0	0.0
777	0.0	0.0
778	0.0	0.0
779	0.0	0.02

- Sales contributions by customer
  Query:

```
select soh.CustomerID , sum(soh.SubTotal) subtotal,

sum(soh.TaxAmt) Taxamt, sum(soh.Freight) Freight,

sum(sod.DiscountPct) discountpercent, sum(soh.TotalDue)
Totaldue

from sales_order_details sod join sales_order_header soh on
sod.salesorderid = soh.salesorderid

group by soh.CustomerID

order by Totaldue desc;
```

  Output: The customer 11241 has contributed to most sales.

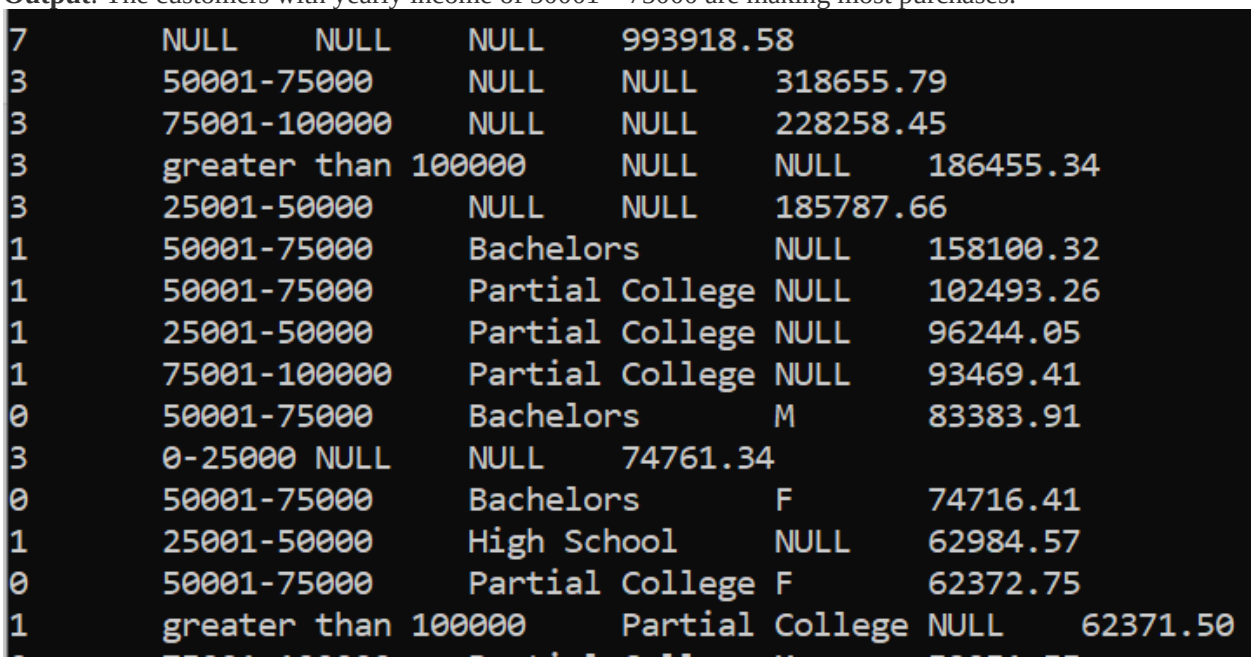


```
11241   31185.098199999997      2494.8079       779.6280999999999       0.02    34459.534199999995
11070   24807.449999999997      1984.5959999999998      620.1863        0.0     27412.232299999996
11245   22538.729999999996      1803.0984       563.4686        0.02    24905.297
11032   21672.28        1733.7823999999998      541.8073999999999       0.0     23947.8698
11109   20545.91        1643.6727999999998      513.6478        0.0     22703.230600000003
11001   20250.409999999996      1620.0328       506.26059999999984      0.0     22376.703400000006
11103   20155.360000000004      1612.4288       503.88429999999994      0.0     22271.673100000004
11117   20067.810000000005      1605.4248       501.69530000000003      0.0     22174.9301
11249   20011.26        1600.9007999999997      500.2817000000001       0.0     22112.442499999997
11080   17862.079999999998      1428.9663999999998      446.55219999999997      0.02    19737.598599999998
11072   17782.019999999997      1422.5616       444.5507        0.0     19649.1323
11093   17681.91        1414.5527999999997      442.04799999999994      0.0     19538.5108
11048   16740.34        1339.2272       418.5088        0.0     18498.076
11039   16346.35        1307.7079999999994      408.6589999999999       0.02    18062.717
11263   16281.429999999997      1302.5144       407.03599999999994      0.0     17990.9804
11282   15680.569999999996      1254.4456       392.01430000000005      0.0     17327.0299
11058   15569.099999999999      1245.528        389.2277        0.0     17203.8557
11060   15452.14        1236.1712       386.3036        0.02    17074.6148
11237   15437.779999999999      1235.0224       385.9445999999999       0.02    17058.747
11171   15375.06        1230.0048       384.37690000000003      0.0     16989.4417
11107   15343.989999999998      1227.5191999999997      383.59979999999996      0.0     16955.109
```

- To Understand customer persona purchasing pattern based on gender, education and yearly income
  Query:

```
select grouping__id, yearlyincome, education, gender,
sum(totalpurchaseytd) sales_value from customer_demo

group by yearlyincome, education,gender

with rollup

order by sales_value desc;
```

Output: The customers with yearly income of 50001 – 75000 are making most purchases.



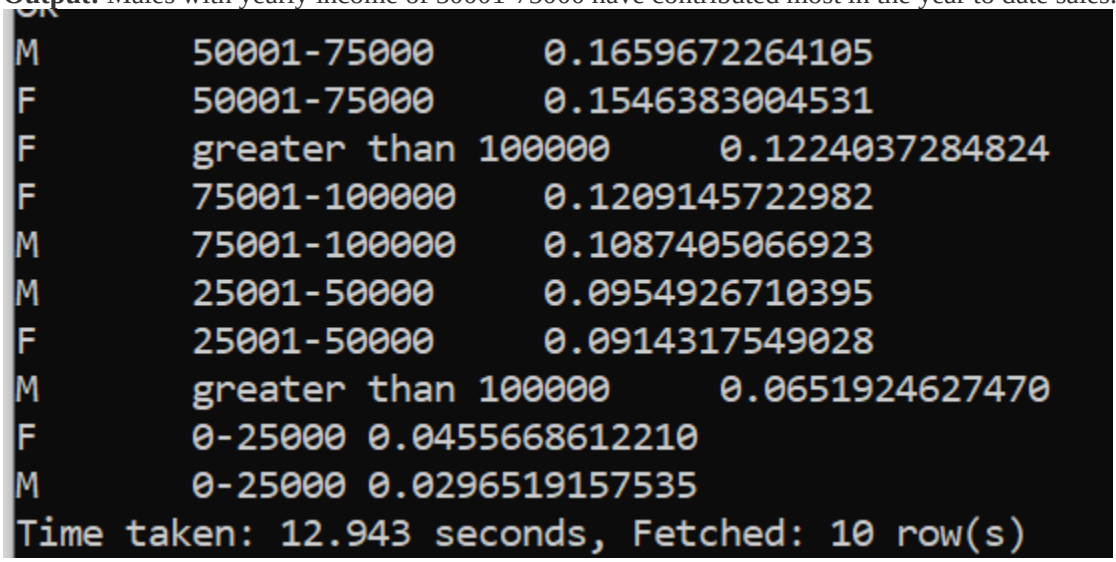
7	NULL	NULL	NULL	993918.58	
3	50001-75000	NULL	NULL	318655.79	
3	75001-100000	NULL	NULL	228258.45	
3	greater than 100000	NULL	NULL	186455.34	
3	25001-50000	NULL	NULL	185787.66	
1	50001-75000	Bachelors	NULL	158100.32	
1	50001-75000	Partial College	NULL	102493.26	
1	25001-50000	Partial College	NULL	96244.05	
1	75001-100000	Partial College	NULL	93469.41	
0	50001-75000	Bachelors	M	83383.91	
3	0-25000	NULL	NULL	74761.34	
0	50001-75000	Bachelors	F	74716.41	
1	25001-50000	High School	NULL	62984.57	
0	50001-75000	Partial College	F	62372.75	
1	greater than 100000	Partial College	NULL	62371.50	

- To find the sales contribution by customers on the overall year to date sales belong to categorized by same gender, yearly income.

Query:

```
select  gender, yearlyincome,

sum(v.percentage) as percentage_of_purchase

from customer_demo cd

join

(select customerid,

 (sum(totalpurchaseytd) over (partition by customerid) /
sum(totalpurchaseytd) over ()) percentage

from customer_demo cd) as v

on v.customerid = cd.customerid

group by gender, yearlyincome

order by percentage_of_purchase desc;
```

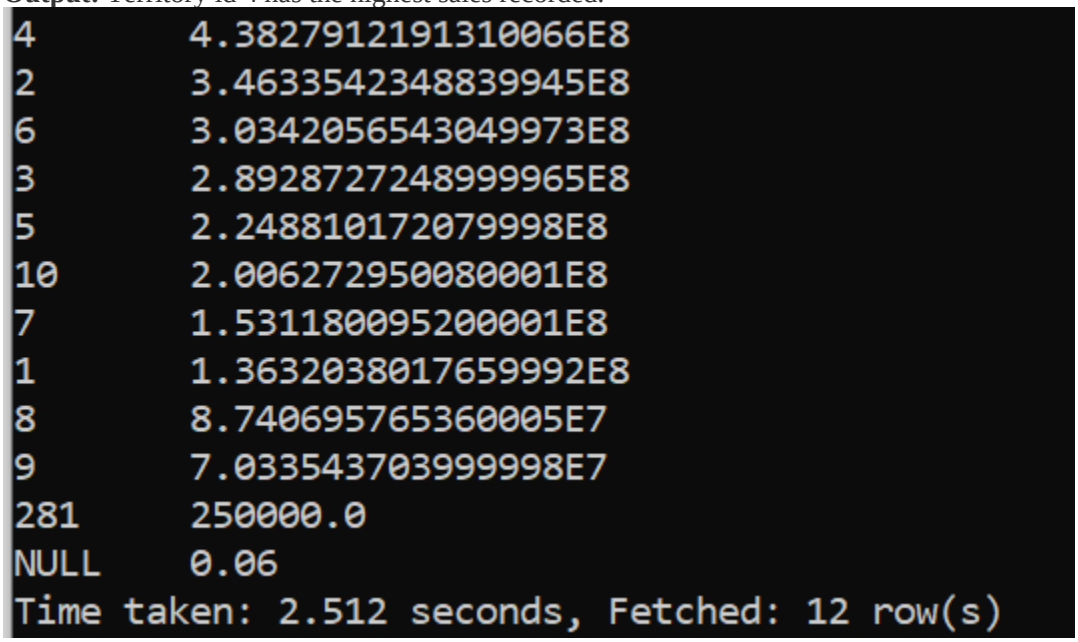
Output: Males with yearly income of 50001-75000 have contributed most in the year to date sales.



- To identify the top performing territory based on sales
  Query:

```
select TerritoryID,sum(SalesYTD) t_sales from store_details
group by TerritoryID order by t_sales desc;
```

Output: Territory id 4 has the highest sales recorded.



- To find the territory-wise sales and their adherence to the defined sales quota.
  Query:

```
select soh.TerritoryID,sum(soh.TotalDue),
sum(stores.targets_acheived) as target_completed

from sales_order_header soh

left join

(select TerritoryID, (sum(SalesYTD) over (partition by
TerritoryID) / sum(SalesQuota) over ()) targets_acheived
from store_details) as stores

on stores.TerritoryID = soh.TerritoryID

group by soh.TerritoryID;
```

Output: