**BIGDATA HADOOP & SPARK DEVELOPER**

**ASSIGNMENT 1**

1) Import mavenmovies from database to HDFS using sqoop

source create\_mavenmovies.sql;

A screenshot of a computer

Description automatically generated

sqoop import-all-tables --exclude-tables "employee10mar,staffinfo,staff15,t31,t3,tname,walmartcopy,walmarttranscations " --connect jdbc:mysql://sqoopdb.slbdh.cloudlabs.com/raveenaprabakartigeranaly -username raveenaprabakartigeranaly --password raveenaprabakartigeranalyneeix --warehouse-dir mavenmoviesdb -m 1

A screenshot of a computer

Description automatically generated­­

2) Convert Data into Parquet format using Spark

urlMySQL = "jdbc:mysql://sqoopdb.slbdh.cloudlabs.com:3306/raveenaprabakartigeranaly" propertiesForMySQL = {"driver":"com.mysql.jdbc.Driver","user":"raveenaprabakartigeranaly","password":"raveenaprabakartigeranalyneeix"}

a) actor = spark.read.jdbc(url=urlMySQL, table='actor', properties=propertiesForMySQL)

actor.write.parquet('actorParquet')

actorParquet = spark.read.parquet('actorParquet')

actorParquet.show()

b) address = spark.read.jdbc(url=urlMySQL, table='address', properties=propertiesForMySQL)

address.write.parquet('addressparquet')

addressparquet = spark.read.parquet('addressparquet')

addressparquet.show()

Graphical user interface, text

Description automatically generated

c) city = spark.read.jdbc(url=urlMySQL, table='city', properties=propertiesForMySQL)

city.write.parquet('cityparquet')

cityparquet = spark.read.parquet('cityparquet')

cityparquet.show()

A screenshot of a computer

Description automatically generated with medium confidence

d) country =spark.read.jdbc(url=urlMySQL, table='country', properties=propertiesForMySQL)

country.write.parquet('countryparquet')

countryparquet = spark.read.parquet('countryparquet')

countryparquet.show()

A screenshot of a computer

Description automatically generated

e) customer = spark.read.jdbc(url=urlMySQL, table='customer', properties=propertiesForMySQL) customer.write.parquet('customerparquet')

customerparquet = spark.read.parquet('customerparquet')

customerparquet.show(5)

Graphical user interface, text

Description automatically generated

f) inventory = spark.read.jdbc(url=urlMySQL, table='inventory', properties=propertiesForMySQL) inventory.write.parquet('inventoryparquet')

inventoryparquet = spark.read.parquet('inventoryparquet')

inventoryparquet.show()

A screenshot of a computer

Description automatically generated

g)category = spark.read.jdbc(url=urlMySQL, table='category', properties=propertiesForMySQL) category.write.parquet('categoryparquet')

categoryparquet = spark.read.parquet('categoryparquet')

categoryparquet.show()

Graphical user interface, text

Description automatically generated

h) film=spark.read.jdbc(url=urlMySQL,table='film',properties=propertiesForMySQL)

film.write.parquet('filmparquet')

filmparquet = spark.read.parquet('filmparquet')

filmparquet.show()

Graphical user interface, text

Description automatically generated

1. payment=spark.read.jdbc(url=urlMySQL,table='payment',properties=propertiesForMySQL)

payment.write.parquet('paymentparquet')

paymentparquet = spark.read.parquet('paymentparquet')

paymentparquet.show()

A screenshot of a computer

Description automatically generated

j) rental=spark.read.jdbc(url=urlMySQL,table='rental',properties=propertiesForMySQL)

rental.write.parquet('rentalparquet')

rentalparquet = spark.read.parquet('rentalparquet')

rentalparquet.show()

Graphical user interface, text

Description automatically generated

K) staff = spark.read.jdbc(url=urlMySQL, table='staff', properties=propertiesForMySQL)

staff.write.parquet('staffparquet')

staffparquet = spark.read.parquet('staffparquet')

staffparquet.show()

Text

Description automatically generated

3) Perform Mid Project Exercise using SparkSql on Parquet data

/\*

1. We will need a list of all staff members, including their first and last names,

email addresses, and the store identification number where they work.

\*/

spark.sql("create temporary view staffview using parquet options (path 'staffparquet')")

spark.sql("select first\_name, last\_name, email, store\_id from staffview").show()

A screenshot of a computer

Description automatically generated A screenshot of a computer

Description automatically generated

/\*

2. We will need separate counts of inventory items held at each of your two stores.

\*/

spark.sql("create temporary view inventoryview using parquet options (path 'inventoryparquet')")

spark.sql("select count(inventory\_id) as inventoryitems , store\_id from inventoryview group by store\_id ").show()

A screenshot of a computer

Description automatically generated

/\*

3. We will need a count of active customers for each of your stores. Separately, please.

\*/

spark.sql("create temporary view customerview using parquet options (path 'customerparquet')")

spark.sql("select count(customer\_id) as activecustomer , store\_id from customerview where active = true group by store\_id").show()

A screenshot of a computer

Description automatically generated

/\*

4. In order to assess the liability of a data breach, we will need you to provide a count

of all customer email addresses stored in the database.

\*/

spark.sql("select count(email) as emailcount from customerview").show()

Text

Description automatically generated

/\*

5. We are interested in how diverse your film offering is as a means of understanding how likely

you are to keep customers engaged in the future. Please provide a count of unique film titles

you have in inventory at each store and then provide a count of the unique categories of films you provide.

\*/

spark.sql("SELECT store\_id, COUNT(DISTINCT film\_id) AS unique\_films FROM inventoryview GROUP BY store\_id").show()

Graphical user interface, text

Description automatically generated

spark.sql("create temporary view categoryview using parquet options (path 'categoryparquet')")

s park.sql("select count(distinct name) AS unique\_categories from categoryview").show()

A screenshot of a computer

Description automatically generated

/\*

6. We would like to understand the replacement cost of your films.

Please provide the replacement cost for the film that is least expensive to replace,

the most expensive to replace, and the average of all films you carry. ``

\*/

spark.sql("create temporary view filmview using parquet options (path 'filmparquet')")

spark.sql("SELECT MIN(replacement\_cost) AS least\_expensive, MAX(replacement\_cost) AS most\_expensive, AVG(replacement\_cost) AS average\_replacement\_cost from filmview").show()

Text

Description automatically generated

/\*

7. We are interested in having you put payment monitoring systems and maximum payment

processing restrictions in place in order to minimize the future risk of fraud by your staff.

Please provide the average payment you process, as well as the maximum payment you have processed.

\*/

spark.sql("create temporary view paymentview using parquet options (path 'paymentparquet')")DataFrame[] >>> spark.sql("SELECT AVG(amount) AS average\_payment, MAX(amount) AS max\_payment FROM paymentview").show()

Text

Description automatically generated

/\*

8. We would like to better understand what your customer base looks like.

Please provide a list of all customer identification values, with a count of rentals

they have made all-time, with your highest volume customers at the top of the list.

spark.sql("create temporary view rentalview using parquet options (path 'rentalparquet')")DataFrame[]>>> spark.sql("SELECT customer\_id, COUNT(rental\_id) AS number\_of\_rentals FROM rentalview GROUP BY customer\_id ORDER BY COUNT(rental\_id) DESC").show()

Graphical user interface, text

Description automatically generated