**Fractal Capstone Project - Retail Case study**

**Dhanuja Pani S D | Data Engineer | B2-Crossover Training | dhanuja.pani@fractal.ai**

**[Note:Please Ctrl+Click on the link to follow the link and view files]**

**Data prep task:**

**1.Create a Git folder and pull the data from Github**

I have created a folder ‘retail’ in local and have pulled the data from github ‘<https://github.com/akgeoinsys/retail.git>’ repository.The files were then extracted in the local

Commands used:

* mkdir retail
* cd retail
* git init
* git remote add dhanujagit ‘<https://github.com/akgeoinsys/retail.git>’
* git pull dhanujagit main
* ls
* unzip Retail-20210521T141851Z-001.zip
* cd Retail
* ls

Graphical user interface, text, application

Description automatically generated

**2.Load Data into MYSQL and HDFS:-**

I have used a shell script to automate the process of loading data to mysql.

mysqlfile.sh

Graphical user interface, text

Description automatically generated with medium confidence

* This script reads through all the files in the directory.
* And for each file it splits the file name and gets the table name
* Its appends the load command to the loading sql file
* The schema creation for mysql tables are done
* Later the data from each file is loaded to mysql table.

mysqlfile.sh-🡪Bash file

<https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/mysqlfile.sh>

retail\_mysqlschema.sql🡪Contains the create queries for MYSQL tables

<https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/retail_mysql_schema.sql>

loadmysql.sql🡪Contains the load commands automated by the bash.

<https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/loadmysql.sql>

The counts are checked using counts.sql

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

<https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/counts.sql>

Thus, all files are successfully loaded.

**Load data into HDFS: -**

**I have created a bash script hdfsload.sh for moving all files from local to hdfs.**

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/hdfsload.sh**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/hdfsload.sh)

Graphical user interface, text

Description automatically generated

Text

Description automatically generated with medium confidence

**Files are loaded to HDFS**

**Hive task :**

**3. Create external table on top of existing HDFS project data as a Raw Table**

**I have created external hive tables on the file in hdfs by running the script.**

**Cmd:-**

hive -f hiveexternaltable.sql

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/hiveexternaltable.sql**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/hiveexternaltable.sql)

Graphical user interface, text, application

Description automatically generated

**Create a optimized staging table to minimize the storage**

I have created hive tables and have stored them in different file formats.

Cmd: -

hive -f hivestagingcompress.sql

<https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/hivestagingcompress.sql>

Text

Description automatically generated

File format Comparison: -

|  |  |
| --- | --- |
| File Format | Size |
| Original file | 7.0MB |
| ORC FILE | 79.5KB |
| Parquet FILE | 230.3KB |
| RCF File | 228.0KB |
| Parquet file | 252.7KB |

Graphical user interface, text, application, email

Description automatically generated

We could see that **ORC file** produces highest level of compression.

**Write 5 business queries with hive builtin functions?**

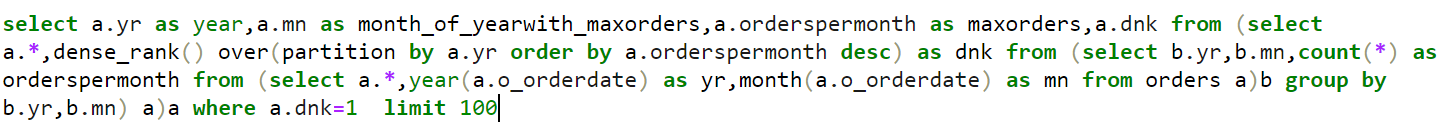
The queries are available in buisnessqueries.sql

<https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/buisnessqueries.sql>

**Query-1:**

**Finding which month of every particular year has the maximum no.of orders.**

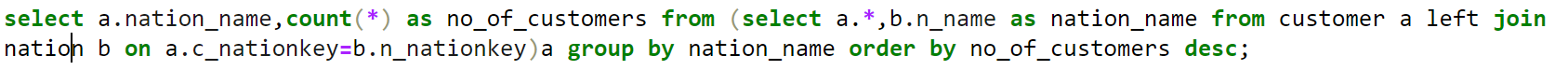
**Built-in-functions used:** year(),month(),dense\_rank()



**Query-2:**

**Finding the no. of customers from each nation**

**Built-in functions: -**count()

­

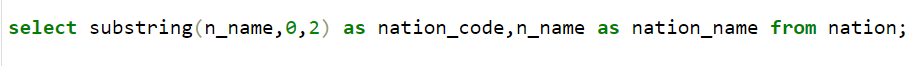
A picture containing text, receipt

Description automatically generated

**Query-3: -**

**Generating Nation codes for each nation.**

**Built-in-function used: -**substring()



Table

Description automatically generated

**Query-4: -**

**Rounding account balance:-**

**Built-in-function: -**round()



Text

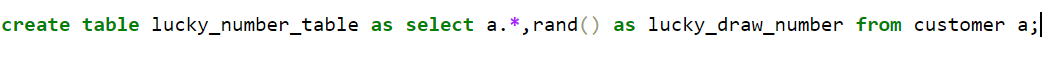
Description automatically generated

**Query-5: -**

**The company has decided to generate random numbers against each customer, pick a random number and gift all the customers with the random number a coupon.**

**Built-in function:-**rand()

**1.Creating a table with lucky\_draw\_numbers**



**2.selecting a lucky number from the list.**

A picture containing text

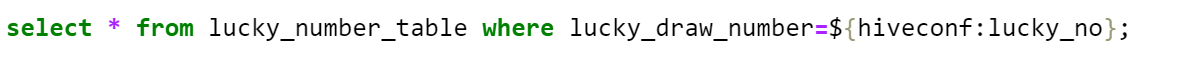
Description automatically generated

**3.Setting the lucky number to be chosen from customers: -**

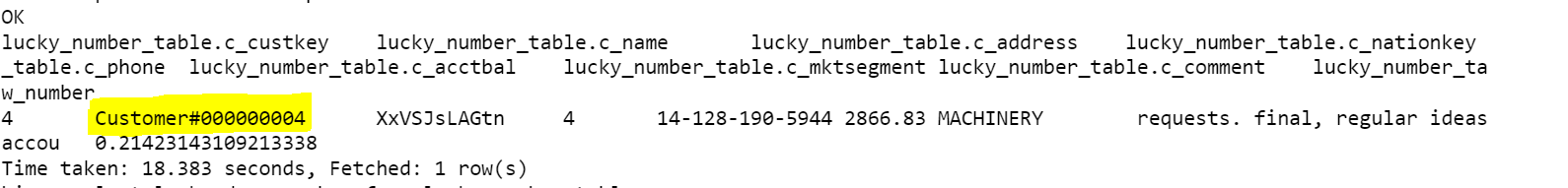
Logo, company name

Description automatically generated with medium confidence

**4.Querying the table for the lucky person: -**



**And the lucky winner is:-**



**4. Create a partition on multiple columns and bucket for optimized storage ?**

**Implement static, dynamic and bucket in a same table**

**State the reason for the usage and column selection**

**We are using the orders dataset.**

**1.Static Partition:-o\_orderpriority[1-URGENT]**

**Reason:-**I have chosen this column to be static as we are going analyse the urgent orders so creating a static patition on o\_orderpriority will help in retrieving the query results fast.

select distinct o\_orderpriority from orders;

Text

Description automatically generated

**Dynamic Partition: - o\_orderstatus**

**Reason: -Under urgent orders we have 3 distinct status, so creating a partition on this will lead to 3 folders so if we have to analyse based on the status it would be easy.**

**select distinct o\_orderstatus from orders;**

Text

Description automatically generated

**Bucketing: - o\_orderdate**

select count (distinct o\_orderdate) from orders where o\_orderpriority='1-URGENT';

Chart

Description automatically generated with low confidence

**We have 1701 distinct date so creating bucket on o\_orderdate would be optimal since we have more distinct column values**

**Based on the hash key of value % no.of buckets using o\_orderdate might leave all the buckets filled .**

Text

Description automatically generated

**Screenshots from warehouse: -**

**STATIC: -**

Graphical user interface, text, application

Description automatically generated

**DYNAMIC: -**

Graphical user interface, text, application

Description automatically generated

**Buckets: -**

A picture containing table

Description automatically generated

**We could see that all the five buckets are of the similar size, which denotes bucketing ‘o\_orderdate’ has used all the buckets almost evenly.**

**5. Do the data ingestion from RDB using sqoop apply all the functionality and incremental options**

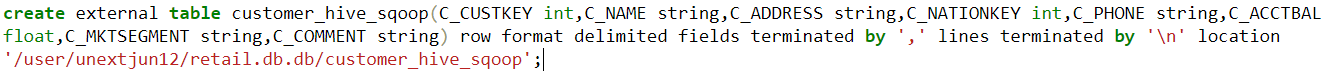
**1.Create a mysql table: -**

**We have distinct c\_custkey from 1 to 1500**

**Currently I have created a table with key <1000 to show incremental append in the below steps.**



**2.Create an external table in hive mapped to a hdfs location so each refresh of data to hdfs is reflected in the hive table.**



**3.Create a sqoop job to migrate data from mysql to the hdfs directory.**

**sqoop job --create custsqoopincappendjob -- import --connect jdbc:mysql://ip-10-1-1-204.ap-south-1.compute.internal:3306/unextjun12 --driver 'com.mysql.jdbc.Driver' -username unextjun12 -password BdhData123 -table customer\_sqoop --target-dir /user/unextjun12/retail.db.db/customer\_hive\_sqoop --incremental append --check-column c\_custkey -m 1**

**4.Check list of jobs;**

sqoop job --list

Text

Description automatically generated

**Job is created.**

**5.Execute the job**

sqoop job --exec custsqoopincappendjob

Text, letter

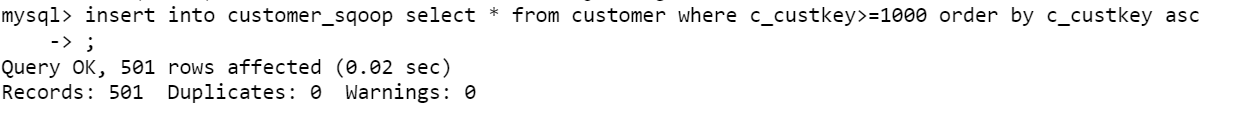
Description automatically generated

**Upper bound value is set to 999.**

Text

Description automatically generated

**6.Lets try inserting new records into mysql table, the records with c\_custkey greater than 1000.**



**7.Execute the sqoop job once again to refresh the data.**

sqoop job --exec custsqoopincappendjob

Text, letter

Description automatically generated

**Lowerbound=999,Upperbound=1500**

Text, letter

Description automatically generated

**HDFS SCREENSHOT:**

Graphical user interface, text, application

Description automatically generated

**HIVE :-**

A screenshot of a computer

Description automatically generated with medium confidence

**The data is loaded from mysql to hive through sqoop.**

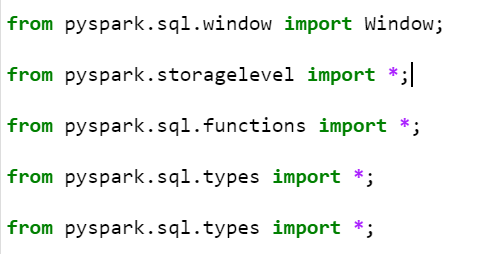
**Spark Task :**

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/dhanujasparktask.py**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/dhanujasparktask.py)

**Creating a spark session**



**Importing :**



**6.Create a dataframe as source from csv file and apply cleansing , scrubbing , filter the bad records , change the date format , replace nulls**

A picture containing background pattern

Description automatically generated

Table

Description automatically generated

**7. Connecting with mysql table and create dataframe apply upper bound and lower bound for partition**

**create hive table and apply partition and bucketing**

A picture containing text

Description automatically generated

Graphical user interface, table

Description automatically generated

**8. Apply transformations using withcoulmn /withcoulmnRenamed**

**dropna/fillna/replace**

**cast/alias/aggregation**

**filter/sort/select**

A picture containing background pattern

Description automatically generated

**Ship mode with most no.of orders:-**

Shape

Description automatically generated

**Joins with join optimization techniques:**

A picture containing text

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

Text, letter

Description automatically generated

**Handling different file formats with compression codecs**

A picture containing scatter chart

Description automatically generated

**BZIP2:**

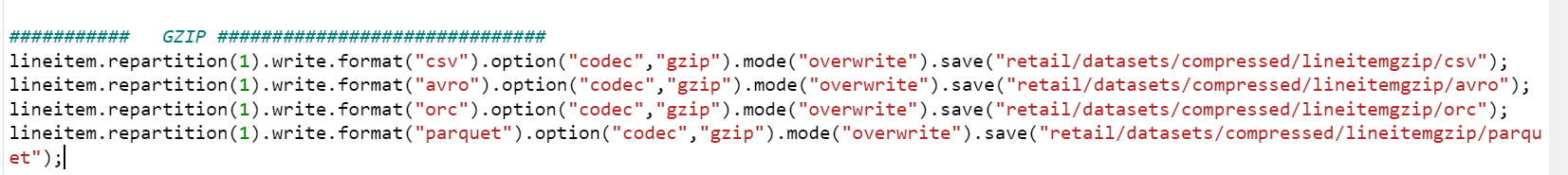
A picture containing chart

Description automatically generated

Graphical user interface, table

Description automatically generated

**GZIP:-**



Graphical user interface

Description automatically generated

**SNAPPY:-**

A picture containing chart

Description automatically generated

Graphical user interface

Description automatically generated with low confidence

**9. Implement Delta table and perform Transactional property**

**Merge schema and time travel**

A picture containing text

Description automatically generated

**Even upon rewriting the delta with new data we are able to see the older version of data**

**Version 0:-**

Table

Description automatically generated

**Version-1:**

A picture containing shape

Description automatically generated

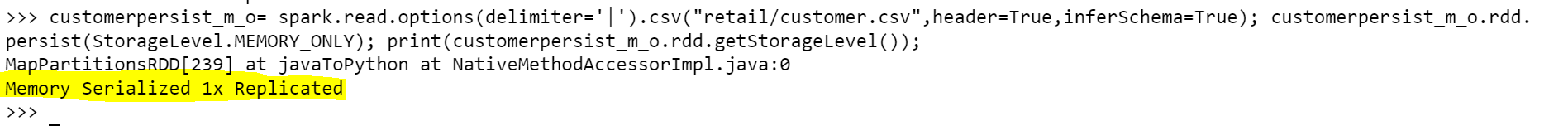
**10. Memory management and optimization**

**Implement persist /cache with effective storage level**

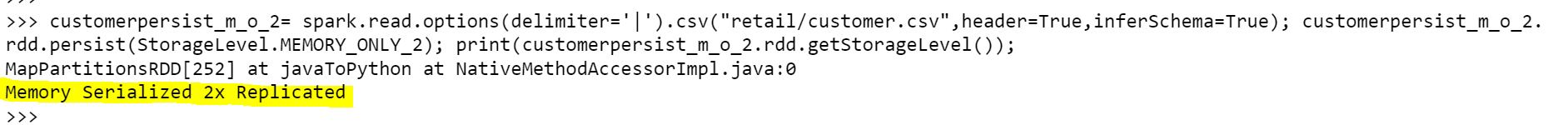
Graphical user interface, text, application, email

Description automatically generated

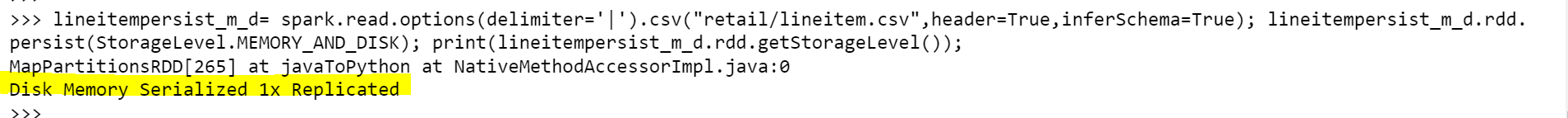
MEMORY ONLY



**MEMORY\_ONLY\_2**



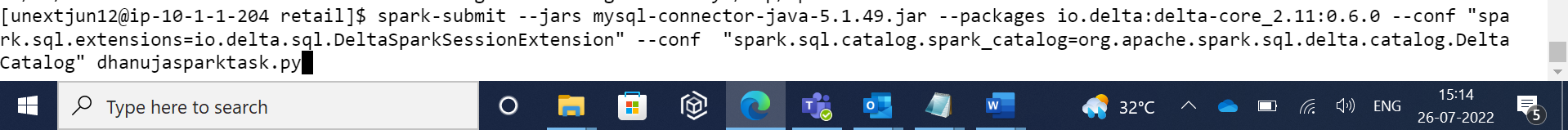
**MEMORY\_AND\_DISK**



**11. Write the .py file for entire spark task and use cache/persist with storage level and deploy that in cluster mode . submit the spark job in YARN cluster**

**Have wrtiten all the spark queries in a .py file 🡪 dhanujasparktask.py**

**I have ran the above file using the below spark submit**



At the same time 15:14 – history server screenshot

Graphical user interface, text, application

Description automatically generated

**All the queries are completed:-**

Graphical user interface

Description automatically generated

**BUISNESS QUERY:-**

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/buisnessquery.py**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/buisnessquery.py)

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

I have created a file buisnessquery.py with all the queries.

Creating dataframe:-

A picture containing graphical user interface

Description automatically generated

**Creating the required column:-**

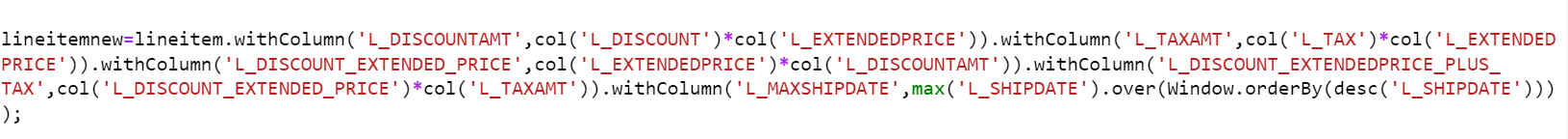
L\_DISCOUNTAMT = >L\_DISCOUNT \* L\_EXTENDEDPRICE

L\_TAXAMT = >L\_TAX \* L\_EXTENDEDPRICE

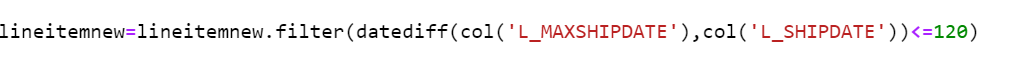
L\_DISCOUNT\_EXTENDED\_PRICE 0 => L\_EXTENDEDPRICE \* L\_DISCOUNTAMT

L\_DISCOUNT\_EXTENDEDPRICE\_PLUS\_ TAX => L\_DISCOUNT\_EXTENDED\_PRICE \* L\_TAXAMT

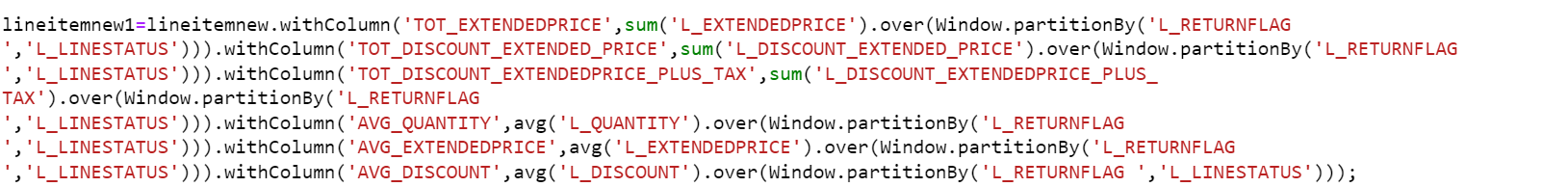
L\_MAXSHIPDATE => MAX(L\_SHIPDATE)



**Filtering the records for 120 days**



**Creating calculated columns:-**



**Joining columns df with count df**

A picture containing diagram

Description automatically generated

**Creating a report with all columns casted to bigint to avoid exponential numbers.**



Table

Description automatically generated

**Count validation with mysql**

Text, letter

Description automatically generated

**12. Kafka task; Spark Integration**

**Create a kafka producer to read the file and topics with replication , partitions**

**Integrate with spark streaming :**

**consume it in spark streams and join with batch data**

**Apply watermark and user defined function (UDF)**

**I have read the files using procer and have consumed the data throgh consumer under the below topics.**

* **order\_unextjun12**
* **lineitem\_unextjun12**

**kafka-topics --create --zookeeper 10.1.1.204 --replication-factor 1 --partitions 1 --topic order\_unextjun12**

**kafka-topics --create --zookeeper 10.1.1.204 --replication-factor 1 --partitions 1 --topic lineitem\_unextjun12**

**I have attached the HTML copy of jupyter notebook for producer and consumer codes.PFA.**

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/Kafka%20Producer%20Lineitem.html**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/Kafka%20Producer%20Lineitem.html)

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/Kafka%20Consumer%20Lineitem.html**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/Kafka%20Consumer%20Lineitem.html)

[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/Kafka%20Producer%20Orders.html**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/Kafka%20Producer%20Orders.html)

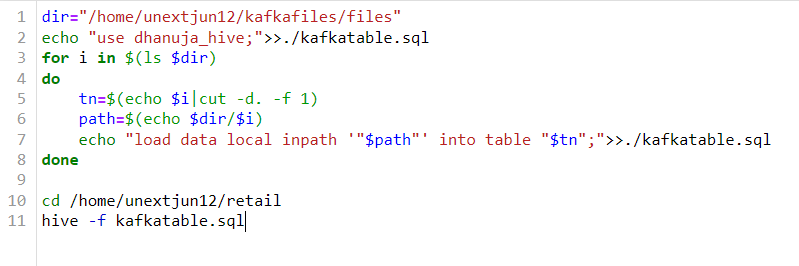
[**https://github.com/DhanujaPani/retail\_capstone\_casestudy\_files/blob/main/Kafka%20Consumer%20Orders.html**](https://github.com/DhanujaPani/retail_capstone_casestudy_files/blob/main/Kafka%20Consumer%20Orders.html)

**Code used to remove double quotes from each line written by consumer:-**

Graphical user interface, text, application

Description automatically generated

**This is the bash script used to load data from the file written by consumer to hive tables:**



Text

Description automatically generated

**Data loaded to hive:-**

Table

Description automatically generated

**13. Implement Jenkins CI/CD**

**1.Created a new pipeline retail\_casestudy\_tar in jenkins**

Graphical user interface, application

Description automatically generated

**Script for pipeline :-**

Graphical user interface, text, application

Description automatically generated

**2.Build now**

**3.Job#1 got triggered and the build was created successfully**

Graphical user interface, application

Description automatically generated

**4.Console Output:-**

Graphical user interface, text, application

Description automatically generated

Graphical user interface, text, application, Word

Description automatically generated

**Validating in the workspace:-**

**/var/lib/jenkins/workspace/retail\_capstone\_tar**

Graphical user interface, text, application, email

Description automatically generated

**Terminal:-**

**Changed diectory to workspace**

Text

Description automatically generated

**Tar file myfile.tar is created in the workspace.**