PROJECT GLADIATOR

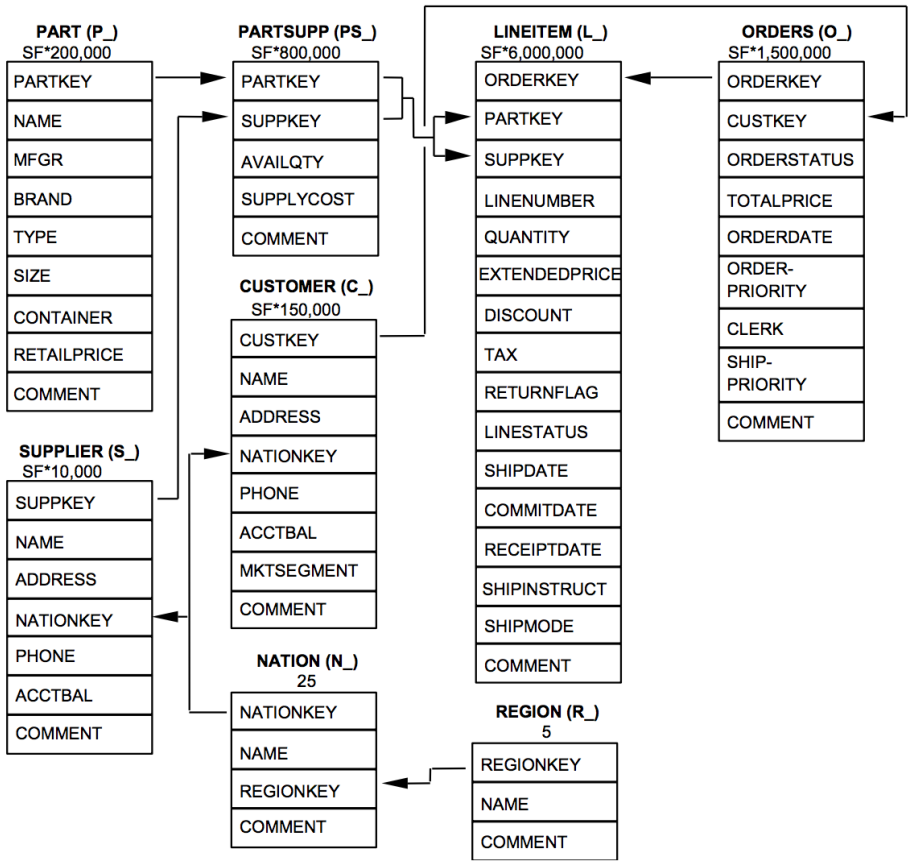
Retail Case Study

Team 8

Team Member
Hansika Khadapkar
Nilesh Kumar
Rahul Noronha

Problem Definition:

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.



Legend:

- The parentheses following each table name contain the prefix of the column names for that table;
- The arrows point in the direction of the one-to-many relationships between tables;
- The number/formula below each table name represents the cardinality (number of rows) of the table. Some are factored by SF, the Scale Factor, to obtain the chosen database size. The cardinality for the LINEITEM table is approximate (see Clause 4.2.5).

Figure 1. TPC-H Schema

Tasks:

1. Create database retail_db and schema retail_schema
2. Create 8 retail tables based on the diagram and apply cluster keys on the most used columns for querying.
3. Add the table definition with constraints , unicode and data retention.
4. Create internal stage and load customer ,lineitem ,nation and orders
5. Load part,partsupp,region and supplier csv files into AWS s3 bucket snow_extstage
6. Load the data from external stage
7. Perform bad records filtering on loading ,file format and compression
8. Create snowpipes to continuously load orders data from external stage from s3 bucket to the order staging table ,maintain the change data capture and merge the data to the consumption table
9. Share your table to new non-snowflake user
10. create clone of the table with time travel before one day and write query to get history data using particular timestamp
11. Create stored procedure to insert the data into table after typecasting date format
12. Create a visualization for sales performance analysis using powerBI and prepare reports /dashboard for business queries

Datasets:

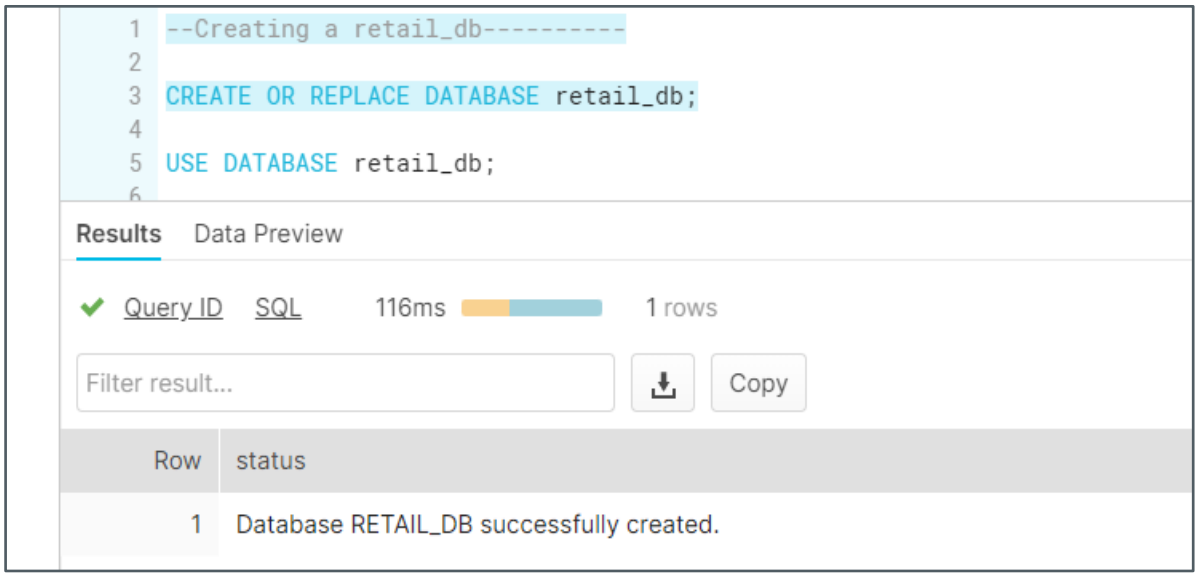
https://drive.google.com/drive/folders/1A3YDOT5ONTsyh6tXPOPEe7pQoGM5N3kj?usp=sharing

We download the dataset in CSV format. There are 8 files, one for each table shown in the TPC-H Schema given in Figure 1.

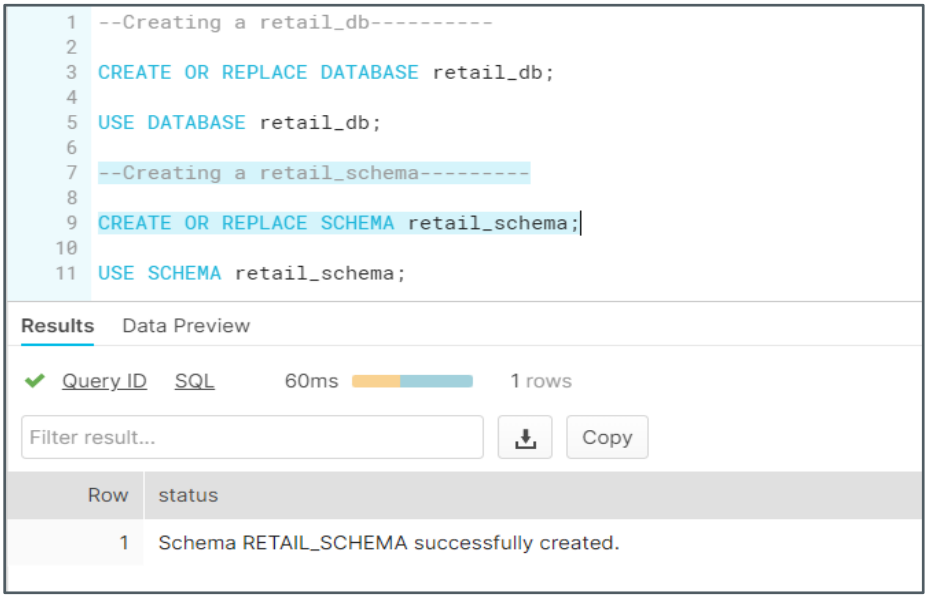
Table Name	Number of Rows
Region	5
Nation	25
Orders	15,000
Customer	1,500
Part	2,000
Supplier	100
PartSupp	8,000
LineItem	60,175

TASK 1: Create database retail_db and schema retail_schema.

1. **The database is created as follows**



2. **The schema is created as follows**

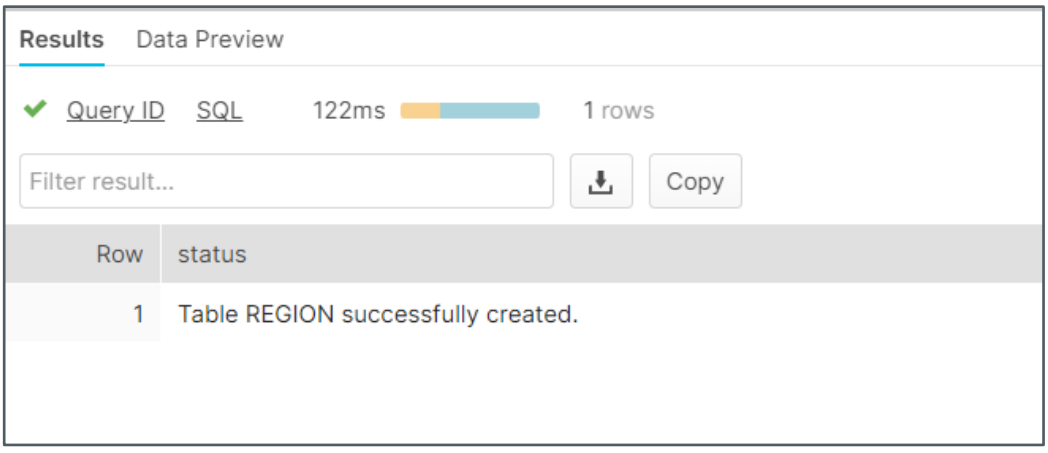


TASK 2: Create 8 retail tables based on the diagram and apply cluster keys on the most used columns for querying.
&
TASK 3: Add the table definition with constraints , unicode and data retention.

1. Region Table: The Region table is created as follows

```
13 CREATE OR REPLACE TABLE Region
14 (
15   R_REGIONKEY number PRIMARY KEY ENFORCED,
16   R_NAME varchar(30),
17   R_COMMENT varchar(1000)
18 )
19 CLUSTER BY (R_NAME)
20 DATA_RETENTION_TIME_IN_DAYS = 90
21 STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
22 ;
```

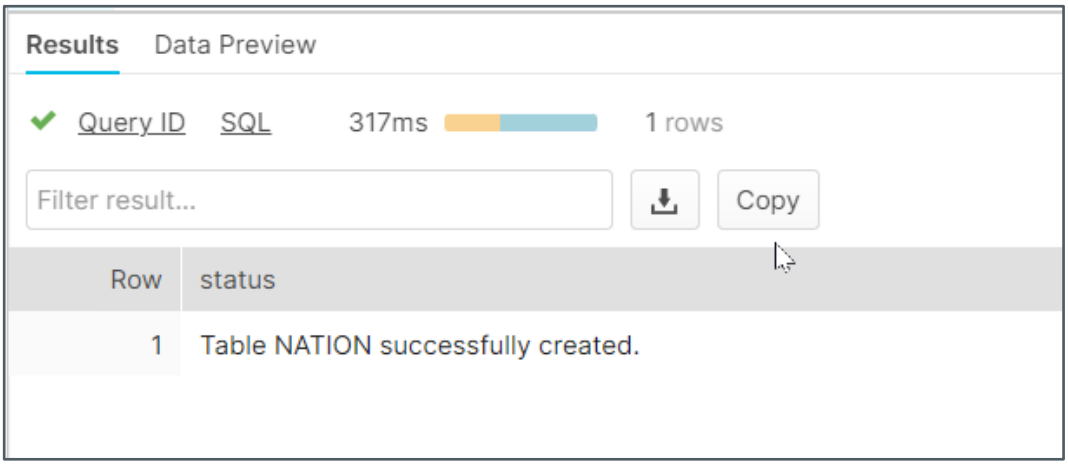
Here we set R_REGIONKEY as the primary key using- PRIMARY KEY
ENFORCED is used to ENFORCE the primary key constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Region table would be the R_NAME since we can identify the region easily using the region name (R_NAME).
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



2. Nation Table: The Nation table is created as follows

```
24 CREATE OR REPLACE TABLE Nation
25 (
26   N_NATIONKEY number PRIMARY KEY ENFORCED,
27   N_NAME varchar(30),
28   N_REGIONKEY number references REGION(R_REGIONKEY) ENFORCED,
29   N_COMMENT varchar(1000)
30 )
31 CLUSTER BY (N_NAME)
32 DATA_RETENTION_TIME_IN_DAYS = 90
33 STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
34 ;
```

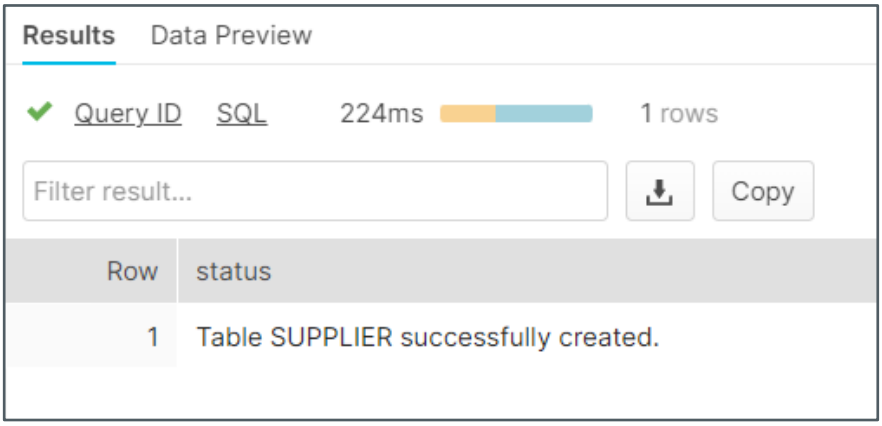
Here we set N_NATIONKEY as the primary key using- PRIMARY KEY
ENFORCED is used to ENFORCE the primary key constraint
We chose the FOREIGN key as N_REGIONKEY which references REGION(R_REGIONKEY) and ENFORCED is used to ENFORCE the constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Nation table would be the N_NAME since we can identify the Nation easily using the Nation name (N_NAME).
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



3. Supplier Table: The Supplier table is created as follows

```
CREATE OR REPLACE TABLE Supplier
(
  S_SUPPKEY number PRIMARY KEY ENFORCED,
  S_NAME varchar(30),
  S_ADDRESS varchar(50),
  S_NATIONKEY number references NATION(N_NATIONKEY) ENFORCED,
  S_PHONE varchar(20),
  S_ACCTBAL float,
  S_COMMENT varchar(1000)
)
CLUSTER BY (S_NAME, S_ACCTBAL)
DATA_RETENTION_TIME_IN_DAYS = 90
STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
;
```

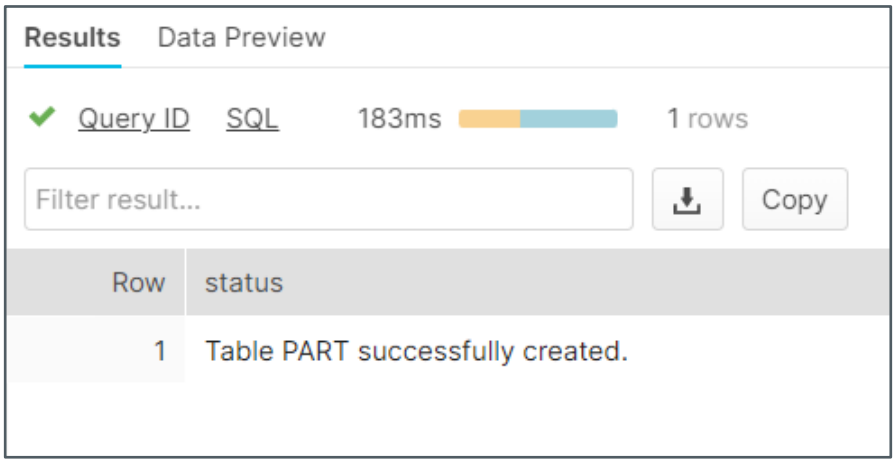
Here we set S_SUPPKEY as the primary key using- PRIMARY KEY
ENFORCED is used to ENFORCE the primary key constraint
We chose the FOREIGN key as S_NATIONKEY which references NATION(N_NATIONKEY) and ENFORCED is used to ENFORCE the constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Supplier table would be the S_NAME and the S_ACCTBAL since we can identify the Supplier easily using the Supplier name (S_NAME) and the S_ACCTBAL seems like it will be queried a lot since it is a quantitative measure of the Supplier which reveals their performance.
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



4. Part Table: The Part table is created as follows

```
CREATE OR REPLACE TABLE Part
(
  P_PARTKEY number PRIMARY KEY ENFORCED,
  P_NAME varchar(60),
  P_MFGR varchar(30),
  P_BRAND varchar(20),
  P_TYPE varchar(30),
  P_SIZE number,
  P_CONTAINER varchar(20),
  P_RETAILPRICE float,
  P_COMMENT varchar(1000)
)
CLUSTER BY (P_NAME, P_SIZE, P_RETAILPRICE)
DATA_RETENTION_TIME_IN_DAYS = 90
STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
;
```

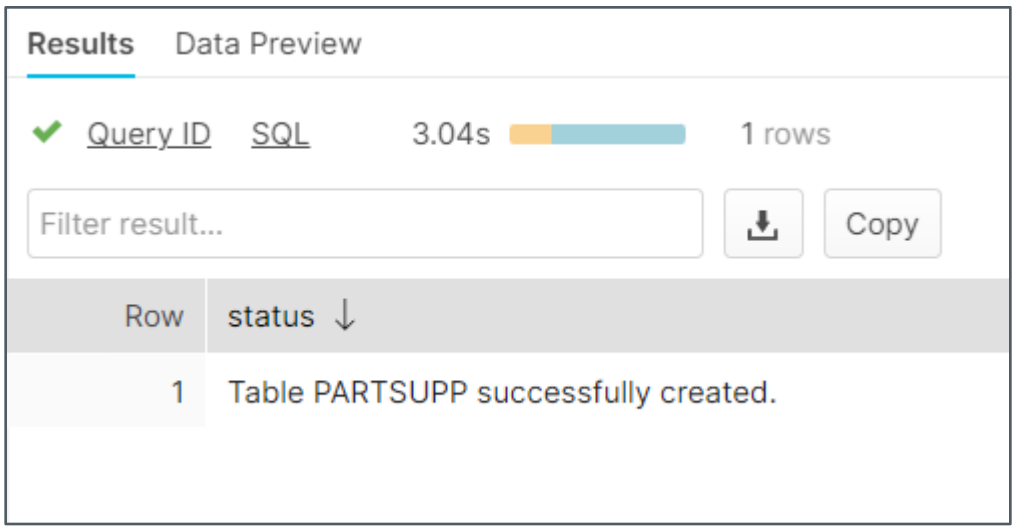
Here we set P_PARTKEY as the primary key using- PRIMARY KEY
ENFORCED is used to ENFORCE the primary key constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Part table would be the P_NAME, P_SIZE, and P_RETAILPRICE since we can identify the Part easily using the Part name (P_NAME) and the fields P_SIZE, and P_RETAILPRICE seem like they will be queried a lot since they are quantitative measures of the Part which reveals about the Part.
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



5. PartSupp Table: The Part-Supplier table is created as follows

```
CREATE OR REPLACE TABLE PartSupp
(
  PS_PARTKEY number UNIQUE references PART(P_PARTKEY) ENFORCED,
  PS_SUPPKEY number UNIQUE references SUPPLIER(S_SUPPKEY) ENFORCED,
  PS_AVAILQTY number,
  PS_SUPPLYCOST float,
  PS_COMMENT varchar(1000)
)
CLUSTER BY (PS_AVAILQTY, PS_SUPPLYCOST)
DATA_RETENTION_TIME_IN_DAYS = 90
STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
;
```

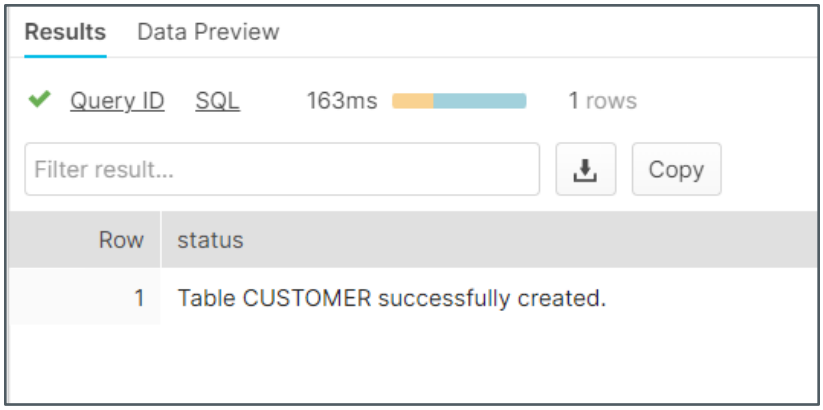
We chose the FOREIGN keys as PS_PARTKEY which references PART(P_PARTKEY) and PS_SUPPKEY which references SUPPLIER(S_SUPPKEY) and ENFORCED is used to ENFORCE the constraint. UNIQUE is added because these are referenced by LINEITEM table which is defined at the end.
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Part table would be the PS_AVAILQTY, and PS_SUPPLYCOST since they seem like they will be queried a lot since they are quantitative measures of the Part-Supplier which reveals about the Part-Supplier.
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



6. Customer Table: The Customer table is created as follows

```
CREATE OR REPLACE TABLE Customer
(
  C_CUSTKEY number PRIMARY KEY ENFORCED,
  C_NAME varchar(30),
  C_ADDRESS varchar(50),
  C_NATIONKEY number references NATION(N_NATIONKEY) ENFORCED,
  C_PHONE varchar(20),
  C_ACCTBAL float,
  C_MKTSEGMENT varchar(20),
  C_COMMENT varchar(1000)
)
CLUSTER BY (C_NAME, C_ACCTBAL)
DATA_RETENTION_TIME_IN_DAYS = 90
STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
;
```

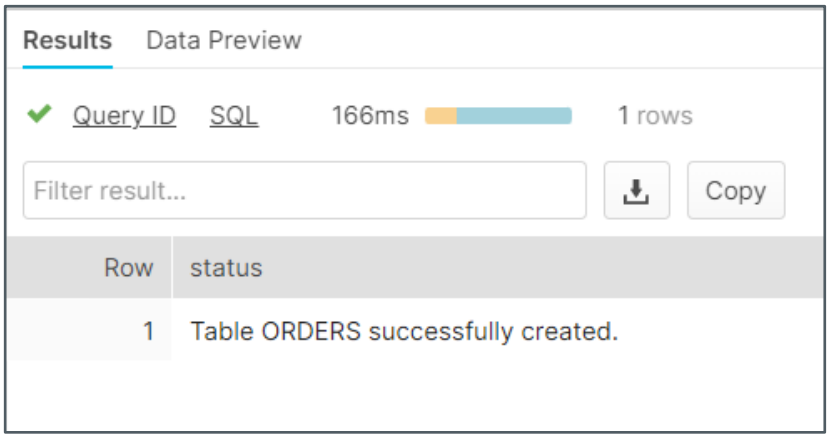
Here we set C_CUSTKEY as the primary key using- PRIMARY KEY
ENFORCED is used to ENFORCE the primary key constraint
We chose the FOREIGN key as C_NATIONKEY which references NATION(N_NATIONKEY) and ENFORCED is used to ENFORCE the constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Customer table would be the C_NAME and the C_ACCTBAL since we can identify the Customer easily using the Customer name (C_NAME) and the C_ACCTBAL seems like it will be queried a lot since it is a quantitative measure of the Customer which reveals their performance.
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



7. Orders Table: The Orders table is created as follows

```
CREATE OR REPLACE TABLE Orders
(
  O_ORDERKEY number PRIMARY KEY ENFORCED,
  O_CUSTKEY number references CUSTOMER(C_CUSTKEY) ENFORCED,
  O_ORDERSTATUS varchar(10),
  O_TOTALPRICE float,
  O_ORDERDATE date,
  O_ORDERPRIORITY varchar(20),
  O_CLERK varchar(20),
  O_SHIPPRIORITY number,
  O_COMMENT varchar(1000)
)
CLUSTER BY (O_ORDERDATE, O_TOTALPRICE )
DATA_RETENTION_TIME_IN_DAYS = 90
STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
;
```

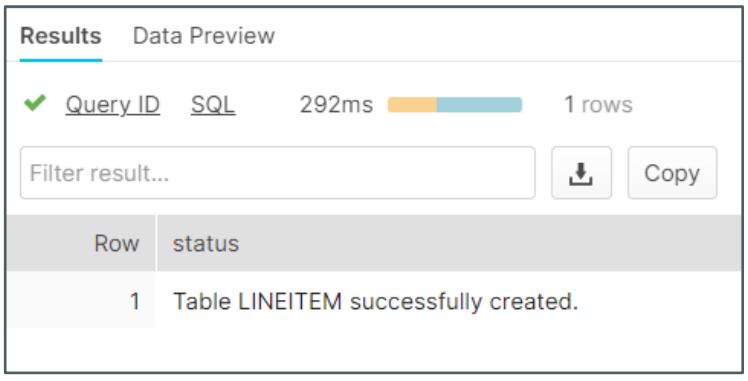
Here we set O_ORDERKEY as the primary key using- PRIMARY KEY
ENFORCED is used to ENFORCE the primary key constraint
We chose the FOREIGN key as O_CUSTKEY which references CUSTOMER(C_CUSTKEY) and ENFORCED is used to ENFORCE the constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the Orders table would be the O_ORDERDATE and the O_TOTALPRICE since they seem like they will be queried a lot being a creation date and quantitative measure respectively of the Customer.
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



8. LineItem Table: The LineItem table is created as follows

```
CREATE OR REPLACE TABLE LineItem
(
  L_ORDERKEY number references ORDERS(O_ORDERKEY) ENFORCED,
  L_PARTKEY number references PARTSUPP(PS_PARTKEY) ENFORCED,
  L_SUPPKEY number references PARTSUPP(PS_SUPPKEY) ENFORCED,
  L_LINENUMBER number,
  L_QUANTITY float,
  L_EXTENDEDPRICE float,
  L_DISCOUNT float,
  L_TAX float,
  L_RETURNFLAG varchar(10),
  L_LINESTATUS varchar(10),
  L_SHIPDATE date,
  L_COMMITDATE date,
  L_RECEIPTDATE date,
  L_SHIPINSTRUCT varchar(30),
  L_SHIPMODE varchar(20),
  L_COMMENT varchar(1000)
)
CLUSTER BY (L_QUANTITY, L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS)
DATA_RETENTION_TIME_IN_DAYS = 90
STAGE_FILE_FORMAT = (TYPE=CSV, ENCODING=UTF8)
;
```

We chose the FOREIGN key as L_ORDERKEY which references ORDERS(O_ORDERKEY), L_PARTKEY which references PART(PS_PARTKEY), and L_SUPPKEY which references PARTSUPP(PS_SUPPKEY) and ENFORCED is used to ENFORCE the constraint
CLUSTER BY (col1, col2, …coln)- Helps us choose cluster keys for our table. We thought the most appropriate CLUSTER KEY for the LineItem table would be L_QUANTITY, L_EXTENDEDPRICE, L_DISCOUNT, L_TAX, L_RETURNFLAG, L_LINESTATUS since they seem like they will be queried a lot being quantitative measures of the LineItem table.
We set the data retention time in days as 90, which is the maximum allowed number of days to retain the data before it goes into fail-safe mode in the Enterprise Edition of Snowflake, which we are using.
We set the STAGE_FILE_FORMAT to TYPE=CSV since we are loading the data from the CSV file and we set the ENCODING to UTF8.



TASK 4: Create Internal stage and load customer, lineitem, nation and orders

1. CREATING FILE FORMAT

```
HANSIKA2000#COMPUTE_WH@(no database).(no schema)>
HANSIKA2000#COMPUTE_WH@(no database).(no schema)>USE DATABASE RETAIL_DB;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
1 Row(s) produced. Time Elapsed: 0.201s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.PUBLIC>USE SCHEMA RETAIL_SCHEMA;
+----------------------------------+
| status                           |
|----------------------------------|
| Statement executed successfully. |
+----------------------------------+
1 Row(s) produced. Time Elapsed: 0.063s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>CREATE FILE FORMAT csv_ff TYPE = 'CSV' COMPRESSION = 'AUTO' FIELD_DELIMITER = '\t'
                                               RECORD_DELIMITER = '\n' SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = 'NONE' TRIM_SPACE = FALSE
                                               ERROR_ON_COLUMN_COUNT_MISMATCH = TRUE ESCAPE = 'NONE' ESCAPE_UNENCLOSED_FIELD = '\134'
                                               DATE_FORMAT = 'AUTO' TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('\\N');
+----------------------------------------+
| status                                 |
|----------------------------------------|
| File format CSV_FF successfully created. |
+----------------------------------------+
1 Row(s) produced. Time Elapsed: 0.097s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

We have created a file format named **csv_ff** that describes a set of staged data to access or load into Snowflake tables.

Parameters:

TYPE='CSV' : TYPE Specifies the format of the input files (for data loading) or output files (for data unloading). We have set this parameter to CSV as the input files are in CSV format.

COMPRESSION='AUTO' : When loading data, specifies the current compression algorithm for the data file and when unloading data, compresses the data file using the specified compression algorithm. We have set this parameter to AUTO so that the compression algorithm is detected automatically. When unloading data, files are automatically compressed using the default, which is gzip.

FIELD_DELIMITER='\t' : This parameter specifies one or more single byte or multibyte characters that separates fields in an input file or unloaded file. We have set this parameter to '\t' since we observed the fields to be separated by '\t'.

RECORD_DELIMITER='\n' : This parameter specifies one or more single byte or multibyte characters that separate records in an input file or unloaded file. We have set this parameter to '\n'.

SKIP_HEADER=1 : Specifies number of lines at the start of the file to skip. We have set this parameter to 1 as we are skipping the header line.

FIELD_OPTIONALLY_ENCLOSED_BY='NONE' : Character used to enclose strings. When a field contains this character, it escapes it using the same character.

TRIM_SPACE= FALSE : Boolean that specifies whether to remove white space from fields. When set to FALSE, snowflake reads the leading space if any, as the beginning of the field.

ERROR_ON_COLUMN_COUNT_MISMATCH=TRUE : Boolean that specifies whether to generate a parsing error if the number of delimited columns (i.e. fields) in an input file does not match the number of columns in the corresponding table. When set to TRUE, an error is generated on column count mismatch.

ESCAPE=NONE : A single byte character string used as the escape character for enclosed or unenclosed field values. You can use the ESCAPE character to interpret instances of the FIELD_DELIMITER or RECORD_DELIMITER characters in the data as literals.

ESCAPE_UNENCLOSED_FIELD=NONE : Specifies the escape character for unenclosed fields only.

DATE_FORMAT=AUTO : Defines the format of date string values in the data files. AUTO specifies that Snowflake attempts to automatically detect the format of dates stored in the system during the session.

TIME_FORMAT=AUTO : Defines the format of timestamp string values in the data files. AUTO specifies that Snowflake attempts to automatically detect the format of timestamps stored in the system during the session.

NULL_IF = ('\\N') : String used to convert to and from SQL NULL. Snowflake replaces these strings in the data load source with SQL NULL.

2. LOADING nation table using named internal stage

In the below step, we have created a named internal stage called my_internal_stage. By specifying a named file format object for the stage, it is not necessary to later specify the same file format options in the COPY command used to load data from the stage.

```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>CREATE OR REPLACE STAGE my_internal_stage FILE_FORMAT = csv_ff;
+-----------------------------------------------+
| status                                        |
|-----------------------------------------------|
| Stage area MY_INTERNAL_STAGE successfully created. |
+-----------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.119s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>PUT file:///D:\Snowflake\nation.csv @my_internal_stage;
+------------+--------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source     | target       | source_size | target_size | source_compression | target_compression | status   | message |
|------------+--------------+-------------+-------------+--------------------+--------------------+----------+---------|
| nation.csv | nation.csv.gz |        2240 |        1008 | NONE               | GZIP               | UPLOADED |         |
+------------+--------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 3.511s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO "RETAIL_DB"."RETAIL_SCHEMA"."NATION" FROM @my_internal_stage/nation.csv.gz
                                        VALIDATION_MODE = 'RETURN_ERRORS';
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
| ERROR | FILE | LINE | CHARACTER | BYTE_OFFSET | CATEGORY | CODE | SQL_STATE | COLUMN_NAME | ROW_NUMBER | ROW_START_LINE | REJECTED_RECORD |
|-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------|
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
0 Row(s) produced. Time Elapsed: 2.683s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO "RETAIL_DB"."RETAIL_SCHEMA"."NATION" FROM @my_internal_stage/nation.csv.gz
                                        PURGE=TRUE ON_ERROR=CONTINUE;
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file                          | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| my_internal_stage/nation.csv.gz | LOADED |          25 |          25 |          25 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 1.402s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

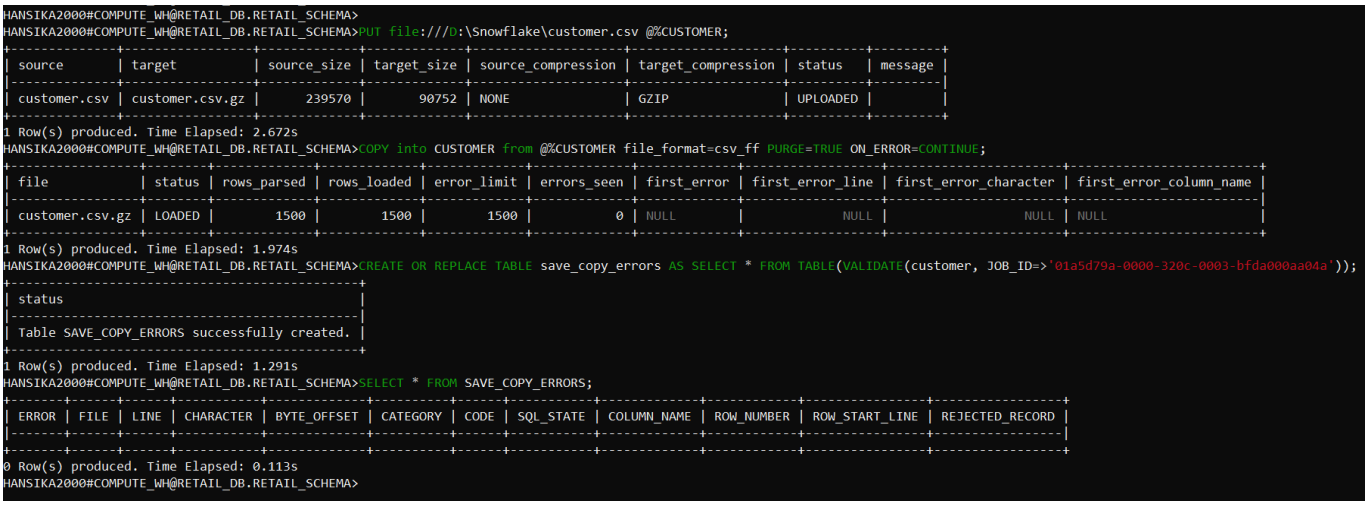
- PUT command is used to upload a file named nation.csv on local machine to a named internal stage called my_internal_stage
- VALIDATION_MODE parameter in a COPY statement is used to validate the staged data rather than load it into the target table.
- COPY INTO <table> loads data from file nation.csv.gz in my_internal_stage into nation table
- PURGE=TRUE is used to purge all files that are successfully loaded into the table
- ON_ERROR = CONTINUE continues to load the file with valid data if errors are found.

3. LOADING lineitem table and customer table using table stage

In the below step, we have loaded the tables lineitem and customer using the table stage.

- Each table has a Snowflake stage allocated to it by default for storing files
- Table stages have the same name as the table
- Unlike named stages, table stages cannot be altered or dropped
- Table stages do not support setting file format options. Instead, you must specify file format and copy options as part of the COPY INTO <table> command

```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>PUT file:///D:\Snowflake\lineitem.csv @%LINEITEM;
+--------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source       | target         | source_size | target_size | source_compression | target_compression | status   | message |
|--------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------|
| lineitem.csv | lineitem.csv.gz |     7384788 |     2042352 | NONE               | GZIP               | UPLOADED |         |
+--------------+----------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 7.062s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY into LINEITEM from @%LINEITEM file_format=csv_ff PURGE=TRUE ON_ERROR=CONTINUE;
+-----------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file            | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|-----------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| lineitem.csv.gz | LOADED |       60175 |       60175 |       60175 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+-----------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 3.200s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>CREATE OR REPLACE TABLE save_copy_errors AS SELECT * FROM TABLE(VALIDATE(lineitem, JOB_ID=>'01a5d75a-0000-3217-0003-bfda0009572e'));
+-------------------------------------------+
| status                                    |
|-------------------------------------------|
| Table SAVE_COPY_ERRORS successfully created. |
+-------------------------------------------+
1 Row(s) produced. Time Elapsed: 0.433s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>SELECT * FROM SAVE_COPY_ERRORS;
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
| ERROR | FILE | LINE | CHARACTER | BYTE_OFFSET | CATEGORY | CODE | SQL_STATE | COLUMN_NAME | ROW_NUMBER | ROW_START_LINE | REJECTED_RECORD |
|-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------|
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
0 Row(s) produced. Time Elapsed: 0.100s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```



```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>PUT file:///D:\Snowflake\customer.csv @%CUSTOMER;
+--------------+-----------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source       | target          | source_size | target_size | source_compression | target_compression | status   | message |
|--------------+-----------------+-------------+-------------+--------------------+--------------------+----------+---------|
| customer.csv | customer.csv.gz |      239570 |       90752 | NONE               | GZIP               | UPLOADED |         |
+--------------+-----------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 2.672s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY into CUSTOMER from @%CUSTOMER file_format=csv_ff PURGE=TRUE ON_ERROR=CONTINUE;
+-----------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+----------------------+--------------------------+
| file            | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|-----------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+----------------------+--------------------------|
| customer.csv.gz | LOADED |        1500 |        1500 |        1500 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+-----------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+----------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 1.974s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>CREATE OR REPLACE TABLE save_copy_errors AS SELECT * FROM TABLE(VALIDATE(customer, JOB_ID=>'01a5d79a-0000-320c-0003-bfda000aa04a'));
+-----------------------------------------+
| status                                  |
|-----------------------------------------|
| Table SAVE_COPY_ERRORS successfully created. |
+-----------------------------------------+
1 Row(s) produced. Time Elapsed: 1.291s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>SELECT * FROM SAVE_COPY_ERRORS;
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
| ERROR | FILE | LINE | CHARACTER | BYTE_OFFSET | CATEGORY | CODE | SQL_STATE | COLUMN_NAME | ROW_NUMBER | ROW_START_LINE | REJECTED_RECORD |
|-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------|
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
0 Row(s) produced. Time Elapsed: 0.113s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

- PUT command is used to upload the files on local machine to the respective stages for the tables lineitem and customer.
- COPY INTO <table> loads data from the table stage using the named csv_ff file format into table
- The VALIDATE table function is used to view all errors encountered during the previous load.
- PURGE=TRUE is used to purge all files that are successfully loaded into the table
- ON_ERROR = CONTINUE continues to load the file with valid data if errors are found.

4. LOADING orders table from user stage

In the below step, we have loaded the table orders using the user stage.

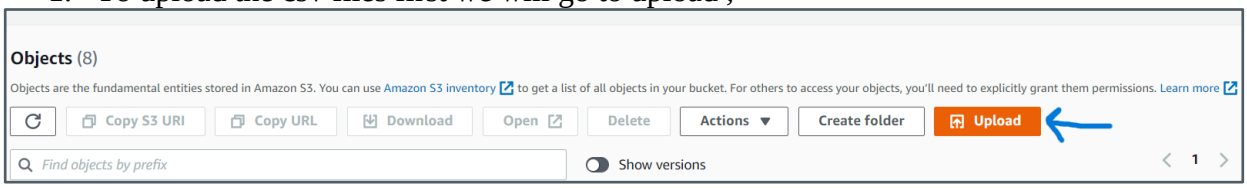
- Each user has a Snowflake stage allocated to them by default for storing files.
- User stages are referenced using @~
- Unlike named stages, user stages cannot be altered or dropped
- User stages do not support setting file format options. Instead, you must specify file format and copy options as part of the COPY INTO <table> command

```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>PUT file:///D:\Snowflake\orders.csv @~;
+------------+--------------+-------------+-------------+--------------------+--------------------+----------+---------+
| source     | target       | source_size | target_size | source_compression | target_compression | status   | message |
|------------+--------------+-------------+-------------+--------------------+--------------------+----------+---------|
| orders.csv | orders.csv.gz |     1644246 |      451744 | NONE               | GZIP               | UPLOADED |         |
+------------+--------------+-------------+-------------+--------------------+--------------------+----------+---------+
1 Row(s) produced. Time Elapsed: 2.212s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO ORDERS FROM @~/orders.csv.gz FILE_FORMAT=csv_ff VALIDATION_MODE = 'RETURN_ERRORS';
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
| ERROR | FILE | LINE | CHARACTER | BYTE_OFFSET | CATEGORY | CODE | SQL_STATE | COLUMN_NAME | ROW_NUMBER | ROW_START_LINE | REJECTED_RECORD |
|-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------|
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
0 Row(s) produced. Time Elapsed: 2.107s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO ORDERS FROM @~/orders.csv.gz FILE_FORMAT=csv_ff PURGE=TRUE ON_ERROR=CONTINUE;
+---------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file          | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|---------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| orders.csv.gz | LOADED |       15000 |       15000 |       15000 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+---------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 2.155s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

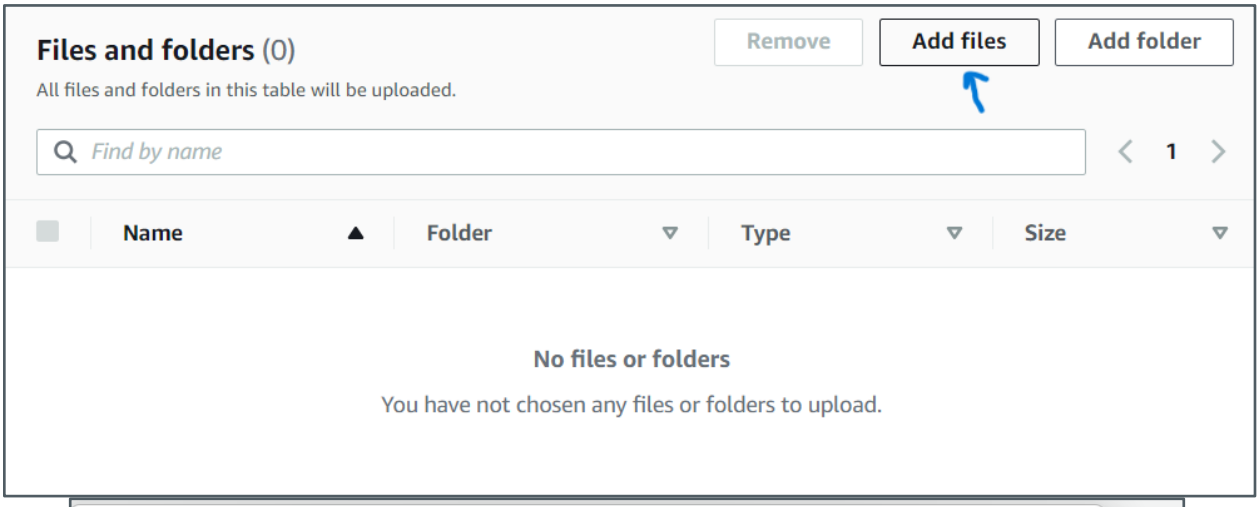
- PUT command is used to upload a file named orders.csv on local machine to the user stage.
- COPY INTO <table> loads data from your user stage using the named csv_ff file format
- PURGE=TRUE is used to purge all files that are successfully loaded into the table
- ON_ERROR = CONTINUE continues to load the file with valid data if errors are found.

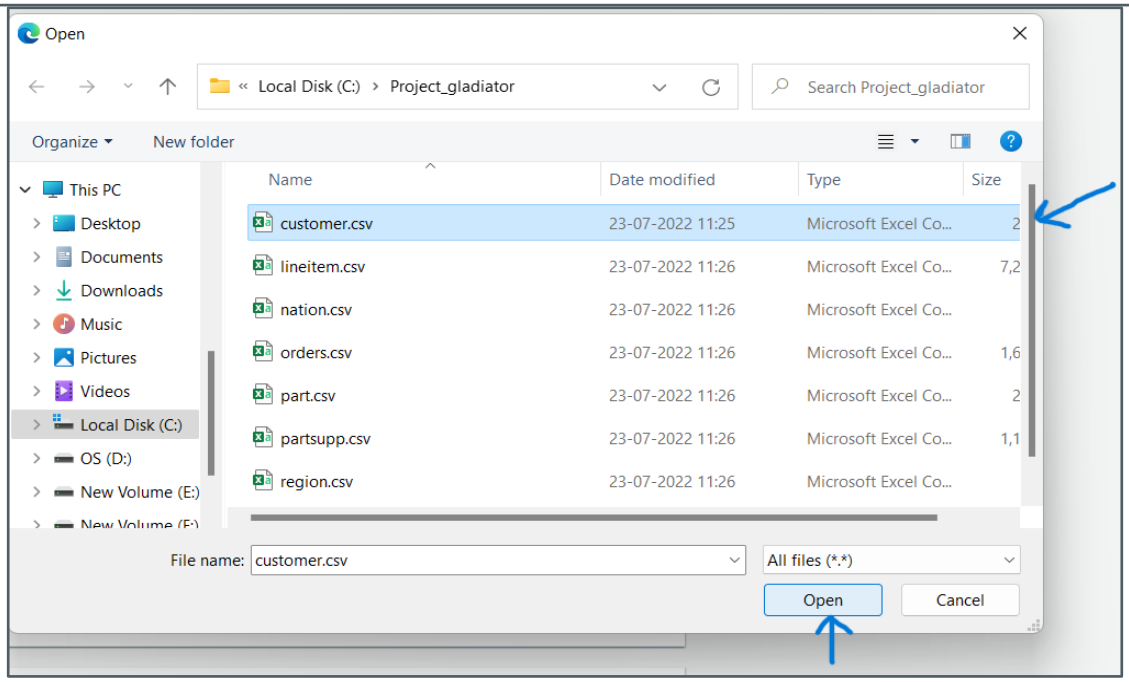
TASK 5: Load part,partsupp,region and supplier csv files into AWS S3 bucket Snow_extstage.

1. To upload the csv files first we will go to upload ,

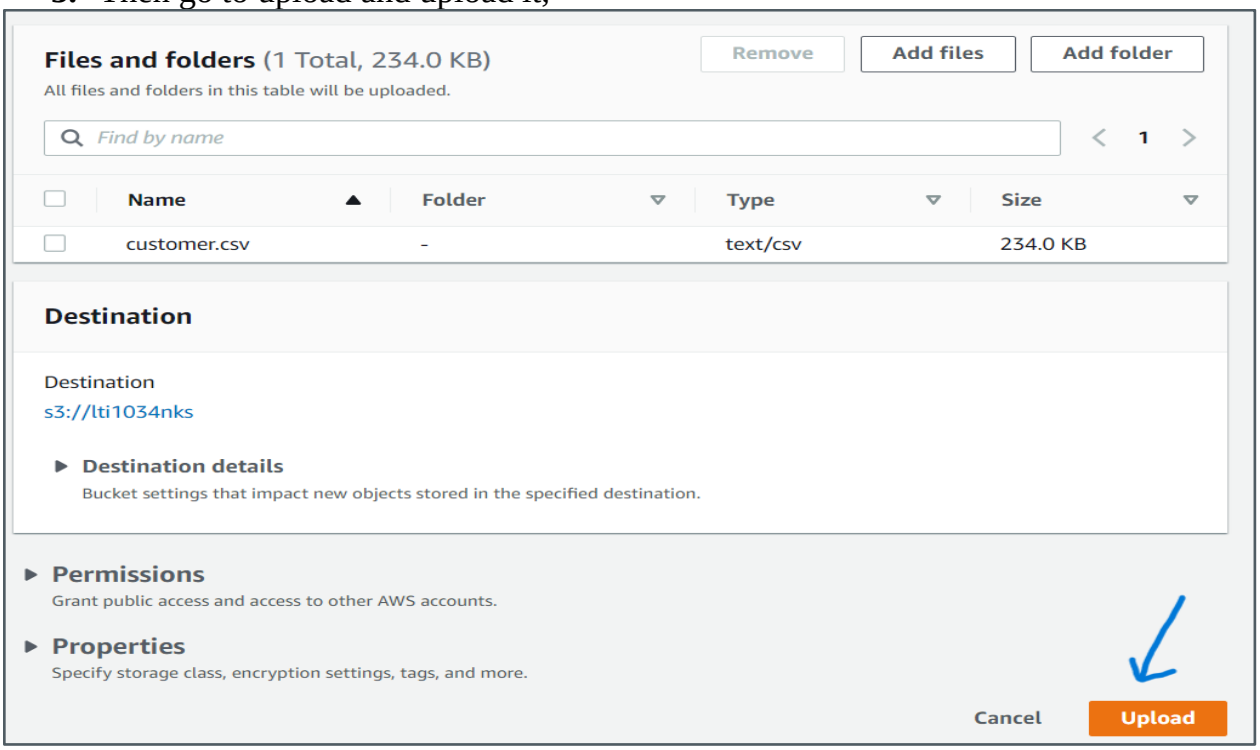


2. Then we will go to Add Files to choose the csv file from our local drive,

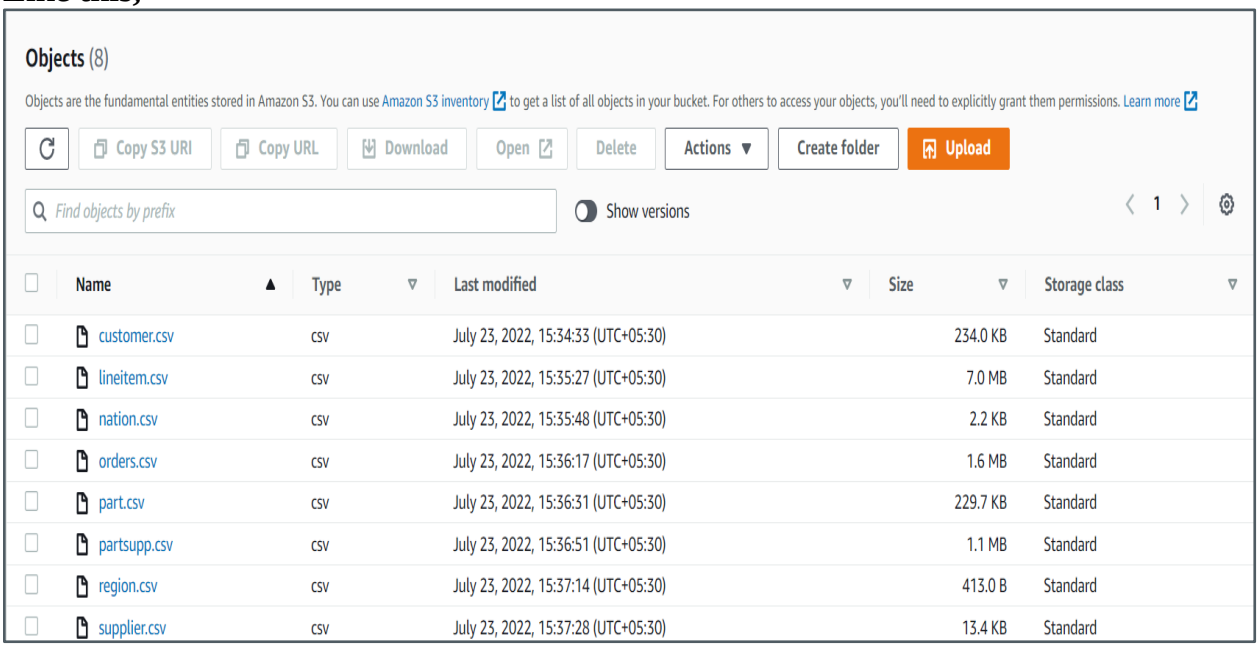




3. Then go to upload and upload it,

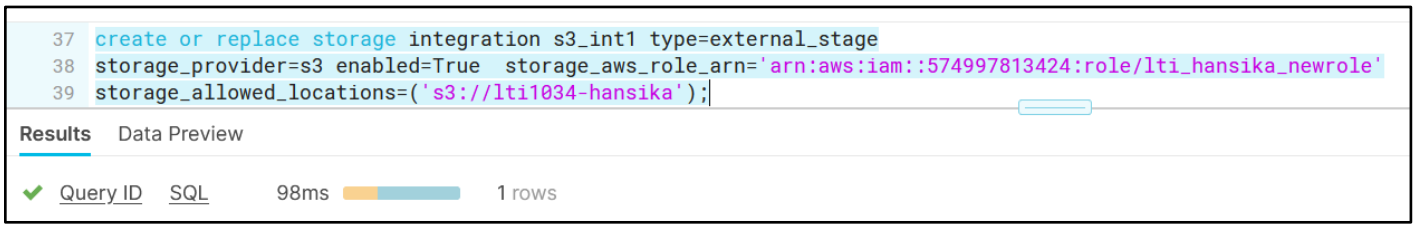


Like this,



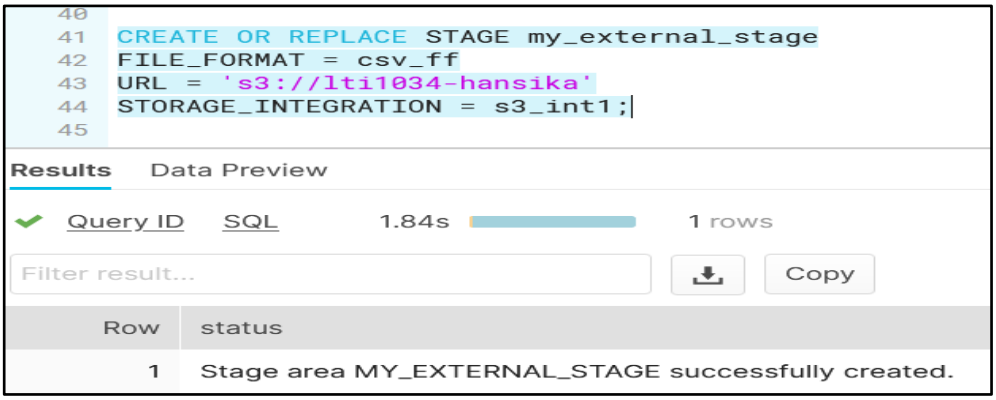
TASK 6: LOAD DATA FROM EXTERNAL STAGE(AWS S3)
&
TASK 7: Perform bad records filtering on loading ,file format and compression

- **Creating storage integration**
  In the below step we have created a new storage integration named s3_int1. This allows users to avoid supplying credentials when creating stages or when loading or unloading data.



  - TYPE = EXTERNAL_STAGE : Specifies the type of integration. EXTERNAL_STAGE creates an interface between Snowflake and an external cloud storage location.
  - STORAGE_PROVIDER = S3 : Specifies the cloud storage provider that stores the data files.
  - ENABLED = TRUE : Specifies whether this storage integration is available for usage in stages. TRUE allows users to create new stages that reference this integration.
  - STORAGE_AWS_ROLE_ARN = *iam_role* : Specifies the Amazon Resource Name (ARN) of the AWS identity and access management (IAM) role that grants privileges on the S3 bucket containing the data files.
  - STORAGE_ALLOWED_LOCATIONS = ('*cloud_specific_url*') : This value includes a cloud storage URL where required files are present

- **Creating external stage**
  In the below step we have created an external stage named my_external_stage that references a S3 bucket.



  - FILE_FORMAT : A file format named csv_ff created earlier is used
  - URL : The cloud storage URL specifies the storage location of the required files
  - Storage_Integration: Secure access to the S3 bucket is provided via the s3_int1 storage integration.

- Loading from external stage

    - Snowflake supports transforming data while loading it into a table using the COPY INTO <table> command. Column reordering, column omission, and casts can be performed using a SELECT statement.
    - LTRIM() removes leading characters, including whitespace, from a string. In our case, this function is used to trim the strings (eg: from 'Manufacturer#1' to '1')
    - ON_ERROR = CONTINUE continues to load the file with valid data if errors are found.
    - FORCE = TRUE is specified as one of the copy options, so that the command does not ignore the staged data files that were already loaded into the table.

```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO PART(P_PARTKEY,P_NAME,P_MFGR,P_BRAND,P_TYPE,P_SIZE,P_CONTAINER,P_RETAILPRICE,P_COMMENT)
                                               FROM (select t.$1,t.$2,LTRIM(t.$3, 'Manufacturer#'),LTRIM(t.$4, 'Brand#'),t.$5,t.$6,t.$7,t.$8,t.$9 from @my_external_stage/part.csv t)
                                               ON_ERROR=CONTINUE FORCE=TRUE;
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file                          | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| s3://lti1034-hansika/part.csv | LOADED |        2000 |        2000 |        2000 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 5.586s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>CREATE OR REPLACE TABLE save_copy_errors AS SELECT * FROM TABLE(VALIDATE(part, JOB_ID=>'01a5d789-0000-321f-0003-bfda000a9022'));

+-------------------------------------------+
| status                                    |
|-------------------------------------------|
| Table SAVE_COPY_ERRORS successfully created. |
+-------------------------------------------+
1 Row(s) produced. Time Elapsed: 4.389s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>SELECT * FROM SAVE_COPY_ERRORS;
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
| ERROR | FILE | LINE | CHARACTER | BYTE_OFFSET | CATEGORY | CODE | SQL_STATE | COLUMN_NAME | ROW_NUMBER | ROW_START_LINE | REJECTED_RECORD |
|-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------|
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
0 Row(s) produced. Time Elapsed: 0.096s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

Hence, the part table is loaded using an external stage which loads data from the s3 bucket.

```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO SUPPLIER(S_SUPPKEY,S_NAME,S_ADDRESS,S_NATIONKEY,S_PHONE,S_ACCTBAL,S_COMMENT)
                                               FROM (select t.$1,LTRIM(t.$2, 'Supplier#0'),t.$3,t.$4,t.$5,t.$6,t.$7 from @my_external_stage/supplier.csv t)
                                               ON_ERROR=CONTINUE FORCE=TRUE;
+---------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file                            | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|---------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| s3://lti1034-hansika/supplier.csv | LOADED |         100 |         100 |         100 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+---------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 0.768s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>CREATE OR REPLACE TABLE save_copy_errors AS SELECT * FROM TABLE(VALIDATE(supplier, JOB_ID=>'01a5d793-0000-3223-0003-bfda000ab016'));

+-------------------------------------------+
| status                                    |
|-------------------------------------------|
| Table SAVE_COPY_ERRORS successfully created. |
+-------------------------------------------+
1 Row(s) produced. Time Elapsed: 4.125s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>SELECT * FROM SAVE_COPY_ERRORS;
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
| ERROR | FILE | LINE | CHARACTER | BYTE_OFFSET | CATEGORY | CODE | SQL_STATE | COLUMN_NAME | ROW_NUMBER | ROW_START_LINE | REJECTED_RECORD |
|-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------|
+-------+------+------+-----------+-------------+----------+------+-----------+-------------+------------+----------------+-----------------+
0 Row(s) produced. Time Elapsed: 0.090s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

Hence, the supplier table is loaded using an external stage which loads data from the s3 bucket.

```
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO PARTSUPP FROM @my_external_stage/partsupp.csv  ON_ERROR=CONTINUE FORCE=TRUE;
+--------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file                           | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|--------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| s3://lti1034-hansika/partsupp.csv | LOADED |        8000 |        8000 |        8000 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+--------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 4.628s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>COPY INTO REGION FROM @my_external_stage/region.csv  ON_ERROR=CONTINUE FORCE=TRUE;
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
| file                          | status | rows_parsed | rows_loaded | error_limit | errors_seen | first_error | first_error_line | first_error_character | first_error_column_name |
|-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------|
| s3://lti1034-hansika/region.csv | LOADED |           5 |           5 |           5 |           0 | NULL        |             NULL |                 NULL | NULL                     |
+-------------------------------+--------+-------------+-------------+-------------+-------------+-------------+------------------+--------------------+--------------------------+
1 Row(s) produced. Time Elapsed: 1.741s
HANSIKA2000#COMPUTE_WH@RETAIL_DB.RETAIL_SCHEMA>
```

Hence, the partsupp and region tables are loaded using an external stage which loads data from the s3 bucket.

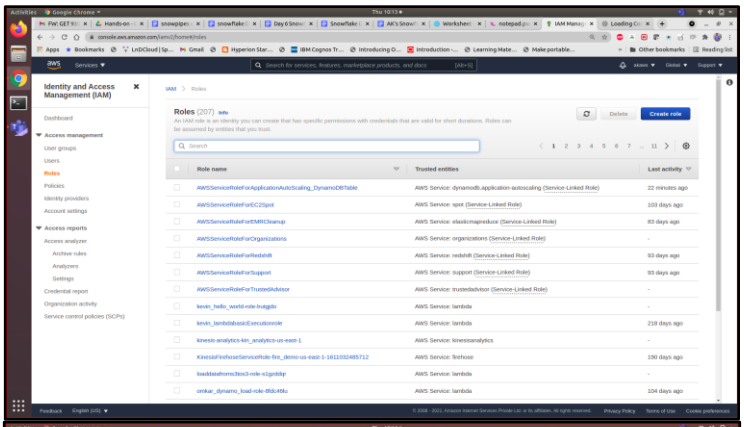
TASK 8: Create snowpipes to continuously load orders data from external stage from S3

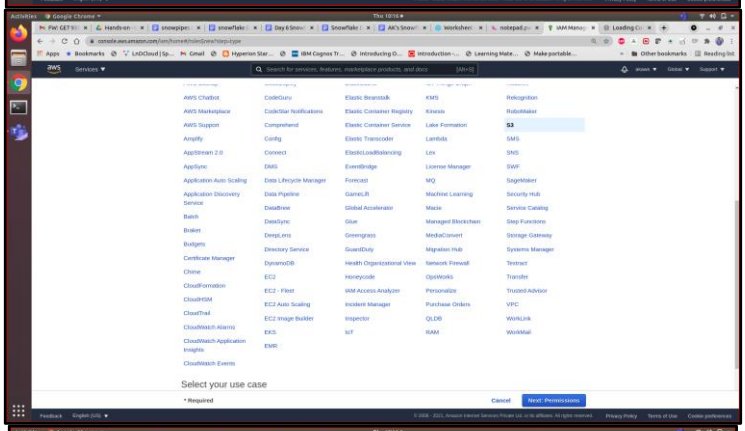
bucket to the order staging table, maintain the change data capture and merge the data to the consumption table.

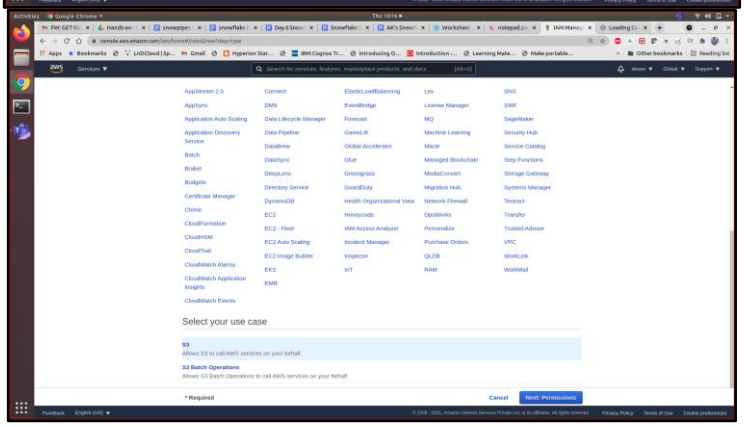
1. To see the details of the storage integration use the following command

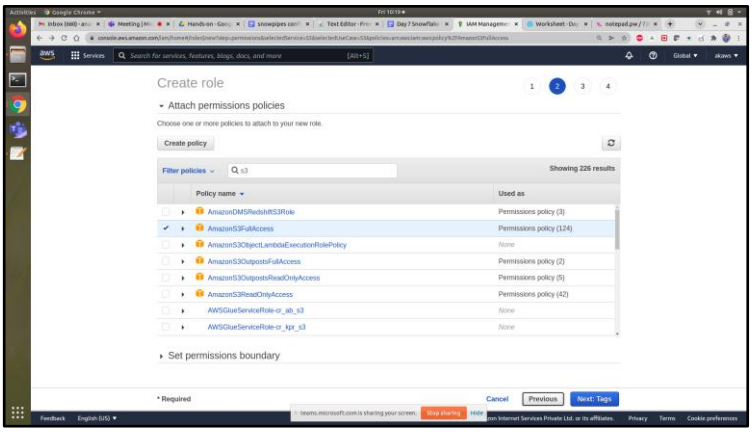
```
desc integration s3_int;
```

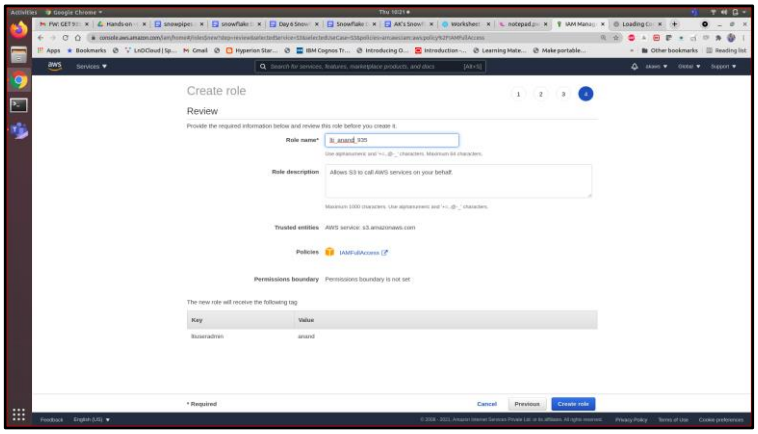
2. Create IAM role : to get arn

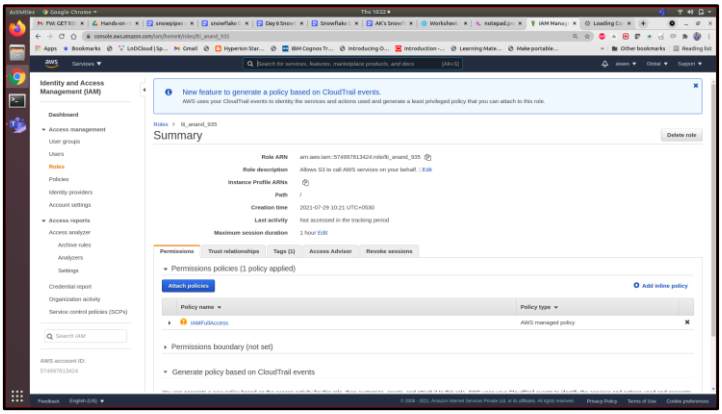






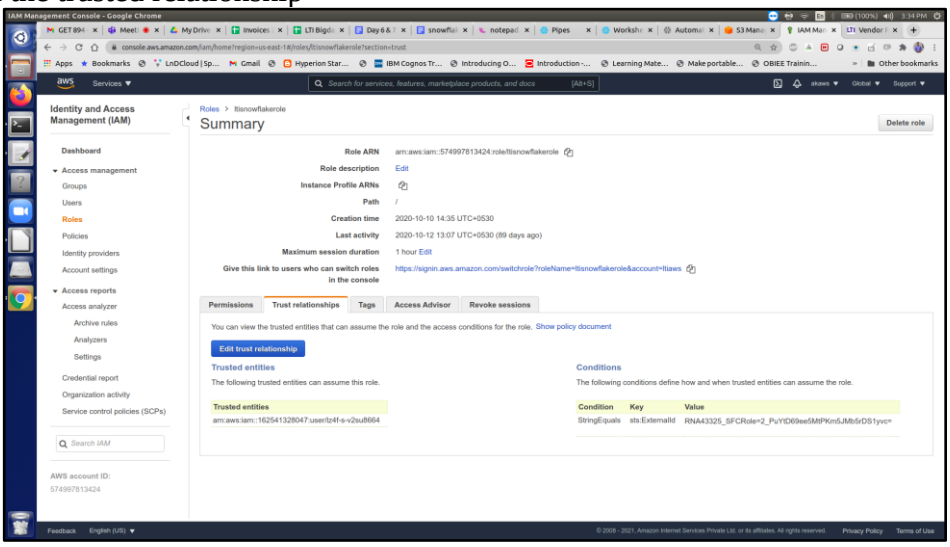




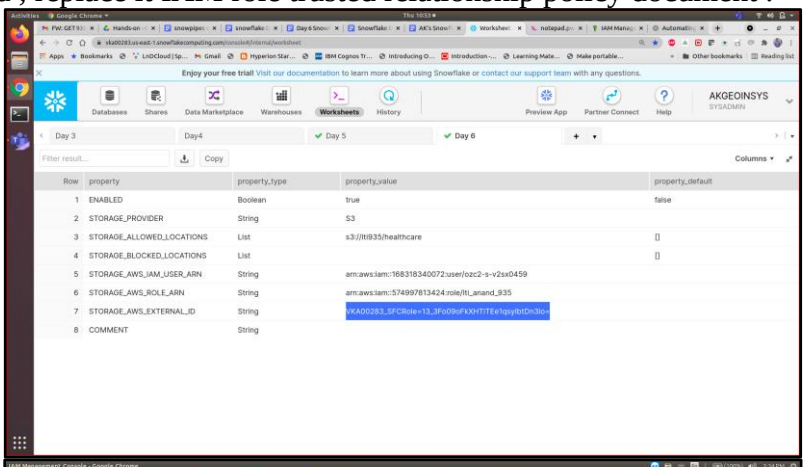


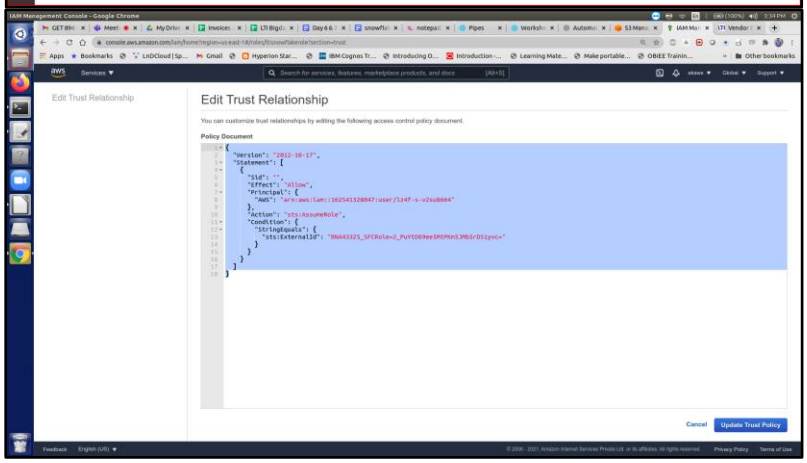
3. create storage integration with iam arn
   create storage integration s3_int
   type = external_stage
   storage_provider = s3
   enabled = true
   storage_aws_role_arn = 'arn:aws:iam::<arn_id>:role/<rolename>'
   storage_allowed_locations = ('s3://<bucketlocation>');
   Desc integration s3_int;

4. Add the trusted relationship



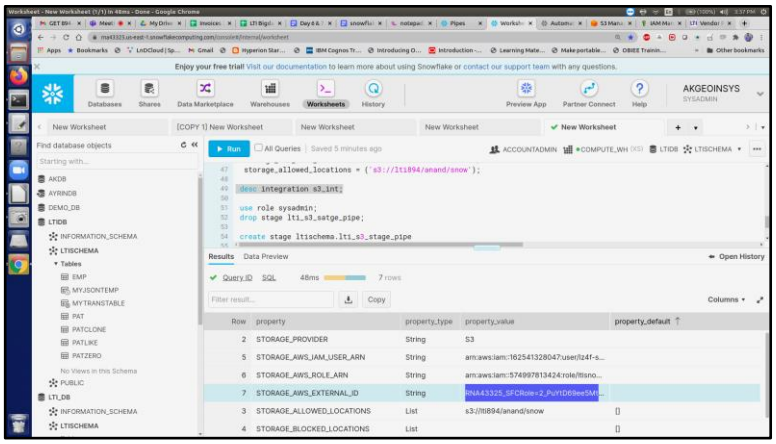
5. run desc integration command in snowflake worksheet and get the snowflake arn and external id , replace it IAM role trusted relationship policy document .





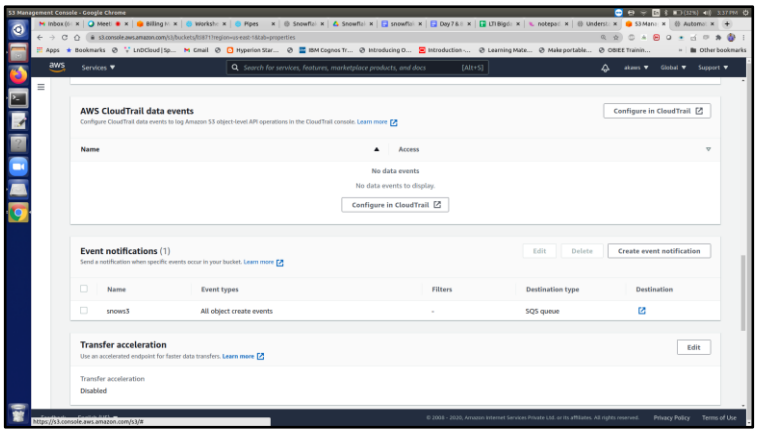
```
{
 "Version": "2012-10-17",
 "Statement": [
  {
   "Sid": "",
   "Effect": "Allow",
   "Principal": {
    "AWS": "<snowflake_user_arn>"
   },
   "Action": "sts:AssumeRole",
   "Condition": {
    "StringEquals": {
     "sts:ExternalId": "<snowflake_external_id>"
    }
   }
  }
 ]
}
```

6. Use the external id and snowflake arn from desc integration s3_int;

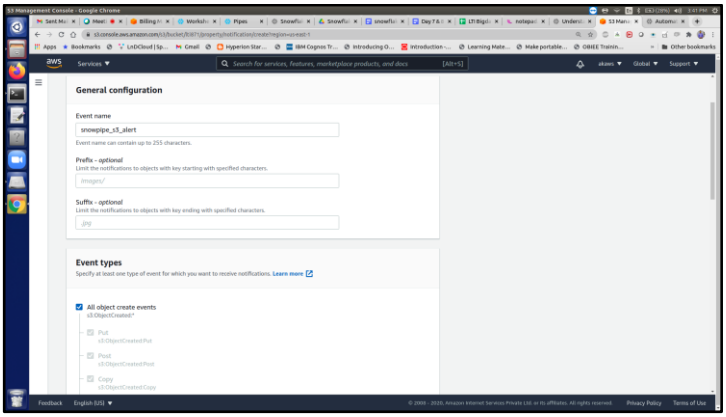


7. Create the external stage for s3 with integration
8. Create the pipe and map source and destination

9. In s3 bucket , create the event notification snowflake with sqs which you get as notification channel from command Show pipes :
    a. Create event in s3 bucket



    b. Select all object creation



    c. Log into the Amazon S3 console.
        i. Configure an event notification for your S3 bucket using the instructions provided in the Amazon S3 documentation. Complete the fields as follows:
        Name: Name of the event notification (e.g. Auto-ingest Snowflake).
        Events: Select the **ObjectCreate (All)** option.
        Send to: Select **SQS Queue** from the dropdown list.
        SQS: Select **Add SQS queue ARN** from the dropdown list.
        SQS queue ARN: Paste the SQS queue name from the SHOW PIPES output.
        ii. snow pipe auto ingest the data from s3 bucket when you create or copy file to the stage location.

10. Create the table CDC_Orders and Orders_History to track the CDC(Changing Data Capture) and the Historical records respectively. They are created as follows

```
CREATE OR REPLACE TABLE CDC_Orders
(
  O_ORDERKEY number PRIMARY KEY ENFORCED,
  O_CUSTKEY number references CUSTOMER(C_CUSTKEY) ENFORCED,
  O_ORDERSTATUS varchar(10),
  O_TOTALPRICE float,
  O_ORDERDATE date,
  O_ORDERPRIORITY varchar(20),
  O_CLERK varchar(20),
  O_SHIPPRIORITY number,
  O_COMMENT varchar(1000),
  update_timestamp timestamp_ntz default current_timestamp()::timestamp_ntz
);
```

```
CREATE OR REPLACE TABLE Orders_History
(
  O_ORDERKEY number PRIMARY KEY ENFORCED,
  O_CUSTKEY number references CUSTOMER(C_CUSTKEY) ENFORCED,
  O_ORDERSTATUS varchar(10),
  O_TOTALPRICE float,
  O_ORDERDATE date,
  O_ORDERPRIORITY varchar(20),
  O_CLERK varchar(20),
  O_SHIPPRIORITY number,
  O_COMMENT varchar(1000),
  start_time timestamp_ntz,
  end_time timestamp_ntz,
  current_flag int
);
```

Note that the default constraint is not of much use because it only works when data is inserted using SQL commands, and Snowflake is not designed to let the default constraint work with data loading from snowpipe, and so the update_timestamp will show NULL until manual updates are done on the table. To tackle this issue we can ask the team to track the timestamp of updates as well.

```
update CDC_Orders SET update_timestamp=current_timestamp();

update Orders_History SET start_time=current_timestamp();
```

Hence we manually update the current timestamp in CDC_Orders and Orders_History.

11. Next create a snowpipe as follows

```
CREATE OR REPLACE FILE FORMAT "RETAIL_DB"."RETAIL_SCHEMA".CSV_FF_one COMPRESSION = 'AUTO' FIELD_DELIMITER = '\t' RECORD_DELIMITER = '\n'
SKIP_HEADER = 1 FIELD_OPTIONALLY_ENCLOSED_BY = 'NONE' TRIM_SPACE = FALSE ERROR_ON_COLUMN_COUNT_MISMATCH = FALSE
ESCAPE = 'NONE' ESCAPE_UNENCLOSED_FIELD = '\134' DATE_FORMAT = 'AUTO' TIMESTAMP_FORMAT = 'AUTO' NULL_IF = ('\\N');

create or replace stage CDC_Orders storage_integration=s3_int
url='s3://lti1034-rahulnoronha/' file_format=csv_ff_one;

create or replace pipe snowpipe_orders auto_ingest=TRUE as copy into CDC_Orders from @CDC_Orders;
```

12. Create a stream on the table CDC_Orders which essentially turns CDC on

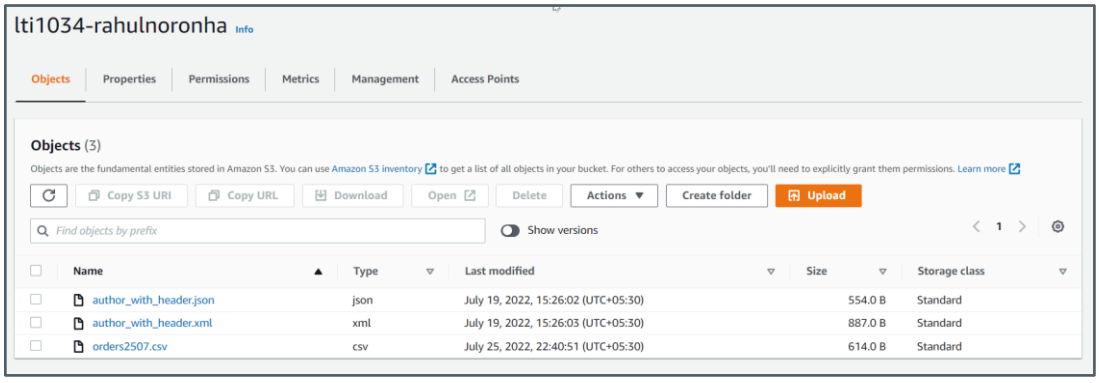
```
use role sysadmin;
create or replace stream orders_table_changes on table CDC_Orders;

show streams;

select * from orders_table_changes;
```

We use role sysadmin to create the stream orders_table_changes.

13. The ORDER_CHANGE_DATA view handles the logic to figure out what needs to be loaded into the ORDERS_HISTORY table. It relies on the data in the ORDERS_TABLE_CHANGES stream.

```
create or replace view order_change_data as
-- This subquery figures out what to do when data is inserted into the CDC_Orders table
-- An insert to the CDC_Orders table results in an INSERT to the Orders_History table
select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY,
O_COMMENT, start_time, end_time, current_flag, 'I' as dml_type
from (select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT,
            update_timestamp as start_time,
            lag(update_timestamp) over (partition by O_ORDERKEY order by update_timestamp desc) as end_time_raw,
            case when end_time_raw is null then '9999-12-31'::timestamp_ntz else end_time_raw end as end_time,
            case when end_time_raw is null then 1 else 0 end as current_flag
      from (select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT, update_timestamp
            from orders_table_changes
            where metadata$action = 'INSERT'
            and metadata$isupdate = 'FALSE'))
union
-- This subquery figures out what to do when data is updated in the CDC_Orders table
-- An update to the NATION table results in an update AND an insert to the Orders_History table
-- The subquery below generates two records, each with a different dml_type
select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT, start_time, end_time,
current_flag, dml_type
from (select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT,
            update_timestamp as start_time,
            lag(update_timestamp) over (partition by O_ORDERKEY order by update_timestamp desc) as end_time_raw,
            case when end_time_raw is null then '9999-12-31'::timestamp_ntz else end_time_raw end as end_time,
            case when end_time_raw is null then 1 else 0 end as current_flag,
            dml_type
      from (-- Identify data to insert into Orders_History table
            select O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE, O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT, update_timestamp,
'I' as dml_type
            from orders_table_changes
            where metadata$action = 'INSERT'
            and metadata$isupdate = 'TRUE'
            union
            -- Identify data in Orders_History table that needs to be updated
            select O_ORDERKEY, null, null, null, null,null, null, null, null, start_time, 'U' as dml_type
            from Orders_History
            where O_ORDERKEY in (select distinct O_ORDERKEY
                                 from orders_table_changes
                                 where metadata$action = 'INSERT'
                                 and metadata$isupdate = 'TRUE')
      and current_flag = 1))
union
-- This subquery figures out what to do when data is deleted from the CDC_Orders table
-- A deletion from the NATION table results in an update to the Orders_History table
select oms.O_ORDERKEY, null, null, null, null, null, null, null, null, oh.start_time, current_timestamp()::timestamp_ntz, null, 'D'
from Orders_History oh
inner join orders_table_changes oms
   on oh.O_ORDERKEY = oms.O_ORDERKEY
where oms.metadata$action = 'DELETE'
and   oms.metadata$isupdate = 'FALSE'
and   oh.current_flag = 1;
```

14. When the new data is loaded into snowpipe from orders2507.csv which is uploaded into the s3 bucket lti1034-rahulnoronha the new records are reflected in CDC_Orders table. Use the select query to view the changes in the table.




```
45
46  SELECT * from CDC_Orders order by O_ORDERKEY DESC LIMIT 10;
47
```

Row	O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORITY	O_CLERK	O_SHIPPRIORITY	O_COMMENT	UPDATE_TIMESTAMP
1	60005	1426	P	299401.66	1995-04-21	2-HIGH	Clerk#000009194	0	usual frets use along...	NULL
2	60004	1426	P	1989401.65	1995-04-21	2-HIGH	Clerk#000008194	0	usual frets use along...	NULL
3	60003	1426	P	99401.64	1995-04-21	2-HIGH	Clerk#000007194	0	usual frets use along...	NULL
4	60002	1426	P	98401.63	1995-04-21	2-HIGH	Clerk#000006194	0	usual frets use along...	NULL
5	60001	1426	P	399401.62	1995-04-21	2-HIGH	Clerk#000005194	0	usual frets use along...	NULL

15. Changes to CDC_Orders are captured in the stream and can be viewed as follows

```
57
58  select * from orders_table_changes;
59
```

Row	O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORIT	O_CLERK	O_SHIPPRIORITY..	O_COMMENT	UPDATE_TIMESTA	METADATA$ACTI	METADATA$ISUPI	METADATA$ROW_
1	60001	1426	P	399401.62	1995-04-21	2-HIGH	Clerk#000005...	0	usual frets use...	NULL	INSERT	FALSE	8360b130886...
2	60002	1426	P	98401.63	1995-04-21	2-HIGH	Clerk#000006...	0	usual frets use...	NULL	INSERT	FALSE	43cf70ceabf0...
3	60003	1426	P	99401.64	1995-04-21	2-HIGH	Clerk#000007...	0	usual frets use...	NULL	INSERT	FALSE	f247dd24f8e1...
4	60004	1426	P	1989401.65	1995-04-21	2-HIGH	Clerk#000008...	0	usual frets use...	NULL	INSERT	FALSE	14421374fe81...
5	60005	1426	P	299401.66	1995-04-21	2-HIGH	Clerk#000009...	0	usual frets use...	NULL	INSERT	FALSE	cbc8a4c037d1...

16. Changes are merged to the History table as follows from the stream. We can set up a task every few minutes to perform the merge operation.

```
create or replace task populate_orders_history warehouse = COMPUTE_WH
schedule = '2 minute' when system$stream_has_data('orders_table_changes')
as
merge into Orders_History oh -- Target table to merge changes from CDC_Orders into
using order_change_data m -- order_change_data is a view that holds the logic that determines what to insert/update into the Orders_History table.
   on  oh.O_ORDERKEY = m.O_ORDERKEY -- O_ORDERKEY and start_time determine whether there is a unique record in the Orders_History table
   and oh.start_time = m.start_time
when matched and m.dml_type = 'U' then update -- Indicates the record has been updated and is no longer current and the end_time needs to be stamped
    set oh.end_time = m.end_time,
        oh.current_flag = 0
when matched and m.dml_type = 'D' then update -- Deletes are essentially logical deletes. The record is stamped and no newer version is inserted
    set oh.end_time = m.end_time,
        oh.current_flag = 0
when not matched and m.dml_type = 'I' then insert -- Inserting a new O_ORDERKEY and updating an existing one both result in an insert
          (O_ORDERKEY, O_CUSTKEY, O_ORDERSTATUS, O_TOTALPRICE, O_ORDERDATE,
           O_ORDERPRIORITY, O_CLERK, O_SHIPPRIORITY, O_COMMENT, start_time, end_time, current_flag)
    values (m.O_ORDERKEY, m.O_CUSTKEY, m.O_ORDERSTATUS, m.O_TOTALPRICE, m.O_ORDERDATE,
    m.O_ORDERPRIORITY, m.O_CLERK, m.O_SHIPPRIORITY, m.O_COMMENT, m.start_time, m.end_time, m.current_flag);

```

Results | Data Preview

Query ID | SQL | 67ms | 1 rows

Filter result... | Copy

Row	status
1	Task POPULATE_ORDERS_HISTORY successfully created.

Row	O_ORDERKEY	O_CUSTKEY	O_ORDERSTATUS	O_TOTALPRICE	O_ORDERDATE	O_ORDERPRIORITY	O_CLERK	O_SHIPPRIORITY	O_COMMENT	START_TIME	END_TIME	CURRENT_FLAG
1	60001	1426	P	399401.62	1995-04-21	2-HIGH	Clerk#000005194	0	usual frets use a...	NULL	9999-12-31 00:...	1
2	60005	1426	P	299401.66	1995-04-21	2-HIGH	Clerk#000009194	0	usual frets use a...	NULL	9999-12-31 00:...	1
3	60002	1426	P	98401.63	1995-04-21	2-HIGH	Clerk#000006194	0	usual frets use a...	NULL	9999-12-31 00:...	1
4	60003	1426	P	99401.64	1995-04-21	2-HIGH	Clerk#000007194	0	usual frets use a...	NULL	9999-12-31 00:...	1
5	60004	1426	P	1989401.65	1995-04-21	2-HIGH	Clerk#000008194	0	usual frets use a...	NULL	9999-12-31 00:...	1

.....

15,000	57796	667	O	70051.07	1996-02-28	4-NOT SPECIFIED	Clerk#0000004...	0	ons according t...	NULL	9999-12-31 00:...	1
15,001	45029	547	F	142822.97	1993-07-04	2-HIGH	Clerk#000000141	0	x carefully slyly ...	NULL	9999-12-31 00:...	1
15,002	47397	1456	O	47593.25	1998-02-16	5-LOW	Clerk#0000000...	0	y pending reque...	NULL	9999-12-31 00:...	1
15,003	42342	68	F	160932.08	1994-09-05	4-NOT SPECIFIED	Clerk#0000004...	0	ajole furiously a...	NULL	9999-12-31 00:...	1
15,004	54052	737	O	217776.51	1996-07-20	1-URGENT	Clerk#0000003...	0	grate carefully p...	NULL	9999-12-31 00:...	1
15,005	10881	1001	F	235385.68	1994-02-16	1-URGENT	Clerk#0000008...	0	quickly final pint...	NULL	9999-12-31 00:...	1

TASK 9: Share your table to a new non-snowflake user.

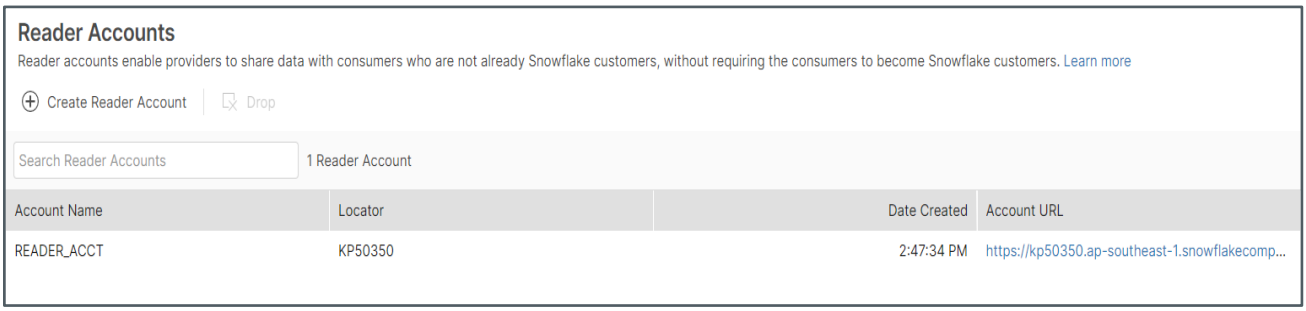
1. So in the provider account first we need to create an outbound share to share the database, table to the consumer and grant access to the database,tables and warehouse.

```
--creating a shared object-------

CREATE SHARE "PROJECT_GLADIATOR_SHARE" COMMENT='';
GRANT USAGE ON DATABASE "RETAIL_DB" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT USAGE ON SCHEMA "RETAIL_DB"."RETAIL_SCHEMA" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."CDC_ORDERS" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."CUSTOMER" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."LINEITEM" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."NATION" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."ORDERS" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."PART" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."PARTSUPP" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."REGION" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT SELECT ON TABLE "RETAIL_DB"."RETAIL_SCHEMA"."SUPPLIER" TO SHARE "PROJECT_GLADIATOR_SHARE";
GRANT USAGE ON WAREHOUSE COMPUTE_WH TO ROLE ACCOUNTADMIN;
```

2. Now we need to create a reader account and add it to provider account,

```
--creating a reader account------

CREATE MANAGED ACCOUNT READER_ACCT admin_name='NKREADER', admin_password=[redacted], type=reader, COMMENT='';

--now adding the reader------

ALTER SHARE "PROJECT_GLADIATOR_SHARE" ADD ACCOUNTS = KP50350;
```

here you can see the reader account is added,



A reader account is intended primarily for querying data shared by the provider of the account. Adding new data to the account and/or updating shared data in the account is not supported.

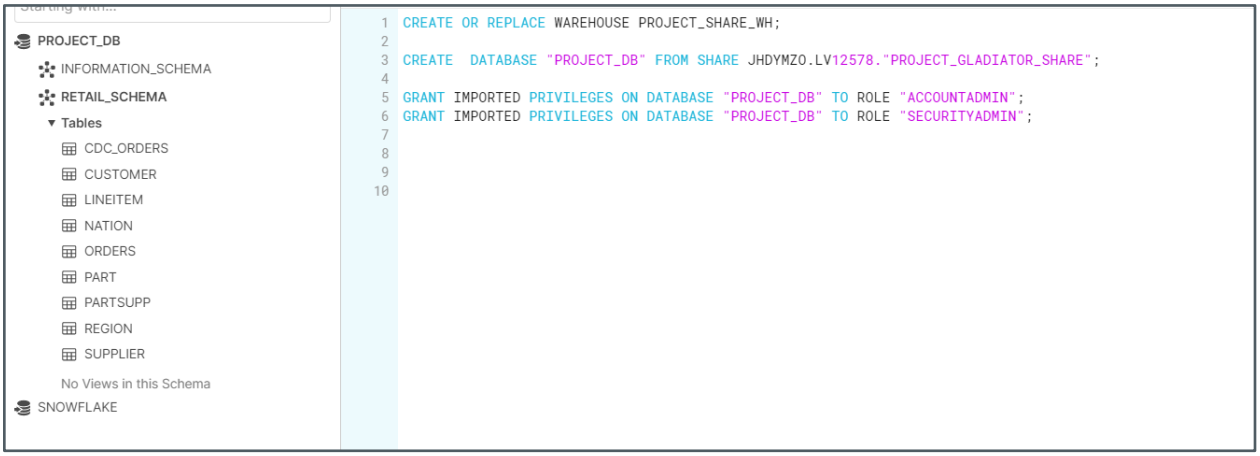
3. Now login to the reader account and create a warehouse
   And we will create a database from secure share in reader account with the role account admin to consume the shared data,

```
CREATE OR REPLACE WAREHOUSE PROJECT_SHARE_WH;

CREATE  DATABASE "PROJECT_DB" FROM SHARE JHDYMZO.LV12578."PROJECT_GLADIATOR_SHARE";
```

Then we will grant privileges to roles in the reader account,

```
4
5 GRANT IMPORTED PRIVILEGES ON DATABASE "PROJECT_DB" TO ROLE "ACCOUNTADMIN";
6 GRANT IMPORTED PRIVILEGES ON DATABASE "PROJECT_DB" TO ROLE "SECURITYADMIN";
7
```

Here we can see all the tables in the reader account,



So now we can access the data as you can see here,

TASK 10: Create clone of the table with time travel before one day and write query to get history data using particular timestamp

These are the ways to perform time travel:

- Using offset
- Using timestamp
- Using statement/query id
- Using Undrop table command

1. Time travel using offset:

To copy into a clone table use CREATE OR REPLACE TABLE clone_name CLONE object_name AT …

We can use the command shown below to travel back in time to retrieve a table. Here use the AT or BEFORE keywords to specify that we want to go back in time. In the bracket we give offset=>-60*60*24. The negative symbol signifies we are going back in time and the value is in seconds, hence we use the number of seconds in a day=60*60*24 to go back 24 hours or 1 day.

```
2
3  CREATE OR REPLACE TABLE Region_Clone_TimeTravel CLONE Region AT (offset=>-60*60*24);
4
```

Row	status
1	Table REGION_CLONE_TIMETRAVEL successfully created.

We use the select command to view the cloned table in the past.

```
4
5  select * from Region_Clone_TimeTravel;
6
```

Row	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regu...
2	1	AMERICA	hs use ironic, even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asy...

2. Time Travel using timestamp:

Here we first insert a new record into the Region table using insert into command.

```
16 INSERT INTO REGION VALUES(5, 'NORTH AMERICA', 'lar deposits. blithely final packages cajole.');
17
```

We see the newly inserted table will have our inserted row using select * command from Region.

```
19
20 select * from REGION;
21
```

Row	R_REGIONKEY	R_NAME	R_COMMENT
1	5	NORTH AMERICA	lar deposits. blithely final packages cajole.
2	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regu...
3	1	AMERICA	hs use ironic, even requests. s
4	2	ASIA	ges. thinly even pinto beans ca
5	3	EUROPE	ly final courts cajole furiously final excuse
6	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asy...

To clone the table into the Region_History_TimeTravel table at the timestamp before we insert the new record we check the query history and get the timestamp. Then we add it in the brackets as timestamp=>TO_TIMESTAMP('2022-07-26 00:23:12.915'))
Here we use the TO_TIMESTAMP to convert the string into a timestamp data type.

```
20 CREATE OR REPLACE TABLE Region_History_TimeTravel CLONE Region AT
21 (timestamp=>TO_TIMESTAMP('2022-07-26 00:23:12.915'));
```

Results | Data Preview

Query ID | SQL | 872ms | 1 rows

Filter result... | Copy

Row	status
1	Table REGION_HISTORY_TIMETRAVEL successfully created.

We can view this cloned table using select * to retrieve the table before the new row was inserted.

```
25 select * from REGION_HISTORY_TIMETRAVEL;
```

Row	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regu...
2	1	AMERICA	hs use ironic, even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asy...

3. Time Travel using Query ID

We can time travel using the query id of a past command by finding its query id from the history tab in Snowflake.
First we delete a row from the table Region. We notice the select * query of Region returns only 5 rows now, since the newly inserted record in the previous step has been deleted.

```
27 DELETE from REGION where R_REGIONKEY=5;
28 select * from REGION;
```

Row	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regu...
2	1	AMERICA	hs use ironic, even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asy...

To get the table before the delete query we use the following command

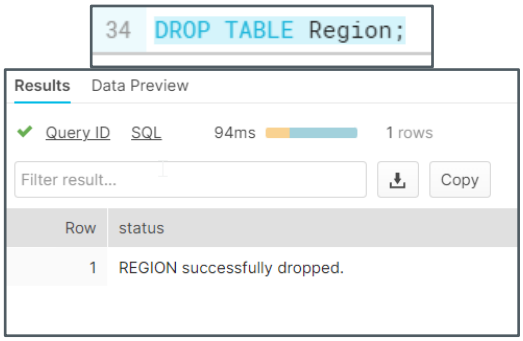
```
29
30 select * from REGION BEFORE (statement=>'01a5dc1f-0000-321a-0003-bfd6000791de');
31
```

Select * Region table now shows 6 records since we restored the deleted record using time travel.

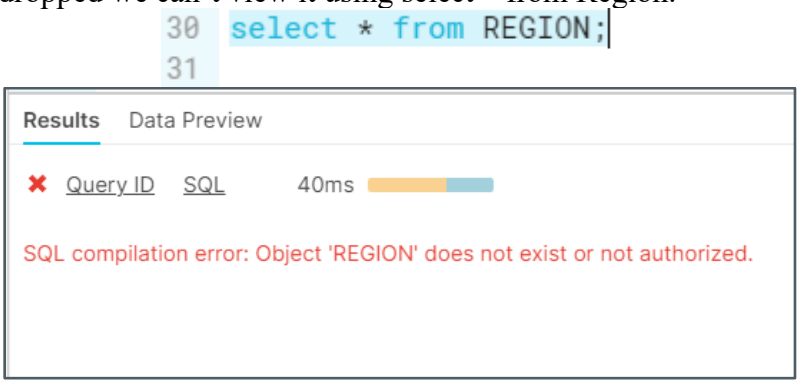
Row	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are final requests. regu...
2	1	AMERICA	hs use ironic, even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. carefully final asy...
6	5	NORTH AMERICA	lar deposits. blithely final packages cajole.

4. Undrop table:

We can restore a table we have dropped using the undrop command as follows.
First we drop the table using DROP TABLE Region;



Once the table is dropped we can't view it using select * from Region.



Use the UNDROP option on the table to restore it

```
35
36 UNDROP TABLE Region;
```

Results Data Preview

Query ID SQL 69ms 1 rows

Filter result... Copy

Row	status
1	Table REGION successfully restored.

Now when we call select * from Region we will be able to see the table values restored.

```
30 select * from REGION;
```

Results Data Preview Open History

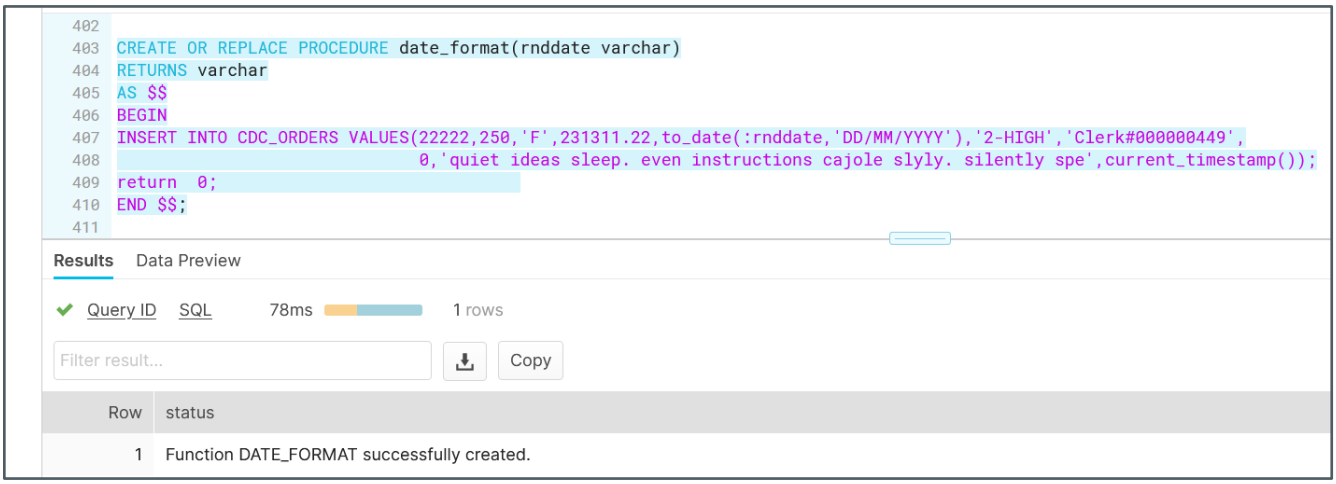
Query ID SQL 60ms 5 rows

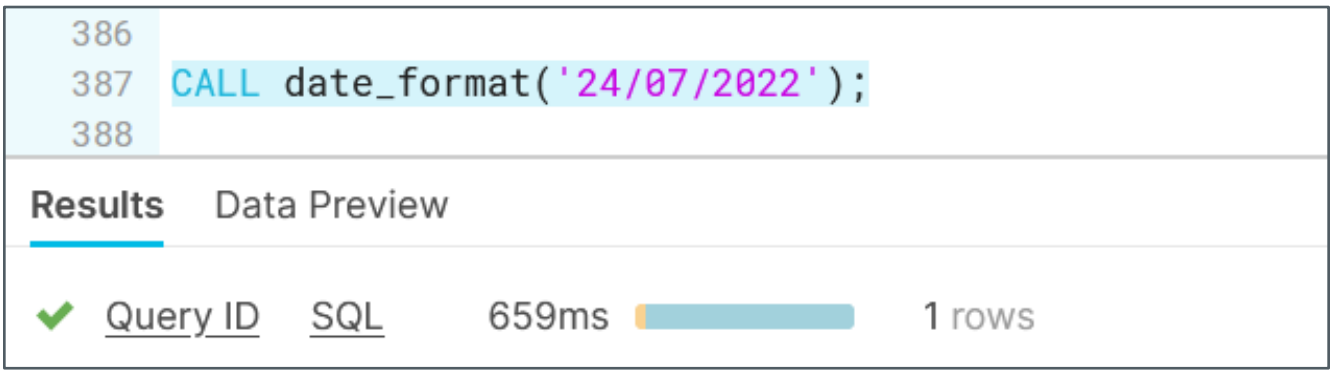
Filter result... Copy Columns

Row	R_REGIONKEY	R_NAME	R_COMMENT
1	0	AFRICA	lar deposits. blithely final packages cajole. regular waters are fin...
2	1	AMERICA	hs use ironic, even requests. s
3	2	ASIA	ges. thinly even pinto beans ca
4	3	EUROPE	ly final courts cajole furiously final excuse
5	4	MIDDLE EAST	uickly special accounts cajole carefully blithely close requests. ...

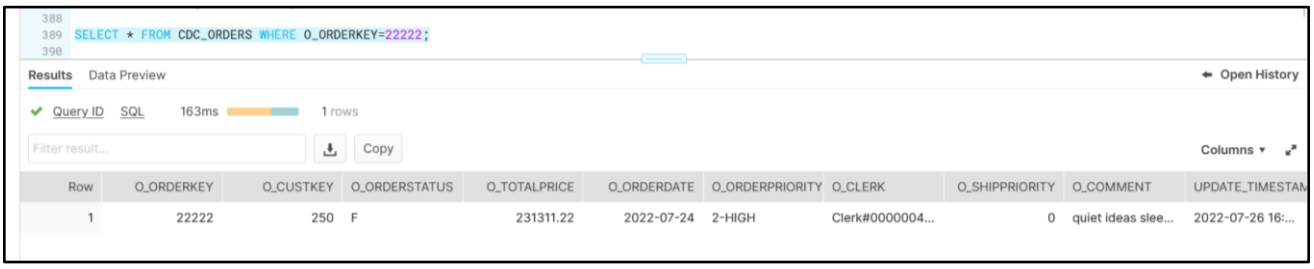
TASK 11: Create stored procedure to insert data into table after typecasting date format

In the below step, a stored procedure named date_format is created.





In the above step the procedure date_format is called. The procedure, when called, inserts a new record into the table CDC_ORDERS after typecasting the date provided in the procedure call.



As we can see that the record got inserted.

TASK 12:Create a visualization for sales performance analysis using powerBI and prepare reports /dashboard for business queries

1. **Snowflake:**

We find the maximum and the minimum l_shipdate and since we get 1998-11-29 as the maximum, we can use it while creating the view. This is because the date is within 60-120 days of the greatest ship date contained in the database, which is also why we chose -90 in the dateadd function.

```
26
27  select max(l_shipdate),min(l_shipdate) from lineitem;
```

Results | Data Preview | Open History

Query ID | SQL | 147ms | 1 rows

Filter result... | Copy | Columns

Row	MAX(L_SHIPDATE)	MIN(L_SHIPDATE)
1	1998-11-29	1992-01-04

We create a view called PowerBIAnalysis which has the aggregates that we require grouped by l_returnflag and l_linestatus and ordered by l_returnflag and l_linestatus.

```
create or replace view PowerBIAnalysis as
select
       l_returnflag,
       l_linestatus,
       sum(l_quantity) as sum_qty,
       sum(l_extendedprice) as sum_base_price,
       sum(l_extendedprice * (1-l_discount)) as sum_disc_price,
       sum(l_extendedprice * (1-l_discount) * (1+l_tax)) as sum_charge,
       avg(l_quantity) as avg_qty,
       avg(l_extendedprice) as avg_price,
       avg(l_discount) as avg_disc,
       count(*) as count_order
 from
       lineitem
 where
       l_shipdate <= dateadd(day, -90, to_date('1998-11-29'))
 group by
       l_returnflag,
       l_linestatus
 order by
       l_returnflag,
       l_linestatus
       ;
```

We can use the select * from PowerBIAnalysis; to get the records of the view.

```
24
25  select * from PowerBIAnalysis;
```

Row	L_RETURNFLAG	L_LINESTATUS	SUM_QTY	SUM_BASE_PRICE	SUM_DISC_PRICE	SUM_CHARGE	AVG_QTY	AVG_PRICE	AVG_DISC	COUNT_ORDER
1	A	F	380456	532348211.65	505822441.4861	526165934.000839	25.575154611	35785.709306937	0.05008133907	14876
2	N	F	8971	12384801.37	11798257.208	12282485.056933	25.778735632	35588.509683908	0.04775862069	348
3	N	O	741582	1039776928.95	988095461.8355	1027703097.97375	25.452429984	35687.017056219	0.04992964031	29136
4	R	F	381449	534594445.35	507996454.4067	528524219.358903	25.597168165	35874.00653268	0.04982753993	14902

In order to visualize this view we import it to Power BI using the Snowflake ODBC connector.

JOINS:

1. In place Join: We create an in-place join between LINEITEM and ORDERS such that only data where L_ORDERKEY=O_ORDERKEY are returned, but we only return one row, i.e., AVG(L_QUANTITY). But the join is still performed. We can later take this aggregate and use it for Power BI visualizations.

```
--In place Join
CREATE OR REPLACE SECURE VIEW InPlaceJoin AS
SELECT AVG(l.L_QUANTITY) as AVERAGE_L_QUANTITY FROM LINEITEM l, ORDERS o WHERE
l.L_ORDERKEY=o.O_ORDERKEY;

SELECT * FROM InPlaceJoin;
```

Row	AVERAGE_L_QUANTITY
1	25.527660989

2. We find the sum of Orders.O_TOTALPRICE in each Nation using a join between Customer and Orders table considering only the sum in the year 1995 using where O.O_ORDERDATE LIKE '1995%' which means the O_ORDERDATE must begin with 1995. Store this Join in a view and later visualize it in Power BI.

```
CREATE OR REPLACE SECURE VIEW CustomerOrdersNationJoin AS
SELECT N.N_NAME, SUM(O.O_TOTALPRICE) AS SUM_O_TOTALPRICE
FROM ORDERS O, CUSTOMER C, NATION N
WHERE
N.N_NATIONKEY=C.C_NATIONKEY
AND C.C_CUSTKEY=O.O_CUSTKEY
AND O.O_ORDERDATE LIKE '1995%'
GROUP BY N.N_NAME;

SELECT * from CustomerOrdersNationJoin;
```

Row	N_NAME	SUM_O_TOTALPRICE
1	EGYPT	15217257.87
2	ARGENTINA	10841034.61
3	MOROCCO	15384463.84
4	CANADA	15258476.25
5	INDONESIA	12560161.91
6	ALGERIA	15055557.51
7	SAUDI ARABIA	12020675.08
8	INDIA	9620709.26
9	JORDAN	15421301.56
10	GERMANY	9767990.36
11	UNITED KINGDOM	14833240.53
12	UNITED STATES	12815118.6
13	JAPAN	13556798.12
14	ROMANIA	12792395.94
15	VIETNAM	13770421.66
16	KENYA	13603047.13
17	ETHIOPIA	15043371

3. We have created a secure view named LARGE_QUANTITY_ORDERS.

The below query ranks customers based on them having placed a large quantity order. Large quantity orders are those orders whose total quantity is above a certain level. In our case we have kept it as 250.

```
472
473 CREATE OR REPLACE SECURE VIEW LARGE_QUANTITY_ORDERS AS
474 SELECT c_name,c_custkey,o_orderkey,o_orderdate,o_totalprice,SUM(l_quantity) AS quantity
475 FROM customer
476 INNER JOIN orders ON c_custkey = o_custkey
477 INNER JOIN lineitem ON o_orderkey = l_orderkey
478 WHERE o_orderkey IN (SELECT l_orderkey FROM lineitem GROUP BY l_orderkey HAVING sum(l_quantity) > 250)
479 GROUP BY c_name, c_custkey, o_orderkey, o_orderdate, o_totalprice
480 ORDER BY o_totalprice DESC, o_orderdate LIMIT 20;
481
482
```

Results Data Preview

✔ Query ID SQL 123ms 1 rows

Filter result... Copy

Row	status
1	View LARGE_QUANTITY_ORDERS successfully created.

```
483
484 SELECT * FROM  LARGE_QUANTITY_ORDERS;
485
```

Results Data Preview

✔ Query ID SQL 421ms 20 rows

Row	C_NAME	C_CUSTKEY	O_ORDERKEY	O_ORDERDATE	O_TOTALPRICE	QUANTITY
1	Customer#000000676	676	52965	1996-09-22	466001.28	271
2	Customer#000000667	667	29158	1995-10-21	439687.23	305
3	Customer#000001013	1013	44707	1997-08-14	431771.98	279
4	Customer#000000953	953	59106	1996-10-24	430619.75	276
5	Customer#000000178	178	6882	1997-04-09	422359.65	303
6	Customer#000001016	1016	39456	1998-02-16	409770.83	266
7	Customer#000001279	1279	39620	1994-10-05	406938.36	272
8	Customer#000001003	1003	15779	1998-02-24	405401.76	256
9	Customer#000000583	583	52480	1993-07-28	403464.01	261
10	Customer#000000166	166	55937	1994-01-27	402930.49	268
11	Customer#000000085	85	4421	1997-04-04	401055.62	255
12	Customer#000001210	1210	10209	1993-11-30	400191.77	263
13	Customer#000000772	772	35424	1996-01-04	397797.8	253
14	Customer#000001303	1303	45670	1992-08-08	397549.76	261

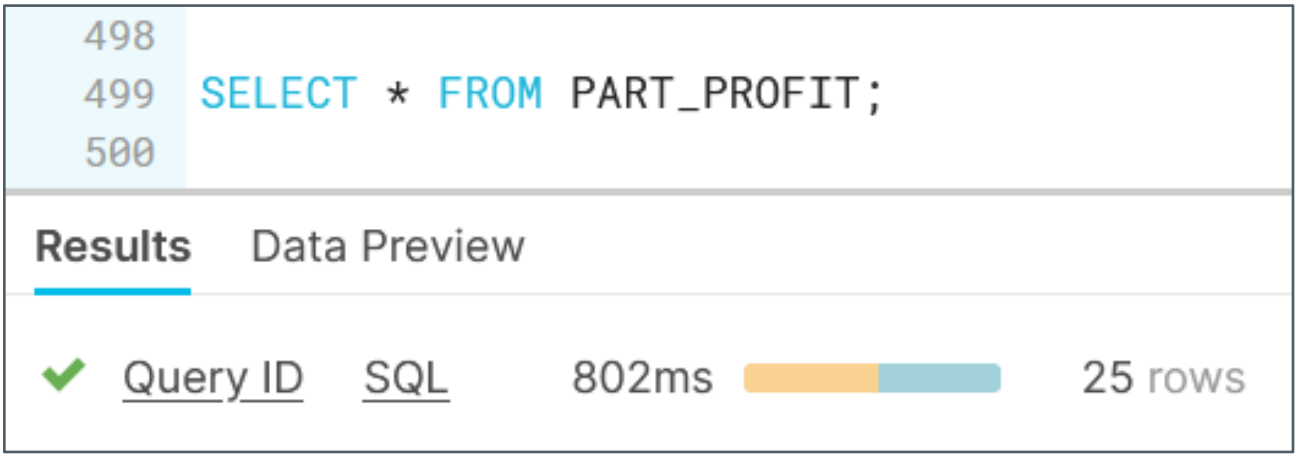
4. Here, we have created a secure view named PART_PROFIT.

The below query determines how much profit is made on a given line of parts, broken out by supplier nation and year. In our case we are computing results from the latest year in a dataset for parts that contain ‘lavender’ in their name.

```
484
485 CREATE OR REPLACE SECURE VIEW PART_PROFIT AS
486 SELECT nation,o_year,SUM(amount) AS sum_profit
487 FROM (
488 SELECT n_name AS nation,
489 extract(year FROM o_orderdate) AS o_year,
490 l_extendedprice * (1 - l_discount) - ps_supplycost * l_quantity AS amount
491 FROM part
492 INNER JOIN lineitem ON p_partkey = l_partkey
493 INNER JOIN supplier ON s_suppkey = l_suppkey
494 INNER JOIN partsupp ON ps_suppkey = l_suppkey and ps_partkey = l_partkey
495 INNER JOIN orders ON o_orderkey = l_orderkey
496 INNER JOIN nation ON s_nationkey = n_nationkey
497 WHERE p_name like '%lavender%' and o_year=extract(year FROM (SELECT max(o_orderdate) FROM orders))) AS profit
498 GROUP BY nation, o_year
499 ORDER BY nation;
500
```

Results Data Preview

Query ID SQL 218ms 1 rows



Row	NATION	O_YEAR	SUM_PROFIT
1	ALGERIA	1998	136898.0411
2	ARGENTINA	1998	335174.7602
3	BRAZIL	1998	173650.8218
4	CANADA	1998	93951.786
5	CHINA	1998	554245.0946
6	EGYPT	1998	469310.116
7	ETHIOPIA	1998	195603.812
8	FRANCE	1998	20835.9696
9	GERMANY	1998	528202.2605
10	INDIA	1998	272162.3933
11	INDONESIA	1998	294951.2822
12	IRAN	1998	189946.9709
13	IRAQ	1998	61114.935
14	JAPAN	1998	145932.9922

5.This query joins PART and LINEITEM on partkey which is the PK for PART and the FK for LINEITEM while performing an operation involving columns in both tables,

```sql
--PK/FK JOIN---------

CREATE OR REPLACE SECURE VIEW PK_FK_JOIN AS
SELECT AVG (P_RETAILPRICE*L_QUANTITY) AS AVERAGE_PR_LQ FROM PART, LINEITEM WHERE P_PARTKEY= L_PARTKEY;

SELECT * FROM PK_FK_JOIN;
```

Columns

Row	AVERAGE_PR_LQ
1	35765.513260823

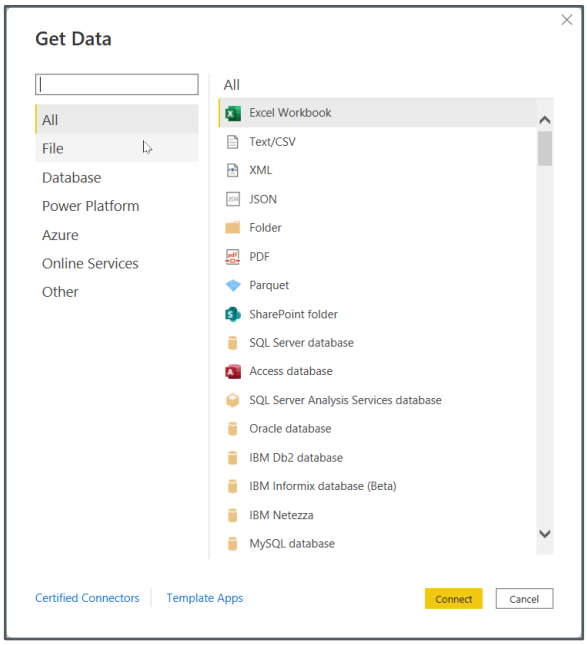
6. This query joins PART and LINEITEM on unrelated columns (p_partkey and l_suppkey) while performing a sum so that only one row is returned that is SUM_L_QUANTITY,

```sql

CREATE OR REPLACE SECURE VIEW P_L_JOIN AS
SELECT SUM(L_QUANTITY) AS SUM_L_QUANTITY FROM PART,LINEITEM
WHERE P_PARTKEY = L_SUPPKEY;

SELECT * FROM P_L_JOIN;
```

Columns

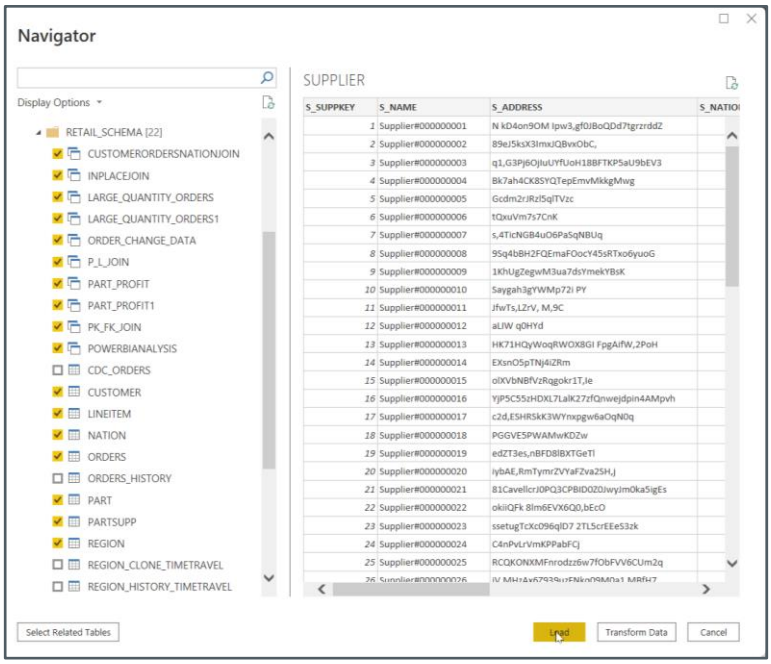
Row	SUM_L_QUANTITY
1	1536127

2. **Power BI Desktop Visualizations**:
    1. Get Data into Power BI from snowflake.
    - Select Get data from other sources.

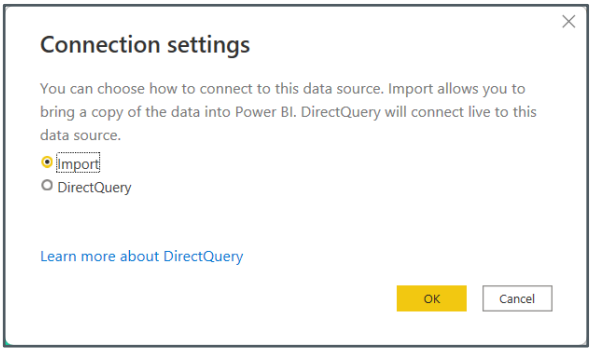


- Find Snowflake

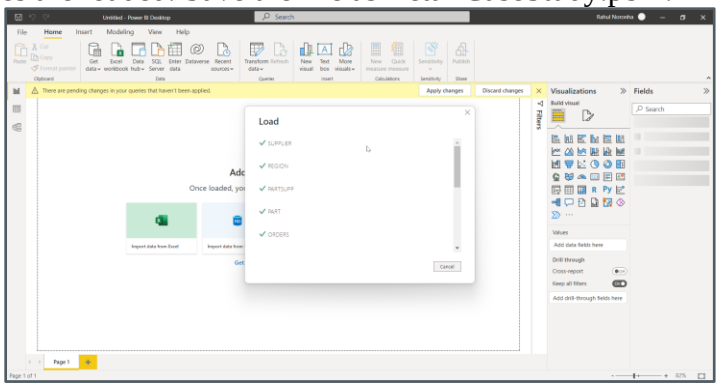
- Enter the server name and warehouse to use server: im69079.ap-south-1.aws.snowflakecomputing.com
  warehouse: COMPUTE_WH
- Select all the tables and views we want.



- Then click on Load, in the pop up click import, since we want to import the data from Power BI rather than DirectQuery the data from Snowflake, because the data is of fixed nature.

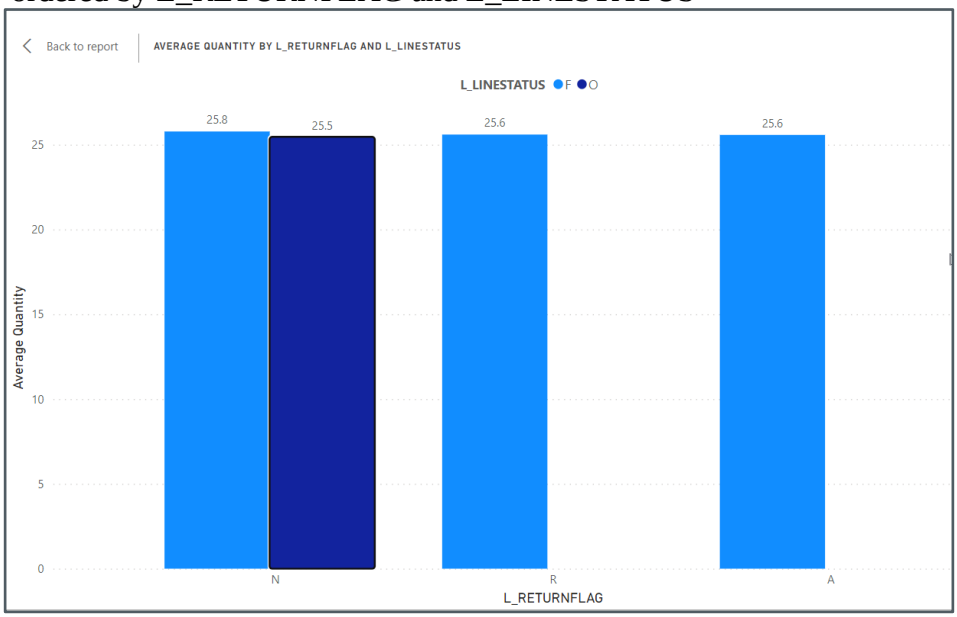


- Then the tables are loaded. Save the file as RetailCaseStudy.pbix.

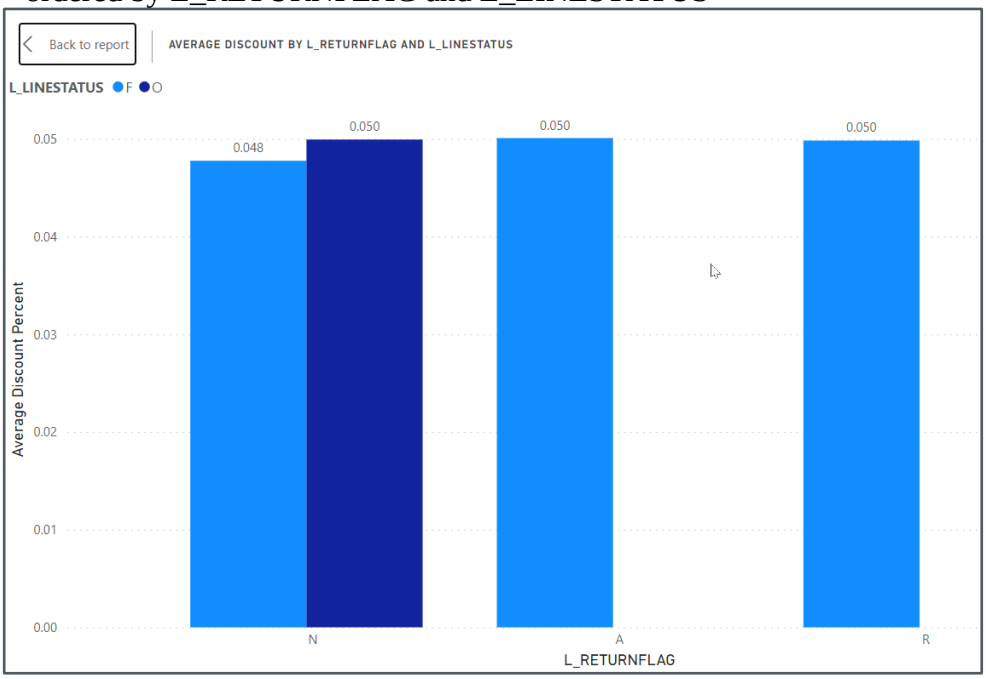


1. Use the view PowerBIAnalysis to create the following visualization

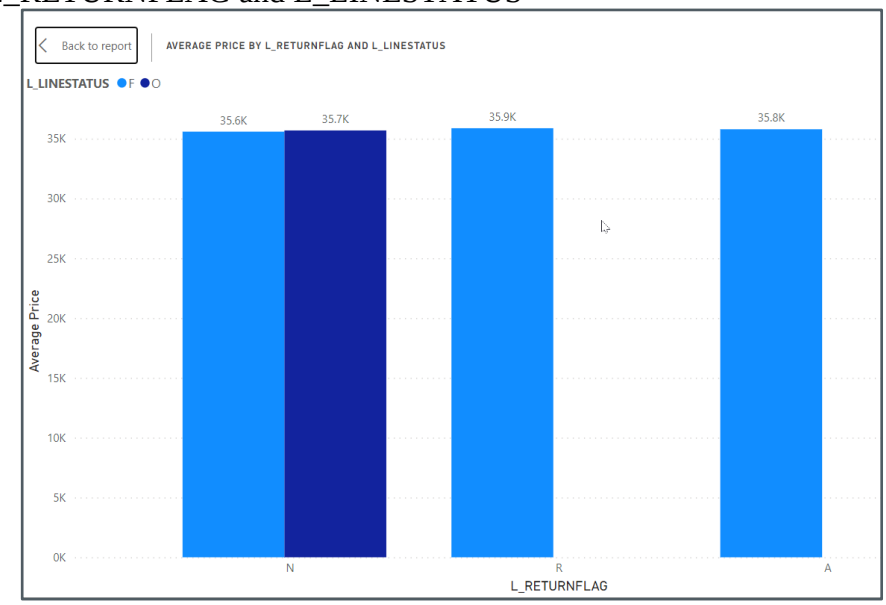
    a. Average Quantity grouped by L_RETURNFLAG and L_LINESTATUS, and ordered by L_RETURNFLAG and L_LINESTATUS



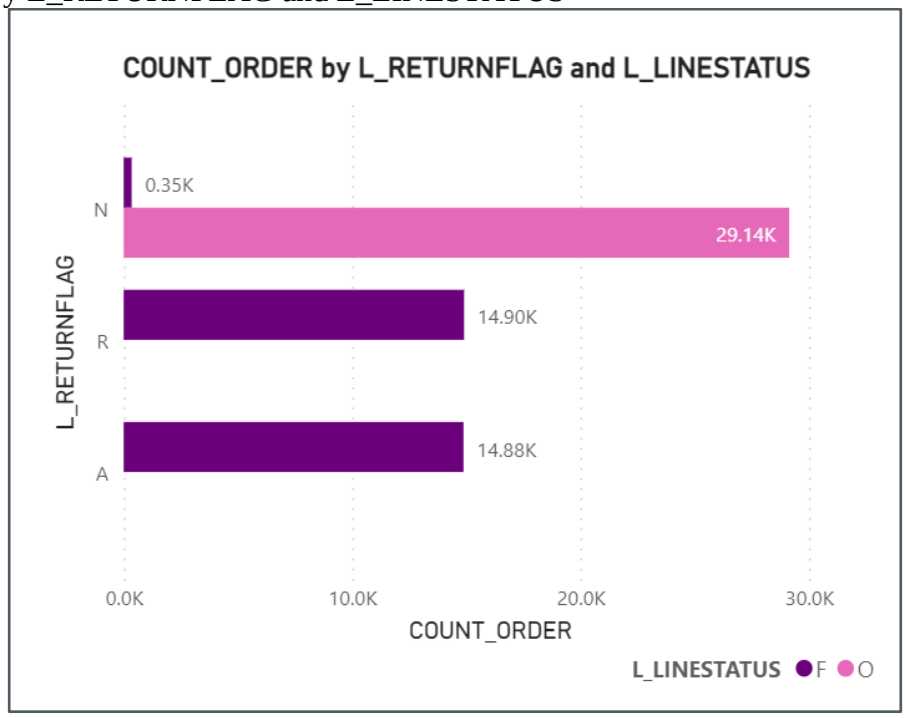
    b. Average Discount grouped by L_RETURNFLAG and L_LINESTATUS, and ordered by L_RETURNFLAG and L_LINESTATUS



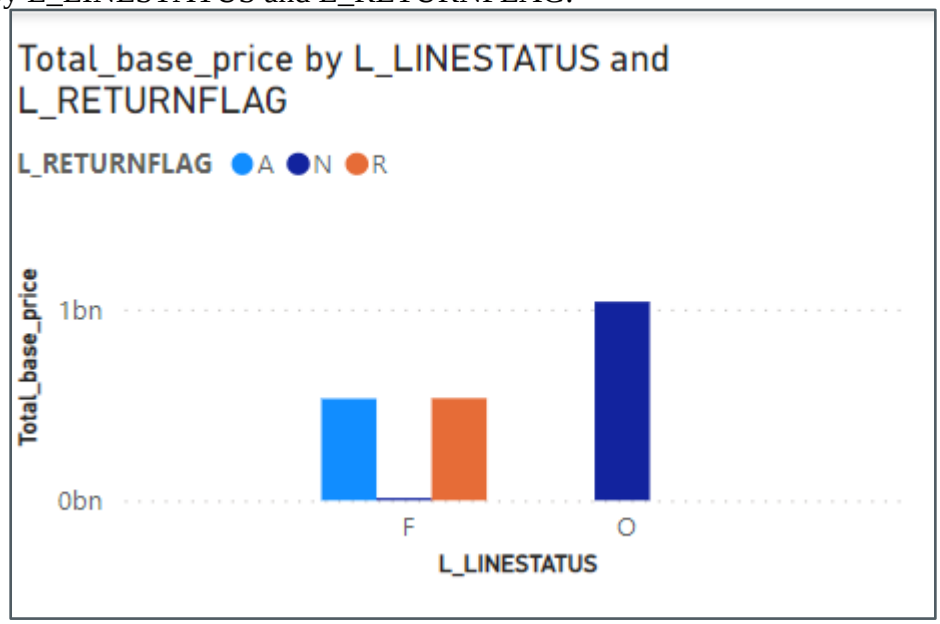
c. Average Price grouped by L_RETURNFLAG and L_LINESTATUS, and ordered by L_RETURNFLAG and L_LINESTATUS



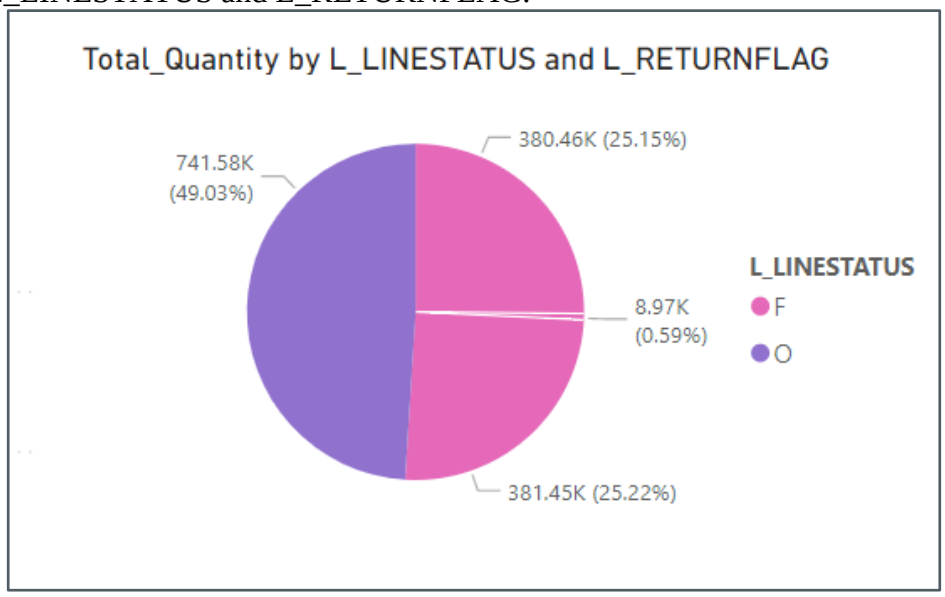
d. Count of Order grouped by L_RETURNFLAG and L_LINESTATUS, and ordered by L_RETURNFLAG and L_LINESTATUS



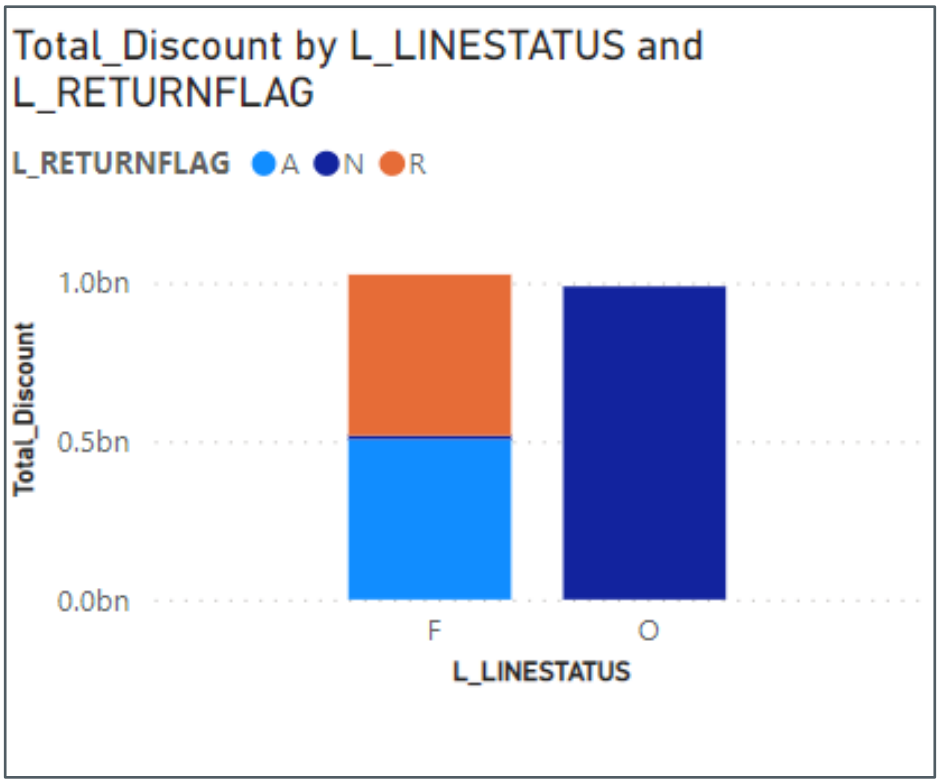
e. Total base price grouped by L_LINESTATUS and L_RETURNFLAG and ordered by L_LINESTATUS and L_RETURNFLAG.



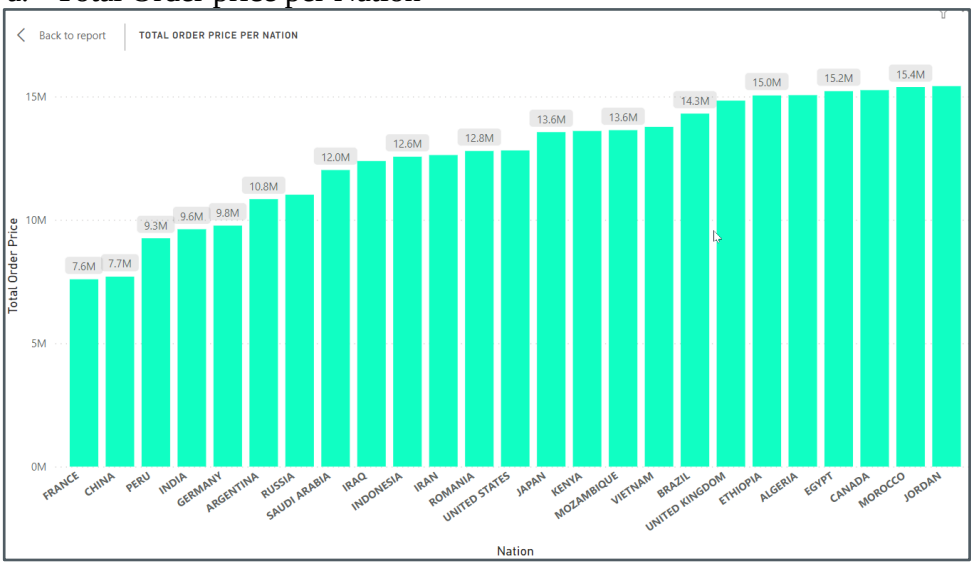
f. Total Quantity grouped by L_LINESTATUS and L_RETURNFLAG and ordered by L_LINESTATUS and L_RETURNFLAG.



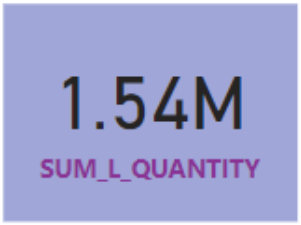
g. Total discount grouped by L_LINESTATUS and L_RETURNFLAG and ordered by L_LINESTATUS and L_RETURNFLAG.



2. Use the view CustomerOrdersNationJoin to create the following visualization
   a. Total Order price per Nation

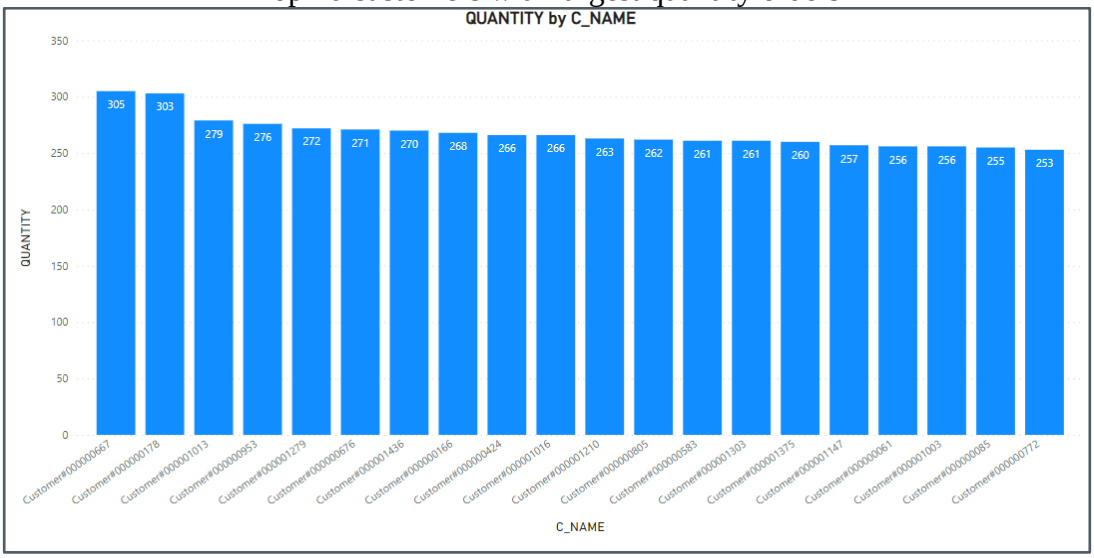


3. Use the view InPlaceJoin to create the following visualizations
   a. Sum of L_QUANTITY from LineItem



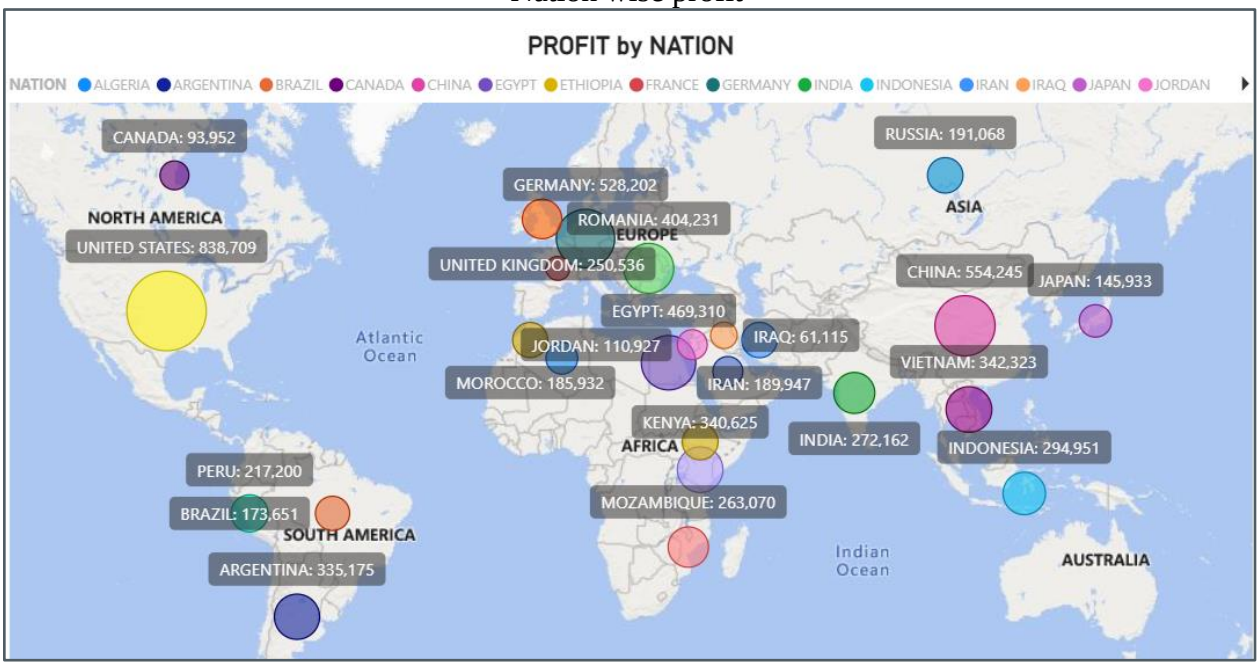
4. Use the view LARGE_QUANTITY_ORDERS to create following visualization

Top 20 customers with largest quantity orders

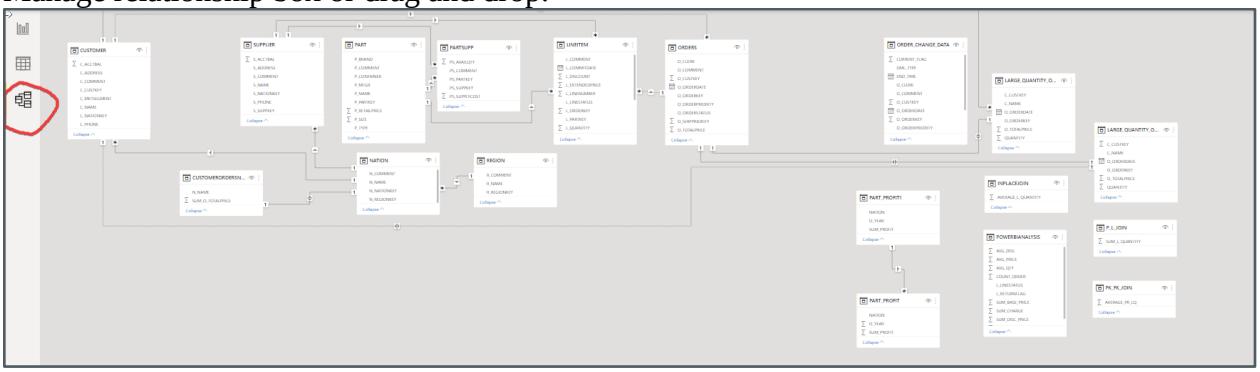


5. Use the view LARGE_QUANTITY_ORDERS to create following visualization

Nation wise profit

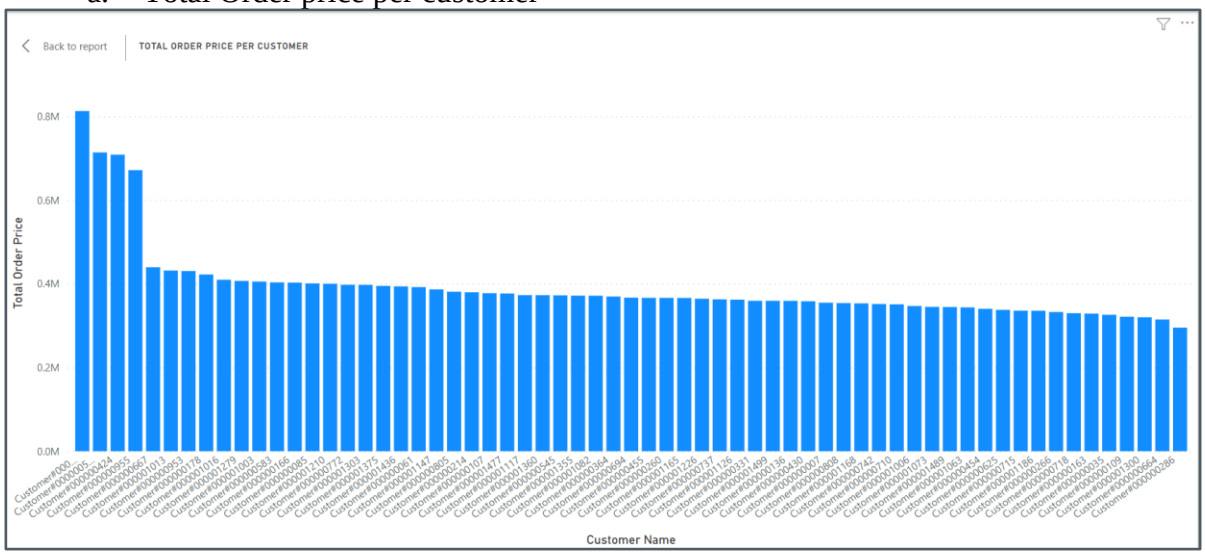


Before checking for new visualizations possible, connect the tables using the Model section via Manage relationship box or drag and drop.

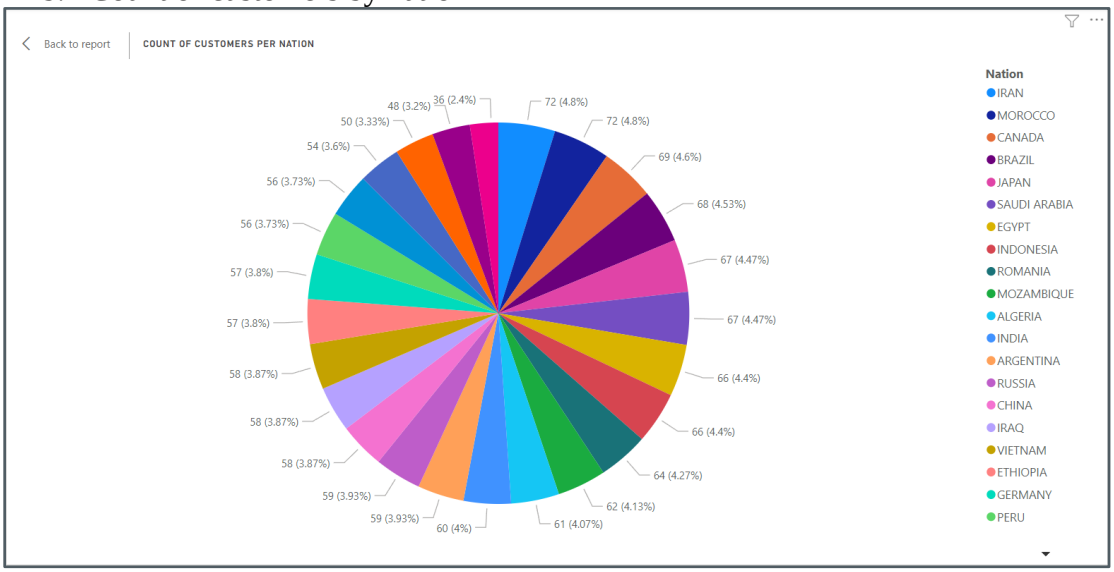


6. Few other queries we can create on the data model are as follows

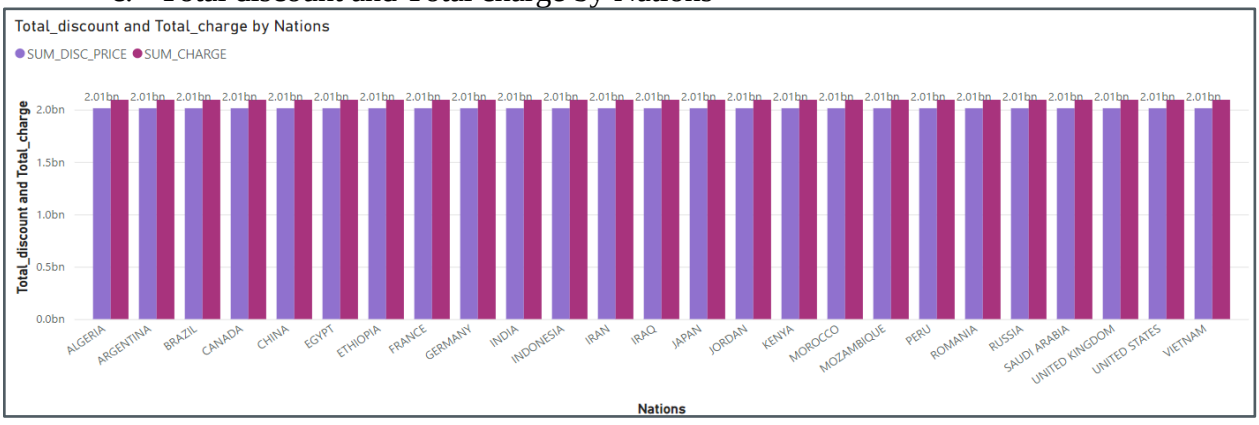
a. Total Order price per customer



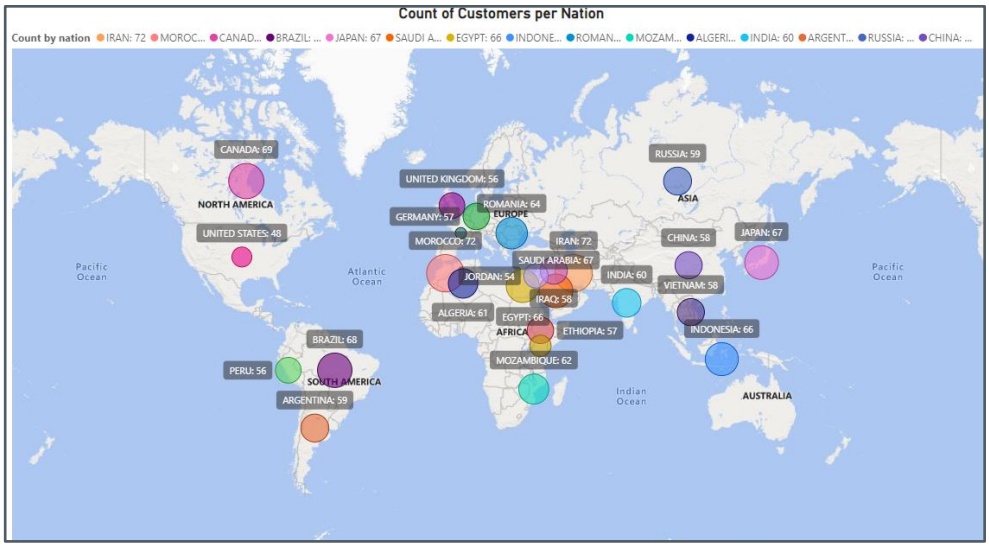
b. Count of customers by nation



c. Total discount and Total charge by Nations

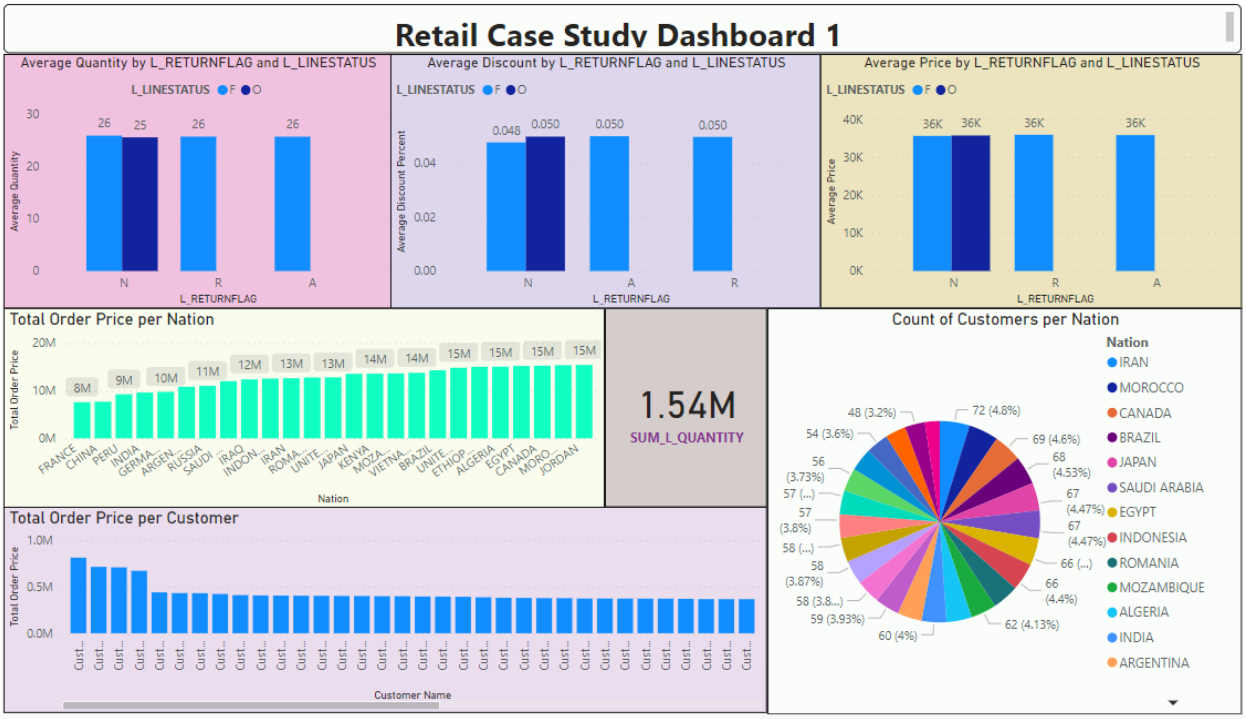


d. Count of customers per Nation

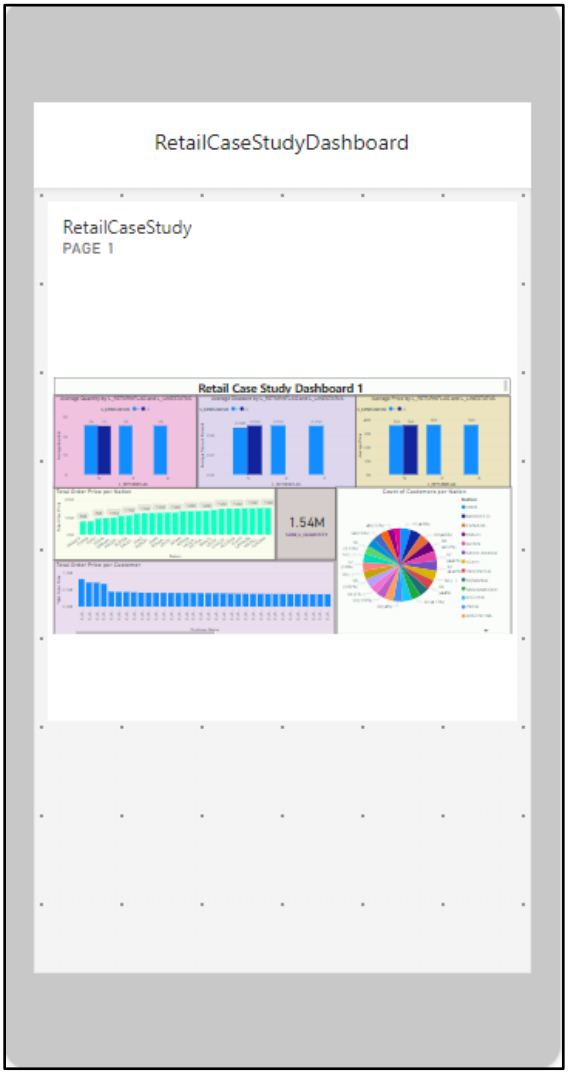


Dashboards:

1. **Retail Case Study Dashboard 1**
   a. **Web View**



b. **Mobile View**



2. **Retail case study Dashboard 2**
    a. **Web View**

b. **Mobile view Retail case study dashboard 2**