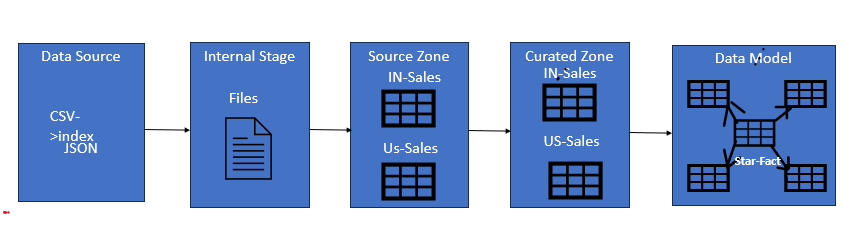
DAY-11-SNOWPARK



drop database sales\_dwh;

Step-1 Create User & Virtual Warehouse

Lets create a virtual warehouse and user account that will be used to run Snowpark ETL workload.

-- create a virtual warehouse

create warehouse snowpark\_etl\_wh

with

warehouse\_size = 'xsmall'

warehouse\_type = 'standard'

auto\_suspend = 60

auto\_resume = true

min\_cluster\_count = 1

max\_cluster\_count = 1

scaling\_policy = 'standard';

use role accountadmin;

--Validate if we are able to use the user and virtual warehouse created in step-1

import snowflake.snowpark as snowpark

from snowflake.snowpark.functions import col

def main(session: snowpark.Session):

# Your code goes here, inside the "main" handler.

df = session.sql("select \* from information\_schema.packages where language='python' ")

# Print a sample of the dataframe to standard output.

df .show()

# Return value will appear in the Results tab.

return df

import snowflake.snowpark as snowpark

def main(session: snowpark.Session):

# Your code goes here, inside the "main" handler.

context\_df = session.sql("select current\_role(), current\_database(), current\_schema(), current\_warehouse()")

context\_df.show(2)

customer\_df = session.sql("select c\_custkey,c\_name,c\_phone,c\_mktsegment from snowflake\_sample\_data.tpch\_sf1.customer limit 10")

customer\_df.show(5)

# Return value will appear in the Results tab.

return context\_df

Step-2 Database & Schema Object

Creating sales\_dwh and 5 schemas under the sales\_dwh.

-- create database

create database if not exists sales\_dwh;

use database sales\_dwh;

create schema if not exists source; -- will have source stage etc

create schema if not exists curated; -- data curation and de-duplication

create schema if not exists consumption; -- fact & dimension

create schema if not exists common; -- for file formats sequence object etc

Step-3.1 Internal Stage In Source Schema

Creating internal stage that will host all the data setup available in our local machine.

-- creating internal stage within source schema.

use schema source;

create or replace stage my\_internal\_stg;

ls @my\_internal\_stg;

rm @my\_internal\_stg;

Step 3.2 Loading Data To Internal Stage

snowsql -a yp20234.us-east-2.aws -u denzel

use database sales\_dwh;

use schema source;

-- following put command can be executed

/\*

-- csv example

put file://C:/Users/HP/Desktop/snowflake/orders-in.csv @my\_internal\_stg auto\_compress=False overwrite=True, parallel=3 ;

-- json example

put file://C:/Users/HP/Desktop/snowflake/orders-us.json @my\_internal\_stg auto\_compress=False overwrite=True, parallel=3 ;

\*/

Step-4 File Format Objects Within Common Schema

Following file formate will be created under common schema and will be used to read and process the data from the internal stage location.

use schema common;

-- create file formats csv (India), json (France), Parquet (USA)

create or replace file format my\_csv\_format

type = csv

field\_delimiter = ','

skip\_header = 1

null\_if = ('null', 'null')

empty\_field\_as\_null = true

field\_optionally\_enclosed\_by = '\042'

compression = auto;

-- json file format with strip outer array true

create or replace file format my\_json\_format

type = json

strip\_outer\_array = true

compression = auto;

-- Internal Stage - Query The CSV Data File Format

use schema source;

select

t.$1::text as order\_id,

t.$2::text as customer\_name,

t.$3::text as mobile\_key,

t.$4::number as order\_quantity,

t.$5::number as unit\_price,

t.$6::number as order\_valaue,

t.$7::text as promotion\_code ,

t.$8::number(10,2) as final\_order\_amount,

t.$9::number(10,2) as tax\_amount,

t.$10::date as order\_dt,

t.$11::text as payment\_status,

t.$12::text as shipping\_status,

t.$13::text as payment\_method,

t.$14::text as payment\_provider,

t.$15::text as mobile,

t.$16::text as shipping\_address

from

@my\_internal\_stg/orders-in.csv

(file\_format => 'sales\_dwh.common.my\_csv\_format') t;

-- Internal Stage - Query The Parquet Data File Format

-- Internal Stage - Query The JSON Data File Format

select

$1:"Order ID"::text as orde\_id,

$1:"Customer Name"::text as customer\_name,

$1:"Mobile Model"::text as mobile\_key,

to\_number($1:"Quantity") as quantity,

to\_number($1:"Price per Unit") as unit\_price,

to\_decimal($1:"Total Price") as total\_price,

$1:"Promotion Code"::text as promotion\_code,

$1:"Order Amount"::number(10,2) as order\_amount,

to\_decimal($1:"Tax") as tax,

$1:"Order Date"::date as order\_dt,

$1:"Payment Status"::text as payment\_status,

$1:"Shipping Status"::text as shipping\_status,

$1:"Payment Method"::text as payment\_method,

$1:"Payment Provider"::text as payment\_provider,

$1:"Phone"::text as phone,

$1:"Delivery Address"::text as shipping\_address

from

@sales\_dwh.source.my\_internal\_stg/orders-us

(file\_format => sales\_dwh.common.my\_json\_format);

Step-5 Foreign Exchange Rate Data

Create Foreign exchange rate data to convert the local currency data (like INR or Euro) to US Dollar so when we create total sales, at global level, so we can build PKI in a single currency and compare the performance.

-- put file://C:/Users/HP/Desktop/snowflake/exchange.csv @my\_internal\_stg auto\_compress=False overwrite=True, parallel=3 ;

list @my\_internal\_stg;

use schema common;

create or replace transient table exchange\_rate(

date date,

usd2usd decimal(10,7),

usd2eu decimal(10,7),

usd2can decimal(10,7),

usd2uk decimal(10,7),

usd2inr decimal(10,7),

usd2jp decimal(10,7)

);

copy into sales\_dwh.common.exchange\_rate

from

(

select

t.$1::date as exchange\_dt,

to\_decimal(t.$2) as usd2usd,

to\_decimal(t.$3,12,10) as usd2eu,

to\_decimal(t.$4,12,10) as usd2can,

to\_decimal(t.$4,12,10) as usd2uk,

to\_decimal(t.$4,12,10) as usd2inr,

to\_decimal(t.$4,12,10) as usd2jp

from

@sales\_dwh.source.my\_internal\_stg/exchange.csv

(file\_format => 'sales\_dwh.common.my\_csv\_format') t

);

select \* from sales\_dwh.common.exchange\_rate;

Step-6.1 Loading Data From Internal Stage to Source Tables

Every time the data moves from internal stage location to source layer within permanent tables, it will add a sequence number that will help to de-duplicate the data set.

-- order table

use schema source;

create or replace sequence in\_sales\_order\_seq

start = 1

increment = 1

comment='This is sequence for India sales order table';

create or replace sequence us\_sales\_order\_seq

start = 1

increment = 1

comment='This is sequence for USA sales order table';

6.2 Source Table DDL Script

show sequences;

-- India Sales Table in Source Schema (CSV File)

create or replace transient table in\_sales\_order (

sales\_order\_key number(38,0),

order\_id varchar(),

customer\_name varchar(),

mobile\_key varchar(),

order\_quantity number(38,0),

unit\_price number(38,0),

order\_valaue number(38,0),

promotion\_code varchar(),

final\_order\_amount number(10,2),

tax\_amount number(10,2),

order\_dt date,

payment\_status varchar(),

shipping\_status varchar(),

payment\_method varchar(),

payment\_provider varchar(),

mobile varchar(),

shipping\_address varchar(),

\_metadata\_file\_name varchar(),

\_metadata\_row\_numer number(38,0),

\_metadata\_last\_modified timestamp\_ntz(9)

);

-- US Sales Table in Source Schema (Parquet File)

create or replace transient table us\_sales\_order (

sales\_order\_key number(38,0),

order\_id varchar(),

customer\_name varchar(),

mobile\_key varchar(),

order\_quantity number(38,0),

unit\_price number(38,0),

order\_valaue number(38,0),

promotion\_code varchar(),

final\_order\_amount number(10,2),

tax\_amount number(10,2),

order\_dt date,

payment\_status varchar(),

shipping\_status varchar(),

payment\_method varchar(),

payment\_provider varchar(),

phone varchar(),

shipping\_address varchar(),

\_metadata\_file\_name varchar(),

\_metadata\_row\_numer number(38,0),

\_metadata\_last\_modified timestamp\_ntz(9)

);

6.3 Snowpark Python Example To Load To Stage

Here is the example of snowpark python example to load the data from local machine, where all amazon mobile order data in 3 different formats, CSV, Parquet & JSON can be moved to snowflake internal stage.

# The Snowpark package is required for Python Worksheets.

# You can add more packages by selecting them using the Packages control and then importing them.

import snowflake.snowpark as snowpark

import sys

import logging

from snowflake.snowpark import Session, DataFrame

from snowflake.snowpark.types import StructType, StringType, StructField, StringType,LongType,DecimalType,DateType,TimestampType

from snowflake.snowpark.functions import col,lit,row\_number, rank

from snowflake.snowpark import Window

def ingest\_in\_sales(session)-> None:

session.sql(" \

copy into sales\_dwh.source.in\_sales\_order from ( \

select \

in\_sales\_order\_seq.nextval, \

t.$1::text as order\_id, \

t.$2::text as customer\_name, \

t.$3::text as mobile\_key,\

t.$4::number as order\_quantity, \

t.$5::number as unit\_price, \

t.$6::number as order\_valaue, \

t.$7::text as promotion\_code , \

t.$8::number(10,2) as final\_order\_amount,\

t.$9::number(10,2) as tax\_amount,\

t.$10::date as order\_dt,\

t.$11::text as payment\_status,\

t.$12::text as shipping\_status,\

t.$13::text as payment\_method,\

t.$14::text as payment\_provider,\

t.$15::text as mobile,\

t.$16::text as shipping\_address,\

metadata$filename as stg\_file\_name,\

metadata$file\_row\_number as stg\_row\_numer,\

metadata$file\_last\_modified as stg\_last\_modified\

from \

@sales\_dwh.source.my\_internal\_stg/orders-in.csv \

( \

file\_format => 'sales\_dwh.common.my\_csv\_format' \

) t ) on\_error = 'Continue' \

"

).collect()

def ingest\_us\_sales(session)-> None:

session.sql(' \

copy into sales\_dwh.source.us\_sales\_order \

from \

( \

select \

us\_sales\_order\_seq.nextval, \

$1:"Order ID"::text as order\_id, \

$1:"Customer Name"::text as customer\_name,\

$1:"Mobile Model"::text as mobile\_key,\

to\_number($1:"Quantity") as quantity,\

to\_number($1:"Price per Unit") as unit\_price,\

to\_decimal($1:"Total Price") as total\_price,\

$1:"Promotion Code"::text as promotion\_code,\

$1:"Order Amount"::number(10,2) as order\_amount,\

to\_decimal($1:"Tax") as tax,\

$1:"Order Date"::date as order\_dt,\

$1:"Payment Status"::text as payment\_status,\

$1:"Shipping Status"::text as shipping\_status,\

$1:"Payment Method"::text as payment\_method,\

$1:"Payment Provider"::text as payment\_provider,\

$1:"Phone"::text as phone,\

$1:"Delivery Address"::text as shipping\_address,\

metadata$filename as stg\_file\_name,\

metadata$file\_row\_number as stg\_row\_numer,\

metadata$file\_last\_modified as stg\_last\_modified\

from \

@sales\_dwh.source.my\_internal\_stg/orders-us.json\

(file\_format => sales\_dwh.common.my\_json\_format)\

) on\_error = continue \

'

).collect()

def main(session: snowpark.Session):

#ingest in sales data

ingest\_in\_sales(session)

#ingest in sales data

ingest\_us\_sales(session)

return "success"

call STAGE\_TO\_SOURCE\_TABLE\_PROC() ;

select \* from sales\_dwh.source.us\_sales\_order;

truncate table sales\_dwh.source.us\_sales\_order;

select \* from sales\_dwh.source.in\_sales\_order;

truncate table sales\_dwh.source.in\_sales\_order;

Step-7 Loading Data From Source To Curated Layer

7.1 Sequence Object Under Curated Schema Layer

-- Following are for curated schema

-- -----------------------------------

use schema curated;

create or replace sequence in\_sales\_order\_seq

start = 1

increment = 1

comment='This is sequence for India sales order table';

create or replace sequence us\_sales\_order\_seq

start = 1

increment = 1

comment='This is sequence for USA sales order table';

7.2 Curated Layer DDL

use schema curated;

-- curated India sales order table

create or replace table in\_sales\_order (

sales\_order\_key number(38,0),

order\_id varchar(),

order\_dt date,

customer\_name varchar(),

mobile\_key varchar(),

country varchar(),

region varchar(),

order\_quantity number(38,0),

local\_currency varchar(),

local\_unit\_price number(38,0),

promotion\_code varchar(),

local\_total\_order\_amt number(10,2),

local\_tax\_amt number(10,2),

exhchange\_rate number(15,7),

us\_total\_order\_amt number(23,8),

usd\_tax\_amt number(23,8),

payment\_status varchar(),

shipping\_status varchar(),

payment\_method varchar(),

payment\_provider varchar(),

conctact\_no varchar(),

shipping\_address varchar()

);

-- curated US sales order table

create or replace table us\_sales\_order (

sales\_order\_key number(38,0),

order\_id varchar(),

order\_dt date,

customer\_name varchar(),

mobile\_key varchar(),

country varchar(),

region varchar(),

order\_quantity number(38,0),

local\_currency varchar(),

local\_unit\_price number(38,0),

promotion\_code varchar(),

local\_total\_order\_amt number(10,2),

local\_tax\_amt number(10,2),

exhchange\_rate number(15,7),

us\_total\_order\_amt number(23,8),

usd\_tax\_amt number(23,8),

payment\_status varchar(),

shipping\_status varchar(),

payment\_method varchar(),

payment\_provider varchar(),

conctact\_no varchar(),

shipping\_address varchar()

);

7.3 Snowpark Python — Source To Curated (IN)

use schema source;

import sys

import logging

import snowflake.snowpark as snowpark

from snowflake.snowpark import Session, DataFrame

from snowflake.snowpark.types import StructType, StringType, StructField, StringType,LongType,DecimalType,DateType,TimestampType

from snowflake.snowpark.functions import col,lit,row\_number, rank

from snowflake.snowpark import Window

def filter\_dataset(df, column\_name, filter\_criterian) -> DataFrame:

# Payment Status = Paid

# Shipping = Delivered

return\_df = df.filter(col(column\_name) == filter\_criterian)

return return\_df

def main(session: snowpark.Session):

sales\_df = session.sql("select \* from in\_sales\_order")

# apply filter to select only paid and delivered sale orders

#select \* from in\_sales\_order where PAYMENT\_STATUS = 'Paid' and SHIPPING\_STATUS = 'Delivered'

paid\_sales\_df = filter\_dataset(sales\_df,'PAYMENT\_STATUS','Paid')

shipped\_sales\_df = filter\_dataset(paid\_sales\_df,'SHIPPING\_STATUS','Delivered')

# adding country and region to the data frame

# select \*, 'IN' as Country, 'APAC' as Region from in\_sales\_order where PAYMENT\_STATUS = 'Paid' and SHIPPING\_STATUS = 'Delivered'

country\_sales\_df = shipped\_sales\_df.with\_column('Country',lit('IN')).with\_column('Region',lit('APAC'))

# join to add forex calculation

forex\_df = session.sql("select \* from sales\_dwh.common.exchange\_rate")

sales\_with\_forext\_df = country\_sales\_df.join(forex\_df,country\_sales\_df['order\_dt']==forex\_df['date'],join\_type='outer')

#sales\_with\_forext\_df.show(2)

#de-duplication

#print(sales\_with\_forext\_df.count())

unique\_orders = sales\_with\_forext\_df.with\_column('order\_rank',rank().over(Window.partitionBy(col("order\_dt")).order\_by(col('\_metadata\_last\_modified').desc()))).filter(col("order\_rank")==1).select(col('SALES\_ORDER\_KEY').alias('unique\_sales\_order\_key'))

final\_sales\_df = unique\_orders.join(sales\_with\_forext\_df,unique\_orders['unique\_sales\_order\_key']==sales\_with\_forext\_df['SALES\_ORDER\_KEY'],join\_type='inner')

final\_sales\_df = final\_sales\_df.select(

col('SALES\_ORDER\_KEY'),

col('ORDER\_ID'),

col('ORDER\_DT'),

col('CUSTOMER\_NAME'),

col('MOBILE\_KEY'),

col('Country'),

col('Region'),

col('ORDER\_QUANTITY'),

lit('INR').alias('LOCAL\_CURRENCY'),

col('UNIT\_PRICE').alias('LOCAL\_UNIT\_PRICE'),

col('PROMOTION\_CODE').alias('PROMOTION\_CODE'),

col('FINAL\_ORDER\_AMOUNT').alias('LOCAL\_TOTAL\_ORDER\_AMT'),

col('TAX\_AMOUNT').alias('local\_tax\_amt'),

col('USD2INR').alias("Exhchange\_Rate"),

(col('FINAL\_ORDER\_AMOUNT')/col('USD2INR')).alias('US\_TOTAL\_ORDER\_AMT'),

(col('TAX\_AMOUNT')/col('USD2INR')).alias('USD\_TAX\_AMT'),

col('payment\_status'),

col('shipping\_status'),

col('payment\_method'),

col('payment\_provider'),

col('mobile').alias('conctact\_no'),

col('shipping\_address')

)

#final\_sales\_df.show(5)

final\_sales\_df.write.save\_as\_table("sales\_dwh.curated.in\_sales\_order",mode="append")

return "Success"

call SOURCE\_CURRATED\_TABLE\_IN\_PROC();

select \* from sales\_dwh.curated.in\_sales\_order;

truncate table sales\_dwh.curated.in\_sales\_order;

7.4 Snowpark Python — Source To Curated (US)

import sys

import logging

import snowflake.snowpark as snowpark

from snowflake.snowpark import Session, DataFrame

from snowflake.snowpark.types import StructType, StringType, StructField, StringType,LongType,DecimalType,DateType,TimestampType

from snowflake.snowpark.functions import col,lit,row\_number, rank

from snowflake.snowpark import Window

def filter\_dataset(df, column\_name, filter\_criterian) -> DataFrame:

# Payment Status = Paid

# Shipping = Delivered

return\_df = df.filter(col(column\_name) == filter\_criterian)

return return\_df

def main(session: snowpark.Session):

sales\_df = session.sql("select \* from us\_sales\_order")

paid\_sales\_df = filter\_dataset(sales\_df,'PAYMENT\_STATUS','Paid')

shipped\_sales\_df = filter\_dataset(paid\_sales\_df,'SHIPPING\_STATUS','Delivered')

# adding country and region to the data frame

# select \*, 'IN' as Country, 'APAC' as Region from us\_sales\_order where PAYMENT\_STATUS = 'Paid' and SHIPPING\_STATUS = 'Delivered'

country\_sales\_df = shipped\_sales\_df.with\_column('Country',lit('US')).with\_column('Region',lit('NA'))

# join to add forex calculation

forex\_df = session.sql("select \* from sales\_dwh.common.exchange\_rate")

sales\_with\_forext\_df = country\_sales\_df.join(forex\_df,country\_sales\_df['order\_dt']==forex\_df['date'],join\_type='outer')

#sales\_with\_forext\_df.show(2)

#de-duplication

print(sales\_with\_forext\_df.count())

unique\_orders = sales\_with\_forext\_df.with\_column('order\_rank',rank().over(Window.partitionBy(col("order\_dt")).order\_by(col('\_metadata\_last\_modified').desc()))).filter(col("order\_rank")==1).select(col('SALES\_ORDER\_KEY').alias('unique\_sales\_order\_key'))

final\_sales\_df = unique\_orders.join(sales\_with\_forext\_df,unique\_orders['unique\_sales\_order\_key']==sales\_with\_forext\_df['SALES\_ORDER\_KEY'],join\_type='inner')

final\_sales\_df = final\_sales\_df.select(

col('SALES\_ORDER\_KEY'),

col('ORDER\_ID'),

col('ORDER\_DT'),

col('CUSTOMER\_NAME'),

col('MOBILE\_KEY'),

col('Country'),

col('Region'),

col('ORDER\_QUANTITY'),

lit('USD').alias('LOCAL\_CURRENCY'),

col('UNIT\_PRICE').alias('LOCAL\_UNIT\_PRICE'),

col('PROMOTION\_CODE').alias('PROMOTION\_CODE'),

col('FINAL\_ORDER\_AMOUNT').alias('LOCAL\_TOTAL\_ORDER\_AMT'),

col('TAX\_AMOUNT').alias('local\_tax\_amt'),

col('USD2INR').alias("Exhchange\_Rate"),

(col('FINAL\_ORDER\_AMOUNT')/col('USD2USD')).alias('US\_TOTAL\_ORDER\_AMT'),

(col('TAX\_AMOUNT')/col('USD2USD')).alias('USD\_TAX\_AMT'),

col('payment\_status'),

col('shipping\_status'),

col('payment\_method'),

col('payment\_provider'),

col('phone').alias('conctact\_no'),

col('shipping\_address')

)

#final\_sales\_df.show(5)

final\_sales\_df.write.save\_as\_table("sales\_dwh.curated.us\_sales\_order",mode="append")

return "Success"

call SOURCE\_CURRATED\_TABLE\_US\_PROC() ;

select \* from sales\_dwh.curated.us\_sales\_order;

truncate table sales\_dwh.curated.us\_sales\_order;

Step-8 Working on Consumption Layer

Step-8.1 Dimension Tables & Sequence Object

-- region dimension

use schema consumption;

create or replace sequence region\_dim\_seq start = 1 increment = 1;

create or replace transient table region\_dim(

region\_id\_pk number primary key,

Country text,

Region text,

isActive text(1)

);

select \* from product\_dim;

-- product dimension

use schema consumption;

create or replace sequence product\_dim\_seq start = 1 increment = 1;

create or replace transient table product\_dim(

product\_id\_pk number primary key,

Mobile\_key text,

Brand text,

Model text,

Color text,

Memory text,

isActive text(1)

);

-- customer dimension

use schema consumption;

create or replace sequence customer\_dim\_seq start = 1 increment = 1;

create or replace transient table customer\_dim(

customer\_id\_pk number primary key,

customer\_name text,

CONCTACT\_NO text,

SHIPPING\_ADDRESS text,

country text,

region text,

isActive text(1)

);

-- payment dimension

use schema consumption;

create or replace sequence payment\_dim\_seq start = 1 increment = 1;

create or replace transient table payment\_dim(

payment\_id\_pk number primary key,

PAYMENT\_METHOD text,

PAYMENT\_PROVIDER text,

country text,

region text,

isActive text(1)

);

-- fact tables

create or replace table sales\_fact (

order\_code varchar(),

region\_id\_fk number(38,0),

customer\_id\_fk number(38,0),

payment\_id\_fk number(38,0),

product\_id\_fk number(38,0),

order\_quantity number(38,0),

local\_total\_order\_amt number(10,2),

local\_tax\_amt number(10,2),

exhchange\_rate number,

us\_total\_order\_amt number,

usd\_tax\_amt number

);

select \* from sales\_dwh.consumption.sales\_fact;

-- Table Containts

alter table sales\_fact add

constraint fk\_sales\_region FOREIGN KEY (REGION\_ID\_FK) REFERENCES region\_dim (REGION\_ID\_PK) NOT ENFORCED;

alter table sales\_fact add

constraint fk\_sales\_customer FOREIGN KEY (CUSTOMER\_ID\_FK) REFERENCES customer\_dim (CUSTOMER\_ID\_PK) NOT ENFORCED;

--

alter table sales\_fact add

constraint fk\_sales\_payment FOREIGN KEY (PAYMENT\_ID\_FK) REFERENCES payment\_dim (PAYMENT\_ID\_PK) NOT ENFORCED;

alter table sales\_fact add

constraint fk\_sales\_product FOREIGN KEY (PRODUCT\_ID\_FK) REFERENCES product\_dim (PRODUCT\_ID\_PK) NOT ENFORCED;

8.2 Curated To Model Snowpark Python Example

import sys

import logging

import snowflake.snowpark as snowpark

from snowflake.snowpark import Session, DataFrame

from snowflake.snowpark.types import StructType, StringType, StructField, StringType,LongType,DecimalType,DateType,TimestampType

from snowflake.snowpark.functions import col,lit,row\_number, rank,split,cast

from snowflake.snowpark import Window

def create\_region\_dim(all\_sales\_df,session)-> None:

region\_dim\_df = all\_sales\_df.groupBy(col("Country"),col("Region")).count()

region\_dim\_df.show(2)

region\_dim\_df = region\_dim\_df.with\_column("isActive",lit('Y'))

region\_dim\_df = region\_dim\_df.selectExpr("sales\_dwh.consumption.region\_dim\_seq.nextval as region\_id\_pk","Country", "Region", "isActive")

#region\_dim\_df.write.save\_as\_table('sales\_dwh.consumption.region\_dim',mode="append")

region\_dim\_df.show(5)

# part 2 where delta data will be processed

existing\_region\_dim\_df = session.sql("select Country, Region from sales\_dwh.consumption.region\_dim")

region\_dim\_df = region\_dim\_df.join(existing\_region\_dim\_df,region\_dim\_df['Country']==existing\_region\_dim\_df['Country'],join\_type='leftanti')

region\_dim\_df.show(5)

intsert\_cnt = int(region\_dim\_df.count())

if intsert\_cnt>0:

region\_dim\_df.write.save\_as\_table("sales\_dwh.consumption.region\_dim",mode="append")

print("save operation ran...")

else:

print("No insert ...Opps...")

# have exclude key

def create\_product\_dim(all\_sales\_df,session)-> None:

product\_dim\_df = all\_sales\_df.with\_column("Brand",split(col('MOBILE\_KEY'),lit('/'))[0]) \

.with\_column("Model",split(col('MOBILE\_KEY'),lit('/'))[1]) \

.with\_column("Color",split(col('MOBILE\_KEY'),lit('/'))[2]) \

.with\_column("Memory",split(col('MOBILE\_KEY'),lit('/'))[3]) \

.select(col('mobile\_key'),col('Brand'),col('Model'),col('Color'),col('Memory'))

product\_dim\_df = product\_dim\_df.select(col('mobile\_key'), \

cast(col('Brand'), StringType()).as\_("Brand"),\

cast(col('Model'), StringType()).as\_("Model"),\

cast(col('Color'), StringType()).as\_("Color"),\

cast(col('Memory'), StringType()).as\_("Memory")\

)

product\_dim\_df = product\_dim\_df.groupBy(col('mobile\_key'),col("Brand"),col("Model"),col("Color"),col("Memory")).count()

product\_dim\_df = product\_dim\_df.with\_column("isActive",lit('Y'))

#fetch existing product dim records.

existing\_product\_dim\_df = session.sql("select mobile\_key, Brand, Model, Color, Memory from sales\_dwh.consumption.product\_dim")

existing\_product\_dim\_df.count()

product\_dim\_df = product\_dim\_df.join(existing\_product\_dim\_df,["mobile\_key", "Brand", "Model", "Color", "Memory"],join\_type='leftanti')

product\_dim\_df.show(5)

product\_dim\_df = product\_dim\_df.selectExpr("sales\_dwh.consumption.product\_dim\_seq.nextval as product\_id\_pk","mobile\_key","Brand", "Model","Color","Memory", "isActive")

product\_dim\_df.show(5)

intsert\_cnt = int(product\_dim\_df.count())

if intsert\_cnt>0:

product\_dim\_df.write.save\_as\_table("sales\_dwh.consumption.product\_dim",mode="append")

print("save operation ran...")

else:

print("No insert ...Opps...")

def create\_customer\_dim(all\_sales\_df, session) -> None:

customer\_dim\_df = all\_sales\_df.groupBy(col("COUNTRY"),col("REGION"),col("CUSTOMER\_NAME"),col("CONCTACT\_NO"),col("SHIPPING\_ADDRESS")).count()

customer\_dim\_df = customer\_dim\_df.with\_column("isActive",lit('Y'))

customer\_dim\_df = customer\_dim\_df.selectExpr("customer\_name", "conctact\_no","shipping\_address","country","region" ,"isactive")

#region\_dim\_df.write.save\_as\_table('sales\_dwh.consumption.region\_dim',mode="append")

customer\_dim\_df.show(5)

# part 2 where delta data will be processed

existing\_customer\_dim\_df = session.sql("select customer\_name,conctact\_no,shipping\_address,country, region from sales\_dwh.consumption.customer\_dim")

customer\_dim\_df = customer\_dim\_df.join(existing\_customer\_dim\_df,["customer\_name","conctact\_no","shipping\_address","country", "region"],join\_type='leftanti')

customer\_dim\_df = customer\_dim\_df.selectExpr("sales\_dwh.consumption.customer\_dim\_seq.nextval as customer\_id\_pk","customer\_name", "conctact\_no","shipping\_address","country","region", "isActive")

customer\_dim\_df.show(5)

intsert\_cnt = int(customer\_dim\_df.count())

if intsert\_cnt>0:

customer\_dim\_df.write.save\_as\_table("sales\_dwh.consumption.customer\_dim",mode="append")

print("save operation ran...")

else:

print("No insert ...Opps...")

def create\_payment\_dim(all\_sales\_df, session) -> None:

payment\_dim\_df = all\_sales\_df.groupBy(col("COUNTRY"),col("REGION"),col("payment\_method"),col("payment\_provider")).count()

payment\_dim\_df = payment\_dim\_df.with\_column("isActive",lit('Y'))

#region\_dim\_df.write.save\_as\_table('sales\_dwh.consumption.region\_dim',mode="append")

payment\_dim\_df.show(5)

# part 2 where delta data will be processed

existing\_payment\_dim\_df = session.sql("select payment\_method,payment\_provider,country, region from sales\_dwh.consumption.payment\_dim")

payment\_dim\_df = payment\_dim\_df.join(existing\_payment\_dim\_df,["payment\_method","payment\_provider","country", "region"],join\_type='leftanti')

payment\_dim\_df = payment\_dim\_df.selectExpr("sales\_dwh.consumption.payment\_dim\_seq.nextval as payment\_id\_pk","payment\_method", "payment\_provider","country","region", "isActive")

intsert\_cnt = int(payment\_dim\_df.count())

if intsert\_cnt>0:

payment\_dim\_df.write.save\_as\_table("sales\_dwh.consumption.payment\_dim",mode="append")

print("save operation ran...")

else:

print("No insert ...Opps...")

def main(session: snowpark.Session):

#get the session object and get dataframe

in\_sales\_df = session.sql("select \* from sales\_dwh.curated.in\_sales\_order")

us\_sales\_df = session.sql("select \* from sales\_dwh.curated.us\_sales\_order")

all\_sales\_df = in\_sales\_df.union(us\_sales\_df)

create\_region\_dim(all\_sales\_df,session) #region dimension

create\_product\_dim(all\_sales\_df,session) #product dimension

create\_customer\_dim(all\_sales\_df,session) #customer dimension

create\_payment\_dim(all\_sales\_df,session) #payment dimension

customer\_dim\_df = session.sql("select customer\_id\_pk, customer\_name, country, region from sales\_dwh.consumption.CUSTOMER\_DIM")

payment\_dim\_df = session.sql("select payment\_id\_pk, payment\_method, payment\_provider, country, region from sales\_dwh.consumption.PAYMENT\_DIM")

product\_dim\_df = session.sql("select product\_id\_pk, mobile\_key from sales\_dwh.consumption.PRODUCT\_DIM")

region\_dim\_df = session.sql("select region\_id\_pk,country, region from sales\_dwh.consumption.REGION\_DIM")

all\_sales\_df = all\_sales\_df.join(customer\_dim\_df, ["customer\_name","region","country"],join\_type='inner')

all\_sales\_df = all\_sales\_df.join(payment\_dim\_df, ["payment\_method", "payment\_provider", "country", "region"],join\_type='inner')

#all\_sales\_df = all\_sales\_df.join(product\_dim\_df, ["brand","model","color","Memory"],join\_type='inner')

all\_sales\_df = all\_sales\_df.join(product\_dim\_df, ["mobile\_key"],join\_type='inner')

all\_sales\_df = all\_sales\_df.join(region\_dim\_df, ["country", "region"],join\_type='inner')

all\_sales\_df= all\_sales\_df[['order\_id','region\_id\_pk','customer\_id\_pk','payment\_id\_pk','product\_id\_pk','order\_quantity'\

,'local\_total\_order\_amt' ,'local\_tax\_amt' ,'exhchange\_rate','us\_total\_order\_amt' ,'usd\_tax\_amt' ]]

all\_sales\_df.write.save\_as\_table("sales\_dwh.consumption.sales\_fact",mode="append")

return all\_sales\_df

CALL CURRATED\_MODEL\_DIM\_FACT\_TABLE\_PROC();

select \* from sales\_dwh.consumption.sales\_fact;

truncate table sales\_dwh.consumption.sales\_fact;