

**SQL Scripts:**-- use sysadmin role.  
use role sysadmin;  
  
-- create a warehouse if not exist   
create warehouse if not exists adhoc\_wh  
 comment = 'This is the adhoc-wh'  
 warehouse\_size = 'x-small'   
 auto\_resume = true   
 auto\_suspend = 60   
 enable\_query\_acceleration = false   
 warehouse\_type = 'standard'   
 min\_cluster\_count = 1   
 max\_cluster\_count = 1   
 scaling\_policy = 'standard'  
 initially\_suspended = true;  
  
-- create development database/schema if does not exist  
create database if not exists sandbox;  
use database sandbox;  
create schema if not exists stage\_sch;  
create schema if not exists clean\_sch;  
create schema if not exists consumption\_sch;  
create schema if not exists common;  
  
use schema stage\_sch;  
  
 -- create file format to process the CSV file  
 create file format if not exists stage\_sch.csv\_file\_format   
 type = 'csv'   
 compression = 'auto'   
 field\_delimiter = ','   
 record\_delimiter = '\n'   
 skip\_header = 1   
 field\_optionally\_enclosed\_by = '\042'   
 null\_if = ('\\N');  
  
create stage stage\_sch.csv\_stg  
 directory = ( enable = true )  
 comment = 'this is the snowflake internal stage';  
  
  
create or replace tag   
 common.pii\_policy\_tag   
 allowed\_values 'PII','PRICE','SENSITIVE','EMAIL'  
 comment = 'This is PII policy tag object';  
  
create or replace masking policy   
 common.pii\_masking\_policy as (pii\_text string)  
 returns string ->   
 to\_varchar('\*\* PII \*\*');  
  
create or replace masking policy   
 common.email\_masking\_policy as (email\_text string)  
 returns string ->   
 to\_varchar('\*\* EAMIL \*\*');  
  
create or replace masking policy   
 common.phone\_masking\_policy as (phone string)  
 returns string ->   
 to\_varchar('\*\* Phone \*\*');

**Sample CSV Files**

Here is the [sample data setup](https://drive.google.com/file/d/1sAdDME9fqmXtVbP0JaGpAlgxILIUF4mK/view?usp=sharing) that must be loaded before runing SQL scripts.

**Location Dimension — SQL Script**

Press enter or click to view image in full size

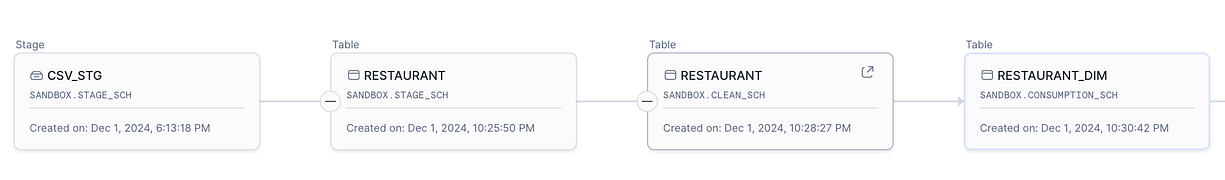


**Location Dimension Data Lineage**

use role sysadmin;  
use schema sandbox.stage\_sch;  
  
create table stage\_sch.location (  
 locationid text,  
 city text,  
 state text,  
 zipcode text,  
 activeflag text,  
 createddate text,  
 modifieddate text,  
 -- audit columns for tracking & debugging  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the location stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.'  
;  
  
create or replace stream stage\_sch.location\_stm   
on table stage\_sch.location  
append\_only = true  
comment = 'this is the append-only stream object on location table that gets delta data based on changes';  
  
select \* from stage\_sch.location;  
  
copy into stage\_sch.location (locationid, city, state, zipcode, activeflag,   
 createddate, modifieddate, \_stg\_file\_name,   
 \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as locationid,  
 t.$2::text as city,  
 t.$3::text as state,  
 t.$4::text as zipcode,  
 t.$5::text as activeflag,  
 t.$6::text as createddate,  
 t.$7::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/location t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
select \*  
from table(information\_schema.copy\_history(table\_name=>'LOCATION', start\_time=> dateadd(hours, -1, current\_timestamp())));  
  
  
select \* from stage\_sch.location;  
select \* from stage\_sch.location\_stm;  
  
use schema clean\_sch;  
  
-- Level 2  
create or replace table clean\_sch.restaurant\_location (  
 restaurant\_location\_sk number autoincrement primary key,  
 location\_id number not null unique,  
 city string(100) not null,  
 state string(100) not null,  
 state\_code string(2) not null,  
 is\_union\_territory boolean not null default false,  
 capital\_city\_flag boolean not null default false,  
 city\_tier text(6),  
 zip\_code string(10) not null,  
 active\_flag string(10) not null,  
 created\_ts timestamp\_tz not null,  
 modified\_ts timestamp\_tz,  
   
 -- additional audit columns  
 \_stg\_file\_name string,  
 \_stg\_file\_load\_ts timestamp\_ntz,  
 \_stg\_file\_md5 string,  
 \_copy\_data\_ts timestamp\_ntz default current\_timestamp  
)  
comment = 'Location entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
create or replace stream clean\_sch.restaurant\_location\_stm   
on table clean\_sch.restaurant\_location  
comment = 'this is a standard stream object on the location table to track insert, update, and delete changes';  
  
  
MERGE INTO clean\_sch.restaurant\_location AS target  
USING (  
 SELECT   
 CAST(LocationID AS NUMBER) AS Location\_ID,  
 CAST(City AS STRING) AS City,  
 CASE   
 WHEN CAST(State AS STRING) = 'Delhi' THEN 'New Delhi'  
 ELSE CAST(State AS STRING)  
 END AS State,  
 -- State Code Mapping  
 CASE   
 WHEN State = 'Delhi' THEN 'DL'  
 WHEN State = 'Maharashtra' THEN 'MH'  
 WHEN State = 'Uttar Pradesh' THEN 'UP'  
 WHEN State = 'Gujarat' THEN 'GJ'  
 WHEN State = 'Rajasthan' THEN 'RJ'  
 WHEN State = 'Kerala' THEN 'KL'  
 WHEN State = 'Punjab' THEN 'PB'  
 WHEN State = 'Karnataka' THEN 'KA'  
 WHEN State = 'Madhya Pradesh' THEN 'MP'  
 WHEN State = 'Odisha' THEN 'OR'  
 WHEN State = 'Chandigarh' THEN 'CH'  
 WHEN State = 'West Bengal' THEN 'WB'  
 WHEN State = 'Sikkim' THEN 'SK'  
 WHEN State = 'Andhra Pradesh' THEN 'AP'  
 WHEN State = 'Assam' THEN 'AS'  
 WHEN State = 'Jammu and Kashmir' THEN 'JK'  
 WHEN State = 'Puducherry' THEN 'PY'  
 WHEN State = 'Uttarakhand' THEN 'UK'  
 WHEN State = 'Himachal Pradesh' THEN 'HP'  
 WHEN State = 'Tamil Nadu' THEN 'TN'  
 WHEN State = 'Goa' THEN 'GA'  
 WHEN State = 'Telangana' THEN 'TG'  
 WHEN State = 'Chhattisgarh' THEN 'CG'  
 WHEN State = 'Jharkhand' THEN 'JH'  
 WHEN State = 'Bihar' THEN 'BR'  
 ELSE NULL  
 END AS state\_code,  
 CASE   
 WHEN State IN ('Delhi', 'Chandigarh', 'Puducherry', 'Jammu and Kashmir') THEN 'Y'  
 ELSE 'N'  
 END AS is\_union\_territory,  
 CASE   
 WHEN (State = 'Delhi' AND City = 'New Delhi') THEN TRUE  
 WHEN (State = 'Maharashtra' AND City = 'Mumbai') THEN TRUE  
 -- Other conditions for capital cities  
 ELSE FALSE  
 END AS capital\_city\_flag,  
 CASE   
 WHEN City IN ('Mumbai', 'Delhi', 'Bengaluru', 'Hyderabad', 'Chennai', 'Kolkata', 'Pune', 'Ahmedabad') THEN 'Tier-1'  
 WHEN City IN ('Jaipur', 'Lucknow', 'Kanpur', 'Nagpur', 'Indore', 'Bhopal', 'Patna', 'Vadodara', 'Coimbatore',   
 'Ludhiana', 'Agra', 'Nashik', 'Ranchi', 'Meerut', 'Raipur', 'Guwahati', 'Chandigarh') THEN 'Tier-2'  
 ELSE 'Tier-3'  
 END AS city\_tier,  
 CAST(ZipCode AS STRING) AS Zip\_Code,  
 CAST(ActiveFlag AS STRING) AS Active\_Flag,  
 TO\_TIMESTAMP\_TZ(CreatedDate, 'YYYY-MM-DD HH24:MI:SS') AS created\_ts,  
 TO\_TIMESTAMP\_TZ(ModifiedDate, 'YYYY-MM-DD HH24:MI:SS') AS modified\_ts,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 CURRENT\_TIMESTAMP AS \_copy\_data\_ts  
 FROM stage\_sch.location\_stm  
) AS source  
ON target.Location\_ID = source.Location\_ID  
WHEN MATCHED AND (  
 target.City != source.City OR  
 target.State != source.State OR  
 target.state\_code != source.state\_code OR  
 target.is\_union\_territory != source.is\_union\_territory OR  
 target.capital\_city\_flag != source.capital\_city\_flag OR  
 target.city\_tier != source.city\_tier OR  
 target.Zip\_Code != source.Zip\_Code OR  
 target.Active\_Flag != source.Active\_Flag OR  
 target.modified\_ts != source.modified\_ts  
) THEN   
 UPDATE SET   
 target.City = source.City,  
 target.State = source.State,  
 target.state\_code = source.state\_code,  
 target.is\_union\_territory = source.is\_union\_territory,  
 target.capital\_city\_flag = source.capital\_city\_flag,  
 target.city\_tier = source.city\_tier,  
 target.Zip\_Code = source.Zip\_Code,  
 target.Active\_Flag = source.Active\_Flag,  
 target.modified\_ts = source.modified\_ts,  
 target.\_stg\_file\_name = source.\_stg\_file\_name,  
 target.\_stg\_file\_load\_ts = source.\_stg\_file\_load\_ts,  
 target.\_stg\_file\_md5 = source.\_stg\_file\_md5,  
 target.\_copy\_data\_ts = source.\_copy\_data\_ts  
WHEN NOT MATCHED THEN  
 INSERT (  
 Location\_ID,  
 City,  
 State,  
 state\_code,  
 is\_union\_territory,  
 capital\_city\_flag,  
 city\_tier,  
 Zip\_Code,  
 Active\_Flag,  
 created\_ts,  
 modified\_ts,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 )  
 VALUES (  
 source.Location\_ID,  
 source.City,  
 source.State,  
 source.state\_code,  
 source.is\_union\_territory,  
 source.capital\_city\_flag,  
 source.city\_tier,  
 source.Zip\_Code,  
 source.Active\_Flag,  
 source.created\_ts,  
 source.modified\_ts,  
 source.\_stg\_file\_name,  
 source.\_stg\_file\_load\_ts,  
 source.\_stg\_file\_md5,  
 source.\_copy\_data\_ts  
 );  
  
create or replace table consumption\_sch.restaurant\_location\_dim (  
 restaurant\_location\_hk NUMBER primary key, -- hash key for the dimension  
 location\_id number(38,0) not null, -- business key  
 city varchar(100) not null, -- city  
 state varchar(100) not null, -- state  
 state\_code varchar(2) not null, -- state code  
 is\_union\_territory boolean not null default false, -- union territory flag  
 capital\_city\_flag boolean not null default false, -- capital city flag  
 city\_tier varchar(6), -- city tier  
 zip\_code varchar(10) not null, -- zip code  
 active\_flag varchar(10) not null, -- active flag (indicating current record)  
 eff\_start\_dt timestamp\_tz(9) not null, -- effective start date for scd2  
 eff\_end\_dt timestamp\_tz(9), -- effective end date for scd2  
 current\_flag boolean not null default true -- indicator of the current record  
)  
comment = 'Dimension table for restaurant location with scd2 (slowly changing dimension) enabled and hashkey as surrogate key';  
  
  
MERGE INTO   
 CONSUMPTION\_SCH.RESTAURANT\_LOCATION\_DIM AS target  
 USING   
 CLEAN\_SCH.RESTAURANT\_LOCATION\_STM AS source  
 ON   
 target.LOCATION\_ID = source.LOCATION\_ID and   
 target.ACTIVE\_FLAG = source.ACTIVE\_FLAG  
 WHEN MATCHED   
 AND source.METADATA$ACTION = 'DELETE' and source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Update the existing record to close its validity period  
 UPDATE SET   
 target.EFF\_END\_DT = CURRENT\_TIMESTAMP(),  
 target.CURRENT\_FLAG = FALSE  
 WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' and source.METADATA$ISUPDATE = 'TRUE'  
 THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 RESTAURANT\_LOCATION\_HK,  
 LOCATION\_ID,  
 CITY,  
 STATE,  
 STATE\_CODE,  
 IS\_UNION\_TERRITORY,  
 CAPITAL\_CITY\_FLAG,  
 CITY\_TIER,  
 ZIP\_CODE,  
 ACTIVE\_FLAG,  
 EFF\_START\_DT,  
 EFF\_END\_DT,  
 CURRENT\_FLAG  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.CITY, source.STATE, source.STATE\_CODE, source.ZIP\_CODE))),  
 source.LOCATION\_ID,  
 source.CITY,  
 source.STATE,  
 source.STATE\_CODE,  
 source.IS\_UNION\_TERRITORY,  
 source.CAPITAL\_CITY\_FLAG,  
 source.CITY\_TIER,  
 source.ZIP\_CODE,  
 source.ACTIVE\_FLAG,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 )  
 WHEN NOT MATCHED AND   
 source.METADATA$ACTION = 'INSERT' and source.METADATA$ISUPDATE = 'FALSE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 RESTAURANT\_LOCATION\_HK,  
 LOCATION\_ID,  
 CITY,  
 STATE,  
 STATE\_CODE,  
 IS\_UNION\_TERRITORY,  
 CAPITAL\_CITY\_FLAG,  
 CITY\_TIER,  
 ZIP\_CODE,  
 ACTIVE\_FLAG,  
 EFF\_START\_DT,  
 EFF\_END\_DT,  
 CURRENT\_FLAG  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.CITY, source.STATE, source.STATE\_CODE, source.ZIP\_CODE))),  
 source.LOCATION\_ID,  
 source.CITY,  
 source.STATE,  
 source.STATE\_CODE,  
 source.IS\_UNION\_TERRITORY,  
 source.CAPITAL\_CITY\_FLAG,  
 source.CITY\_TIER,  
 source.ZIP\_CODE,  
 source.ACTIVE\_FLAG,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 );  
  
-- Part-2  
copy into stage\_sch.location (locationid, city, state, zipcode, activeflag,   
 createddate, modifieddate, \_stg\_file\_name,   
 \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as locationid,  
 t.$2::text as city,  
 t.$3::text as state,  
 t.$4::text as zipcode,  
 t.$5::text as activeflag,  
 t.$6::text as createddate,  
 t.$7::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/location/delta-day02-2rows-update.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Restaurant Dimension — SQL Script**

Press enter or click to view image in full size

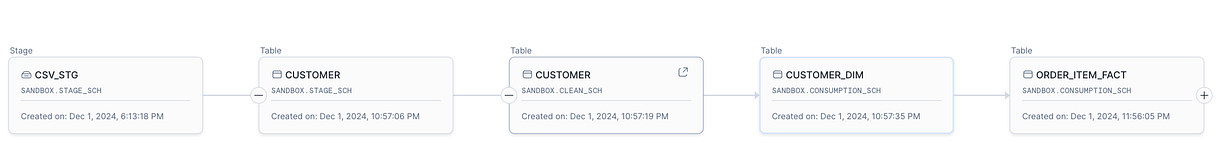


**Restaurant Dimension Data Lineage**

-- change context  
use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
-- create restaurant table under stage location, with all text value + audit column for copy command  
create or replace table stage\_sch.restaurant (  
 restaurantid text,   
 name text , -- restaurant name, required field  
 cuisinetype text, -- type of cuisine offered  
 pricing\_for\_2 text, -- pricing for two people as text  
 restaurant\_phone text WITH TAG (common.pii\_policy\_tag = 'SENSITIVE'), -- phone number as text  
 operatinghours text, -- restaurant operating hours  
 locationid text , -- location id, default as text  
 activeflag text , -- active status  
 openstatus text , -- open status  
 locality text, -- locality as text  
 restaurant\_address text, -- address as text  
 latitude text, -- latitude as text for precision  
 longitude text, -- longitude as text for precision  
 createddate text, -- record creation date  
 modifieddate text, -- last modified date  
  
 -- audit columns for debugging  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the restaurant stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.'  
;  
  
-- Stream object to capture the changes.   
create or replace stream stage\_sch.restaurant\_stm   
on table stage\_sch.restaurant  
append\_only = true  
comment = 'This is the append-only stream object on restaurant table that only gets delta data';  
  
  
-- run copy command to load the data into stage-restaurant table.  
copy into stage\_sch.restaurant (restaurantid, name, cuisinetype, pricing\_for\_2, restaurant\_phone,   
 operatinghours, locationid, activeflag, openstatus,   
 locality, restaurant\_address, latitude, longitude,   
 createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as restaurantid, -- restaurantid as the first column  
 t.$2::text as name,  
 t.$3::text as cuisinetype,  
 t.$4::text as pricing\_for\_2,  
 t.$5::text as restaurant\_phone,  
 t.$6::text as operatinghours,  
 t.$7::text as locationid,  
 t.$8::text as activeflag,  
 t.$9::text as openstatus,  
 t.$10::text as locality,  
 t.$11::text as restaurant\_address,  
 t.$12::text as latitude,  
 t.$13::text as longitude,  
 t.$14::text as createddate,  
 t.$15::text as modifieddate,  
 -- audit columns for tracking & debugging  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp() as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/restaurant/restaurant-delhi+NCR.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
  
-- the restaurant table where data types are defined.   
create or replace table clean\_sch.restaurant (  
 restaurant\_sk number autoincrement primary key, -- primary key with auto-increment  
 restaurant\_id number unique, -- restaurant id without auto-increment  
 name string(100) not null, -- restaurant name, required field  
 cuisine\_type string, -- type of cuisine offered  
 pricing\_for\_two number(10, 2), -- pricing for two people, up to 10 digits with 2 decimal places  
 restaurant\_phone string(15) WITH TAG (common.pii\_policy\_tag = 'SENSITIVE'), -- phone number, supports 10-digit or international format  
 operating\_hours string(100), -- restaurant operating hours  
 location\_id\_fk number, -- reference id for location, defaulted to 1  
 active\_flag string(10), -- indicates if the restaurant is active  
 open\_status string(10), -- indicates if the restaurant is currently open  
 locality string(100), -- locality of the restaurant  
 restaurant\_address string, -- address of the restaurant, supports longer text  
 latitude number(9, 6), -- latitude with 6 decimal places for precision  
 longitude number(9, 6), -- longitude with 6 decimal places for precision  
 created\_dt timestamp\_tz, -- record creation date  
 modified\_dt timestamp\_tz, -- last modified date, allows null if not modified  
  
 -- additional audit columns  
 \_stg\_file\_name string, -- file name for audit  
 \_stg\_file\_load\_ts timestamp\_ntz, -- file load timestamp for audit  
 \_stg\_file\_md5 string, -- md5 hash for file content for audit  
 \_copy\_data\_ts timestamp\_ntz default current\_timestamp -- timestamp when data is copied, defaults to current timestamp  
)  
comment = 'Restaurant entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
create or replace stream clean\_sch.restaurant\_stm   
on table clean\_sch.restaurant  
comment = 'This is a standard stream object on the clean restaurant table to track insert, update, and delete changes';  
  
-- following is the insert statement..  
insert into clean\_sch.restaurant (  
 restaurant\_id,  
 name,  
 cuisine\_type,  
 pricing\_for\_two,  
 restaurant\_phone,  
 operating\_hours,  
 location\_id\_fk,  
 active\_flag,  
 open\_status,  
 locality,  
 restaurant\_address,  
 latitude,  
 longitude,  
 created\_dt,  
 modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5  
)  
select   
 try\_cast(restaurantid as number) as restaurant\_id,  
 try\_cast(name as string) as name,  
 try\_cast(cuisinetype as string) as cuisine\_type,  
 try\_cast(pricing\_for\_2 as number(10, 2)) as pricing\_for\_two,  
 try\_cast(restaurant\_phone as string) as restaurant\_phone,  
 try\_cast(operatinghours as string) as operating\_hours,  
 try\_cast(locationid as number) as location\_id\_fk,  
 try\_cast(activeflag as string) as active\_flag,  
 try\_cast(openstatus as string) as open\_status,  
 try\_cast(locality as string) as locality,  
 try\_cast(restaurant\_address as string) as restaurant\_address,  
 try\_cast(latitude as number(9, 6)) as latitude,  
 try\_cast(longitude as number(9, 6)) as longitude,  
 try\_to\_timestamp\_ntz(createddate, 'YYYY-MM-DD HH24:MI:SS.FF9') as created\_dt,  
 try\_to\_timestamp\_ntz(modifieddate, 'YYYY-MM-DD HH24:MI:SS.FF9') as modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5  
from   
 stage\_sch.restaurant;  
   
-- here is the merge statement  
MERGE INTO clean\_sch.restaurant AS target  
USING (  
 SELECT   
 try\_cast(restaurantid AS number) AS restaurant\_id,  
 try\_cast(name AS string) AS name,  
 try\_cast(cuisinetype AS string) AS cuisine\_type,  
 try\_cast(pricing\_for\_2 AS number(10, 2)) AS pricing\_for\_two,  
 try\_cast(restaurant\_phone AS string) AS restaurant\_phone,  
 try\_cast(operatinghours AS string) AS operating\_hours,  
 try\_cast(locationid AS number) AS location\_id\_fk,  
 try\_cast(activeflag AS string) AS active\_flag,  
 try\_cast(openstatus AS string) AS open\_status,  
 try\_cast(locality AS string) AS locality,  
 try\_cast(restaurant\_address AS string) AS restaurant\_address,  
 try\_cast(latitude AS number(9, 6)) AS latitude,  
 try\_cast(longitude AS number(9, 6)) AS longitude,  
 try\_to\_timestamp\_ntz(createddate, 'YYYY-MM-DD HH24:MI:SS.FF9') AS created\_dt,  
 try\_to\_timestamp\_ntz(modifieddate, 'YYYY-MM-DD HH24:MI:SS.FF9') AS modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5  
 FROM   
 stage\_sch.restaurant\_stm  
) AS source  
ON target.restaurant\_id = source.restaurant\_id  
WHEN MATCHED THEN   
 UPDATE SET   
 target.name = source.name,  
 target.cuisine\_type = source.cuisine\_type,  
 target.pricing\_for\_two = source.pricing\_for\_two,  
 target.restaurant\_phone = source.restaurant\_phone,  
 target.operating\_hours = source.operating\_hours,  
 target.location\_id\_fk = source.location\_id\_fk,  
 target.active\_flag = source.active\_flag,  
 target.open\_status = source.open\_status,  
 target.locality = source.locality,  
 target.restaurant\_address = source.restaurant\_address,  
 target.latitude = source.latitude,  
 target.longitude = source.longitude,  
 target.created\_dt = source.created\_dt,  
 target.modified\_dt = source.modified\_dt,  
 target.\_stg\_file\_name = source.\_stg\_file\_name,  
 target.\_stg\_file\_load\_ts = source.\_stg\_file\_load\_ts,  
 target.\_stg\_file\_md5 = source.\_stg\_file\_md5  
WHEN NOT MATCHED THEN   
 INSERT (  
 restaurant\_id,  
 name,  
 cuisine\_type,  
 pricing\_for\_two,  
 restaurant\_phone,  
 operating\_hours,  
 location\_id\_fk,  
 active\_flag,  
 open\_status,  
 locality,  
 restaurant\_address,  
 latitude,  
 longitude,  
 created\_dt,  
 modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5  
 )  
 VALUES (  
 source.restaurant\_id,  
 source.name,  
 source.cuisine\_type,  
 source.pricing\_for\_two,  
 source.restaurant\_phone,  
 source.operating\_hours,  
 source.location\_id\_fk,  
 source.active\_flag,  
 source.open\_status,  
 source.locality,  
 source.restaurant\_address,  
 source.latitude,  
 source.longitude,  
 source.created\_dt,  
 source.modified\_dt,  
 source.\_stg\_file\_name,  
 source.\_stg\_file\_load\_ts,  
 source.\_stg\_file\_md5  
 );  
  
-- now define dim table for restaurant.  
CREATE OR REPLACE TABLE CONSUMPTION\_SCH.RESTAURANT\_DIM (  
 RESTAURANT\_HK NUMBER primary key, -- Hash key for the restaurant location  
 RESTAURANT\_ID NUMBER, -- Restaurant ID without auto-increment  
 NAME STRING(100), -- Restaurant name  
 CUISINE\_TYPE STRING, -- Type of cuisine offered  
 PRICING\_FOR\_TWO NUMBER(10, 2), -- Pricing for two people  
 RESTAURANT\_PHONE STRING(15) WITH TAG (common.pii\_policy\_tag = 'SENSITIVE'), -- Restaurant phone number  
 OPERATING\_HOURS STRING(100), -- Restaurant operating hours  
 LOCATION\_ID\_FK NUMBER, -- Foreign key reference to location  
 ACTIVE\_FLAG STRING(10), -- Indicates if the restaurant is active  
 OPEN\_STATUS STRING(10), -- Indicates if the restaurant is currently open  
 LOCALITY STRING(100), -- Locality of the restaurant  
 RESTAURANT\_ADDRESS STRING, -- Full address of the restaurant  
 LATITUDE NUMBER(9, 6), -- Latitude for the restaurant's location  
 LONGITUDE NUMBER(9, 6), -- Longitude for the restaurant's location  
 EFF\_START\_DATE TIMESTAMP\_TZ, -- Effective start date for the record  
 EFF\_END\_DATE TIMESTAMP\_TZ, -- Effective end date for the record (NULL if active)  
 IS\_CURRENT BOOLEAN -- Indicates whether the record is the current version  
)  
COMMENT = 'Dimensional table for Restaurant entity with hash keys and SCD enabled.';  
  
-- how many changes are available.  
select count(\*) from CLEAN\_SCH.RESTAURANT\_STM;  
  
-- merge statement  
MERGE INTO   
 CONSUMPTION\_SCH.RESTAURANT\_DIM AS target  
USING   
 CLEAN\_SCH.RESTAURANT\_STM AS source  
ON   
 target.RESTAURANT\_ID = source.RESTAURANT\_ID AND   
 target.NAME = source.NAME AND   
 target.CUISINE\_TYPE = source.CUISINE\_TYPE AND   
 target.PRICING\_FOR\_TWO = source.PRICING\_FOR\_TWO AND   
 target.RESTAURANT\_PHONE = source.RESTAURANT\_PHONE AND   
 target.OPERATING\_HOURS = source.OPERATING\_HOURS AND   
 target.LOCATION\_ID\_FK = source.LOCATION\_ID\_FK AND   
 target.ACTIVE\_FLAG = source.ACTIVE\_FLAG AND   
 target.OPEN\_STATUS = source.OPEN\_STATUS AND   
 target.LOCALITY = source.LOCALITY AND   
 target.RESTAURANT\_ADDRESS = source.RESTAURANT\_ADDRESS AND   
 target.LATITUDE = source.LATITUDE AND   
 target.LONGITUDE = source.LONGITUDE  
WHEN MATCHED   
 AND source.METADATA$ACTION = 'DELETE' AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Update the existing record to close its validity period  
 UPDATE SET   
 target.EFF\_END\_DATE = CURRENT\_TIMESTAMP(),  
 target.IS\_CURRENT = FALSE  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 RESTAURANT\_HK,  
 RESTAURANT\_ID,  
 NAME,  
 CUISINE\_TYPE,  
 PRICING\_FOR\_TWO,  
 RESTAURANT\_PHONE,  
 OPERATING\_HOURS,  
 LOCATION\_ID\_FK,  
 ACTIVE\_FLAG,  
 OPEN\_STATUS,  
 LOCALITY,  
 RESTAURANT\_ADDRESS,  
 LATITUDE,  
 LONGITUDE,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.RESTAURANT\_ID, source.NAME, source.CUISINE\_TYPE,   
 source.PRICING\_FOR\_TWO, source.RESTAURANT\_PHONE, source.OPERATING\_HOURS,   
 source.LOCATION\_ID\_FK, source.ACTIVE\_FLAG, source.OPEN\_STATUS, source.LOCALITY,   
 source.RESTAURANT\_ADDRESS, source.LATITUDE, source.LONGITUDE))),  
 source.RESTAURANT\_ID,  
 source.NAME,  
 source.CUISINE\_TYPE,  
 source.PRICING\_FOR\_TWO,  
 source.RESTAURANT\_PHONE,  
 source.OPERATING\_HOURS,  
 source.LOCATION\_ID\_FK,  
 source.ACTIVE\_FLAG,  
 source.OPEN\_STATUS,  
 source.LOCALITY,  
 source.RESTAURANT\_ADDRESS,  
 source.LATITUDE,  
 source.LONGITUDE,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 )  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' AND source.METADATA$ISUPDATE = 'FALSE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 RESTAURANT\_HK,  
 RESTAURANT\_ID,  
 NAME,  
 CUISINE\_TYPE,  
 PRICING\_FOR\_TWO,  
 RESTAURANT\_PHONE,  
 OPERATING\_HOURS,  
 LOCATION\_ID\_FK,  
 ACTIVE\_FLAG,  
 OPEN\_STATUS,  
 LOCALITY,  
 RESTAURANT\_ADDRESS,  
 LATITUDE,  
 LONGITUDE,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.RESTAURANT\_ID, source.NAME, source.CUISINE\_TYPE,   
 source.PRICING\_FOR\_TWO, source.RESTAURANT\_PHONE, source.OPERATING\_HOURS,   
 source.LOCATION\_ID\_FK, source.ACTIVE\_FLAG, source.OPEN\_STATUS, source.LOCALITY,   
 source.RESTAURANT\_ADDRESS, source.LATITUDE, source.LONGITUDE))),  
 source.RESTAURANT\_ID,  
 source.NAME,  
 source.CUISINE\_TYPE,  
 source.PRICING\_FOR\_TWO,  
 source.RESTAURANT\_PHONE,  
 source.OPERATING\_HOURS,  
 source.LOCATION\_ID\_FK,  
 source.ACTIVE\_FLAG,  
 source.OPEN\_STATUS,  
 source.LOCALITY,  
 source.RESTAURANT\_ADDRESS,  
 source.LATITUDE,  
 source.LONGITUDE,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 );  
  
  
-- load the delta data  
  
list @stage\_sch.csv\_stg/daily/restaurant/;  
  
copy into stage\_sch.restaurant (restaurantid, name, cuisinetype, pricing\_for\_2, restaurant\_phone,   
 operatinghours, locationid, activeflag, openstatus,   
 locality, restaurant\_address, latitude, longitude,   
 createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as restaurantid, -- restaurantid as the first column  
 t.$2::text as name,  
 t.$3::text as cuisinetype,  
 t.$4::text as pricing\_for\_2,  
 t.$5::text as restaurant\_phone,  
 t.$6::text as operatinghours,  
 t.$7::text as locationid,  
 t.$8::text as activeflag,  
 t.$9::text as openstatus,  
 t.$10::text as locality,  
 t.$11::text as restaurant\_address,  
 t.$12::text as latitude,  
 t.$13::text as longitude,  
 t.$14::text as createddate,  
 t.$15::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp() as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/daily/restaurant/day-02-upsert-restaurant-delhi+NCR.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
  
// ------------------------------------------------  
// Part -2 loading the delta data  
  
list @stage\_sch.csv\_stg/delta/restaurant/;  
copy into stage\_sch.restaurant (restaurantid, name, cuisinetype, pricing\_for\_2, restaurant\_phone,   
 operatinghours, locationid, activeflag, openstatus,   
 locality, restaurant\_address, latitude, longitude,   
 createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as restaurantid, -- restaurantid as the first column  
 t.$2::text as name,  
 t.$3::text as cuisinetype,  
 t.$4::text as pricing\_for\_2,  
 t.$5::text as restaurant\_phone,  
 t.$6::text as operatinghours,  
 t.$7::text as locationid,  
 t.$8::text as activeflag,  
 t.$9::text as openstatus,  
 t.$10::text as locality,  
 t.$11::text as restaurant\_address,  
 t.$12::text as latitude,  
 t.$13::text as longitude,  
 t.$14::text as createddate,  
 t.$15::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp() as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/restaurant/day-02-upsert-restaurant-delhi+NCR.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
select \*  
from table(information\_schema.copy\_history(table\_name=>'RESTAURANT', start\_time=> dateadd(hours, -1, current\_timestamp())));

**Customer Dimension — SQL Script**

Press enter or click to view image in full size

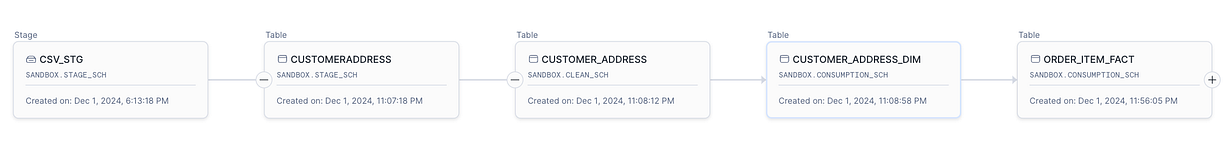


**Customer Dimension Data Lineage**

use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
-- create restaurant table under stage, with all text value + audit column for copy command  
create or replace table stage\_sch.customer (  
 customerid text, -- primary key as text  
 name text, -- name as text  
 mobile text WITH TAG (common.pii\_policy\_tag = 'PII'), -- mobile number as text  
 email text WITH TAG (common.pii\_policy\_tag = 'EMAIL'), -- email as text  
 loginbyusing text, -- login method as text  
 gender text WITH TAG (common.pii\_policy\_tag = 'PII'), -- gender as text  
 dob text WITH TAG (common.pii\_policy\_tag = 'PII'), -- date of birth as text  
 anniversary text, -- anniversary as text  
 preferences text, -- preferences as text  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the customer stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
-- Stream object to capture the changes.   
create or replace stream stage\_sch.customer\_stm   
on table stage\_sch.customer  
append\_only = true  
comment = 'This is the append-only stream object on customer table that only gets delta data';  
  
  
-- run copy command to load the data into stage-customer table.  
copy into stage\_sch.customer (customerid, name, mobile, email, loginbyusing, gender, dob, anniversary,   
 preferences, createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as customerid,  
 t.$2::text as name,  
 t.$3::text as mobile,  
 t.$4::text as email,  
 t.$5::text as loginbyusing,  
 t.$6::text as gender,  
 t.$7::text as dob,  
 t.$8::text as anniversary,  
 t.$9::text as preferences,  
 t.$10::text as createddate,  
 t.$11::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/customer/customers-initial.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
select \* from stage\_sch.customer limit 10;  
select count(\*) from stage\_sch.customer; -- 99899  
select count(\*) from stage\_sch.customer\_stm;   
  
  
-- Part-2 Clean Layer  
--   
  
CREATE OR REPLACE TABLE CLEAN\_SCH.CUSTOMER (  
   
 CUSTOMER\_SK NUMBER AUTOINCREMENT PRIMARY KEY, -- Auto-incremented primary key  
 CUSTOMER\_ID STRING NOT NULL, -- Customer ID  
 NAME STRING(100) NOT NULL, -- Customer name  
 MOBILE STRING(15) WITH TAG (common.pii\_policy\_tag = 'PII'), -- Mobile number, accommodating international format  
 EMAIL STRING(100) WITH TAG (common.pii\_policy\_tag = 'EMAIL'), -- Email  
 LOGIN\_BY\_USING STRING(50), -- Method of login (e.g., Social, Google, etc.)  
 GENDER STRING(10) WITH TAG (common.pii\_policy\_tag = 'PII'), -- Gender  
 DOB DATE WITH TAG (common.pii\_policy\_tag = 'PII'), -- Date of birth in DATE format  
 ANNIVERSARY DATE, -- Anniversary in DATE format  
 PREFERENCES STRING, -- Customer preferences  
 CREATED\_DT TIMESTAMP\_TZ DEFAULT CURRENT\_TIMESTAMP, -- Record creation timestamp  
 MODIFIED\_DT TIMESTAMP\_TZ, -- Record modification timestamp, allows NULL if not modified  
  
 -- Additional audit columns  
 \_STG\_FILE\_NAME STRING, -- File name for audit  
 \_STG\_FILE\_LOAD\_TS TIMESTAMP\_NTZ, -- File load timestamp  
 \_STG\_FILE\_MD5 STRING, -- MD5 hash for file content  
 \_COPY\_DATA\_TS TIMESTAMP\_NTZ DEFAULT CURRENT\_TIMESTAMP -- Copy data timestamp  
)  
comment = 'Customer entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
-- Stream object to capture the changes.   
create or replace stream CLEAN\_SCH.customer\_stm   
on table CLEAN\_SCH.customer  
comment = 'This is the stream object on customer entity to track insert, update, and delete changes';  
  
  
insert into clean\_sch.customer (  
 customer\_id,  
 name,  
 mobile,  
 email,  
 login\_by\_using,  
 gender,  
 dob,  
 anniversary,  
 preferences,  
 created\_dt,  
 modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
)  
select   
 customerid::string,  
 name::string,  
 mobile::string,  
 email::string,  
 loginbyusing::string,  
 gender::string,  
 try\_to\_date(dob, 'YYYY-MM-DD') as dob, -- converting dob to date  
 try\_to\_date(anniversary, 'YYYY-MM-DD') as anniversary, -- converting anniversary to date  
 preferences::string,  
 try\_to\_timestamp\_tz(createddate, 'YYYY-MM-DD HH24:MI:SS') as created\_dt, -- timestamp conversion  
 try\_to\_timestamp\_tz(modifieddate, 'YYYY-MM-DD HH24:MI:SS') as modified\_dt, -- timestamp conversion  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
from stage\_sch.customer;  
  
  
MERGE INTO CLEAN\_SCH.CUSTOMER AS target  
USING (  
 SELECT   
 CUSTOMERID::STRING AS CUSTOMER\_ID,  
 NAME::STRING AS NAME,  
 MOBILE::STRING AS MOBILE,  
 EMAIL::STRING AS EMAIL,  
 LOGINBYUSING::STRING AS LOGIN\_BY\_USING,  
 GENDER::STRING AS GENDER,  
 TRY\_TO\_DATE(DOB, 'YYYY-MM-DD') AS DOB,   
 TRY\_TO\_DATE(ANNIVERSARY, 'YYYY-MM-DD') AS ANNIVERSARY,   
 PREFERENCES::STRING AS PREFERENCES,  
 TRY\_TO\_TIMESTAMP\_TZ(CREATEDDATE, 'YYYY-MM-DD"T"HH24:MI:SS.FF6') AS CREATED\_DT,   
 TRY\_TO\_TIMESTAMP\_TZ(MODIFIEDDATE, 'YYYY-MM-DD"T"HH24:MI:SS.FF6') AS MODIFIED\_DT,   
 \_STG\_FILE\_NAME,  
 \_STG\_FILE\_LOAD\_TS,  
 \_STG\_FILE\_MD5,  
 \_COPY\_DATA\_TS  
 FROM STAGE\_SCH.CUSTOMER\_STM  
) AS source  
ON target.CUSTOMER\_ID = source.CUSTOMER\_ID  
WHEN MATCHED THEN  
 UPDATE SET   
 target.NAME = source.NAME,  
 target.MOBILE = source.MOBILE,  
 target.EMAIL = source.EMAIL,  
 target.LOGIN\_BY\_USING = source.LOGIN\_BY\_USING,  
 target.GENDER = source.GENDER,  
 target.DOB = source.DOB,  
 target.ANNIVERSARY = source.ANNIVERSARY,  
 target.PREFERENCES = source.PREFERENCES,  
 target.CREATED\_DT = source.CREATED\_DT,  
 target.MODIFIED\_DT = source.MODIFIED\_DT,  
 target.\_STG\_FILE\_NAME = source.\_STG\_FILE\_NAME,  
 target.\_STG\_FILE\_LOAD\_TS = source.\_STG\_FILE\_LOAD\_TS,  
 target.\_STG\_FILE\_MD5 = source.\_STG\_FILE\_MD5,  
 target.\_COPY\_DATA\_TS = source.\_COPY\_DATA\_TS  
WHEN NOT MATCHED THEN  
 INSERT (  
 CUSTOMER\_ID,  
 NAME,  
 MOBILE,  
 EMAIL,  
 LOGIN\_BY\_USING,  
 GENDER,  
 DOB,  
 ANNIVERSARY,  
 PREFERENCES,  
 CREATED\_DT,  
 MODIFIED\_DT,  
 \_STG\_FILE\_NAME,  
 \_STG\_FILE\_LOAD\_TS,  
 \_STG\_FILE\_MD5,  
 \_COPY\_DATA\_TS  
 )  
 VALUES (  
 source.CUSTOMER\_ID,  
 source.NAME,  
 source.MOBILE,  
 source.EMAIL,  
 source.LOGIN\_BY\_USING,  
 source.GENDER,  
 source.DOB,  
 source.ANNIVERSARY,  
 source.PREFERENCES,  
 source.CREATED\_DT,  
 source.MODIFIED\_DT,  
 source.\_STG\_FILE\_NAME,  
 source.\_STG\_FILE\_LOAD\_TS,  
 source.\_STG\_FILE\_MD5,  
 source.\_COPY\_DATA\_TS  
 );  
  
-- create dim table   
CREATE OR REPLACE TABLE CONSUMPTION\_SCH.CUSTOMER\_DIM (  
 CUSTOMER\_HK NUMBER PRIMARY KEY, -- Surrogate key for the customer  
 CUSTOMER\_ID STRING NOT NULL, -- Natural key for the customer  
 NAME STRING(100) NOT NULL, -- Customer name  
 MOBILE STRING(15) WITH TAG (common.pii\_policy\_tag = 'PII'), -- Mobile number  
 EMAIL STRING(100) WITH TAG (common.pii\_policy\_tag = 'EMAIL'), -- Email  
 LOGIN\_BY\_USING STRING(50), -- Method of login  
 GENDER STRING(10) WITH TAG (common.pii\_policy\_tag = 'PII'), -- Gender  
 DOB DATE WITH TAG (common.pii\_policy\_tag = 'PII'), -- Date of birth  
 ANNIVERSARY DATE, -- Anniversary  
 PREFERENCES STRING, -- Preferences  
 EFF\_START\_DATE TIMESTAMP\_TZ, -- Effective start date  
 EFF\_END\_DATE TIMESTAMP\_TZ, -- Effective end date (NULL if active)  
 IS\_CURRENT BOOLEAN -- Flag to indicate the current record  
)  
COMMENT = 'Customer Dimension table with SCD Type 2 handling for historical tracking.';  
  
  
  
MERGE INTO   
 CONSUMPTION\_SCH.CUSTOMER\_DIM AS target  
USING   
 CLEAN\_SCH.CUSTOMER\_STM AS source  
ON   
 target.CUSTOMER\_ID = source.CUSTOMER\_ID AND  
 target.NAME = source.NAME AND  
 target.MOBILE = source.MOBILE AND  
 target.EMAIL = source.EMAIL AND  
 target.LOGIN\_BY\_USING = source.LOGIN\_BY\_USING AND  
 target.GENDER = source.GENDER AND  
 target.DOB = source.DOB AND  
 target.ANNIVERSARY = source.ANNIVERSARY AND  
 target.PREFERENCES = source.PREFERENCES  
WHEN MATCHED   
 AND source.METADATA$ACTION = 'DELETE' AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Update the existing record to close its validity period  
 UPDATE SET   
 target.EFF\_END\_DATE = CURRENT\_TIMESTAMP(),  
 target.IS\_CURRENT = FALSE  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 CUSTOMER\_HK,  
 CUSTOMER\_ID,  
 NAME,  
 MOBILE,  
 EMAIL,  
 LOGIN\_BY\_USING,  
 GENDER,  
 DOB,  
 ANNIVERSARY,  
 PREFERENCES,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.CUSTOMER\_ID, source.NAME, source.MOBILE,   
 source.EMAIL, source.LOGIN\_BY\_USING, source.GENDER, source.DOB,   
 source.ANNIVERSARY, source.PREFERENCES))),  
 source.CUSTOMER\_ID,  
 source.NAME,  
 source.MOBILE,  
 source.EMAIL,  
 source.LOGIN\_BY\_USING,  
 source.GENDER,  
 source.DOB,  
 source.ANNIVERSARY,  
 source.PREFERENCES,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 )  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' AND source.METADATA$ISUPDATE = 'FALSE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 CUSTOMER\_HK,  
 CUSTOMER\_ID,  
 NAME,  
 MOBILE,  
 EMAIL,  
 LOGIN\_BY\_USING,  
 GENDER,  
 DOB,  
 ANNIVERSARY,  
 PREFERENCES,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.CUSTOMER\_ID, source.NAME, source.MOBILE,   
 source.EMAIL, source.LOGIN\_BY\_USING, source.GENDER, source.DOB,   
 source.ANNIVERSARY, source.PREFERENCES))),  
 source.CUSTOMER\_ID,  
 source.NAME,  
 source.MOBILE,  
 source.EMAIL,  
 source.LOGIN\_BY\_USING,  
 source.GENDER,  
 source.DOB,  
 source.ANNIVERSARY,  
 source.PREFERENCES,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 );  
  
// ----------------------------------------------------------  
// ----------------------------------------------------------  
-- delta processing check  
  
list @stage\_sch.csv\_stg/delta/customer/;  
  
copy into stage\_sch.customer (customerid, name, mobile, email, loginbyusing, gender, dob, anniversary,   
 preferences, createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as customerid,  
 t.$2::text as name,  
 t.$3::text as mobile,  
 t.$4::text as email,  
 t.$5::text as loginbyusing,  
 t.$6::text as gender,  
 t.$7::text as dob,  
 t.$8::text as anniversary,  
 t.$9::text as preferences,  
 t.$10::text as createddate,  
 t.$11::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/customer/day-01-insert-customer.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
  
-- ------------------------------------------------  
-- Part -2 loading the delta data  
  
list @stage\_sch.csv\_stg/delta/customer/;  
  
copy into stage\_sch.customer (customerid, name, mobile, email, loginbyusing, gender, dob, anniversary,   
 preferences, createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as customerid,  
 t.$2::text as name,  
 t.$3::text as mobile,  
 t.$4::text as email,  
 t.$5::text as loginbyusing,  
 t.$6::text as gender,  
 t.$7::text as dob,  
 t.$8::text as anniversary,  
 t.$9::text as preferences,  
 t.$10::text as createddate,  
 t.$11::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/customer/day-02-insert-update.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Customer Address Dimension — SQL Scripts**

Press enter or click to view image in full size

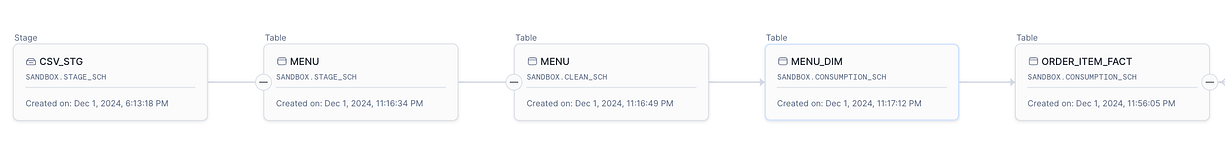


**Customer Address Dimension Data Lineage**

use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
  
create or replace table stage\_sch.customeraddress (  
 addressid text, -- primary key as text  
 customerid text comment 'Customer FK (Source Data)', -- foreign key reference as text (no constraint in snowflake)  
 flatno text, -- flat number as text  
 houseno text, -- house number as text  
 floor text, -- floor as text  
 building text, -- building name as text  
 landmark text, -- landmark as text  
 locality text, -- locality as text  
 city text, -- city as text  
 state text, -- state as text  
 pincode text, -- pincode as text  
 coordinates text, -- coordinates as text  
 primaryflag text, -- primary flag as text  
 addresstype text, -- address type as text  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the customer address stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
create or replace stream stage\_sch.customeraddress\_stm   
on table stage\_sch.customeraddress  
append\_only = true  
comment = 'This is the append-only stream object on customer address table that only gets delta data';  
  
  
select \* from stage\_sch.customeraddress\_stm;  
copy into stage\_sch.customeraddress (addressid, customerid, flatno, houseno, floor, building,   
 landmark, locality,city,pincode, state, coordinates, primaryflag, addresstype,   
 createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as addressid,  
 t.$2::text as customerid,  
 t.$3::text as flatno,  
 t.$4::text as houseno,  
 t.$5::text as floor,  
 t.$6::text as building,  
 t.$7::text as landmark,  
 t.$8::text as locality,  
 t.$9::text as city,  
 t.$10::text as State,  
 t.$11::text as Pincode,  
 t.$12::text as coordinates,  
 t.$13::text as primaryflag,  
 t.$14::text as addresstype,  
 t.$15::text as createddate,  
 t.$16::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/customer-address t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
  
-- 2nd layer  
CREATE OR REPLACE TABLE CLEAN\_SCH.CUSTOMER\_ADDRESS (  
 CUSTOMER\_ADDRESS\_SK NUMBER AUTOINCREMENT PRIMARY KEY comment 'Surrogate Key (EWH)', -- Auto-incremented primary key  
 ADDRESS\_ID INT comment 'Primary Key (Source Data)', -- Primary key as string  
 CUSTOMER\_ID\_FK INT comment 'Customer FK (Source Data)', -- Foreign key reference as string (no constraint in Snowflake)  
 FLAT\_NO STRING, -- Flat number as string  
 HOUSE\_NO STRING, -- House number as string  
 FLOOR STRING, -- Floor as string  
 BUILDING STRING, -- Building name as string  
 LANDMARK STRING, -- Landmark as string  
 locality STRING, -- locality as string  
 CITY STRING, -- City as string  
 STATE STRING, -- State as string  
 PINCODE STRING, -- Pincode as string  
 COORDINATES STRING, -- Coordinates as string  
 PRIMARY\_FLAG STRING, -- Primary flag as string  
 ADDRESS\_TYPE STRING, -- Address type as string  
 CREATED\_DATE TIMESTAMP\_TZ, -- Created date as timestamp with time zone  
 MODIFIED\_DATE TIMESTAMP\_TZ, -- Modified date as timestamp with time zone  
  
 -- Audit columns with appropriate data types  
 \_STG\_FILE\_NAME STRING,  
 \_STG\_FILE\_LOAD\_TS TIMESTAMP,  
 \_STG\_FILE\_MD5 STRING,  
 \_COPY\_DATA\_TS TIMESTAMP DEFAULT CURRENT\_TIMESTAMP  
)  
comment = 'Customer address entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
  
-- Stream object to capture the changes.   
create or replace stream CLEAN\_SCH.CUSTOMER\_ADDRESS\_STM  
on table CLEAN\_SCH.CUSTOMER\_ADDRESS  
comment = 'This is the stream object on customer address entity to track insert, update, and delete changes';  
  
  
MERGE INTO clean\_sch.customer\_address AS clean  
USING (  
 SELECT   
 CAST(addressid AS INT) AS address\_id,  
 CAST(customerid AS INT) AS customer\_id\_fk,  
 flatno AS flat\_no,  
 houseno AS house\_no,  
 floor,  
 building,  
 landmark,  
 locality,  
 city,  
 state,  
 pincode,  
 coordinates,  
 primaryflag AS primary\_flag,  
 addresstype AS address\_type,  
 TRY\_TO\_TIMESTAMP\_TZ(createddate, 'YYYY-MM-DD"T"HH24:MI:SS') AS created\_date,  
 TRY\_TO\_TIMESTAMP\_TZ(modifieddate, 'YYYY-MM-DD"T"HH24:MI:SS') AS modified\_date,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 FROM stage\_sch.customeraddress\_stm   
) AS stage  
ON clean.address\_id = stage.address\_id  
-- Insert new records  
WHEN NOT MATCHED THEN  
 INSERT (  
 address\_id,  
 customer\_id\_fk,  
 flat\_no,  
 house\_no,  
 floor,  
 building,  
 landmark,  
 locality,  
 city,  
 state,  
 pincode,  
 coordinates,  
 primary\_flag,  
 address\_type,  
 created\_date,  
 modified\_date,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 )  
 VALUES (  
 stage.address\_id,  
 stage.customer\_id\_fk,  
 stage.flat\_no,  
 stage.house\_no,  
 stage.floor,  
 stage.building,  
 stage.landmark,  
 stage.locality,  
 stage.city,  
 stage.state,  
 stage.pincode,  
 stage.coordinates,  
 stage.primary\_flag,  
 stage.address\_type,  
 stage.created\_date,  
 stage.modified\_date,  
 stage.\_stg\_file\_name,  
 stage.\_stg\_file\_load\_ts,  
 stage.\_stg\_file\_md5,  
 stage.\_copy\_data\_ts  
 )  
-- Update existing records  
WHEN MATCHED THEN  
 UPDATE SET  
 clean.flat\_no = stage.flat\_no,  
 clean.house\_no = stage.house\_no,  
 clean.floor = stage.floor,  
 clean.building = stage.building,  
 clean.landmark = stage.landmark,  
 clean.locality = stage.locality,  
 clean.city = stage.city,  
 clean.state = stage.state,  
 clean.pincode = stage.pincode,  
 clean.coordinates = stage.coordinates,  
 clean.primary\_flag = stage.primary\_flag,  
 clean.address\_type = stage.address\_type,  
 clean.created\_date = stage.created\_date,  
 clean.modified\_date = stage.modified\_date,  
 clean.\_stg\_file\_name = stage.\_stg\_file\_name,  
 clean.\_stg\_file\_load\_ts = stage.\_stg\_file\_load\_ts,  
 clean.\_stg\_file\_md5 = stage.\_stg\_file\_md5,  
 clean.\_copy\_data\_ts = stage.\_copy\_data\_ts;  
  
  
  
CREATE OR REPLACE TABLE CONSUMPTION\_SCH.CUSTOMER\_ADDRESS\_DIM (  
 CUSTOMER\_ADDRESS\_HK NUMBER PRIMARY KEY comment 'Customer Address HK (EDW)', -- Surrogate key (hash key)  
 ADDRESS\_ID INT comment 'Primary Key (Source System)', -- Original primary key  
 CUSTOMER\_ID\_FK STRING comment 'Customer FK (Source System)', -- Surrogate key from Customer Dimension (Foreign Key)  
 FLAT\_NO STRING, -- Flat number  
 HOUSE\_NO STRING, -- House number  
 FLOOR STRING, -- Floor  
 BUILDING STRING, -- Building name  
 LANDMARK STRING, -- Landmark  
 LOCALITY STRING, -- Locality  
 CITY STRING, -- City  
 STATE STRING, -- State  
 PINCODE STRING, -- Pincode  
 COORDINATES STRING, -- Geo-coordinates  
 PRIMARY\_FLAG STRING, -- Whether it's the primary address  
 ADDRESS\_TYPE STRING, -- Type of address (e.g., Home, Office)  
  
 -- SCD2 Columns  
 EFF\_START\_DATE TIMESTAMP\_TZ, -- Effective start date  
 EFF\_END\_DATE TIMESTAMP\_TZ, -- Effective end date (NULL if active)  
 IS\_CURRENT BOOLEAN -- Flag to indicate the current record  
);  
  
-- select \* from CLEAN\_SCH.CUSTOMER\_ADDRESS\_STM;  
MERGE INTO   
 CONSUMPTION\_SCH.CUSTOMER\_ADDRESS\_DIM AS target  
USING   
 CLEAN\_SCH.CUSTOMER\_ADDRESS\_STM AS source  
ON   
 target.ADDRESS\_ID = source.ADDRESS\_ID AND  
 target.CUSTOMER\_ID\_FK = source.CUSTOMER\_ID\_FK AND  
 target.FLAT\_NO = source.FLAT\_NO AND  
 target.HOUSE\_NO = source.HOUSE\_NO AND  
 target.FLOOR = source.FLOOR AND  
 target.BUILDING = source.BUILDING AND  
 target.LANDMARK = source.LANDMARK AND  
 target.LOCALITY = source.LOCALITY AND  
 target.CITY = source.CITY AND  
 target.STATE = source.STATE AND  
 target.PINCODE = source.PINCODE AND  
 target.COORDINATES = source.COORDINATES AND  
 target.PRIMARY\_FLAG = source.PRIMARY\_FLAG AND  
 target.ADDRESS\_TYPE = source.ADDRESS\_TYPE  
WHEN MATCHED   
 AND source.METADATA$ACTION = 'DELETE' AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Update the existing record to close its validity period  
 UPDATE SET   
 target.EFF\_END\_DATE = CURRENT\_TIMESTAMP(),  
 target.IS\_CURRENT = FALSE  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 CUSTOMER\_ADDRESS\_HK,  
 ADDRESS\_ID,  
 CUSTOMER\_ID\_FK,  
 FLAT\_NO,  
 HOUSE\_NO,  
 FLOOR,  
 BUILDING,  
 LANDMARK,  
 LOCALITY,  
 CITY,  
 STATE,  
 PINCODE,  
 COORDINATES,  
 PRIMARY\_FLAG,  
 ADDRESS\_TYPE,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.ADDRESS\_ID, source.CUSTOMER\_ID\_FK, source.FLAT\_NO,   
 source.HOUSE\_NO, source.FLOOR, source.BUILDING, source.LANDMARK,   
 source.LOCALITY, source.CITY, source.STATE, source.PINCODE,   
 source.COORDINATES, source.PRIMARY\_FLAG, source.ADDRESS\_TYPE))),  
 source.ADDRESS\_ID,  
 source.CUSTOMER\_ID\_FK,  
 source.FLAT\_NO,  
 source.HOUSE\_NO,  
 source.FLOOR,  
 source.BUILDING,  
 source.LANDMARK,  
 source.LOCALITY,  
 source.CITY,  
 source.STATE,  
 source.PINCODE,  
 source.COORDINATES,  
 source.PRIMARY\_FLAG,  
 source.ADDRESS\_TYPE,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 )  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT' AND source.METADATA$ISUPDATE = 'FALSE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 CUSTOMER\_ADDRESS\_HK,  
 ADDRESS\_ID,  
 CUSTOMER\_ID\_FK,  
 FLAT\_NO,  
 HOUSE\_NO,  
 FLOOR,  
 BUILDING,  
 LANDMARK,  
 LOCALITY,  
 CITY,  
 STATE,  
 PINCODE,  
 COORDINATES,  
 PRIMARY\_FLAG,  
 ADDRESS\_TYPE,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.ADDRESS\_ID, source.CUSTOMER\_ID\_FK, source.FLAT\_NO,   
 source.HOUSE\_NO, source.FLOOR, source.BUILDING, source.LANDMARK,   
 source.LOCALITY, source.CITY, source.STATE, source.PINCODE,   
 source.COORDINATES, source.PRIMARY\_FLAG, source.ADDRESS\_TYPE))),  
 source.ADDRESS\_ID,  
 source.CUSTOMER\_ID\_FK,  
 source.FLAT\_NO,  
 source.HOUSE\_NO,  
 source.FLOOR,  
 source.BUILDING,  
 source.LANDMARK,  
 source.LOCALITY,  
 source.CITY,  
 source.STATE,  
 source.PINCODE,  
 source.COORDINATES,  
 source.PRIMARY\_FLAG,  
 source.ADDRESS\_TYPE,  
 CURRENT\_TIMESTAMP(),  
 NULL,  
 TRUE  
 );  
  
--  
select \* from stage\_sch.customeraddressbook;  
select \* from CLEAN\_SCH.CUSTOMER\_ADDRESS\_BOOK;  
select \* from CONSUMPTION\_SCH.CUSTOMER\_ADDRESS\_BOOK\_DIM;  
  
  
list @stage\_sch.csv\_stg/delta/customer-address;  
copy into stage\_sch.customeraddress (addressid, customerid, flatno, houseno, floor, building,   
 landmark, locality,city,pincode, state, coordinates, primaryflag, addresstype,   
 createddate, modifieddate,   
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as addressid,  
 t.$2::text as customerid,  
 t.$3::text as flatno,  
 t.$4::text as houseno,  
 t.$5::text as floor,  
 t.$6::text as building,  
 t.$7::text as landmark,  
 t.$8::text as locality,  
 t.$9::text as city,  
 t.$10::text as State,  
 t.$11::text as Pincode,  
 t.$12::text as coordinates,  
 t.$13::text as primaryflag,  
 t.$14::text as addresstype,  
 t.$15::text as createddate,  
 t.$16::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/customer-address/ t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Menu Dimension — SQL Script**

Press enter or click to view image in full size

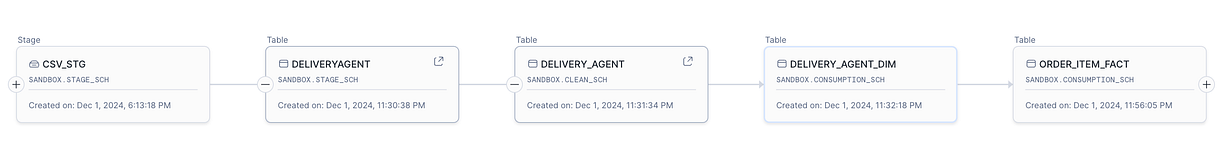


**Menu Dimension Data Lineage**

use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
  
create or replace table stage\_sch.menu (  
 menuid text comment 'Primary Key (Source System)', -- primary key as text  
 restaurantid text comment 'Restaurant FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 itemname text, -- item name as text  
 description text, -- description as text  
 price text, -- price as text (no decimal constraint)  
 category text, -- category as text  
 availability text, -- availability as text  
 itemtype text, -- item type as text  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the menu stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
-- Stream object to capture the changes.   
create or replace stream stage\_sch.menu\_stm   
on table stage\_sch.menu  
append\_only = true  
comment = 'This is the append-only stream object on menu entity that only gets delta data';  
  
  
list @stage\_sch.csv\_stg/initial/menu;  
  
copy into stage\_sch.menu (menuid, restaurantid, itemname, description, price, category,   
 availability, itemtype, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as menuid,  
 t.$2::text as restaurantid,  
 t.$3::text as itemname,  
 t.$4::text as description,  
 t.$5::text as price,  
 t.$6::text as category,  
 t.$7::text as availability,  
 t.$8::text as itemtype,  
 t.$9::text as createddate,  
 t.$10::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/menu t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
select \* from menu limit 10;  
  
select \*  
from table(information\_schema.copy\_history(table\_name=>'MENU', start\_time=> dateadd(hours, -1, current\_timestamp())));  
  
  
CREATE OR REPLACE TABLE clean\_sch.menu (  
 Menu\_SK INT AUTOINCREMENT PRIMARY KEY comment 'Surrogate Key (EDW)', -- Auto-incrementing primary key for internal tracking  
 Menu\_ID INT NOT NULL UNIQUE comment 'Primary Key (Source System)' , -- Unique and non-null Menu\_ID  
 Restaurant\_ID\_FK INT comment 'Restaurant FK(Source System)' , -- Identifier for the restaurant  
 Item\_Name STRING not null, -- Name of the menu item  
 Description STRING not null, -- Description of the menu item  
 Price DECIMAL(10, 2) not null, -- Price as a numeric value with 2 decimal places  
 Category STRING, -- Food category (e.g., North Indian)  
 Availability BOOLEAN, -- Availability status (True/False)  
 Item\_Type STRING, -- Dietary classification (e.g., Vegan)  
 Created\_dt TIMESTAMP\_NTZ, -- Date when the record was created  
 Modified\_dt TIMESTAMP\_NTZ, -- Date when the record was last modified  
  
 -- Audit columns for traceability  
 \_STG\_FILE\_NAME STRING, -- Source file name  
 \_STG\_FILE\_LOAD\_TS TIMESTAMP\_NTZ, -- Timestamp when data was loaded from the staging layer  
 \_STG\_FILE\_MD5 STRING, -- MD5 hash of the source file  
 \_COPY\_DATA\_TS TIMESTAMP\_NTZ DEFAULT CURRENT\_TIMESTAMP -- Timestamp when data was copied to the clean layer  
)  
comment = 'Menu entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
create or replace stream CLEAN\_SCH.menu\_stm   
on table CLEAN\_SCH.menu  
comment = 'This is the stream object on menu table table to track insert, update, and delete changes';  
  
  
MERGE INTO clean\_sch.menu AS target  
USING (  
 SELECT   
 TRY\_CAST(menuid AS INT) AS Menu\_ID,  
 TRY\_CAST(restaurantid AS INT) AS Restaurant\_ID\_FK,  
 TRIM(itemname) AS Item\_Name,  
 TRIM(description) AS Description,  
 TRY\_CAST(price AS DECIMAL(10, 2)) AS Price,  
 TRIM(category) AS Category,  
 CASE   
 WHEN LOWER(availability) = 'true' THEN TRUE  
 WHEN LOWER(availability) = 'false' THEN FALSE  
 ELSE NULL  
 END AS Availability,  
 TRIM(itemtype) AS Item\_Type,  
 TRY\_CAST(createddate AS TIMESTAMP\_NTZ) AS Created\_dt, -- Renamed column  
 TRY\_CAST(modifieddate AS TIMESTAMP\_NTZ) AS Modified\_dt, -- Renamed column  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 FROM stage\_sch.menu  
) AS source  
ON target.Menu\_ID = source.Menu\_ID  
WHEN MATCHED THEN  
 UPDATE SET  
 Restaurant\_ID\_FK = source.Restaurant\_ID\_FK,  
 Item\_Name = source.Item\_Name,  
 Description = source.Description,  
 Price = source.Price,  
 Category = source.Category,  
 Availability = source.Availability,  
 Item\_Type = source.Item\_Type,  
 Created\_dt = source.Created\_dt,   
 Modified\_dt = source.Modified\_dt,   
 \_STG\_FILE\_NAME = source.\_stg\_file\_name,  
 \_STG\_FILE\_LOAD\_TS = source.\_stg\_file\_load\_ts,  
 \_STG\_FILE\_MD5 = source.\_stg\_file\_md5,  
 \_COPY\_DATA\_TS = CURRENT\_TIMESTAMP  
WHEN NOT MATCHED THEN  
 INSERT (  
 Menu\_ID,  
 Restaurant\_ID\_FK,  
 Item\_Name,  
 Description,  
 Price,  
 Category,  
 Availability,  
 Item\_Type,  
 Created\_dt,   
 Modified\_dt,   
 \_STG\_FILE\_NAME,  
 \_STG\_FILE\_LOAD\_TS,  
 \_STG\_FILE\_MD5,  
 \_COPY\_DATA\_TS  
 )  
 VALUES (  
 source.Menu\_ID,  
 source.Restaurant\_ID\_FK,  
 source.Item\_Name,  
 source.Description,  
 source.Price,  
 source.Category,  
 source.Availability,  
 source.Item\_Type,  
 source.Created\_dt,   
 source.Modified\_dt,   
 source.\_stg\_file\_name,  
 source.\_stg\_file\_load\_ts,  
 source.\_stg\_file\_md5,  
 CURRENT\_TIMESTAMP  
 );  
  
  
CREATE OR REPLACE TABLE consumption\_sch.menu\_dim (  
 Menu\_Dim\_HK NUMBER primary key comment 'Menu Dim HK (EDW)', -- Hash key generated for Menu Dim table  
 Menu\_ID INT NOT NULL comment 'Primary Key (Source System)', -- Unique and non-null Menu\_ID  
 Restaurant\_ID\_FK INT NOT NULL comment 'Restaurant FK (Source System)', -- Identifier for the restaurant  
 Item\_Name STRING, -- Name of the menu item  
 Description STRING, -- Description of the menu item  
 Price DECIMAL(10, 2), -- Price as a numeric value with 2 decimal places  
 Category STRING, -- Food category (e.g., North Indian)  
 Availability BOOLEAN, -- Availability status (True/False)  
 Item\_Type STRING, -- Dietary classification (e.g., Vegan)  
 EFF\_START\_DATE TIMESTAMP\_NTZ, -- Effective start date of the record  
 EFF\_END\_DATE TIMESTAMP\_NTZ, -- Effective end date of the record  
 IS\_CURRENT BOOLEAN -- Flag to indicate if the record is current (True/False)  
)  
COMMENT = 'This table stores the dimension data for the menu items, tracking historical changes using SCD Type 2. Each menu item has an effective start and end date, with a flag indicating if it is the current record or historical. The hash key (Menu\_Dim\_HK) is generated based on Menu\_ID and Restaurant\_ID.';  
  
  
MERGE INTO   
 consumption\_sch.MENU\_DIM AS target  
USING   
 CLEAN\_SCH.MENU\_STM AS source  
ON   
 target.Menu\_ID = source.Menu\_ID AND  
 target.Restaurant\_ID\_FK = source.Restaurant\_ID\_FK AND  
 target.Item\_Name = source.Item\_Name AND  
 target.Description = source.Description AND  
 target.Price = source.Price AND  
 target.Category = source.Category AND  
 target.Availability = source.Availability AND  
 target.Item\_Type = source.Item\_Type  
WHEN MATCHED   
 AND source.METADATA$ACTION = 'DELETE'   
 AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Update the existing record to close its validity period  
 UPDATE SET   
 target.EFF\_END\_DATE = CURRENT\_TIMESTAMP(),  
 target.IS\_CURRENT = FALSE  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT'   
 AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 Menu\_Dim\_HK, -- Hash key  
 Menu\_ID,  
 Restaurant\_ID\_FK,  
 Item\_Name,  
 Description,  
 Price,  
 Category,  
 Availability,  
 Item\_Type,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.Menu\_ID, source.Restaurant\_ID\_FK,   
 source.Item\_Name, source.Description, source.Price,   
 source.Category, source.Availability, source.Item\_Type))), -- Hash key  
 source.Menu\_ID,  
 source.Restaurant\_ID\_FK,  
 source.Item\_Name,  
 source.Description,  
 source.Price,  
 source.Category,  
 source.Availability,  
 source.Item\_Type,  
 CURRENT\_TIMESTAMP(), -- Effective start date  
 NULL, -- Effective end date (NULL for current record)  
 TRUE -- IS\_CURRENT = TRUE for new record  
 )  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT'   
 AND source.METADATA$ISUPDATE = 'FALSE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 Menu\_Dim\_HK, -- Hash key  
 Menu\_ID,  
 Restaurant\_ID\_FK,  
 Item\_Name,  
 Description,  
 Price,  
 Category,  
 Availability,  
 Item\_Type,  
 EFF\_START\_DATE,  
 EFF\_END\_DATE,  
 IS\_CURRENT  
 )  
 VALUES (  
 hash(SHA1\_hex(CONCAT(source.Menu\_ID, source.Restaurant\_ID\_FK,   
 source.Item\_Name, source.Description, source.Price,   
 source.Category, source.Availability, source.Item\_Type))), -- Hash key  
 source.Menu\_ID,  
 source.Restaurant\_ID\_FK,  
 source.Item\_Name,  
 source.Description,  
 source.Price,  
 source.Category,  
 source.Availability,  
 source.Item\_Type,  
 CURRENT\_TIMESTAMP(), -- Effective start date  
 NULL, -- Effective end date (NULL for current record)  
 TRUE -- IS\_CURRENT = TRUE for new record  
 );  
  
  
-- Part -2  
list @stage\_sch.csv\_stg/delta/menu;  
copy into stage\_sch.menu (menuid, restaurantid, itemname, description, price, category,   
 availability, itemtype, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as menuid,  
 t.$2::text as restaurantid,  
 t.$3::text as itemname,  
 t.$4::text as description,  
 t.$5::text as price,  
 t.$6::text as category,  
 t.$7::text as availability,  
 t.$8::text as itemtype,  
 t.$9::text as createddate,  
 t.$10::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/menu/ t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Delivery Agent Dimension — SQL Script**

Press enter or click to view image in full size

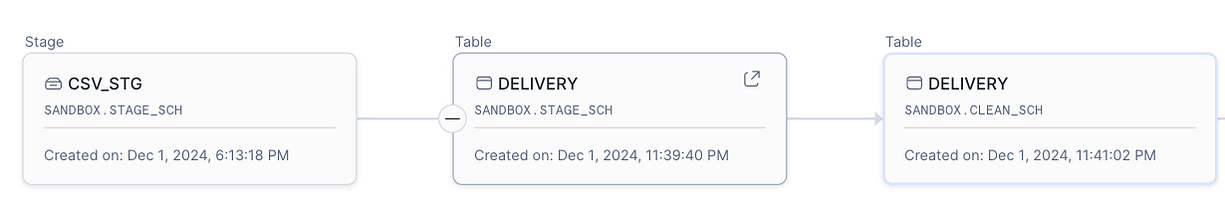


**Delivery Agent Dimension Data Lineage**

use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
create or replace table stage\_sch.deliveryagent (  
 deliveryagentid text comment 'Primary Key (Source System)', -- primary key as text  
 name text, -- name as text, required field  
 phone text, -- phone as text, unique constraint indicated  
 vehicletype text, -- vehicle type as text  
 locationid text, -- foreign key reference as text (no constraint in snowflake)  
 status text, -- status as text  
 gender text, -- status as text  
 rating text, -- rating as text  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the delivery stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
create or replace stream stage\_sch.deliveryagent\_stm   
on table stage\_sch.deliveryagent  
append\_only = true  
comment = 'This is the append-only stream object on delivery agent table that only gets delta data';  
  
  
copy into stage\_sch.deliveryagent (deliveryagentid, name, phone, vehicletype, locationid,   
 status, gender, rating, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as deliveryagentid,  
 t.$2::text as name,  
 t.$3::text as phone,  
 t.$4::text as vehicletype,  
 t.$5::text as locationid,  
 t.$6::text as status,  
 t.$7::text as gender,  
 t.$8::text as rating,  
 t.$9::text as createddate,  
 t.$10::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/delivery-agent t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
select count(\*) from stage\_sch.deliveryagent;  
  
select \* from stage\_sch.deliveryagent\_stm;  
  
  
CREATE OR REPLACE TABLE clean\_sch.delivery\_agent (  
 delivery\_agent\_sk INT AUTOINCREMENT PRIMARY KEY comment 'Surrogate Key (EDW)', -- Primary key with auto-increment  
 delivery\_agent\_id INT NOT NULL UNIQUE comment 'Primary Key (Source System)', -- Delivery agent ID as integer  
 name STRING NOT NULL, -- Name as string, required field  
 phone STRING NOT NULL, -- Phone as string, unique constraint  
 vehicle\_type STRING NOT NULL, -- Vehicle type as string  
 location\_id\_fk INT comment 'Location FK(Source System)', -- Location ID as integer  
 status STRING, -- Status as string  
 gender STRING, -- Gender as string  
 rating number(4,2), -- Rating as float  
 created\_dt TIMESTAMP\_NTZ, -- Created date as timestamp without timezone  
 modified\_dt TIMESTAMP\_NTZ, -- Modified date as timestamp without timezone  
  
 -- Audit columns with appropriate data types  
 \_stg\_file\_name STRING, -- Staging file name as string  
 \_stg\_file\_load\_ts TIMESTAMP, -- Staging file load timestamp  
 \_stg\_file\_md5 STRING, -- Staging file MD5 hash as string  
 \_copy\_data\_ts TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Data copy timestamp with default value  
)  
comment = 'Delivery entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
  
create or replace stream CLEAN\_SCH.delivery\_agent\_stm   
on table CLEAN\_SCH.delivery\_agent  
comment = 'This is the stream object on delivery agent table table to track insert, update, and delete changes';  
  
  
MERGE INTO clean\_sch.delivery\_agent AS target  
USING stage\_sch.deliveryagent\_stm AS source  
ON target.delivery\_agent\_id = source.deliveryagentid  
WHEN MATCHED THEN  
 UPDATE SET  
 target.phone = source.phone,  
 target.vehicle\_type = source.vehicletype,  
 target.location\_id\_fk = TRY\_TO\_NUMBER(source.locationid),  
 target.status = source.status,  
 target.gender = source.gender,  
 target.rating = TRY\_TO\_DECIMAL(source.rating,4,2),  
 target.created\_dt = TRY\_TO\_TIMESTAMP(source.createddate),  
 target.modified\_dt = TRY\_TO\_TIMESTAMP(source.modifieddate),  
 target.\_stg\_file\_name = source.\_stg\_file\_name,  
 target.\_stg\_file\_load\_ts = source.\_stg\_file\_load\_ts,  
 target.\_stg\_file\_md5 = source.\_stg\_file\_md5,  
 target.\_copy\_data\_ts = source.\_copy\_data\_ts  
WHEN NOT MATCHED THEN  
 INSERT (  
 delivery\_agent\_id,  
 name,  
 phone,  
 vehicle\_type,  
 location\_id\_fk,  
 status,  
 gender,  
 rating,  
 created\_dt,  
 modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 )  
 VALUES (  
 TRY\_TO\_NUMBER(source.deliveryagentid),  
 source.name,  
 source.phone,  
 source.vehicletype,  
 TRY\_TO\_NUMBER(source.locationid),  
 source.status,  
 source.gender,  
 TRY\_TO\_NUMBER(source.rating),  
 TRY\_TO\_TIMESTAMP(source.createddate),  
 TRY\_TO\_TIMESTAMP(source.modifieddate),  
 source.\_stg\_file\_name,  
 source.\_stg\_file\_load\_ts,  
 source.\_stg\_file\_md5,  
 CURRENT\_TIMESTAMP()  
 );  
  
select \* from CLEAN\_SCH.delivery\_agent\_stm ;  
  
  
CREATE OR REPLACE TABLE consumption\_sch.delivery\_agent\_dim (  
 delivery\_agent\_hk number primary key comment 'Delivery Agend Dim HK (EDW)', -- Hash key for unique identification  
 delivery\_agent\_id NUMBER not null comment 'Primary Key (Source System)', -- Business key  
 name STRING NOT NULL, -- Delivery agent name  
 phone STRING UNIQUE, -- Phone number, unique  
 vehicle\_type STRING, -- Type of vehicle  
 location\_id\_fk NUMBER NOT NULL comment 'Location FK (Source System)', -- Location ID  
 status STRING, -- Current status of the delivery agent  
 gender STRING, -- Gender  
 rating NUMBER(4,2), -- Rating with one decimal precision  
 eff\_start\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP, -- Effective start date  
 eff\_end\_date TIMESTAMP, -- Effective end date (NULL for active record)  
 is\_current BOOLEAN DEFAULT TRUE  
)  
comment = 'Dim table for delivery agent entity with SCD2 support.';  
  
  
MERGE INTO consumption\_sch.delivery\_agent\_dim AS target  
USING CLEAN\_SCH.delivery\_agent\_stm AS source  
ON   
 target.delivery\_agent\_id = source.delivery\_agent\_id AND  
 target.name = source.name AND  
 target.phone = source.phone AND  
 target.vehicle\_type = source.vehicle\_type AND  
 target.location\_id\_fk = source.location\_id\_fk AND  
 target.status = source.status AND  
 target.gender = source.gender AND  
 target.rating = source.rating  
WHEN MATCHED   
 AND source.METADATA$ACTION = 'DELETE'   
 AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Update the existing record to close its validity period  
 UPDATE SET   
 target.eff\_end\_date = CURRENT\_TIMESTAMP,  
 target.is\_current = FALSE  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT'   
 AND source.METADATA$ISUPDATE = 'TRUE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 delivery\_agent\_hk, -- Hash key  
 delivery\_agent\_id,  
 name,  
 phone,  
 vehicle\_type,  
 location\_id\_fk,  
 status,  
 gender,  
 rating,  
 eff\_start\_date,  
 eff\_end\_date,  
 is\_current  
 )  
 VALUES (  
 hash(SHA1\_HEX(CONCAT(source.delivery\_agent\_id, source.name, source.phone,   
 source.vehicle\_type, source.location\_id\_fk, source.status,   
 source.gender, source.rating))), -- Hash key  
 delivery\_agent\_id,  
 source.name,  
 source.phone,  
 source.vehicle\_type,  
 location\_id\_fk,  
 source.status,  
 source.gender,  
 source.rating,  
 CURRENT\_TIMESTAMP, -- Effective start date  
 NULL, -- Effective end date (NULL for current record)  
 TRUE -- IS\_CURRENT = TRUE for new record  
 )  
WHEN NOT MATCHED   
 AND source.METADATA$ACTION = 'INSERT'   
 AND source.METADATA$ISUPDATE = 'FALSE' THEN  
 -- Insert new record with current data and new effective start date  
 INSERT (  
 delivery\_agent\_hk, -- Hash key  
 delivery\_agent\_id,  
 name,  
 phone,  
 vehicle\_type,  
 location\_id\_fk,  
 status,  
 gender,  
 rating,  
 eff\_start\_date,  
 eff\_end\_date,  
 is\_current  
 )  
 VALUES (  
 hash(SHA1\_HEX(CONCAT(source.delivery\_agent\_id, source.name, source.phone,   
 source.vehicle\_type, source.location\_id\_fk, source.status,  
 source.gender, source.rating))), -- Hash key  
 source.delivery\_agent\_id,  
 source.name,  
 source.phone,  
 source.vehicle\_type,  
 source.location\_id\_fk,  
 source.status,  
 source.gender,  
 source.rating,  
 CURRENT\_TIMESTAMP, -- Effective start date  
 NULL, -- Effective end date (NULL for current record)  
 TRUE -- IS\_CURRENT = TRUE for new record  
 );  
  
  
  
  
-- part-2  
  
copy into deliveryagent (deliveryagentid, name, phone, vehicletype, locationid,   
 status, gender, rating, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as deliveryagentid,  
 t.$2::text as name,  
 t.$3::text as phone,  
 t.$4::text as vehicletype,  
 t.$5::text as locationid,  
 t.$6::text as status,  
 t.$7::text as gender,  
 t.$8::text as rating,  
 t.$9::text as createddate,  
 t.$10::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/delivery-agent/day-02-delivery-agent.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Delivery Entity — SQL Scripts**

Press enter or click to view image in full size

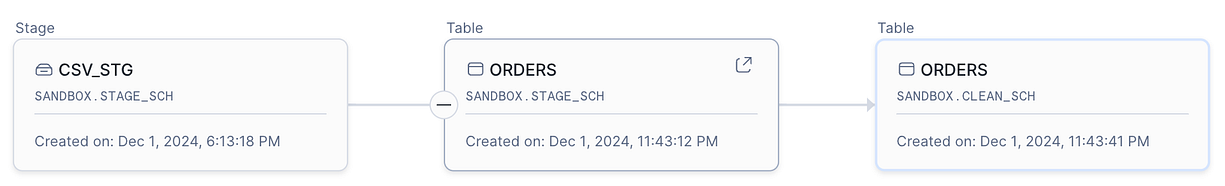


**Delivery Entity Data Lineage**

use role sysadmin;  
  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
list @stage\_sch.csv\_stg/initial/delivery/;  
  
-- this table may have additional information like picked time, accept time etc.  
create or replace table stage\_sch.delivery (  
 deliveryid text comment 'Primary Key (Source System)', -- foreign key reference as text (no constraint in snowflake)  
 orderid text comment 'Order FK (Source System)', -- foreign key reference as text (no constraint in snowflake)  
 deliveryagentid text comment 'Delivery Agent FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 deliverystatus text, -- delivery status as text  
 estimatedtime text, -- estimated time as text  
 addressid text comment 'Customer Address FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 deliverydate text, -- delivery date as text  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the delivery stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
create or replace stream stage\_sch.delivery\_stm   
on table stage\_sch.delivery  
append\_only = true  
comment = 'this is the append-only stream object on delivery table that only gets delta data';  
  
  
copy into stage\_sch.delivery (deliveryid,orderid, deliveryagentid, deliverystatus,   
 estimatedtime, addressid, deliverydate, createddate,   
 modifieddate, \_stg\_file\_name, \_stg\_file\_load\_ts,   
 \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as deliveryid,  
 t.$2::text as orderid,  
 t.$3::text as deliveryagentid,  
 t.$4::text as deliverystatus,  
 t.$5::text as estimatedtime,  
 t.$6::text as addressid,  
 t.$7::text as deliverydate,  
 t.$8::text as createddate,  
 t.$9::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/delivery/delivery-initial-load.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
  
CREATE OR REPLACE TABLE clean\_sch.delivery (  
 delivery\_sk INT AUTOINCREMENT PRIMARY KEY comment 'Surrogate Key (EDW)', -- Primary key with auto-increment  
 delivery\_id INT NOT NULL comment 'Primary Key (Source System)',  
 order\_id\_fk NUMBER NOT NULL comment 'Order FK (Source System)', -- Foreign key reference, converted to numeric type  
 delivery\_agent\_id\_fk NUMBER NOT NULL comment 'Delivery Agent FK (Source System)', -- Foreign key reference, converted to numeric type  
 delivery\_status STRING, -- Delivery status, stored as a string  
 estimated\_time STRING, -- Estimated time, stored as a string  
 customer\_address\_id\_fk NUMBER NOT NULL comment 'Customer Address FK (Source System)', -- Foreign key reference, converted to numeric type  
 delivery\_date TIMESTAMP, -- Delivery date, converted to timestamp  
 created\_date TIMESTAMP, -- Created date, converted to timestamp  
 modified\_date TIMESTAMP, -- Modified date, converted to timestamp  
  
 -- Audit columns with appropriate data types  
 \_stg\_file\_name STRING, -- Source file name  
 \_stg\_file\_load\_ts TIMESTAMP, -- Source file load timestamp  
 \_stg\_file\_md5 STRING, -- MD5 checksum of the source file  
 \_copy\_data\_ts TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Metadata timestamp  
)  
comment = 'Delivery entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
create or replace stream CLEAN\_SCH.delivery\_stm   
on table CLEAN\_SCH.delivery  
comment = 'This is the stream object on delivery agent table table to track insert, update, and delete changes';  
  
MERGE INTO   
 clean\_sch.delivery AS target  
USING   
 stage\_sch.delivery\_stm AS source  
ON   
 target.delivery\_id = TO\_NUMBER(source.deliveryid) and  
 target.order\_id\_fk = TO\_NUMBER(source.orderid) and  
 target.delivery\_agent\_id\_fk = TO\_NUMBER(source.deliveryagentid)  
WHEN MATCHED THEN  
 -- Update the existing record with the latest data  
 UPDATE SET  
 delivery\_status = source.deliverystatus,  
 estimated\_time = source.estimatedtime,  
 customer\_address\_id\_fk = TO\_NUMBER(source.addressid),  
 delivery\_date = TO\_TIMESTAMP(source.deliverydate),  
 created\_date = TO\_TIMESTAMP(source.createddate),  
 modified\_date = TO\_TIMESTAMP(source.modifieddate),  
 \_stg\_file\_name = source.\_stg\_file\_name,  
 \_stg\_file\_load\_ts = source.\_stg\_file\_load\_ts,  
 \_stg\_file\_md5 = source.\_stg\_file\_md5,  
 \_copy\_data\_ts = source.\_copy\_data\_ts  
WHEN NOT MATCHED THEN  
 -- Insert new record if no match is found  
 INSERT (  
 delivery\_id,  
 order\_id\_fk,  
 delivery\_agent\_id\_fk,  
 delivery\_status,  
 estimated\_time,  
 customer\_address\_id\_fk,  
 delivery\_date,  
 created\_date,  
 modified\_date,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 )  
 VALUES (  
 TO\_NUMBER(source.deliveryid),  
 TO\_NUMBER(source.orderid),  
 TO\_NUMBER(source.deliveryagentid),  
 source.deliverystatus,  
 source.estimatedtime,  
 TO\_NUMBER(source.addressid),  
 TO\_TIMESTAMP(source.deliverydate),  
 TO\_TIMESTAMP(source.createddate),  
 TO\_TIMESTAMP(source.modifieddate),  
 source.\_stg\_file\_name,  
 source.\_stg\_file\_load\_ts,  
 source.\_stg\_file\_md5,  
 source.\_copy\_data\_ts  
 );

**Order Entity — SQL Script**

Press enter or click to view image in full size

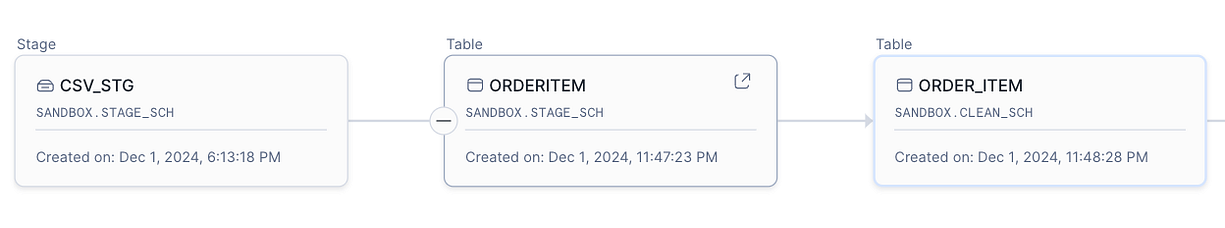


**Order Entity Data Lineage**

use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
create or replace table stage\_sch.orders (  
 orderid text comment 'Primary Key (Source System)', -- primary key as text  
 customerid text comment 'Customer FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 restaurantid text comment 'Restaurant FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 orderdate text, -- order date as text  
 totalamount text, -- total amount as text (no decimal constraint)  
 status text, -- status as text  
 paymentmethod text, -- payment method as text  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the order stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
  
create or replace stream stage\_sch.orders\_stm   
on table stage\_sch.orders  
append\_only = true  
comment = 'This is the append-only stream object on orders entity that only gets delta data';  
  
list @stage\_sch.csv\_stg/initial/orders/orders-initial.csv;  
  
  
copy into stage\_sch.orders (orderid, customerid, restaurantid, orderdate, totalamount,   
 status, paymentmethod, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as orderid,  
 t.$2::text as customerid,  
 t.$3::text as restaurantid,  
 t.$4::text as orderdate,  
 t.$5::text as totalamount,  
 t.$6::text as status,  
 t.$7::text as paymentmethod,  
 t.$8::text as createddate,  
 t.$9::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/orders/orders-initial.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
  
CREATE OR REPLACE TABLE CLEAN\_SCH.ORDERS (  
 ORDER\_SK NUMBER AUTOINCREMENT PRIMARY KEY comment 'Surrogate Key (EDW)', -- Auto-incremented primary key  
 ORDER\_ID BIGINT UNIQUE comment 'Primary Key (Source System)', -- Primary key inferred as BIGINT  
 CUSTOMER\_ID\_FK BIGINT comment 'Customer FK(Source System)', -- Foreign key inferred as BIGINT  
 RESTAURANT\_ID\_FK BIGINT comment 'Restaurant FK(Source System)', -- Foreign key inferred as BIGINT  
 ORDER\_DATE TIMESTAMP, -- Order date inferred as TIMESTAMP  
 TOTAL\_AMOUNT DECIMAL(10, 2), -- Total amount inferred as DECIMAL with two decimal places  
 STATUS STRING, -- Status as STRING  
 PAYMENT\_METHOD STRING, -- Payment method as STRING  
 created\_dt timestamp\_tz, -- record creation date  
 modified\_dt timestamp\_tz, -- last modified date, allows null if not modified  
  
 -- additional audit columns  
 \_stg\_file\_name string, -- file name for audit  
 \_stg\_file\_load\_ts timestamp\_ntz, -- file load timestamp for audit  
 \_stg\_file\_md5 string, -- md5 hash for file content for audit  
 \_copy\_data\_ts timestamp\_ntz default current\_timestamp -- timestamp when data is copied, defaults to current timestamp  
)  
comment = 'Order entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
-- Stream object to capture the changes.   
create or replace stream CLEAN\_SCH.ORDERS\_stm   
on table CLEAN\_SCH.ORDERS  
comment = 'This is the stream object on ORDERS table table to track insert, update, and delete changes';  
  
  
MERGE INTO CLEAN\_SCH.ORDERS AS target  
USING STAGE\_SCH.ORDERS\_STM AS source  
 ON target.ORDER\_ID = TRY\_TO\_NUMBER(source.ORDERID) -- Match based on ORDER\_ID  
WHEN MATCHED THEN  
 -- Update existing records  
 UPDATE SET  
 TOTAL\_AMOUNT = TRY\_TO\_DECIMAL(source.TOTALAMOUNT),  
 STATUS = source.STATUS,  
 PAYMENT\_METHOD = source.PAYMENTMETHOD,  
 MODIFIED\_DT = TRY\_TO\_TIMESTAMP\_TZ(source.MODIFIEDDATE),  
 \_STG\_FILE\_NAME = source.\_STG\_FILE\_NAME,  
 \_STG\_FILE\_LOAD\_TS = source.\_STG\_FILE\_LOAD\_TS,  
 \_STG\_FILE\_MD5 = source.\_STG\_FILE\_MD5,  
 \_COPY\_DATA\_TS = CURRENT\_TIMESTAMP  
WHEN NOT MATCHED THEN  
 -- Insert new records  
 INSERT (  
 ORDER\_ID,  
 CUSTOMER\_ID\_FK,  
 RESTAURANT\_ID\_FK,  
 ORDER\_DATE,  
 TOTAL\_AMOUNT,  
 STATUS,  
 PAYMENT\_METHOD,  
 CREATED\_DT,  
 MODIFIED\_DT,  
 \_STG\_FILE\_NAME,  
 \_STG\_FILE\_LOAD\_TS,  
 \_STG\_FILE\_MD5,  
 \_COPY\_DATA\_TS  
 )  
 VALUES (  
 TRY\_TO\_NUMBER(source.ORDERID),  
 TRY\_TO\_NUMBER(source.CUSTOMERID),  
 TRY\_TO\_NUMBER(source.RESTAURANTID),  
 TRY\_TO\_TIMESTAMP(source.ORDERDATE),  
 TRY\_TO\_DECIMAL(source.TOTALAMOUNT),  
 source.STATUS,  
 source.PAYMENTMETHOD,  
 TRY\_TO\_TIMESTAMP\_TZ(source.CREATEDDATE),  
 TRY\_TO\_TIMESTAMP\_TZ(source.MODIFIEDDATE),  
 source.\_STG\_FILE\_NAME,  
 source.\_STG\_FILE\_LOAD\_TS,  
 source.\_STG\_FILE\_MD5,  
 CURRENT\_TIMESTAMP  
 );  
  
  
-- part-2  
list @stage\_sch.csv\_stg/delta/orders/;  
copy into stage\_sch.orders (orderid, customerid, restaurantid, orderdate, totalamount,   
 status, paymentmethod, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as orderid,  
 t.$2::text as customerid,  
 t.$3::text as restaurantid,  
 t.$4::text as orderdate,  
 t.$5::text as totalamount,  
 t.$6::text as status,  
 t.$7::text as paymentmethod,  
 t.$8::text as createddate,  
 t.$9::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/orders/day-02-orders.csv t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Order Item — SQL Script**

Press enter or click to view image in full size



**Order Item Entity Data Lineage**

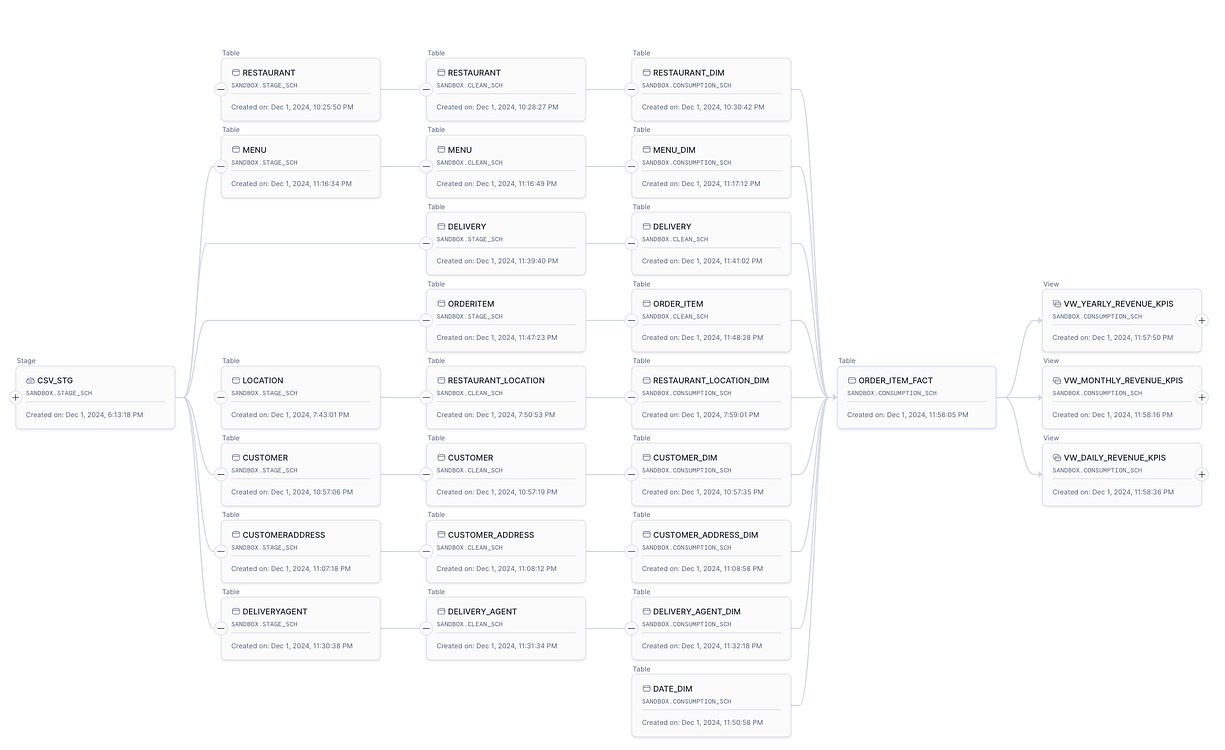
use role sysadmin;  
use database sandbox;  
use schema stage\_sch;  
use warehouse adhoc\_wh;  
  
create or replace table stage\_sch.orderitem (  
 orderitemid text comment 'Primary Key (Source System)', -- primary key as text  
 orderid text comment 'Order FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 menuid text comment 'Menu FK(Source System)', -- foreign key reference as text (no constraint in snowflake)  
 quantity text, -- quantity as text  
 price text, -- price as text (no decimal constraint)  
 subtotal text, -- subtotal as text (no decimal constraint)  
 createddate text, -- created date as text  
 modifieddate text, -- modified date as text  
  
 -- audit columns with appropriate data types  
 \_stg\_file\_name text,  
 \_stg\_file\_load\_ts timestamp,  
 \_stg\_file\_md5 text,  
 \_copy\_data\_ts timestamp default current\_timestamp  
)  
comment = 'This is the order item stage/raw table where data will be copied from internal stage using copy command. This is as-is data represetation from the source location. All the columns are text data type except the audit columns that are added for traceability.';  
  
create or replace stream stage\_sch.orderitem\_stm   
on table stage\_sch.orderitem  
append\_only = true  
comment = 'This is the append-only stream object on order item table that only gets delta data';  
  
list @stage\_sch.csv\_stg/initial/order-item;  
  
copy into stage\_sch.orderitem (orderitemid, orderid, menuid, quantity, price,   
 subtotal, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as orderitemid,  
 t.$2::text as orderid,  
 t.$3::text as menuid,  
 t.$4::text as quantity,  
 t.$5::text as price,  
 t.$6::text as subtotal,  
 t.$7::text as createddate,  
 t.$8::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/initial/order-items/ t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;  
  
select \* from stage\_sch.orderitem;  
select \* from stage\_sch.orderitem\_stm;  
  
CREATE OR REPLACE TABLE clean\_sch.order\_item (  
 order\_item\_sk NUMBER AUTOINCREMENT primary key comment 'Surrogate Key (EDW)', -- Auto-incremented unique identifier for each order item  
 order\_item\_id NUMBER NOT NULL UNIQUE comment 'Primary Key (Source System)',  
 order\_id\_fk NUMBER NOT NULL comment 'Order FK(Source System)', -- Foreign key reference for Order ID  
 menu\_id\_fk NUMBER NOT NULL comment 'Menu FK(Source System)', -- Foreign key reference for Menu ID  
 quantity NUMBER(10, 2), -- Quantity as a decimal number  
 price NUMBER(10, 2), -- Price as a decimal number  
 subtotal NUMBER(10, 2), -- Subtotal as a decimal number  
 created\_dt TIMESTAMP, -- Created date of the order item  
 modified\_dt TIMESTAMP, -- Modified date of the order item  
  
 -- Audit columns  
 \_stg\_file\_name VARCHAR(255), -- File name of the staging file  
 \_stg\_file\_load\_ts TIMESTAMP, -- Timestamp when the file was loaded  
 \_stg\_file\_md5 VARCHAR(255), -- MD5 hash of the file for integrity check  
 \_copy\_data\_ts TIMESTAMP DEFAULT CURRENT\_TIMESTAMP -- Timestamp when data is copied into the clean layer  
)  
comment = 'Order item entity under clean schema with appropriate data type under clean schema layer, data is populated using merge statement from the stage layer location table. This table does not support SCD2';  
  
create or replace stream CLEAN\_SCH.order\_item\_stm   
on table CLEAN\_SCH.order\_item  
comment = 'This is the stream object on order\_item table table to track insert, update, and delete changes';  
  
  
select \* from clean\_sch.order\_item\_stm;  
  
  
MERGE INTO clean\_sch.order\_item AS target  
USING stage\_sch.orderitem\_stm AS source  
ON   
 target.order\_item\_id = source.orderitemid and  
 target.order\_id\_fk = source.orderid and  
 target.menu\_id\_fk = source.menuid  
WHEN MATCHED THEN  
 -- Update the existing record with new data  
 UPDATE SET   
 target.quantity = source.quantity,  
 target.price = source.price,  
 target.subtotal = source.subtotal,  
 target.created\_dt = source.createddate,  
 target.modified\_dt = source.modifieddate,  
 target.\_stg\_file\_name = source.\_stg\_file\_name,  
 target.\_stg\_file\_load\_ts = source.\_stg\_file\_load\_ts,  
 target.\_stg\_file\_md5 = source.\_stg\_file\_md5,  
 target.\_copy\_data\_ts = source.\_copy\_data\_ts  
WHEN NOT MATCHED THEN  
 -- Insert new record if no match is found  
 INSERT (  
 order\_item\_id,  
 order\_id\_fk,  
 menu\_id\_fk,  
 quantity,  
 price,  
 subtotal,  
 created\_dt,  
 modified\_dt,  
 \_stg\_file\_name,  
 \_stg\_file\_load\_ts,  
 \_stg\_file\_md5,  
 \_copy\_data\_ts  
 )  
 VALUES (  
 source.orderitemid,  
 source.orderid,  
 source.menuid,  
 source.quantity,  
 source.price,  
 source.subtotal,  
 source.createddate,  
 source.modifieddate,  
 source.\_stg\_file\_name,  
 source.\_stg\_file\_load\_ts,  
 source.\_stg\_file\_md5,  
 CURRENT\_TIMESTAMP()  
 );  
  
  
-- part-2  
list @stage\_sch.csv\_stg/delta/order-items/;  
  
copy into stage\_sch.orderitem (orderitemid, orderid, menuid, quantity, price,   
 subtotal, createddate, modifieddate,  
 \_stg\_file\_name, \_stg\_file\_load\_ts, \_stg\_file\_md5, \_copy\_data\_ts)  
from (  
 select   
 t.$1::text as orderitemid,  
 t.$2::text as orderid,  
 t.$3::text as menuid,  
 t.$4::text as quantity,  
 t.$5::text as price,  
 t.$6::text as subtotal,  
 t.$7::text as createddate,  
 t.$8::text as modifieddate,  
 metadata$filename as \_stg\_file\_name,  
 metadata$file\_last\_modified as \_stg\_file\_load\_ts,  
 metadata$file\_content\_key as \_stg\_file\_md5,  
 current\_timestamp as \_copy\_data\_ts  
 from @stage\_sch.csv\_stg/delta/order-items/ t  
)  
file\_format = (format\_name = 'stage\_sch.csv\_file\_format')  
on\_error = abort\_statement;

**Date Dimension — SQL**

use role sysadmin;  
use warehouse adhoc\_wh;  
use database sandbox;  
use schema CONSUMPTION\_SCH;  
  
  
CREATE OR REPLACE TABLE CONSUMPTION\_SCH.DATE\_DIM (  
 DATE\_DIM\_HK NUMBER PRIMARY KEY comment 'Menu Dim HK (EDW)', -- Surrogate key for date dimension  
 CALENDAR\_DATE DATE UNIQUE, -- The actual calendar date  
 YEAR NUMBER, -- Year  
 QUARTER NUMBER, -- Quarter (1-4)  
 MONTH NUMBER, -- Month (1-12)  
 WEEK NUMBER, -- Week of the year  
 DAY\_OF\_YEAR NUMBER, -- Day of the year (1-365/366)  
 DAY\_OF\_WEEK NUMBER, -- Day of the week (1-7)  
 DAY\_OF\_THE\_MONTH NUMBER, -- Day of the month (1-31)  
 DAY\_NAME STRING -- Name of the day (e.g., Monday)  
)  
comment = 'Date dimension table created using min of order data.';  
  
insert into CONSUMPTION\_SCH.DATE\_DIM   
with recursive my\_date\_dim\_cte as   
(  
 -- anchor clause  
 select   
 current\_date() as today,  
 year(today) as year,  
 quarter(today) as quarter,  
 month(today) as month,  
 week(today) as week,  
 dayofyear(today) as day\_of\_year,  
 dayofweek(today) as day\_of\_week,  
 day(today) as day\_of\_the\_month,  
 dayname(today) as day\_name  
  
 union all  
  
 -- recursive clause  
 select   
 dateadd('day', -1, today) as today\_r,  
 year(today\_r) as year,  
 quarter(today\_r) as quarter,  
 month(today\_r) as month,  
 week(today\_r) as week,  
 dayofyear(today\_r) as day\_of\_year,  
 dayofweek(today\_r) as day\_of\_week,  
 day(today\_r) as day\_of\_the\_month,  
 dayname(today\_r) as day\_name  
 from   
 my\_date\_dim\_cte  
 where   
 today\_r > (select date(min(order\_date)) from clean\_sch.orders)  
)  
select   
 hash(SHA1\_hex(today)) as DATE\_DIM\_HK,  
 today , -- The actual calendar date  
 YEAR, -- Year  
 QUARTER, -- Quarter (1-4)  
 MONTH, -- Month (1-12)  
 WEEK, -- Week of the year  
 DAY\_OF\_YEAR, -- Day of the year (1-365/366)  
 DAY\_OF\_WEEK, -- Day of the week (1-7)  
 DAY\_OF\_THE\_MONTH, -- Day of the month (1-31)  
 DAY\_NAME   
from my\_date\_dim\_cte;

**Order Item Fact — SQL Scripts**

Press enter or click to view image in full size



**Order Fact Data Lineage**

use role sysadmin;  
use warehouse adhoc\_wh;  
use database sandbox;  
use schema consumption\_sch;  
  
  
CREATE OR REPLACE TABLE consumption\_sch.order\_item\_fact (  
 order\_item\_fact\_sk NUMBER AUTOINCREMENT comment 'Surrogate Key (EDW)', -- Surrogate key for the fact table  
 order\_item\_id NUMBER comment 'Order Item FK (Source System)', -- Natural key from the source data  
 order\_id NUMBER comment 'Order FK (Source System)', -- Reference to the order dimension  
 customer\_dim\_key NUMBER comment 'Order FK (Source System)', -- Reference to the customer dimension  
 customer\_address\_dim\_key NUMBER, -- Reference to the customer dimension  
 restaurant\_dim\_key NUMBER, -- Reference to the restaurant dimension  
 restaurant\_location\_dim\_key NUMBER, -- Reference to the restaurant dimension  
 menu\_dim\_key NUMBER, -- Reference to the menu dimension  
 delivery\_agent\_dim\_key NUMBER, -- Reference to the delivery agent dimension  
 order\_date\_dim\_key NUMBER, -- Reference to the date dimension  
 quantity NUMBER, -- Measure  
 price NUMBER(10, 2), -- Measure  
 subtotal NUMBER(10, 2), -- Measure  
 delivery\_status VARCHAR, -- Delivery information  
 estimated\_time VARCHAR -- Delivery information  
)  
comment = 'The item order fact table that has item level price, quantity and other details';  
  
  
MERGE INTO consumption\_sch.order\_item\_fact AS target  
USING (  
 SELECT   
 oi.Order\_Item\_ID AS order\_item\_id,  
 oi.Order\_ID\_fk AS order\_id,  
 c.CUSTOMER\_HK AS customer\_dim\_key,  
 ca.CUSTOMER\_ADDRESS\_HK AS customer\_address\_dim\_key,  
 r.RESTAURANT\_HK AS restaurant\_dim\_key,   
 rl.restaurant\_location\_hk as restaurant\_location\_dim\_key,  
 m.Menu\_Dim\_HK AS menu\_dim\_key,  
 da.DELIVERY\_AGENT\_HK AS delivery\_agent\_dim\_key,  
 dd.DATE\_DIM\_HK AS order\_date\_dim\_key,  
 oi.Quantity::number(2) AS quantity,  
 oi.Price AS price,  
 oi.Subtotal AS subtotal,  
 o.PAYMENT\_METHOD,  
 d.delivery\_status AS delivery\_status,  
 d.estimated\_time AS estimated\_time,  
 FROM   
 clean\_sch.order\_item\_stm oi  
 JOIN   
 clean\_sch.orders\_stm o ON oi.Order\_ID\_fk = o.Order\_ID  
 JOIN   
 clean\_sch.delivery\_stm d ON o.Order\_ID = d.Order\_ID\_fk  
 JOIN   
 consumption\_sch.CUSTOMER\_DIM c on o.Customer\_ID\_fk = c.customer\_id  
 JOIN   
 consumption\_sch.CUSTOMER\_ADDRESS\_DIM ca on c.Customer\_ID = ca.CUSTOMER\_ID\_fk  
 JOIN   
 consumption\_sch.restaurant\_dim r on o.Restaurant\_ID\_fk = r.restaurant\_id  
 JOIN   
 consumption\_sch.menu\_dim m ON oi.MENU\_ID\_fk = m.menu\_id  
 JOIN   
 consumption\_sch.delivery\_agent\_dim da ON d.Delivery\_Agent\_ID\_fk = da.delivery\_agent\_id  
 JOIN   
 consumption\_sch.restaurant\_location\_dim rl on r.LOCATION\_ID\_FK = rl.location\_id  
 JOIN   
 CONSUMPTION\_SCH.DATE\_DIM dd on dd.calendar\_date = date(o.order\_date)  
) AS source\_stm  
ON   
 target.order\_item\_id = source\_stm.order\_item\_id and   
 target.order\_id = source\_stm.order\_id  
WHEN MATCHED THEN  
 -- Update existing fact record  
 UPDATE SET  
 target.customer\_dim\_key = source\_stm.customer\_dim\_key,  
 target.customer\_address\_dim\_key = source\_stm.customer\_address\_dim\_key,  
 target.restaurant\_dim\_key = source\_stm.restaurant\_dim\_key,  
 target.restaurant\_location\_dim\_key = source\_stm.restaurant\_location\_dim\_key,  
 target.menu\_dim\_key = source\_stm.menu\_dim\_key,  
 target.delivery\_agent\_dim\_key = source\_stm.delivery\_agent\_dim\_key,  
 target.order\_date\_dim\_key = source\_stm.order\_date\_dim\_key,  
 target.quantity = source\_stm.quantity,  
 target.price = source\_stm.price,  
 target.subtotal = source\_stm.subtotal,  
 target.delivery\_status = source\_stm.delivery\_status,  
 target.estimated\_time = source\_stm.estimated\_time  
WHEN NOT MATCHED THEN  
 -- Insert new fact record  
 INSERT (  
 order\_item\_id,  
 order\_id,  
 customer\_dim\_key,  
 customer\_address\_dim\_key,  
 restaurant\_dim\_key,  
 restaurant\_location\_dim\_key,  
 menu\_dim\_key,  
 delivery\_agent\_dim\_key,  
 order\_date\_dim\_key,  
 quantity,  
 price,  
 subtotal,  
 delivery\_status,  
 estimated\_time  
 )  
 VALUES (  
 source\_stm.order\_item\_id,  
 source\_stm.order\_id,  
 source\_stm.customer\_dim\_key,  
 source\_stm.customer\_address\_dim\_key,  
 source\_stm.restaurant\_dim\_key,  
 source\_stm.restaurant\_location\_dim\_key,  
 source\_stm.menu\_dim\_key,  
 source\_stm.delivery\_agent\_dim\_key,  
 source\_stm.order\_date\_dim\_key,  
 source\_stm.quantity,  
 source\_stm.price,  
 source\_stm.subtotal,  
 source\_stm.delivery\_status,  
 source\_stm.estimated\_time  
 );  
  
  
-- start with   
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_customer\_dim  
 foreign key (customer\_dim\_key)  
 references consumption\_sch.customer\_dim (customer\_hk);  
  
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_customer\_address\_dim  
 foreign key (customer\_address\_dim\_key)  
 references consumption\_sch.customer\_address\_dim (CUSTOMER\_ADDRESS\_HK);  
  
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_restaurant\_dim  
 foreign key (restaurant\_dim\_key)  
 references consumption\_sch.restaurant\_dim (restaurant\_hk);  
  
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_restaurant\_location\_dim  
 foreign key (restaurant\_location\_dim\_key)  
 references consumption\_sch.restaurant\_location\_dim (restaurant\_location\_hk);  
  
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_menu\_dim  
 foreign key (menu\_dim\_key)  
 references consumption\_sch.menu\_dim (menu\_dim\_hk);  
  
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_delivery\_agent\_dim  
 foreign key (delivery\_agent\_dim\_key)  
 references consumption\_sch.delivery\_agent\_dim (delivery\_agent\_hk);  
  
alter table consumption\_sch.order\_item\_fact  
 add constraint fk\_order\_item\_fact\_delivery\_date\_dim  
 foreign key (order\_date\_dim\_key)  
 references consumption\_sch.date\_dim (date\_dim\_hk);

**Final View — SQL Scripts**

use role sysadmin;  
use warehouse adhoc\_wh;  
use database sandbox;  
use schema consumption\_sch;  
  
select \* from consumption\_sch.order\_item\_fact limit 100;  
  
create or replace view consumption\_sch.vw\_yearly\_revenue\_kpis as  
select  
 d.year as year, -- fetch year from date\_dim  
 sum(fact.subtotal) as total\_revenue,  
 count(distinct fact.order\_id) as total\_orders,  
 round(sum(fact.subtotal) / count(distinct fact.order\_id), 2) as avg\_revenue\_per\_order,  
 round(sum(fact.subtotal) / count(fact.order\_item\_id), 2) as avg\_revenue\_per\_item,  
 max(fact.subtotal) as max\_order\_value  
from  
 consumption\_sch.order\_item\_fact fact  
join  
 consumption\_sch.date\_dim d  
on  
 fact.order\_date\_dim\_key = d.date\_dim\_hk -- join fact table with date\_dim table  
where DELIVERY\_STATUS = 'Delivered'  
group by  
 d.year  
order by  
 d.year;  
  
  
CREATE OR REPLACE VIEW consumption\_sch.vw\_monthly\_revenue\_kpis AS  
SELECT  
 d.YEAR AS year, -- Fetch year from DATE\_DIM  
 d.MONTH AS month, -- Fetch month from DATE\_DIM  
 SUM(fact.subtotal) AS total\_revenue,  
 COUNT(DISTINCT fact.order\_id) AS total\_orders,  
 ROUND(SUM(fact.subtotal) / COUNT(DISTINCT fact.order\_id), 2) AS avg\_revenue\_per\_order,  
 ROUND(SUM(fact.subtotal) / COUNT(fact.order\_item\_id), 2) AS avg\_revenue\_per\_item,  
 MAX(fact.subtotal) AS max\_order\_value  
FROM  
 consumption\_sch.order\_item\_fact fact  
JOIN  
 consumption\_sch.DATE\_DIM d  
ON  
 fact.order\_date\_dim\_key = d.DATE\_DIM\_HK -- Join fact table with DATE\_DIM table  
where DELIVERY\_STATUS = 'Delivered'  
GROUP BY  
 d.YEAR, d.MONTH  
ORDER BY  
 d.YEAR, d.MONTH;  
  
  
CREATE OR REPLACE VIEW consumption\_sch.vw\_daily\_revenue\_kpis AS  
SELECT  
 d.YEAR AS year, -- Fetch year from DATE\_DIM  
 d.MONTH AS month, -- Fetch month from DATE\_DIM  
 d.DAY\_OF\_THE\_MONTH AS day, -- Fetch day from DATE\_DIM  
 SUM(fact.subtotal) AS total\_revenue,  
 COUNT(DISTINCT fact.order\_id) AS total\_orders,  
 ROUND(SUM(fact.subtotal) / COUNT(DISTINCT fact.order\_id), 2) AS avg\_revenue\_per\_order,  
 ROUND(SUM(fact.subtotal) / COUNT(fact.order\_item\_id), 2) AS avg\_revenue\_per\_item,  
 MAX(fact.subtotal) AS max\_order\_value  
FROM  
 consumption\_sch.order\_item\_fact fact  
JOIN  
 consumption\_sch.DATE\_DIM d  
ON  
 fact.order\_date\_dim\_key = d.DATE\_DIM\_HK -- Join fact table with DATE\_DIM table  
 where DELIVERY\_STATUS = 'Delivered'  
GROUP BY  
 d.YEAR, d.MONTH, d.DAY\_OF\_THE\_MONTH -- Group by year, month, and day  
ORDER BY  
 d.YEAR, d.MONTH, d.DAY\_OF\_THE\_MONTH; -- Order by year, month, and day  
  
  
CREATE OR REPLACE VIEW consumption\_sch.vw\_day\_revenue\_kpis AS  
SELECT  
 d.YEAR AS year, -- Fetch year from DATE\_DIM  
 d.MONTH AS month, -- Fetch month from DATE\_DIM  
 d.DAY\_NAME AS DAY\_NAME, -- Fetch day from DATE\_DIM-DAY\_NAME  
 SUM(fact.subtotal) AS total\_revenue,  
 COUNT(DISTINCT fact.order\_id) AS total\_orders,  
 ROUND(SUM(fact.subtotal) / COUNT(DISTINCT fact.order\_id), 2) AS avg\_revenue\_per\_order,  
 ROUND(SUM(fact.subtotal) / COUNT(fact.order\_item\_id), 2) AS avg\_revenue\_per\_item,  
 MAX(fact.subtotal) AS max\_order\_value  
FROM  
 consumption\_sch.order\_item\_fact fact  
JOIN  
 consumption\_sch.DATE\_DIM d  
ON  
 fact.order\_date\_dim\_key = d.DATE\_DIM\_HK -- Join fact table with DATE\_DIM table  
GROUP BY  
 d.YEAR, d.MONTH, d.DAY\_NAME -- Group by year, month, and day  
ORDER BY  
 d.YEAR, d.MONTH, d.DAY\_NAME; -- Order by year, month, and day  
  
  
  
CREATE OR REPLACE VIEW consumption\_sch.vw\_monthly\_revenue\_by\_restaurant AS  
SELECT  
 d.YEAR AS year, -- Fetch year from DATE\_DIM  
 d.MONTH AS month, -- Fetch month from DATE\_DIM  
 fact.DELIVERY\_STATUS,  
 r.name as restaurant\_name,  
 SUM(fact.subtotal) AS total\_revenue,  
 COUNT(DISTINCT fact.order\_id) AS total\_orders,  
 ROUND(SUM(fact.subtotal) / COUNT(DISTINCT fact.order\_id), 2) AS avg\_revenue\_per\_order,  
 ROUND(SUM(fact.subtotal) / COUNT(fact.order\_item\_id), 2) AS avg\_revenue\_per\_item,  
 MAX(fact.subtotal) AS max\_order\_value  
FROM  
 consumption\_sch.order\_item\_fact fact  
JOIN  
 consumption\_sch.DATE\_DIM d  
ON  
 fact.order\_date\_dim\_key = d.DATE\_DIM\_HK   
JOIN  
 consumption\_sch.restaurant\_dim r  
ON  
 fact.restaurant\_dim\_key = r.RESTAURANT\_HK   
GROUP BY  
 d.YEAR, d.MONTH,fact.DELIVERY\_STATUS,restaurant\_name  
ORDER BY  
 d.YEAR, d.MONTH;

**Streamlit Code**

# Import python packages  
import streamlit as st  
import pandas as pd  
import altair as alt  
from snowflake.snowpark.context import get\_active\_session  
  
# App Title  
st.title("Revenue Dashboard")  
  
# Get the current credentials  
session = get\_active\_session()  
  
def format\_revenue(revenue):  
 #return f"₹{revenue / 1\_000\_000:.1f}M"  
 return f"₹{revenue:.1f}"  
  
# Function to alternate row colors  
def highlight\_rows(row):  
 color = '#f2f2f2' if row.name % 2 == 0 else 'white' # Alternate rows  
 return ['background-color: {}'.format(color)] \* len(row)  
  
# Function to fetch KPI data from Snowflake  
def fetch\_kpi\_data():  
 query = """  
 SELECT   
 year,  
 total\_revenue,  
 total\_orders,  
 avg\_revenue\_per\_order,  
 avg\_revenue\_per\_item,  
 max\_order\_value  
 FROM sandbox.consumption\_sch.vw\_yearly\_revenue\_kpis  
 ORDER BY year;  
 """  
 return session.sql(query).collect()  
  
#TO\_CHAR(TO\_DATE(month::text, 'MM'), 'Mon') AS month\_abbr, -- Converts month number to abbreviated month name  
def fetch\_monthly\_kpi\_data(year):  
 query = f"""  
 SELECT   
 month::number(2) as month,  
 total\_revenue::NUMBER(10) AS TOTAL\_REVENUE  
 FROM   
 sandbox.consumption\_sch.vw\_monthly\_revenue\_kpis  
 WHERE year = {year}  
 ORDER BY month;  
 """  
 return session.sql(query).collect()  
  
  
def fetch\_unique\_months(year):  
 query = f"""  
 SELECT   
 DISTINCT MONTH FROM   
 sandbox.consumption\_sch.vw\_monthly\_revenue\_by\_restaurant   
 WHERE YEAR = {year}   
 ORDER BY MONTH;  
 """  
 return session.sql(query).collect()  
   
def fetch\_top\_restaurants(year, month):  
 query = f"""  
 SELECT  
 restaurant\_name,  
 total\_revenue,  
 total\_orders,  
 avg\_revenue\_per\_order,  
 avg\_revenue\_per\_item,  
 max\_order\_value  
 FROM  
 sandbox.consumption\_sch.vw\_monthly\_revenue\_by\_restaurant  
 WHERE  
 YEAR = {year}  
 AND MONTH = {month}  
 ORDER BY  
 total\_revenue DESC  
 LIMIT 10;  
 """  
 return session.sql(query).collect()  
   
# Function to convert Snowpark DataFrame to Pandas DataFrame  
def snowpark\_to\_pandas(snowpark\_df):  
 return pd.DataFrame(  
 snowpark\_df,  
 columns=[  
 'Restaurant Name',  
 'Total Revenue (₹)',  
 'Total Orders',  
 'Avg Revenue per Order (₹)',  
 'Avg Revenue per Item (₹)',  
 'Max Order Value (₹)'  
 ]  
 )  
# Fetch data  
sf\_df = fetch\_kpi\_data()  
df = pd.DataFrame(  
 sf\_df,  
 columns=['YEAR','TOTAL\_REVENUE','TOTAL\_ORDERS','AVG\_REVENUE\_PER\_ORDER','AVG\_REVENUE\_PER\_ITEM','MAX\_ORDER\_VALUE']  
)  
  
# Aggregate Metrics for All Years  
#st.subheader("Aggregate KPIs: Overall Performance")  
col1, col2, col3 = st.columns(3)  
  
with col1:  
 st.metric("Total Revenue (All Years)", format\_revenue(df['TOTAL\_REVENUE'].sum()))  
with col2:  
 st.metric("Total Orders (All Years)", f"{df['TOTAL\_ORDERS'].sum():,}")  
with col3:  
 st.metric("Max Order Value (Overall)", f"₹{df['MAX\_ORDER\_VALUE'].max():,.0f}")  
  
st.divider()  
  
# Year Selection Box  
years = df["YEAR"].unique()  
default\_year = max(years) # Select the most recent year by default  
selected\_year = st.selectbox("Select Year", sorted(years), index=list(years).index(default\_year))  
  
# Filter data for selected year  
year\_data = df[df["YEAR"] == selected\_year]  
total\_revenue = year\_data["TOTAL\_REVENUE"].iloc[0]  
total\_orders = year\_data["TOTAL\_ORDERS"].iloc[0]  
avg\_revenue\_per\_order = year\_data["AVG\_REVENUE\_PER\_ORDER"].iloc[0]  
avg\_revenue\_per\_item = year\_data["AVG\_REVENUE\_PER\_ITEM"].iloc[0]  
max\_order\_value = year\_data["MAX\_ORDER\_VALUE"].iloc[0]  
  
# Get previous year data  
previous\_year = selected\_year - 1  
previous\_year\_data = df[df["YEAR"] == previous\_year]  
  
# If previous year data exists, calculate differences  
if not previous\_year\_data.empty:  
 prev\_total\_revenue = previous\_year\_data["TOTAL\_REVENUE"].iloc[0]  
 prev\_total\_orders = previous\_year\_data["TOTAL\_ORDERS"].iloc[0]  
 prev\_avg\_revenue\_per\_order = previous\_year\_data["AVG\_REVENUE\_PER\_ORDER"].iloc[0]  
 prev\_avg\_revenue\_per\_item = previous\_year\_data["AVG\_REVENUE\_PER\_ITEM"].iloc[0]  
 prev\_max\_order\_value = previous\_year\_data["MAX\_ORDER\_VALUE"].iloc[0]  
  
 # Calculate differences  
 revenue\_diff = total\_revenue - prev\_total\_revenue  
 orders\_diff = total\_orders - prev\_total\_orders  
 avg\_rev\_order\_diff = avg\_revenue\_per\_order - prev\_avg\_revenue\_per\_order  
 avg\_rev\_item\_diff = avg\_revenue\_per\_item - prev\_avg\_revenue\_per\_item  
 max\_order\_diff = max\_order\_value - prev\_max\_order\_value  
else:  
 # If previous year data is not found, set differences to None or zero  
 revenue\_diff = orders\_diff = avg\_rev\_order\_diff = avg\_rev\_item\_diff = max\_order\_diff = None  
  
  
# Display Metrics for Selected Year with Comparison to Previous Year  
# st.subheader(f"KPI Scorecard for {selected\_year}")  
col1, col2, col3 = st.columns(3)  
  
with col1:  
 st.metric(  
 "Total Revenue",   
 format\_revenue(total\_revenue),   
 delta=f"₹{revenue\_diff:.1f}" if revenue\_diff is not None else None  
 )  
 st.metric("Total Orders", f"{total\_orders:,}", delta=f"{orders\_diff:,}" if orders\_diff is not None else None)  
  
 #st.metric("Total Revenue", f"₹{total\_revenue:,.0f}", delta=f"₹{revenue\_diff:,.0f}" if revenue\_diff is not None else None)  
 #st.metric("Total Orders", f"{total\_orders:,}", delta=f"{orders\_diff:,}" if orders\_diff is not None else None)  
  
with col2:  
 st.metric("Avg Revenue per Order", f"₹{avg\_revenue\_per\_order:,.0f}", delta=f"₹{avg\_rev\_order\_diff:,.0f}" if avg\_rev\_order\_diff is not None else None)  
 st.metric("Avg Revenue per Item", f"₹{avg\_revenue\_per\_item:,.0f}", delta=f"₹{avg\_rev\_item\_diff:,.0f}" if avg\_rev\_item\_diff is not None else None)  
  
with col3:  
 st.metric("Max Order Value", f"₹{max\_order\_value:,.0f}", delta=f"₹{max\_order\_diff:,.0f}" if max\_order\_diff is not None else None)  
  
  
  
st.divider()  
# -----------------------------------------  
  
  
# Fetch and prepare data  
month\_sf\_df = fetch\_monthly\_kpi\_data(selected\_year)  
month\_df = pd.DataFrame(  
 month\_sf\_df,  
 columns=['Month', 'Total Monthly Revenue']  
)  
  
# Map numeric months to abbreviated month names  
month\_mapping = {  
 1: 'Jan', 2: 'Feb', 3: 'Mar', 4: 'Apr',  
 5: 'May', 6: 'Jun', 7: 'Jul', 8: 'Aug',  
 9: 'Sep', 10: 'Oct', 11: 'Nov', 12: 'Dec'  
}  
month\_df['Month'] = month\_df['Month'].map(month\_mapping)  
  
# Ensure months are in the correct chronological order  
month\_df['Month'] = pd.Categorical(  
 month\_df['Month'],  
 categories=['Jan', 'Feb', 'Mar', 'Apr', 'May', 'Jun', 'Jul', 'Aug', 'Sep', 'Oct', 'Nov', 'Dec'],  
 ordered=True  
)  
month\_df = month\_df.sort\_values('Month') # Sort by chronological month order  
  
# Convert revenue to millions  
month\_df['Total Monthly Revenue'] = month\_df['Total Monthly Revenue']   
  
# Plot Monthly Revenue Trend using Bar Chart  
st.subheader(f"{selected\_year} - Monthly Revenue Trend")  
# Create the Altair Bar Chart with Custom Color  
bar\_chart = alt.Chart(month\_df).mark\_bar(color="#ff5200").encode(  
 x=alt.X('Month', sort='ascending', title='Month'),  
 y=alt.Y('Total Monthly Revenue', title='Revenue (₹)')  
).properties(  
 width=700,  
 height=400  
)  
  
# Display the chart in Streamlit  
st.altair\_chart(bar\_chart, use\_container\_width=True)  
  
# Add a Trending Chart using Altair  
st.subheader(f"{selected\_year} - Monthly Revenue Trend")  
  
trend\_chart = alt.Chart(month\_df).mark\_line(color="#ff5200", point=alt.OverlayMarkDef(color="#ff5200")).encode(  
 x=alt.X('Month', sort='ascending', title='Month'),  
 y=alt.Y('Total Monthly Revenue', title='Revenue (₹)', scale=alt.Scale(domain=[0, month\_df['Total Monthly Revenue'].max()])),  
 tooltip=[  
 alt.Tooltip('Month', title='Month'),  
 alt.Tooltip('Total Monthly Revenue', title='Revenue (₹M)', format='.2f') # Format to 2 decimal places  
 ]  
).properties(  
 width=700,  
 height=400  
).configure\_point(  
 size=60  
)  
  
st.altair\_chart(trend\_chart, use\_container\_width=True)  
  
# Month Selection based on the selected year  
if selected\_year:  
  
 #get the unique months  
 month\_sf\_df = fetch\_unique\_months(selected\_year)  
 print(month\_sf\_df)  
 #convert into df  
 month\_df = pd.DataFrame(  
 month\_sf\_df,  
 columns=['MONTH']  
 )  
 print(month\_df)  
  
 # Year Selection Box  
 months = month\_df["MONTH"].unique()  
 default\_month = max(months) # Select the most recent year by default  
 selected\_month = st.selectbox(f"Select Month For {selected\_year}", sorted(months), index=list(months).index(default\_month))  
  
 # Fetch and Display Data  
 if selected\_month:  
 st.subheader(f"Top 10 Restaurants for {selected\_month}/{selected\_year}")  
 top\_restaurants = fetch\_top\_restaurants(selected\_year, selected\_month)  
 if top\_restaurants:  
 top\_restaurants\_df = snowpark\_to\_pandas(top\_restaurants)  
 # Remove index from DataFrame by resetting it and dropping the index column  
 #top\_restaurants\_df\_reset = top\_restaurants\_df.reset\_index(drop=True)  
  
 # Display the DataFrame without index  
 #st.dataframe(top\_restaurants\_df\_reset)  
 #st.dataframe(top\_restaurants\_df)  
  
 # Apply the alternate color style  
 styled\_df = top\_restaurants\_df.style.apply(highlight\_rows, axis=1)  
  
 # Display the styled DataFrame  
 st.dataframe(styled\_df, hide\_index= True)  
 else:  
 st.warning("No data found for the selected year and month.")