CSV files we used : Customers(Has customers data), Products(Has Products data), Sales(Has sales data i.e which customer which product at what cost in which store), Stores(Has stores data), Dimdates(Has timestamps in different formats)

Step 0 : The sales table forms many to many relationship between products, customers and stores tables. It has information about which customer purchased which product from which store at what cost? The entity relationship diagram is as follows:

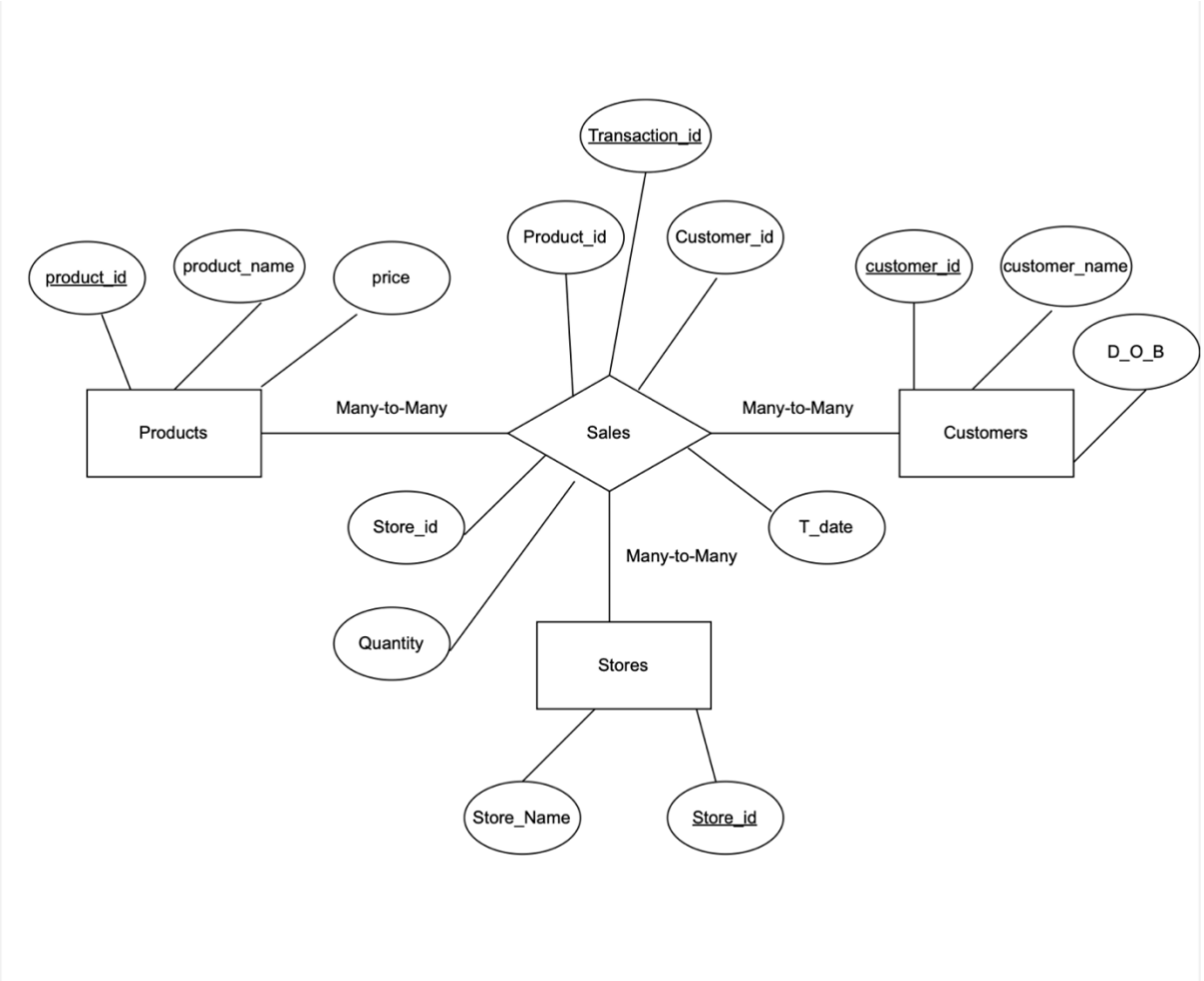

Fig : Entity Relationship Diagram

Figuring out Primary and Foreign Keys:
1. Customers Table:
customer_id -> Primary Key

2. Products Table:
Product_id -> Primary Key

3. Stores Table:
Store_id -> Primary Key

4. Sales Table:
transaction_id -> Primary Key
product_id -> Foreign Key

customer_id -> Foreign Key
store_id -> Foreign Key

Step 1 : Integrate S3 bucket with snowflake with necessary IAM policies and roles in order to get the data from s3 bucket to staging layer. Follow the instructions provided in the following link :
https://docs.snowflake.com/en/user-guide/data-load-s3-config-storage-integration

```
CREATE OR REPLACE STORAGE INTEGRATION s3_stage_demo
  TYPE = EXTERNAL_STAGE
  STORAGE_PROVIDER = 'S3'
  ENABLED = TRUE
  STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::383493188589:role/aashvik-snowflake'
  STORAGE_ALLOWED_LOCATIONS = ('s3://ssttrainingdata/');


DESC INTEGRATION s3_stage_demo;


USE DATABASE SSTRAINING;
CREATE OR REPLACE FILE FORMAT my_csv_format
  TYPE = 'CSV'
  FIELD_OPTIONALLY_ENCLOSED_BY = '"'
  RECORD_DELIMITER = '\n'
  SKIP_HEADER = 1
  NULL_IF = ('NULL', 'null')
  EMPTY_FIELD_AS_NULL = TRUE
  TRIM_SPACE = TRUE
  --parse_header = TRUE;


CREATE OR REPLACE STAGE my_s3_stage
  STORAGE_INTEGRATION = s3_stage_demo
  URL = 's3://ssttrainingdata/'
 FILE_FORMAT = my_csv_format;

ls @my_s3_stage;
```

Step 2 : Now initialize the staging layer tables to get the data from the staging layer(s3 bucket mount) to these tables.

```
CREATE OR REPLACE TABLE customers(
"CUSTOMER_ID" TEXT,
"CUSTOMER_NAME" TEXT,
"DATE_OF_BIRTH" DATE,
"LOADED_AT" TIMESTAMP_NTZ
)

CREATE OR REPLACE TABLE products(
"PRODUCT_ID" TEXT,
"PRODUCT_NAME" TEXT,
"PRICE" NUMBER(4, 2),
"LOADED_AT" TIMESTAMP_NTZ
)
```

Step 3 : Now copy the data from the csv files that are present in the staging layer to the staging layer tables after creation of schema for staging layer tables and also make sure to create a new column(I have used “loaded_at” naming convention) to track the timestamp a record is loaded in. Now create a stored procedure for this whole thing and then schedule a task to run the stored procedure on daily basis. In staging layer we will ensure that we maintain only 7 days of data, there can be duplicates and the maintenance of data i.e. it is

not restricted to 7 days of data it can be more or less than that it depends on the business use case and also the amount of storage that a company wants to make use of.

```
COPY INTO customers(CUSTOMER_ID, CUSTOMER_NAME, DATE_OF_BIRTH, LOADED_AT)
FROM(
SELECT $1, $2, $3,CONVERT_TIMEZONE('UTC', 'America/Chicago', CURRENT_TIMESTAMP())
FROM @my_s3_stage/customers.csv
)
FILE_FORMAT = (FORMAT_NAME = my_csv_format);

COPY INTO products(PRODUCT_ID,PRODUCT_NAME, PRICE, LOADED_AT)
FROM(
SELECT $1, $2, $3,CONVERT_TIMEZONE('UTC', 'America/Chicago', CURRENT_TIMESTAMP())
FROM @my_s3_stage/products.csv
)
FILE_FORMAT = (FORMAT_NAME = my_csv_format);
```

Step 4 : Now initialize the bronze layer tables by creating schema for them and then copy the data from staging layer tables to the bronze layer tables. Ensure that the data is de-duplicated(de-duplication can be achieved by using GROUP BY clause i.e by grouping all columns together(except for timestamp column)) and also ensure that if a customer record is deleted from the customer table in bronze layer then that customer related records must be removed from sales table in bronze layer. Now create a stored procedure for this whole thing and then schedule a task to run the stored procedure on daily basis. In bronze layer, we strictly maintain the updated data on daily basis i.e. this layer contains only very most recent data.

```
CREATE TABLE IF NOT EXISTS customers_bronze LIKE customers;

CREATE TABLE IF NOT EXISTS products_bronze LIKE products;

CREATE TABLE IF NOT EXISTS sales_bronze LIKE sales;

CREATE TABLE IF NOT EXISTS stores_bronze LIKE stores;

CREATE TABLE IF NOT EXISTS dimdates_bronze LIKE dimdates;
```

```
INSERT INTO customers_bronze
SELECT c.customer_id, c.customer_name, c.date_of_birth, MAX(c.LOADED_AT) AS loaded_at
FROM customers c
WHERE DATE(LOADED_AT) = (SELECT MAX(DATE(LOADED_AT)) FROM customers) AND NOT EXISTS (
    SELECT 1
    FROM customers_bronze cb
    WHERE cb.customer_id = c.customer_id
    AND cb.customer_name = c.customer_name
    AND cb.date_of_birth = c.date_of_birth
    AND cb.loaded_at = c.loaded_at
)
GROUP BY c.customer_id, c.customer_name, c.date_of_birth;

DELETE FROM customers_bronze
WHERE DATE(LOADED_AT) <> (SELECT MAX(DATE(LOADED_AT)) FROM customers);
```

Step 5 : Now initialize the silver layer tables by creating schema for them. In silver layer we have to implement type-2 SCD's which means we have to add three more new columns 'start_time', 'end_time' and 'current_flag_status' where 'start_time' represents when a record is loaded and 'end_time' represents when that record is deleted, by default it's value is '9999-12-31' which means that the record is not deleted yet and might get deleted in future and 'current_flag_status' represents either that record is active('True') or inactive('Inactive'). Type-2 SCD's typically used to show whether a record is active or inactive and the time period of that record. Moreover, in silver layer surrogate keys acts as primary keys instead of natural keys being primary keys. In order to produce surrogate keys we can use 'SEQUENCE' to generate unique surrogate keys or use 'UUID' function for a unique identifier. I have used 'SEQUENCE' to generate unique surrogate keys.

https://www.phdata.io/blog/implementing-slowly-changing-dimensions-in-snowflake/

```
--Silver tables initialisation
-- Sequence for Customers SCD Surrogate Key
CREATE SEQUENCE customer_scd_seq;

CREATE OR REPLACE TABLE customers_scd(
  "SURROGATE_KEY" NUMBER PRIMARY KEY DEFAULT customer_scd_seq.nextval,
  "CUSTOMER_ID" TEXT ,
  "CUSTOMER_NAME" TEXT,
  "DATE_OF_BIRTH" DATE,
  "start_time" TIMESTAMP_NTZ,
  "end_time" TIMESTAMP_NTZ,
  "current_flag_status" STRING
)
```

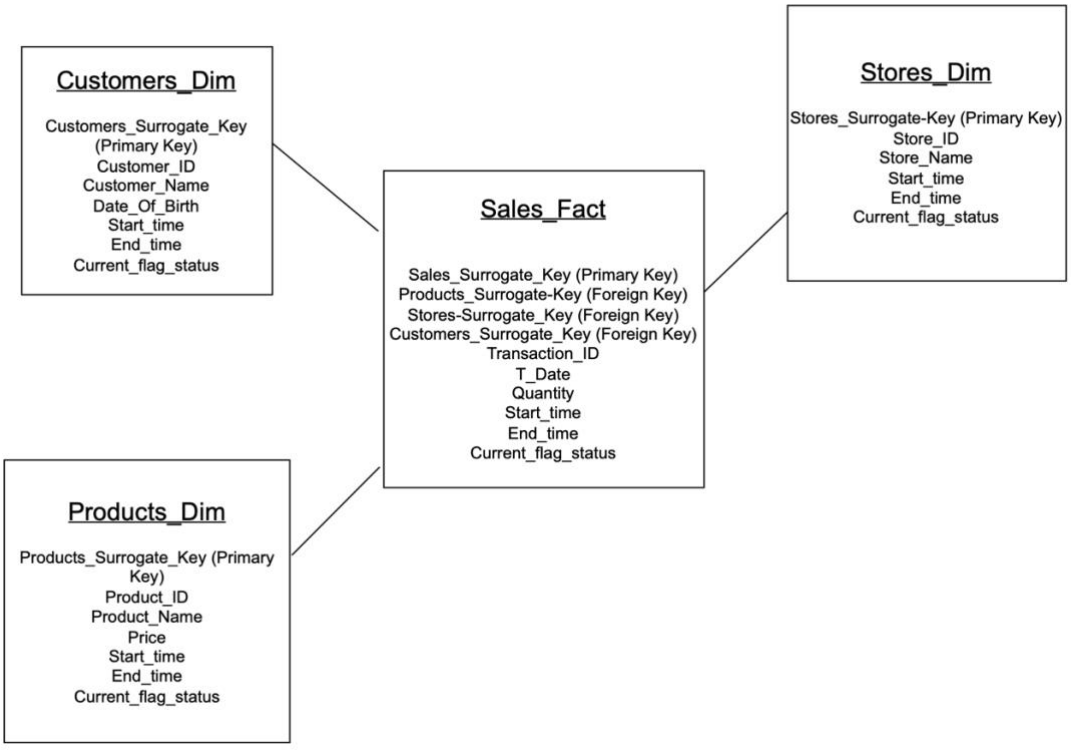

Fig : Silver Layer Modelling

Step 6 : In order to create type-2 SCD's we follow the following approach:

1. Initial Insertion: When a record is first inserted, it is marked as active. This typically involves setting a `current_flag_status` to 'True' and an `end_time` far in the future (like '9999-12-31') to signify that it is the current active record.

2. Deletion: If a record is deleted from the source, the typical Type 2 SCD response is not to physically delete the record in the data warehouse but to mark the existing record as inactive. This would involve setting the `current_flag_status` to 'False' and updating the `end_time` to the deletion timestamp, indicating that the record is no longer active as of that time.

3. Re-insertion: If the same record (identified by a unique key) is inserted again in the future:
 - A new row for this record would be inserted into the data warehouse to reflect the new active state.
 - This new row would typically have the `current_flag_status` set to 'True', a new `start_time` reflecting the insertion timestamp, and an `end_time` set far in the future to indicate it's the current version.
 - The previously inactivated row remains in the table with its `current_flag_status` set to 'False' and its `end_time` indicating when it was inactivated.

This approach allows the data warehouse to maintain a complete history of each record, including any periods during which it was not present in the source system. It's important to design the ETL process to differentiate between a true "new" insertion and a "re-activation" of a previously inactivated record, possibly by checking other fields for changes or maintaining a deletion log.

```
--Pushing data into Silver layer tables based on constraints
MERGE INTO customers_scd cscd
USING (SELECT * from customers_check) cchk
ON cscd.customer_id = cchk.customer_id
AND cscd.date_of_birth = cchk.date_of_birth
-- Handling updates or setting inactive for deleted records
WHEN MATCHED AND (cchk.metadata$action = 'DELETE')
THEN
UPDATE SET "end_time" = CONVERT_TIMEZONE('UTC', 'America/Chicago', CURRENT_TIMESTAMP()),
"current_flag_status" = 'False'
-- Handling reactivation of previously inactive records
WHEN MATCHED AND "current_flag_status" = 'False' AND (cchk.metadata$action = 'INSERT') THEN
UPDATE SET
"start_time" = CONVERT_TIMEZONE('UTC', 'America/Chicago', CURRENT_TIMESTAMP()),
"end_time" = '9999-12-31',
"current_flag_status" = 'True'
-- Handling new records inserts
WHEN NOT MATCHED AND (cchk.metadata$action='INSERT')
THEN
INSERT("CUSTOMER_ID", "CUSTOMER_NAME", "DATE_OF_BIRTH", "start_time", "end_time", "current_flag_status")
VALUES(cchk.customer_id, cchk.customer_name, cchk.date_of_birth, CONVERT_TIMEZONE('UTC', 'America/Chicago', CURRENT_TIMESTAMP()), '9999-12-
31', 'True' );
```

Step 7 : Now create a stored procedure for the above type-2 SCD creation process and then schedule it as a task to run on daily basis.

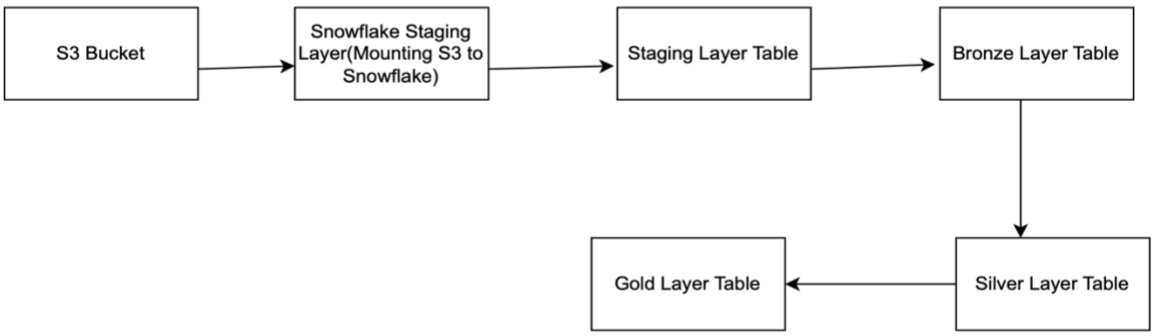

Fig : Flow Chart Of This Project