# 1. SQL DDL commands for Table Creation

CREATE TABLE Complaint (

Complaint\_id INT NOT NULL IDENTITY(1,1),

Complaint\_text VARCHAR(255) NOT NULL,

Complaint\_status VARCHAR(50) NOT NULL,

Filed\_date\_time DATETIME NOT NULL,

User\_id INT NOT NULL,

PRIMARY KEY(Complaint\_id)

);

Create table Complaints\_On\_Shop

(Complaint\_id int not null,

Shop\_id int not null,

Primary Key (Complaint\_id),

);

Create table Complaints\_On\_Restaurant

(Complaint\_id int not null,

Outlet\_id int not null,

Primary Key(Complaint\_id),

);

CREATE TABLE User\_Relation(

User\_id1 INT NOT NULL,

User\_id2 INT NOT NULL,

Relationship varchar(50) NOT NULL,

Primary Key(User\_id1, User\_id2)

);

Create table Shop\_Transaction

(User\_id int not null,

Shop\_id int not null,

Amount\_spent int not null,

Date\_time\_in datetime not null,

Date\_time\_out datetime not null,

Primary Key (User\_id, Shop\_id, Date\_time\_in)

);

Create table Restaurant\_Outlet

(Outlet\_id int not null,

Restaurant\_id int not null,

Mall\_id int not null,

Primary Key(Outlet\_id)

);

CREATE TABLE User\_Account (

User\_id INTEGER NOT NULL IDENTITY(1,1),

Username VARCHAR(255) NOT NULL,

Gender VARCHAR(1) NOT NULL,

DOB DATE NOT NULL,

PRIMARY KEY(User\_id)

);

CREATE TABLE Dine (

User\_id INTEGER NOT NULL,

Outlet\_id INTEGER NOT NULL,

Amount\_spent FLOAT NOT NULL,

Date\_time\_in DATETIME NOT NULL,

Date\_time\_out DATETIME NOT NULL,

PRIMARY KEY(User\_id,Outlet\_id,Date\_time\_in)

);

CREATE TABLE Shop (

Shop\_id INTEGER NOT NULL,

Category VARCHAR(128) NOT NULL,

Mall\_id INTEGER NOT NULL,

PRIMARY KEY(Shop\_id)

);

CREATE TABLE Voucher (

Voucher\_id INTEGER NOT NULL,

Details VARCHAR(255),

Date\_issued DATE NOT NULL,

Date\_expiring DATE NOT NULL,

PRIMARY KEY(Voucher\_id)

);

CREATE TABLE Purchase\_Voucher (

Voucher\_id INTEGER NOT NULL,

Purchase\_discount FLOAT NOT NULL,

User\_id INTEGER NOT NULL,

Date\_time\_used DATETIME,

PRIMARY KEY(Voucher\_id)

);

CREATE TABLE Dine\_Voucher (

Voucher\_id INTEGER NOT NULL,

Cash\_discount INTEGER NOT NULL,

User\_id INTEGER NOT NULL,

Date\_time\_used DATETIME NOT NULL,

PRIMARY KEY(Voucher\_id)

);

CREATE TABLE Group\_Voucher(

Voucher\_id INTEGER NOT NULL,

Group\_size INTEGER NOT NULL,

Group\_discount FLOAT NOT NULL,

Date\_time\_used DATETIME,

PRIMARY KEY (Voucher\_id)

);

CREATE TABLE Package\_Voucher(

Voucher\_id INTEGER NOT NULL,

Package\_discount FLOAT NOT NULL,

PRIMARY KEY(Voucher\_id)

);

CREATE TABLE Recommendation(

Recommendation\_id INTEGER NOT NULL,

Mall\_id INTEGER NOT NULL,

Outlet\_id INTEGER NOT NULL,

DayPackage\_id INTEGER NOT NULL,

Voucher\_id INTEGER NOT NULL,

Date\_expired DATE NOT NULL,

Date\_issued DATE NOT NULL,

PRIMARY KEY(Recommendation\_id)

);

CREATE TABLE UserAccount\_Use\_Recommendation(

User\_id INTEGER NOT NULL,

Recommendation\_id INTEGER NOT NULL,

PRIMARY KEY(User\_id,Recommendation\_id)

);

CREATE TABLE Mall(

Mall\_id INTEGER NOT NULL,

Company\_ID INTEGER NOT NULL,

Address\_of\_mall VARCHAR(255) NOT NULL,

NumShops INTEGER NOT NULL

);

CREATE TABLE Mall\_Has\_DayPackage(

DayPackage\_id INTEGER NOT NULL,

Mall\_id INTEGER NOT NULL,

PRIMARY KEY(DayPackage\_id, Mall\_id)

);

CREATE TABLE Day\_Package(

DayPackage\_id INTEGER NOT NULL,

User\_id INTEGER NOT NULL,

Voucher\_id INTEGER NOT NULL,

Description VARCHAR(255) NOT NULL,

PRIMARY KEY(DayPackage\_id)

);

CREATE TABLE Day\_Package\_Has\_RestaurantOutlet(

DayPackage\_id INTEGER NOT NULL,

Outlet\_id INTEGER NOT NULL,

PRIMARY KEY(DayPackage\_id,Outlet\_id)

);

CREATE TABLE Mall\_Mgmt\_Company(

Company\_id INTEGER NOT NULL,

Company\_Address VARCHAR(255) NOT NULL,

PRIMARY KEY(Company\_id)

);

CREATE TABLE Restaurant\_Chain(

RestaurantChain\_id INTEGER NOT NULL,

RestaurantChain\_Address VARCHAR(255) NOT NULL,

PRIMARY KEY(RestaurantChain\_id)

);

# 2.SQL statements to solve queries

* Each query should be immediately followed by the query output
* Explain how output is obtained

## Most popular day packages where all participants are related to one another as either family members or members of the same club

SELECT

DP.DayPackage\_id,

DP.Description,

COUNT(\*) AS ParticipantsCount

FROM

Day\_Package DP

INNER JOIN

User\_Relation UR1 ON DP.User\_id = UR1.User\_id1

INNER JOIN

User\_Relation UR2 ON DP.User\_id = UR2.User\_id2

WHERE

(UR1.Relationship IN ('Married', 'Parent-Child', 'Siblings') OR UR2.Relationship IN ('Married', 'Parent-Child', 'Siblings'))

OR

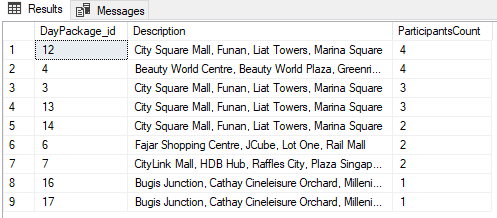
(UR1.Relationship = 'Clubmates' OR UR2.Relationship = 'Clubmates')

GROUP BY

DP.DayPackage\_id, DP.Description

ORDER BY

ParticipantsCount DESC;



Explanation:

* Three columns(DayPackage\_id, Description, Count(\*) AS ParticipantsCount) are selected from the Day\_Package table named DP
* Two inner joins are performed on two User Relations tables named UR1 and UR2, and the joins connect to DP based on the User\_id
* The WHERE clause filters the rows based on:

1. whether the relationship in UR1 or UR2 is either ‘Married’, ‘Siblings’ or ‘Parent-Child’
2. weather the relationship in UR1 or UR2 is Clubmates

* The GROUP BY clause groups the result by DayPackage\_id and Description so that COUNT can be applied to each group
* The ORDER BY clause orders the result by ParticipantsCount in descending order so that the most popular day packages, with the highest ParticipantCount, are shown at the top

## Families who frequently shopped and dined together with or without day packages

* indicate whether these families use day packages or not as part of your output
* frequently means at least 50% of the time

WITH f AS

( SELECT Dine.User\_id, Dine.Date\_time\_in, Dine.Date\_time\_out

FROM Dine

UNION

SELECT Shop\_Transaction.User\_id, Shop\_Transaction.Date\_time\_in, Shop\_Transaction.Date\_time\_out

FROM Shop\_Transaction

RIGHT JOIN User\_Relation ON User\_id = User\_Relation.User\_id1 OR User\_id = User\_Relation.User\_id2),

x AS

( SELECT User\_id, COUNT(\*) AS count1

FROM

( SELECT User\_id

FROM f

WHERE Date\_time\_in IN (

SELECT Date\_time\_in

FROM f

GROUP BY Date\_time\_in

HAVING COUNT(\*) > 1)) AS sub

GROUP BY User\_id),

y AS

( SELECT User\_id, COUNT(\*) AS count2

FROM

( SELECT User\_id

FROM f

WHERE Date\_time\_in IN (

SELECT Date\_time\_in

FROM f

GROUP BY Date\_time\_in

HAVING COUNT(\*) = 1)) AS sub

GROUP BY User\_id),

xy AS

( SELECT User\_id, count1, NULL as count2

FROM x

UNION

SELECT User\_id, NULL as count1, count2

FROM y)

SELECT sub.User\_id, User\_Account.Username,

CASE WHEN EXISTS (SELECT \*

FROM Day\_Package

WHERE sub.User\_id = Day\_Package.User\_id)

THEN 1 ELSE 0 END AS usedPackage

FROM

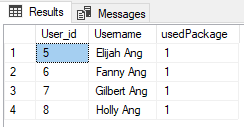
( SELECT User\_id, MAX(count1) AS count1, MAX(count2) AS count2

FROM xy

GROUP BY User\_id) AS sub

LEFT JOIN User\_Account ON User\_Account.User\_id = sub.User\_id

WHERE count1 >= count2



Explanation:

* First create temporary table f which contains every shop and dining transaction records by users who have a relation with another user,
* Then create temporary table x which counts the amount of transactions that have overlapping date\_time\_in for each user,
* As well as temporary table y which counts the amount of transactions that does not have overlapping date\_time\_in,
* Create table xy which is a union between the two,
* Lastly select user id, username as well as a case for whether the user has a day package or not from table xy, join with day\_package, then filter out users who have more transactions alone than in a group.

## Most popular recommendations from the app regarding malls

SELECT Recommendation.Recommendation\_id,Recommendation.Mall\_id, Recommendation.Outlet\_id,

Recommendation.DayPackage\_id, Recommendation.Voucher\_id, Recommendation.Date\_expired,

Recommendation.Date\_issued

FROM Recommendation

JOIN UserAccount\_Use\_Recommendation ON Recommendation.Recommendation\_id = UserAccount\_Use\_Recommendation.Recommendation\_id

JOIN Mall ON Recommendation.Mall\_id = Mall.Mall\_id

GROUP BY Recommendation.Recommendation\_id,Recommendation.Mall\_id, Recommendation.Outlet\_id,

Recommendation.DayPackage\_id, Recommendation.Voucher\_id, Recommendation.Date\_expired,

Recommendation.Date\_issued

HAVING COUNT(DISTINCT UserAccount\_Use\_Recommendation.User\_id) = (

SELECT MAX(num\_users)

FROM (

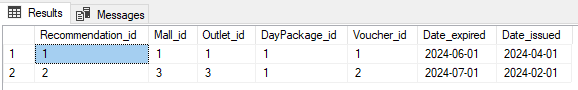
SELECT COUNT(DISTINCT User\_id) AS num\_users

FROM UserAccount\_Use\_Recommendation

GROUP BY Recommendation\_id

) AS subquery

)



Explanation:

* The select statement retrieves the desired columns from the recommendation table, which would have been joined with other tables, explained further below.
* The recommendation table is joined with UserAccount\_Use\_Recommendation based on the Recommendation\_id column, and the Mall table based on the Mall\_id column.
* The GROUP BY clause is used to allow us to filter the records by the HAVING clause.
* We filter grouped records to only include records whose count of distinct User\_id are equal to the maximum count of distinct user IDs across all the recommendation ids using the HAVING clause.
* Those grouped records mentioned above are obtained by the subquery that calculates the maximum count of distinct user IDs, grouped by Recommendation\_ids from the UserAccount\_Use\_Reccomendation table.
* In the output, only the most popular recommendation is shown. In this case, there are two recommendations with the max number of users.

## Youngest shoppers who have visited a certain mall more than 5 times during December 2023 and the amount they spent in total

/\*Find 3 youngest compulsive shoppers and amount spent each\*/

WITH Dec\_Transactions AS (

SELECT DISTINCT User\_id, SUM(Amount\_spent) AS Amount\_spent

FROM Shop\_Transaction, Shop

WHERE MONTH(Date\_time\_in)=12 AND YEAR(Date\_time\_in)=2023

AND Shop.Shop\_id = Shop\_Transaction.Shop\_id

GROUP BY User\_id, Mall\_id

HAVING COUNT(User\_id)>5

),

Compulsive\_Shoppers AS (

SELECT TOP 3 User\_Account.User\_id, Username, DOB

FROM Dec\_Transactions, User\_Account

WHERE Dec\_Transactions.User\_id = User\_Account.User\_id

GROUP BY User\_Account.User\_id, Username, DOB

ORDER BY DOB DESC

) SELECT Compulsive\_Shoppers.User\_id, Compulsive\_Shoppers.Username, SUM(Amount\_spent) AS Amount\_spent

FROM Compulsive\_Shoppers, Shop\_Transaction

WHERE Compulsive\_Shoppers.User\_id = Shop\_Transaction.User\_id

GROUP BY Compulsive\_Shoppers.User\_id, Compulsive\_Shoppers.Username



/\*Find amount spent by the top 3 compulsive shoppers in total, in December\*/

WITH Dec\_Transactions AS (

SELECT DISTINCT User\_id, SUM(Amount\_spent) AS Amount\_spent

FROM Shop\_Transaction, Shop

WHERE MONTH(Date\_time\_in)=12 AND YEAR(Date\_time\_in)=2023

AND Shop.Shop\_id = Shop\_Transaction.Shop\_id

GROUP BY User\_id, Mall\_id

HAVING COUNT(User\_id)>5

),

Compulsive\_Shoppers AS (

SELECT TOP 3 User\_Account.User\_id, Username, DOB

FROM Dec\_Transactions, User\_Account

WHERE Dec\_Transactions.User\_id = User\_Account.User\_id

GROUP BY User\_Account.User\_id, Username, DOB

ORDER BY DOB DESC

), Amount\_Spent\_In\_Dec AS (

SELECT Compulsive\_Shoppers.User\_id, Compulsive\_Shoppers.Username, SUM(Amount\_spent) AS Amount\_spent

FROM Compulsive\_Shoppers, Shop\_Transaction

WHERE Compulsive\_Shoppers.User\_id = Shop\_Transaction.User\_id

GROUP BY Compulsive\_Shoppers.User\_id, Compulsive\_Shoppers.Username

)SELECT SUM(Amount\_spent) AS Total\_spent

FROM Amount\_Spent\_In\_Dec;



Explanation:

* We retrieve transactions that occurred in December 2023, using MONTH(Date\_time\_in)=12 AND YEAR(Date\_time\_in)=2023
* Then, count the number times a particular mall was visited by joining shop and mall tables with Shop.Shop\_id = Shop\_Transaction.Shop\_id, and assigning the data to users
* A user belongs to the compulsive shopper category if they visit a particular mall more than 5 times as seen by COUNT(User\_id)>5
* Then, we retrieve information about the user's age by joining the DOB column Dec\_Transactions.User\_id = User\_Account.User\_id and getting the youngest users by sorting in descending order
* The amount spent reflected in the first output is the amount spent at the respective malls in which they visited more than 5 times and does not fully account for the total amount spent as transactions in other malls have not been accounted for
* Next, to get the total amount spent by the top 3 youngest compulsive shoppers in December 2023 across all malls, we summed the amount spent for each of these users from the shop transactions table using SELECT SUM(Amount\_spent) AS Total\_spent

FROM Amount\_Spent\_In\_Dec; therefore producing the second output

## Users who have dined in all the restaurants in some malls and never dined in any restaurants in some other malls

-- find total number of restaurants for each mall

WITH Total\_restaurants AS

(SELECT Mall\_id, COUNT(DISTINCT Outlet\_id) AS total\_resturants

FROM Restaurant\_Outlet

GROUP BY Mall\_id)

-- find number of unique restaurants each user has dined in for each mall

, User\_restaurants\_dined AS

(SELECT User\_id, Mall\_id, COUNT(DISTINCT Dine.Outlet\_id) AS restaurants\_dined\_in

FROM Dine JOIN Restaurant\_Outlet ON Dine.Outlet\_id = Restaurant\_Outlet.Outlet\_id

GROUP BY User\_id, Mall\_id)

-- compare counts for each user

SELECT ua.User\_id, ua.Username, COUNT(DISTINCT Dine.Outlet\_id) AS Outlets\_dined\_in

FROM User\_Account AS ua

JOIN Dine ON ua.User\_id = Dine.User\_id

JOIN Restaurant\_Outlet AS ro ON Dine.Outlet\_id = ro.Outlet\_id

JOIN User\_restaurants\_dined AS urd ON Dine.User\_id = urd.User\_id

JOIN Total\_restaurants AS tr ON ro.Mall\_id = tr.Mall\_id

GROUP BY ro.Mall\_id, ua.User\_id, ua.Username, restaurants\_dined\_in, total\_resturants

HAVING restaurants\_dined\_in = total\_resturants -- dined in all restaurants at a mall

AND ua.User\_id NOT IN (

SELECT ua.User\_id

FROM User\_Account AS ua

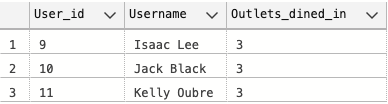
JOIN Dine ON ua.User\_id = Dine.User\_id

JOIN Restaurant\_Outlet AS ro ON Dine.Outlet\_id = ro.Outlet\_id

GROUP BY ua.User\_id

HAVING COUNT(DISTINCT ro.Mall\_id) > 1 -- dined in more than 1 mall

);



Explanation

* The query works by finding the total number of restaurants at each mall and the number of unique restaurants each user has dined in for each mall using the ‘COUNT(DISTINCT)’ aggregate function.
* The ‘HAVING’ clause is used to check if: the number of unique restaurants a user has dined in = the total number of restaurants at any malls.
* The subsequent ‘NOT IN’ clause checks for users who met the previous condition but also have dined in restaurants in other malls, and removes them from the result.

## Top 3 highest earning malls and restaurants

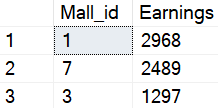
SELECT TOP 3 Mall\_id, SUM(Amount\_spent) AS Earnings

FROM Shop, Shop\_Transaction

WHERE Shop.Shop\_id = Shop\_Transaction.Shop\_id

GROUP BY Mall\_id

ORDER BY Earnings DESC;



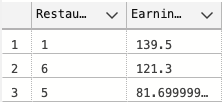
SELECT TOP 3 Restaurant\_id, SUM(Amount\_spent) AS Earnings

FROM Restaurant\_Outlet, Dine

WHERE Restaurant\_Outlet.Outlet\_id = Dine.Outlet\_id

GROUP BY Restaurant\_id

ORDER BY Earnings DESC;



Explanation

* The query works by using the ‘SUM’ aggregate function to get the total amount\_spent at each shop/restaurant.
* The ‘TOP 3’ keyword limits the output to only 3 results. Since the results are ordered by earnings in descending order, this yields the top 3 shops/restaurants with the highest earnings.
* To get the highest earning mall, the Shop table is joined in order to retrieve the shop’s Mall\_id.

## Shops that received the most complaints in December 2023

SELECT cs.Shop\_id, COUNT(\*) AS Number\_of\_complaints

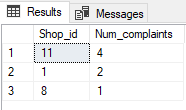
FROM Complaints\_On\_Shop AS cs

JOIN Complaint AS c ON cs.Complaint\_id = c.Complaint\_id

WHERE MONTH(c.Filed\_date\_time)=12 AND YEAR(c.Filed\_date\_time)=2023

GROUP BY cs.Shop\_id

ORDER BY Number\_of\_complaints DESC;

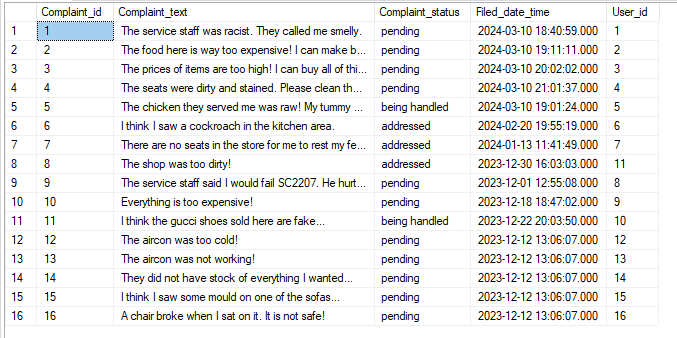


Explanation

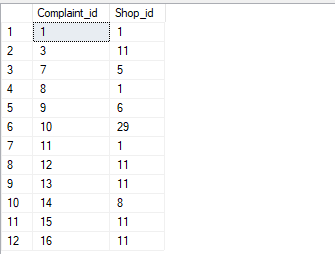
* Two columns(cs.Shop\_id, Count(\*) as Number\_of\_complaints) are selected
* The data is taken from two tables(Complaints\_On\_Shop, Complaint). Complaints\_On\_Shop is named ‘cs’ while Complaint is named ‘c’
* cs is then joined with c based on Complaint\_id
* The WHERE clause then filters the rows so that only complaints in December 2023 are obtained. The month and year is obtained using Filed\_date\_time from c.
* The GROUP BY clause causes the result to be grouped based on cs.Shop\_id so that COUNT can be applied to each group to get the number of complaints per shop
* Finally, the result is sorted by the Number\_of complaints in descending order so the shop with the most complaints is at the top

# 3.Printout of all Table Records

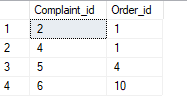
SELECT \* FROM Complaint;



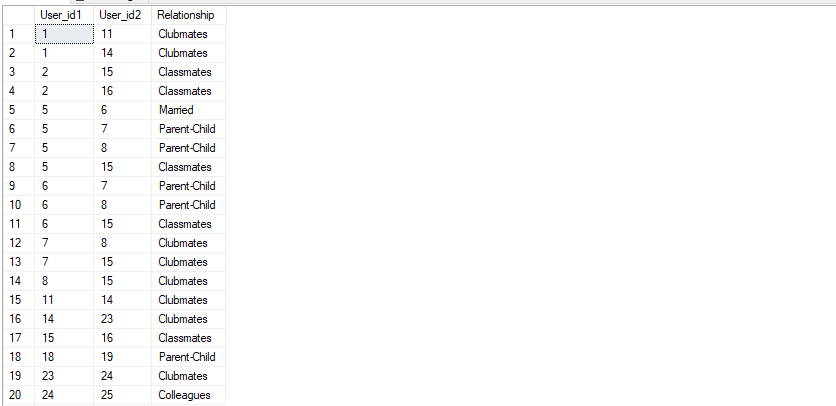
SELECT \* FROM Complaints\_On\_Shop;



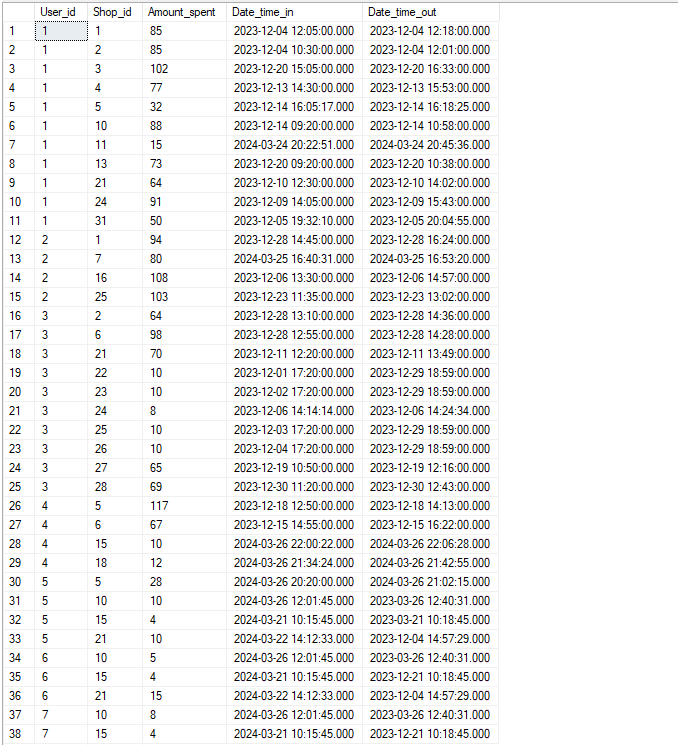
SELECT \* FROM Complaints\_On\_Restaurant;

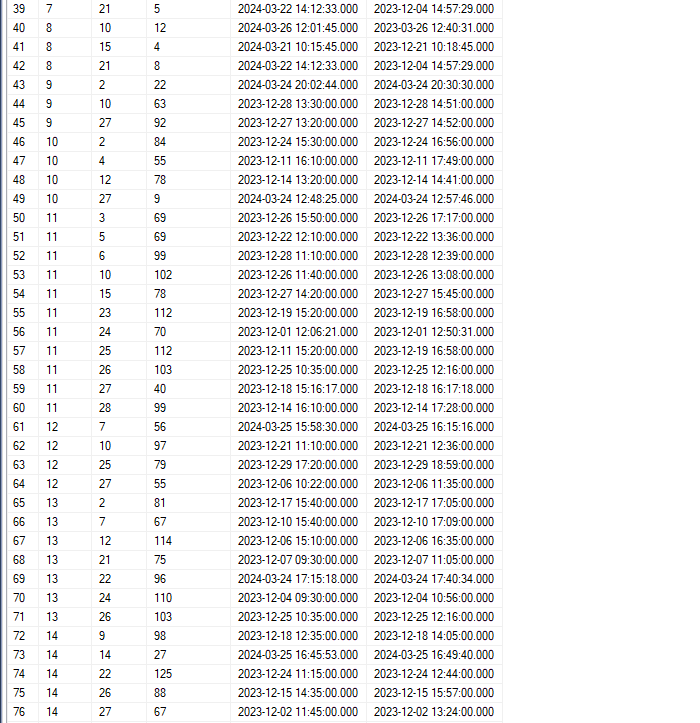


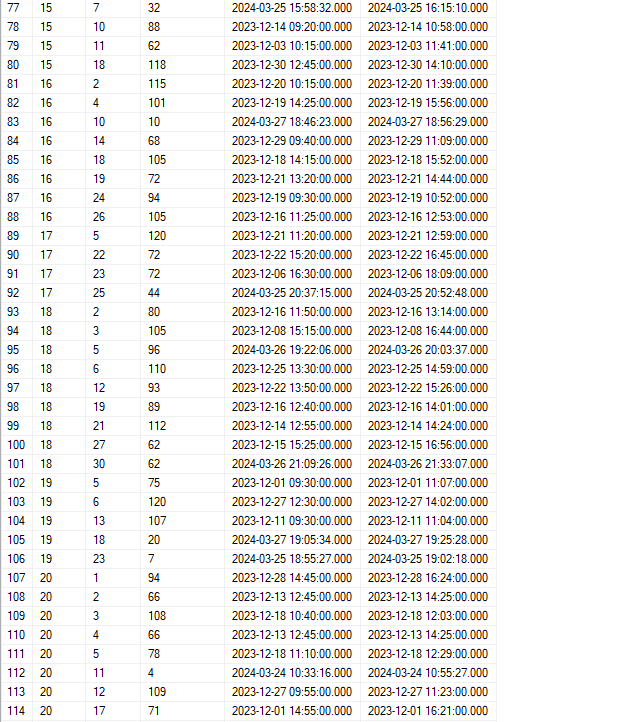
SELECT \* FROM User\_Relation;

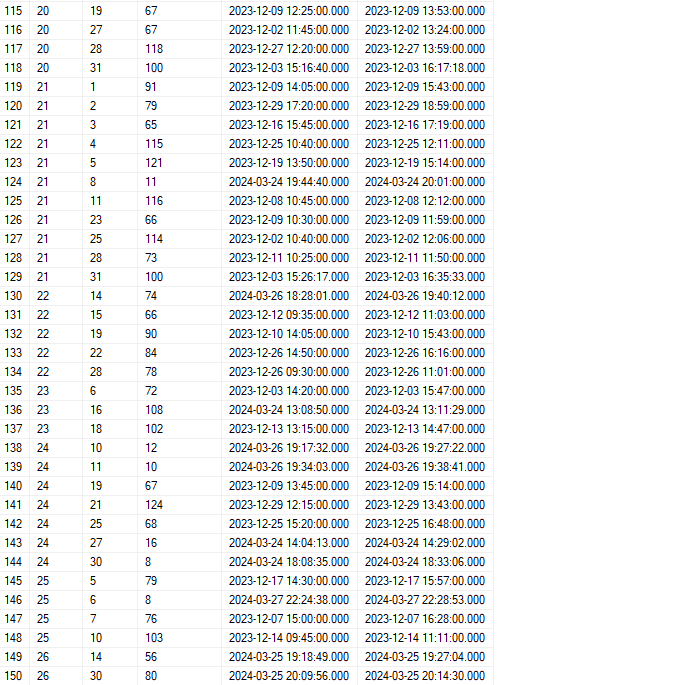


SELECT \* FROM Shop\_Transaction ;

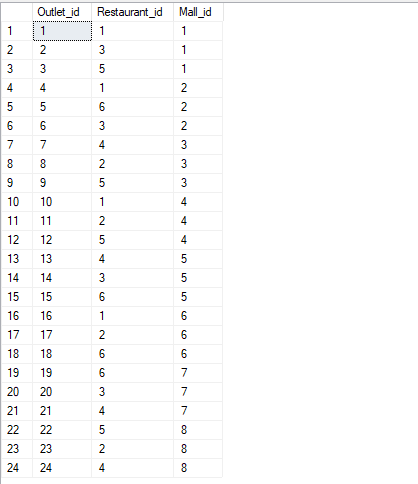




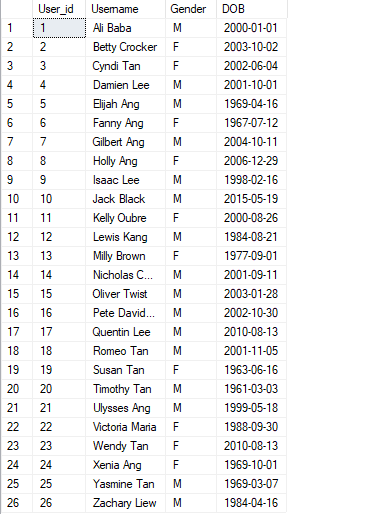




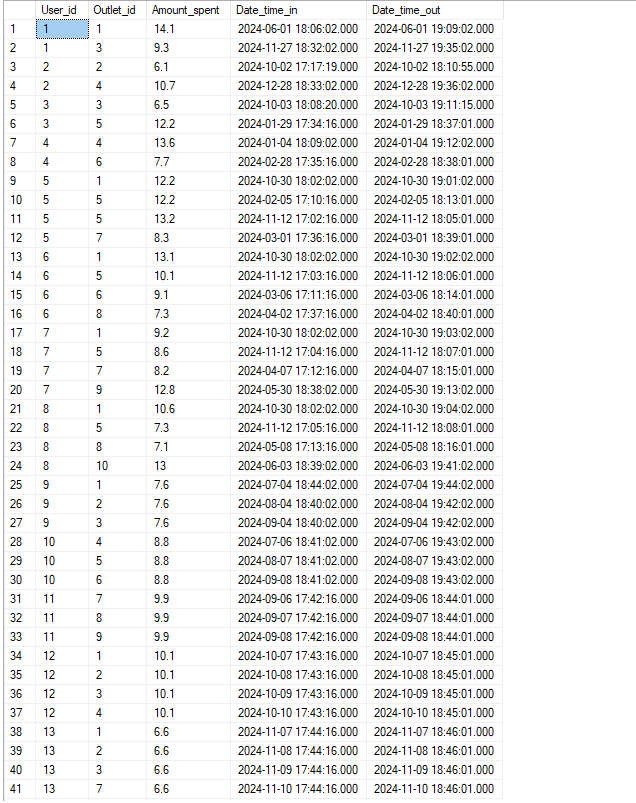
SELECT \* FROM Restaurant\_Outlet ;

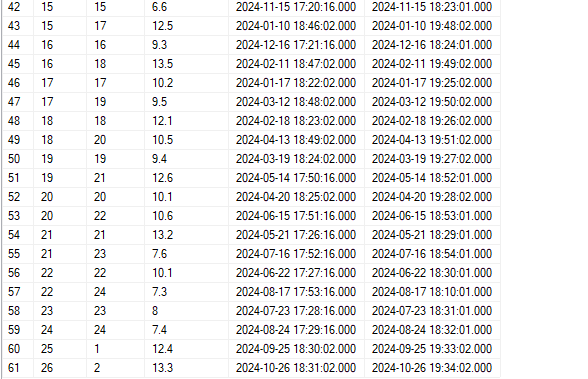


SELECT \* FROM User\_Account;

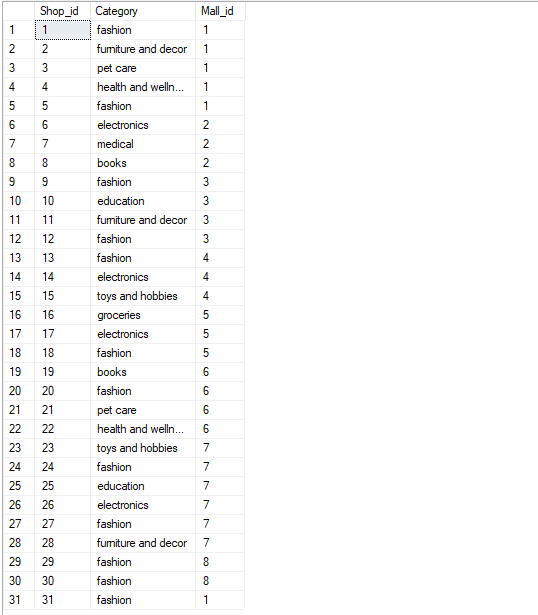


SELECT \* FROM Dine ;

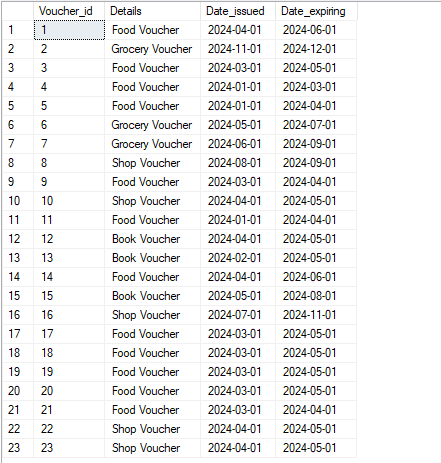




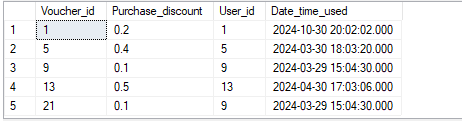
SELECT \* FROM Shop ;



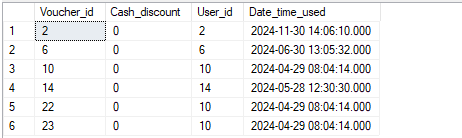
SELECT \* FROM Voucher ;



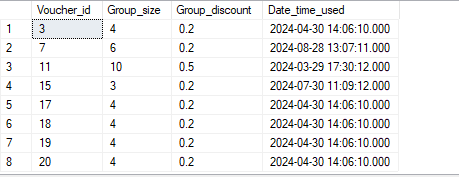
SELECT \* FROM Purchase\_Voucher ;



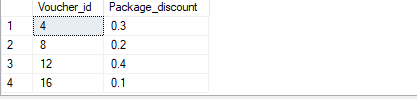
SELECT \* FROM Dine\_Voucher ;



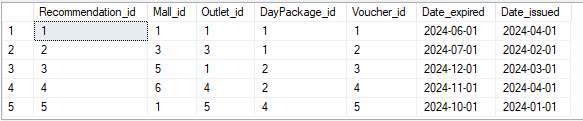
SELECT \* FROM Group\_Voucher;



SELECT \* FROM Package\_Voucher;



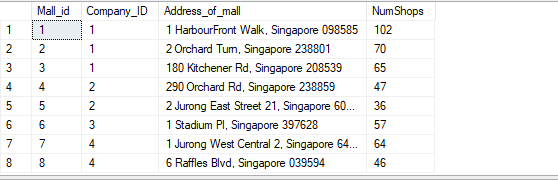
SELECT \* FROM Recommendation;



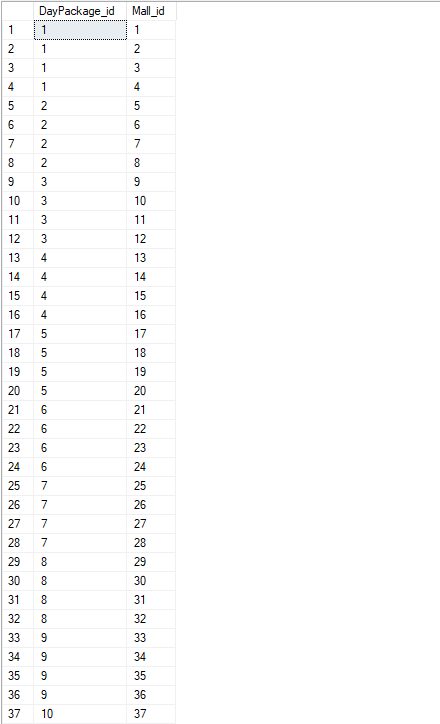
SELECT \* FROM UserAccount\_Use\_Recommendation;

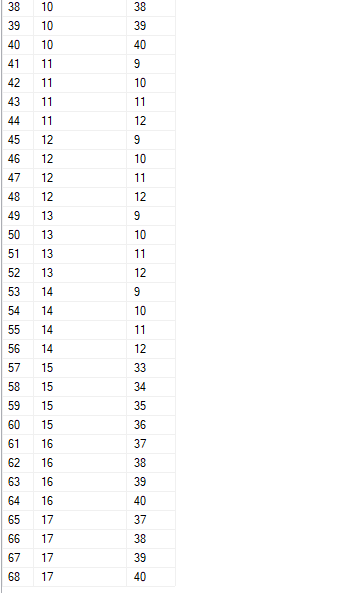


SELECT \* FROM Mall;

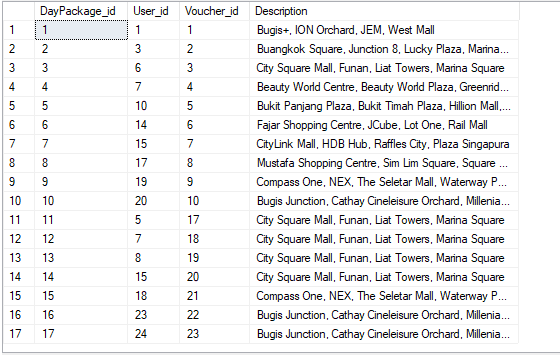


SELECT \* FROM Mall\_Has\_DayPackage;

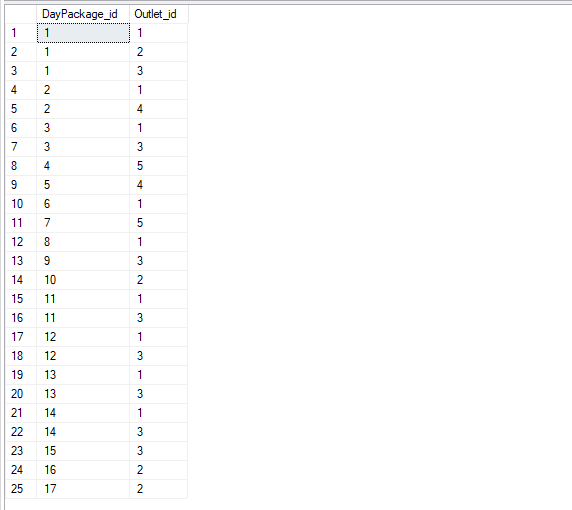




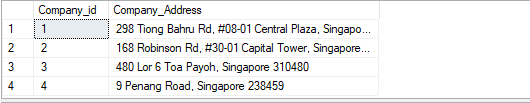
SELECT \* FROM Day\_Package;



SELECT \* FROM Day\_Package\_Has\_RestaurantOutlet;



SELECT \* FROM Mall\_Mgmt\_Company;



SELECT \* FROM Restaurant\_Chain;



# 4.Description of any additional efforts made(optional)

## Auto incrementing primary keys

Usage of keyword “IDENTITY(1,1)” to auto-increment primary keys. The parameters indicate the starting index and the number to increment by. This was used for some tables like User\_Account and Complaint