NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

SC2207: Introduction to Databases
Lab 5 Report

Name	Email	Matric Number
Pu Fanyi	FPU001@e.ntu.edu.sg	U2220175K
Jin Qingyang	JINQ0003@e.ntu.edu.sg	U2220239A
Tang Yutong	TANG0513@e.ntu.edu.sg	U2220495H
Ye Yuhan	YYE016@e.ntu.edu.sg	U2220885J
Ting Ruo Chee	RTING002@e.ntu.edu.sg	U2220572C
Soo Ying Xi	D220001@e.ntu.edu.sg	U2220021D
Qian Jianheng Oscar	QIAN0081@e.ntu.edu.sg	U2220109K

Tutorial Group: SCSD
Group Number: #5
Lab Supervisor: Sourav Saha Bhowmick
Teaching Assistant: Yu Weiping

School of Computer Science and Engineering
Nanyang Technological University

2023/2024 Semester 2

Table of Contents

SQL DDL Commands for Table Creation

```
-- Drop the database 'TestDB'
-- Connect to the 'master' database to run this snippet
USE master
GO
-- Uncomment the ALTER DATABASE statement below to set the database to SINGLE_USER mode if the drop database command fails because the database is in use.
ALTER DATABASE TestDB SET SINGLE_USER WITH ROLLBACK IMMEDIATE;
-- Drop the database if it exists
IF EXISTS (
  SELECT name
   FROM sys.databases
   WHERE name = N'TestDB'
)
DROP DATABASE TestDB
GO

-- Create a new database called 'TestDB'
-- Connect to the 'master' database to run this snippet
USE master
GO
-- Create the new database if it does not exist already
IF NOT EXISTS (
  SELECT name
    FROM sys.databases
    WHERE name = N'TestDB'
)
CREATE DATABASE TestDB
GO

USE TestDB
GO
-- Create a new table called 'user_account' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.user_account', 'U') IS NOT NULL
DROP TABLE dbo.user_account
GO
-- Create the table in the specified schema
CREATE TABLE dbo.user_account
(
  user_account_id INT IDENTITY(1,1) PRIMARY KEY, -- primary key column
  gender BIT NOT NULL, -- 0 for male, 1 for female
  dob DATE NOT NULL,
  name NVARCHAR(150) NOT NULL,
);
GO

-- Create a new table called 'mall_mgmt_company' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.mall_mgmt_company', 'U') IS NOT NULL
DROP TABLE dbo.mall_mgmt_company
GO
-- Create the table in the specified schema
CREATE TABLE dbo.mall_mgmt_company
(
  mall_mgmt_company_id INT IDENTITY(1,1) PRIMARY KEY, -- primary key
column
  address NVARCHAR(500) NOT NULL,
);
GO

-- Create a new table called 'mall' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.mall', 'U') IS NOT NULL
DROP TABLE dbo.mall
GO
-- Create the table in the specified schema
CREATE TABLE dbo.mall
(
  mall_id INT IDENTITY(1,1) PRIMARY KEY, -- primary key column
  address NVARCHAR(500) NOT NULL,
  num_shops INT NOT NULL,
  mall_mgmt_company_id INT NOT NULL,
  FOREIGN KEY (mall_mgmt_company_id) REFERENCES
mall_mgmt_company(mall_mgmt_company_id)
);
GO

-- Create a new table called 'restaurant_chain' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.restaurant_chain', 'U') IS NOT NULL
DROP TABLE dbo.restaurant_chain
GO
-- Create the table in the specified schema
CREATE TABLE dbo.restaurant_chain
(
  restaurant_chain_id INT IDENTITY(1,1) PRIMARY KEY, -- primary key
column
  address NVARCHAR(500) NOT NULL,
);
GO

-- Create a new table called 'voucher' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.voucher', 'U') IS NOT NULL
DROP TABLE dbo.voucher
GO
-- Create the table in the specified schema
CREATE TABLE dbo.voucher
(
  voucher_id INT IDENTITY(1,1) PRIMARY KEY, -- primary key column
  description NVARCHAR(500) NOT NULL,
  status VARCHAR(50) NOT NULL,
  date_issued DATE NOT NULL,
);
GO

-- Create a new table called 'purchase_voucher' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.purchase_voucher', 'U') IS NOT NULL
DROP TABLE dbo.purchase_voucher
GO
-- Create the table in the specified schema
CREATE TABLE dbo.purchase_voucher
(
  voucher_id INT NOT NULL PRIMARY KEY, -- primary key column
  purchase_discount INT NOT NULL, -- discount in percentage
  date_time DATETIME NOT NULL,
  user_account_id INT NOT NULL, -- foreign key column referencing
user_account table
  FOREIGN KEY (user_account_id) REFERENCES
user_account(user_account_id),
  FOREIGN KEY (voucher_id) REFERENCES voucher(voucher_id)
);
GO

-- Create a new table called 'dine_voucher' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.dine_voucher', 'U') IS NOT NULL
DROP TABLE dbo.dine_voucher
GO
-- Create the table in the specified schema
CREATE TABLE dbo.dine_voucher
(
  voucher_id INT NOT NULL PRIMARY KEY, -- primary key column
  cash_discount MONEY NOT NULL, -- discount in currency
  date_time DATETIME NOT NULL,
  user_account_id INT NOT NULL, -- foreign key column referencing
user_account table
  FOREIGN KEY (voucher_id) REFERENCES voucher(voucher_id),
  FOREIGN KEY (user_account_id) REFERENCES user_account(user_account_id)
);
GO

-- Create a new table called 'package_voucher' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.package_voucher', 'U') IS NOT NULL
DROP TABLE dbo.package_voucher
GO
-- Create the table in the specified schema
CREATE TABLE dbo.package_voucher
(
  voucher_id INT NOT NULL PRIMARY KEY, -- primary key column
  package_discount INT NOT NULL, -- discount in percentage
  FOREIGN KEY (voucher_id) REFERENCES voucher(voucher_id)
);
GO

-- Create a new table called 'group_voucher' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.group_voucher', 'U') IS NOT NULL
DROP TABLE dbo.group_voucher
GO
-- Create the table in the specified schema
CREATE TABLE dbo.group_voucher
(
  voucher_id INT NOT NULL PRIMARY KEY, -- primary key column
  group_size INT NOT NULL,
  group_discount INT NOT NULL, -- discount in percentage
  date_time DATETIME NOT NULL,
  user_account_id INT NOT NULL, -- foreign key column referencing
user_account table
  FOREIGN KEY (voucher_id) REFERENCES voucher(voucher_id),
  FOREIGN KEY (user_account_id) REFERENCES user_account(user_account_id)
);
GO

-- Create a new table called 'complaint' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.complaint', 'U') IS NOT NULL
DROP TABLE dbo.complaint
```

```
GO
-- Create the table in the specified schema
CREATE TABLE dbo.complaint
(
  complaint_id INT NOT NULL IDENTITY(1,1) PRIMARY KEY, -- primary key
column
  text NVARCHAR(500) NOT NULL,
  status VARCHAR(50) NOT NULL,
  filed_date_time DATETIME NOT NULL,
  user_account_id INT NOT NULL, -- foreign key column referencing
user_account table
  FOREIGN KEY (user_account_id) REFERENCES user_account(user_account_id)
);
GO

-- Create a new table called 'shop' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.shop', 'U') IS NOT NULL
DROP TABLE dbo.shop
GO
-- Create the table in the specified schema
CREATE TABLE dbo.shop
(
  shop_id INT NOT NULL IDENTITY(1,1) PRIMARY KEY, -- primary key column
  type VARCHAR(50) NOT NULL,
  mall_id INT NOT NULL, -- foreign key column referencing mall table
  FOREIGN KEY (mall_id) REFERENCES mall(mall_id)
);
GO

-- Create a new table called 'restaurant_outlet' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.restaurant_outlet', 'U') IS NOT NULL
DROP TABLE dbo.restaurant_outlet
GO
-- Create the table in the specified schema
CREATE TABLE dbo.restaurant_outlet
(
  restaurant_outlet_id INT NOT NULL PRIMARY KEY, -- primary key column
  mall_id INT NOT NULL, -- foreign key column referencing mall table
  restaurant_chain_id INT NOT NULL, -- foreign key column referencing
restaurant_chain table
  FOREIGN KEY (mall_id) REFERENCES mall(mall_id),
  FOREIGN KEY (restaurant_chain_id) REFERENCES
restaurant_chain(restaurant_chain_id)
);
```

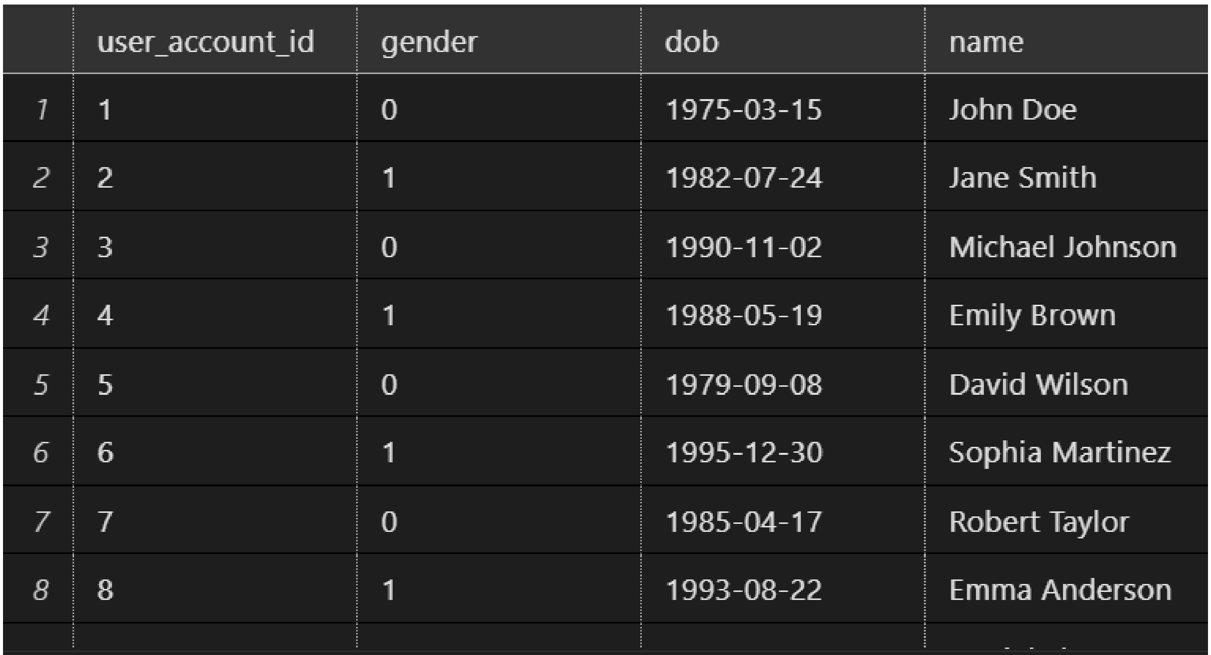
```
GO

-- Create a new table called 'day_package' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.day_package', 'U') IS NOT NULL
DROP TABLE dbo.day_package
GO
-- Create the table in the specified schema
CREATE TABLE dbo.day_package
(
  day_package_id INT NOT NULL IDENTITY(1,1) PRIMARY KEY, -- primary key column
  description NVARCHAR(500) NOT NULL,
  user_account_id INT NOT NULL, -- foreign key column referencing user_account table
  package_voucher_id INT NOT NULL, -- foreign key column referencing package_voucher table
  FOREIGN KEY (user_account_id) REFERENCES user_account(user_account_id),
  FOREIGN KEY (package_voucher_id) REFERENCES package_voucher(voucher_id)
);
GO

-- Create a new table called 'complaints_on_shop' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.complaints_on_shop', 'U') IS NOT NULL
DROP TABLE dbo.complaints_on_shop
GO
-- Create the table in the specified schema
CREATE TABLE dbo.complaints_on_shop
(
  complaint_id INT NOT NULL, -- composite primary key column
  shop_id INT NOT NULL, -- foreign key column referencing shop table
  FOREIGN KEY (complaint_id) REFERENCES complaint(complaint_id),
  FOREIGN KEY (shop_id) REFERENCES shop(shop_id)
);
GO

-- Create a new table called 'complaints_on_restaurant' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.complaints_on_restaurant', 'U') IS NOT NULL
DROP TABLE dbo.complaints_on_restaurant
GO
-- Create the table in the specified schema
CREATE TABLE dbo.complaints_on_restaurant
```

```
(
  complaint_id INT NOT NULL, -- composite primary key column
  restaurant_outlet_id INT NOT NULL, -- foreign key column referencing
restaurant_outlet table
  FOREIGN KEY (complaint_id) REFERENCES complaint(complaint_id),
  FOREIGN KEY (restaurant_outlet_id) REFERENCES
restaurant_outlet(restaurant_outlet_id)
);
GO

-- Create a new table called 'shop_record' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.shop_record', 'U') IS NOT NULL
DROP TABLE dbo.shop_record
GO
-- Create the table in the specified schema
CREATE TABLE dbo.shop_record
(
  shop_id INT NOT NULL, -- composite primary key column
  user_account_id INT NOT NULL, -- composite primary key column
  date_time_in DATETIME NOT NULL, -- composite primary key column
  amount_spent MONEY NOT NULL,
  date_time_out DATETIME NOT NULL,
  FOREIGN KEY (shop_id) REFERENCES shop(shop_id),
  FOREIGN KEY (user_account_id) REFERENCES
user_account(user_account_id),
  PRIMARY KEY (shop_id, user_account_id, date_time_in)
);
GO

-- Create a new table called 'dine_record' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.dine_record', 'U') IS NOT NULL
DROP TABLE dbo.dine_record
GO
-- Create the table in the specified schema
CREATE TABLE dbo.dine_record
(
  user_account_id INT NOT NULL, -- composite primary key column
  restaurant_outlet_id INT NOT NULL, -- composite primary key column
  date_time_in DATETIME NOT NULL, -- composite primary key column
  amount_spent MONEY NOT NULL,
  date_time_out DATETIME NOT NULL,
  FOREIGN KEY (user_account_id) REFERENCES
user_account(user_account_id),
  FOREIGN KEY (restaurant_outlet_id) REFERENCES
```

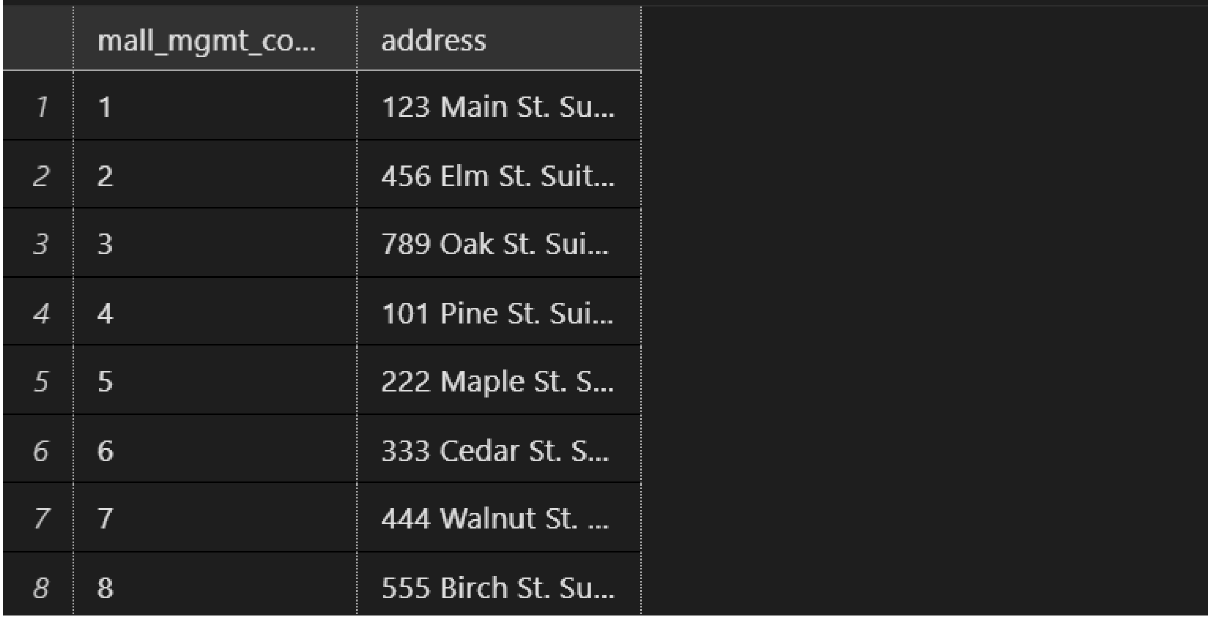
```sql
restaurant_outlet(restaurant_outlet_id),
  PRIMARY KEY (user_account_id, restaurant_outlet_id, date_time_in)
);
GO

-- Create a new table called 'user_relationship' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.user_relationship', 'U') IS NOT NULL
DROP TABLE dbo.user_relationship
GO
-- Create the table in the specified schema
CREATE TABLE dbo.user_relationship
(
  user_account_id1 INT NOT NULL, -- composite primary key column
  user_account_id2 INT NOT NULL, -- composite primary key column
  type VARCHAR(50) NOT NULL,
  PRIMARY KEY (user_account_id1, user_account_id2)
);
GO

-- Create a new table called 'recommendation' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.recommendation', 'U') IS NOT NULL
DROP TABLE dbo.recommendation
GO
-- Create the table in the specified schema
CREATE TABLE dbo.recommendation
(
  recommendation_id INT NOT NULL IDENTITY(1,1) PRIMARY KEY, -- primary
key column
  valid_period INT NOT NULL, -- in days
  date_issued DATE NOT NULL,
  user_account_id INT NOT NULL, -- foreign key column referencing
user_account table
  mall_id INT NOT NULL, -- foreign key column referencing mall table
  restaurant_outlet_id INT NOT NULL, -- foreign key column referencing
restaurant_outlet table
  day_package_id INT NOT NULL, -- foreign key column referencing
day_package table
  voucher_id INT NOT NULL, -- foreign key column referencing voucher
table
  FOREIGN KEY (user_account_id) REFERENCES
user_account(user_account_id),
  FOREIGN KEY (mall_id) REFERENCES mall(mall_id),
  FOREIGN KEY (restaurant_outlet_id) REFERENCES
restaurant_outlet(restaurant_outlet_id),
  FOREIGN KEY (day_package_id) REFERENCES day_package(day_package_id),
  FOREIGN KEY (voucher_id) REFERENCES voucher(voucher_id)
);
GO

-- Create a new table called 'user_use_recommendation' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.user_use_recommendation', 'U') IS NOT NULL
DROP TABLE dbo.user_use_recommendation
GO
-- Create the table in the specified schema
CREATE TABLE dbo.user_use_recommendation
(
  user_account_id INT NOT NULL, -- composite primary key column
  recommendation_id INT NOT NULL, -- composite primary key column
  FOREIGN KEY (user_account_id) REFERENCES
user_account(user_account_id),
  FOREIGN KEY (recommendation_id) REFERENCES
recommendation(recommendation_id),
  PRIMARY KEY (user_account_id, recommendation_id)
);
GO

-- Create a new table called 'restaurant_outlet_has_day_package' in
schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.restaurant_outlet_has_day_package', 'U') IS NOT NULL
DROP TABLE dbo.restaurant_outlet_has_day_package
GO
-- Create the table in the specified schema
CREATE TABLE dbo.restaurant_outlet_has_day_package
(
  day_package_id INT NOT NULL, -- composite primary key column
  restaurant_outlet_id INT NOT NULL, -- composite primary key column
  FOREIGN KEY (day_package_id) REFERENCES day_package(day_package_id),
  FOREIGN KEY (restaurant_outlet_id) REFERENCES
restaurant_outlet(restaurant_outlet_id),
  PRIMARY KEY (day_package_id, restaurant_outlet_id)
);
GO

-- Create a new table called 'mall_has_day_package' in schema 'dbo'
-- Drop the table if it already exists
IF OBJECT_ID('dbo.mall_has_day_package', 'U') IS NOT NULL
DROP TABLE dbo.mall_has_day_package
GO
```

```
-- Create the table in the specified schema
CREATE TABLE dbo.mall_has_day_package
(
  day_package_id INT NOT NULL, -- composite primary key column
  mall_id INT NOT NULL, -- composite primary key column
  FOREIGN KEY (day_package_id) REFERENCES day_package(day_package_id),
  FOREIGN KEY (mall_id) REFERENCES mall(mall_id),
  PRIMARY KEY (day_package_id, mall_id)
);
GO
```

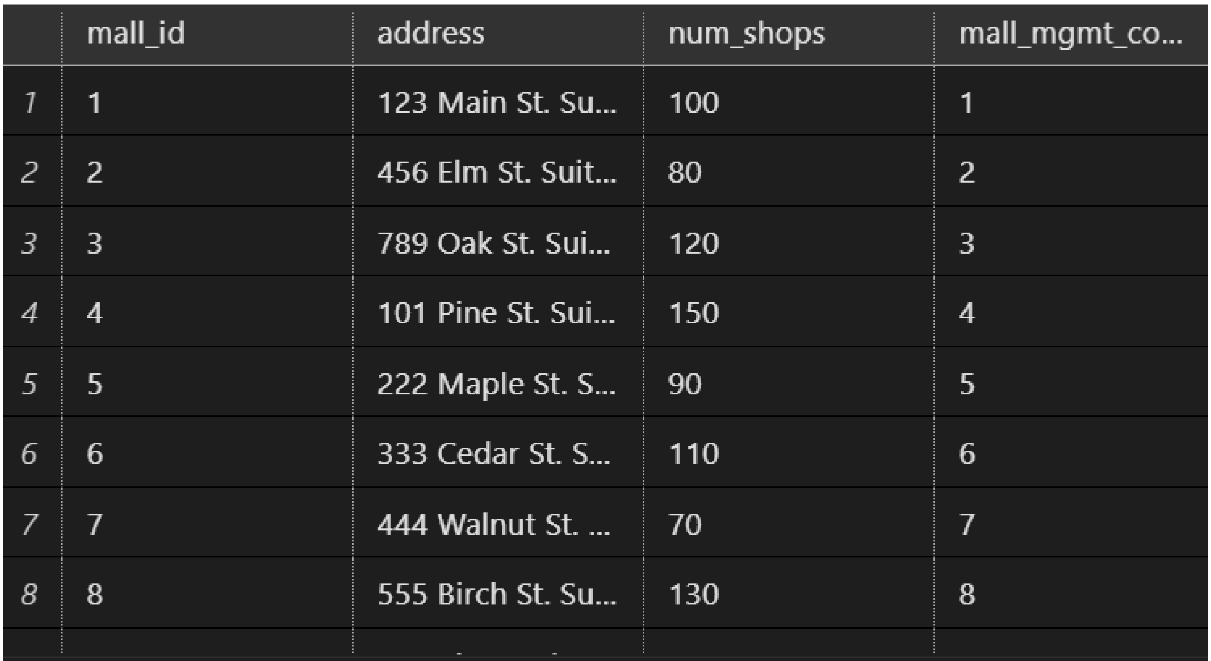
Table Records



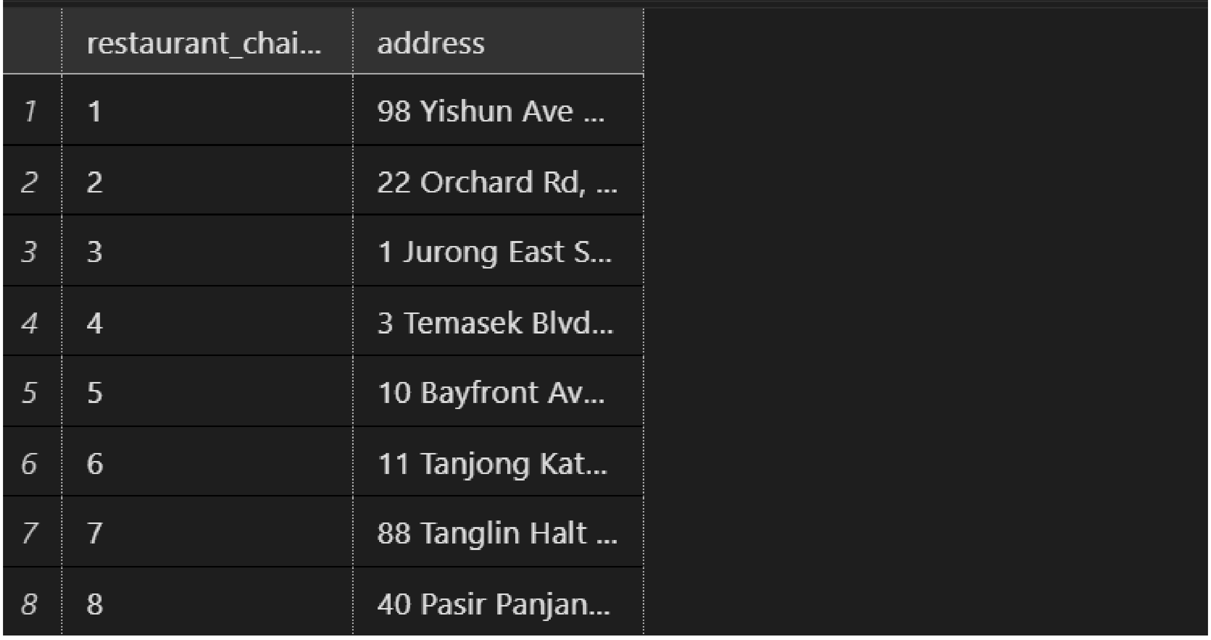
	user_account_id	gender	dob	name
1	1	0	1975-03-15	John Doe
2	2	1	1982-07-24	Jane Smith
3	3	0	1990-11-02	Michael Johnson
4	4	1	1988-05-19	Emily Brown
5	5	0	1979-09-08	David Wilson
6	6	1	1995-12-30	Sophia Martinez
7	7	0	1985-04-17	Robert Taylor
8	8	1	1993-08-22	Emma Anderson



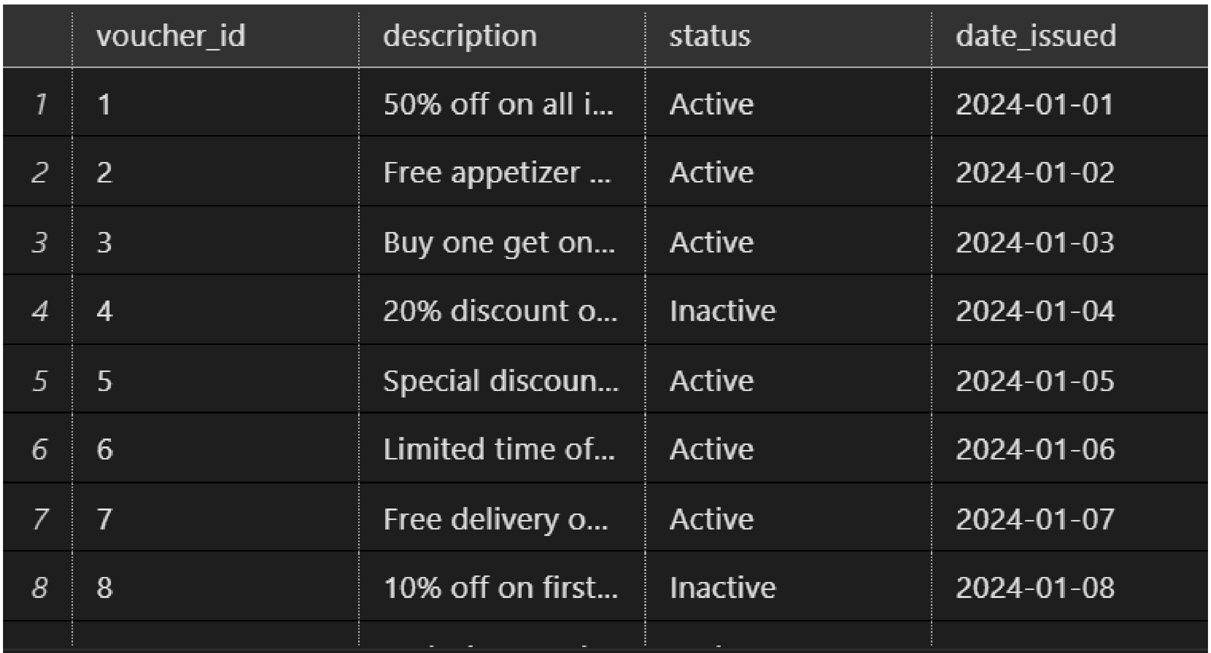
	mall_mgmt_co...	address
1	1	123 Main St. Su...
2	2	456 Elm St. Suit...
3	3	789 Oak St. Sui...
4	4	101 Pine St. Sui...
5	5	222 Maple St. S...
6	6	333 Cedar St. S...
7	7	444 Walnut St. ...
8	8	555 Birch St. Su...



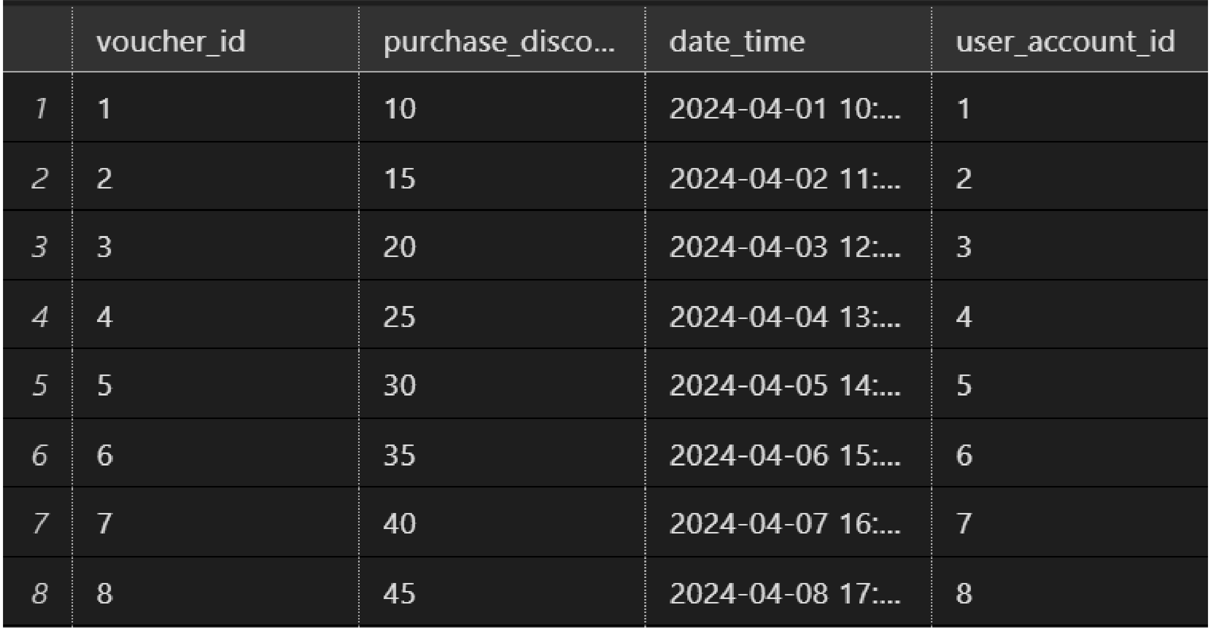
	mall_id	address	num_shops	mall_mgmt_co...
1	1	123 Main St. Su...	100	1
2	2	456 Elm St. Suit...	80	2
3	3	789 Oak St. Sui...	120	3
4	4	101 Pine St. Sui...	150	4
5	5	222 Maple St. S...	90	5
6	6	333 Cedar St. S...	110	6
7	7	444 Walnut St. ...	70	7
8	8	555 Birch St. Su...	130	8



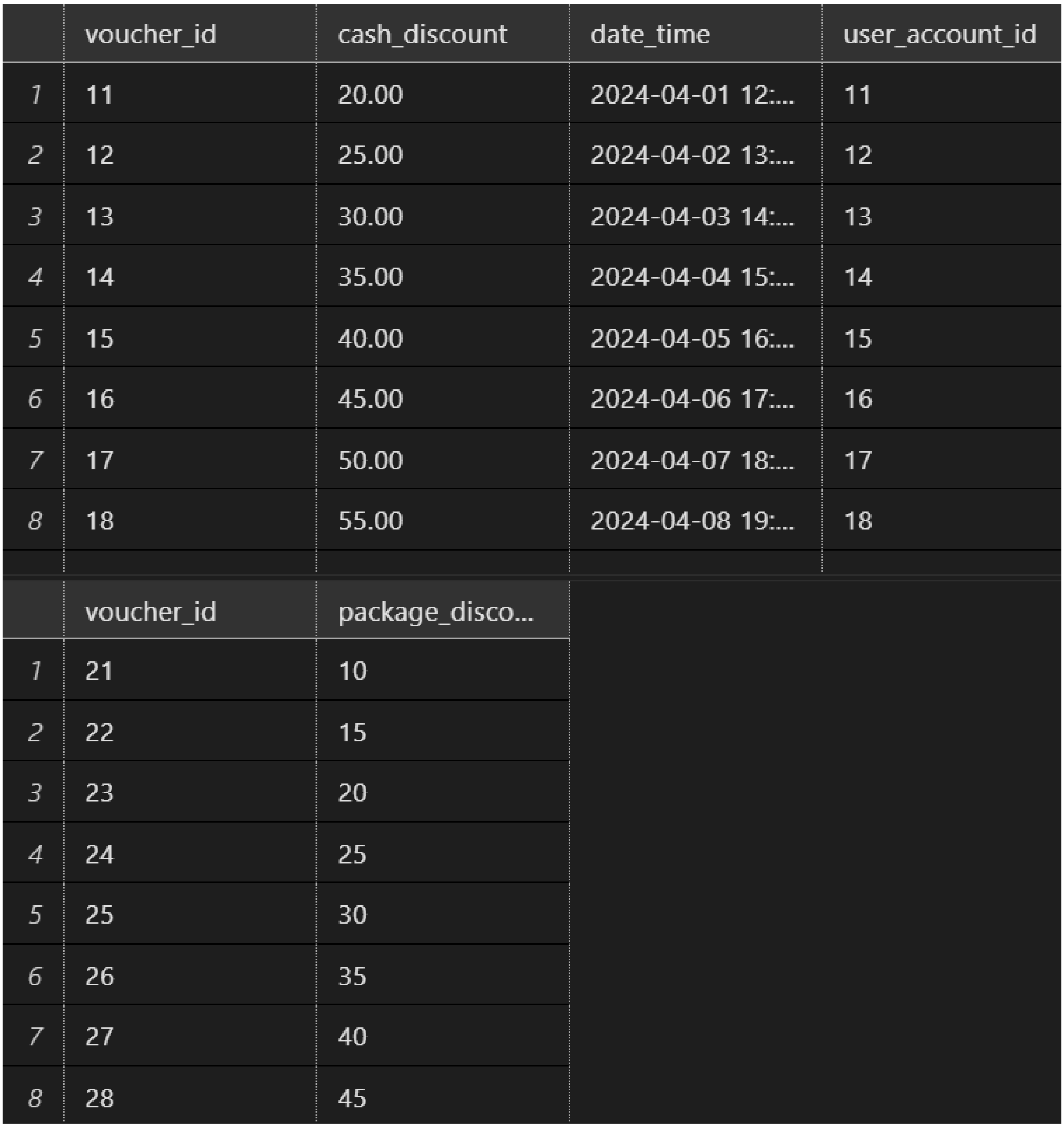
	restaurant_chai...	address
1	1	98 Yishun Ave ...
2	2	22 Orchard Rd, ...
3	3	1 Jurong East S...
4	4	3 Temasek Blvd...
5	5	10 Bayfront Av...
6	6	11 Tanjong Kat...
7	7	88 Tanglin Halt ...
8	8	40 Pasir Panjan...



	voucher_id	description	status	date_issued
1	1	50% off on all i...	Active	2024-01-01
2	2	Free appetizer ...	Active	2024-01-02
3	3	Buy one get on...	Active	2024-01-03
4	4	20% discount o...	Inactive	2024-01-04
5	5	Special discoun...	Active	2024-01-05
6	6	Limited time of...	Active	2024-01-06
7	7	Free delivery o...	Active	2024-01-07
8	8	10% off on first...	Inactive	2024-01-08

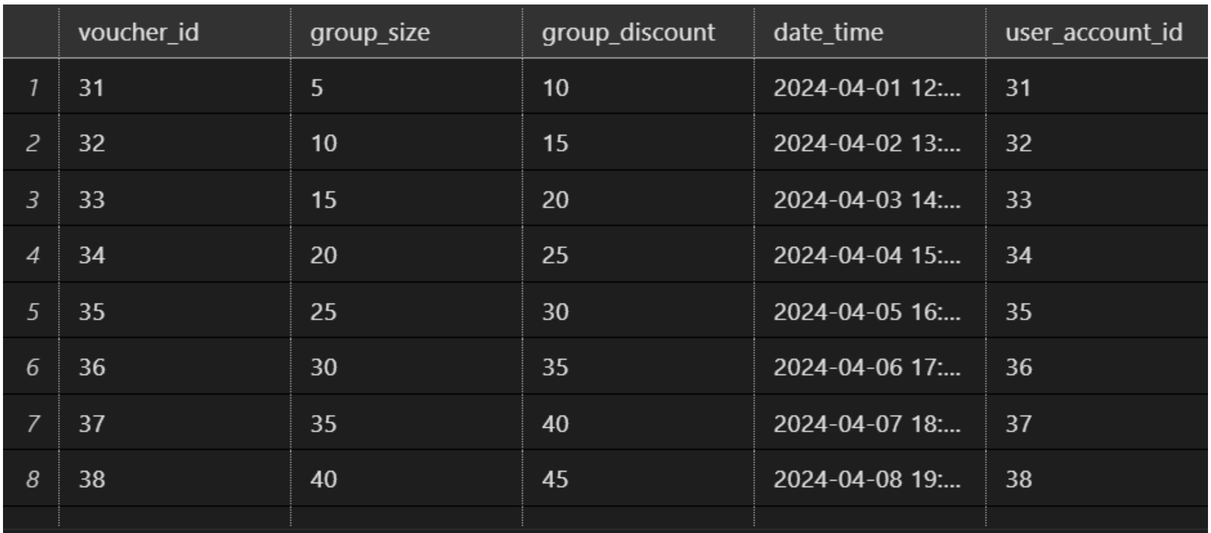


	voucher_id	purchase_disco...	date_time	user_account_id
1	1	10	2024-04-01 10:...	1
2	2	15	2024-04-02 11:...	2
3	3	20	2024-04-03 12:...	3
4	4	25	2024-04-04 13:...	4
5	5	30	2024-04-05 14:...	5
6	6	35	2024-04-06 15:...	6
7	7	40	2024-04-07 16:...	7
8	8	45	2024-04-08 17:...	8

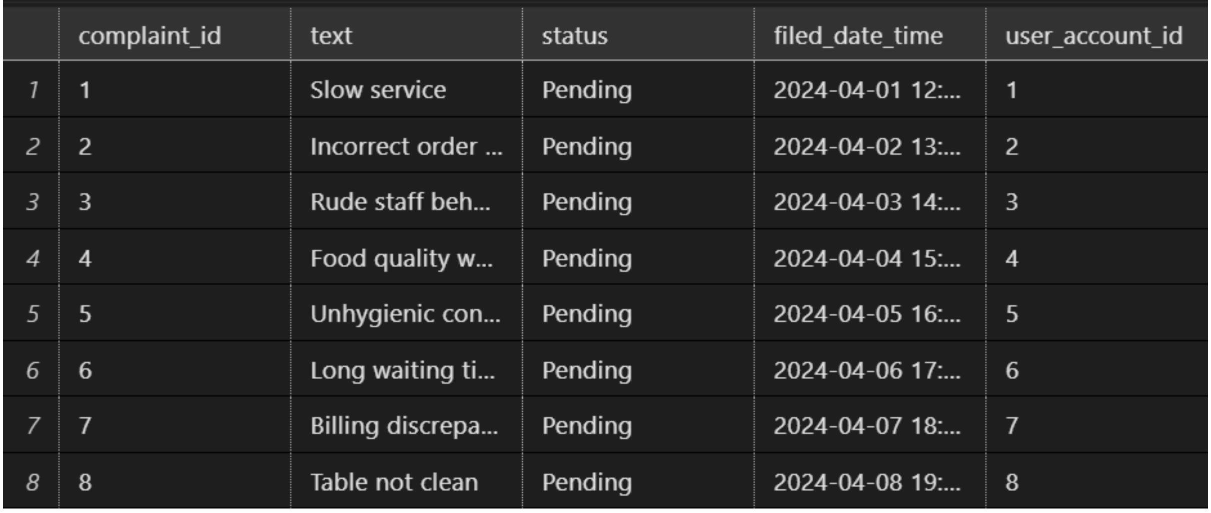


	voucher_id	cash_discount	date_time	user_account_id
1	11	20.00	2024-04-01 12:...	11
2	12	25.00	2024-04-02 13:...	12
3	13	30.00	2024-04-03 14:...	13
4	14	35.00	2024-04-04 15:...	14
5	15	40.00	2024-04-05 16:...	15
6	16	45.00	2024-04-06 17:...	16
7	17	50.00	2024-04-07 18:...	17
8	18	55.00	2024-04-08 19:...	18

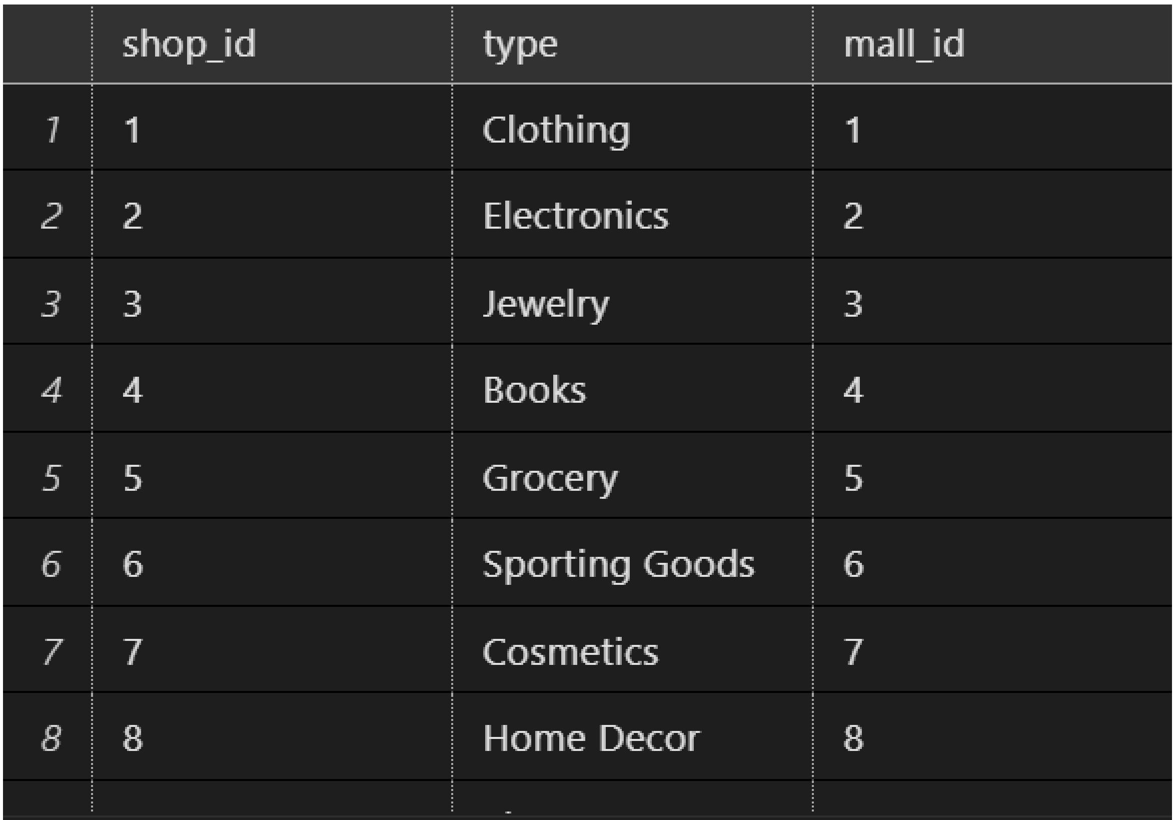
	voucher_id	package_disco...
1	21	10
2	22	15
3	23	20
4	24	25
5	25	30
6	26	35
7	27	40
8	28	45



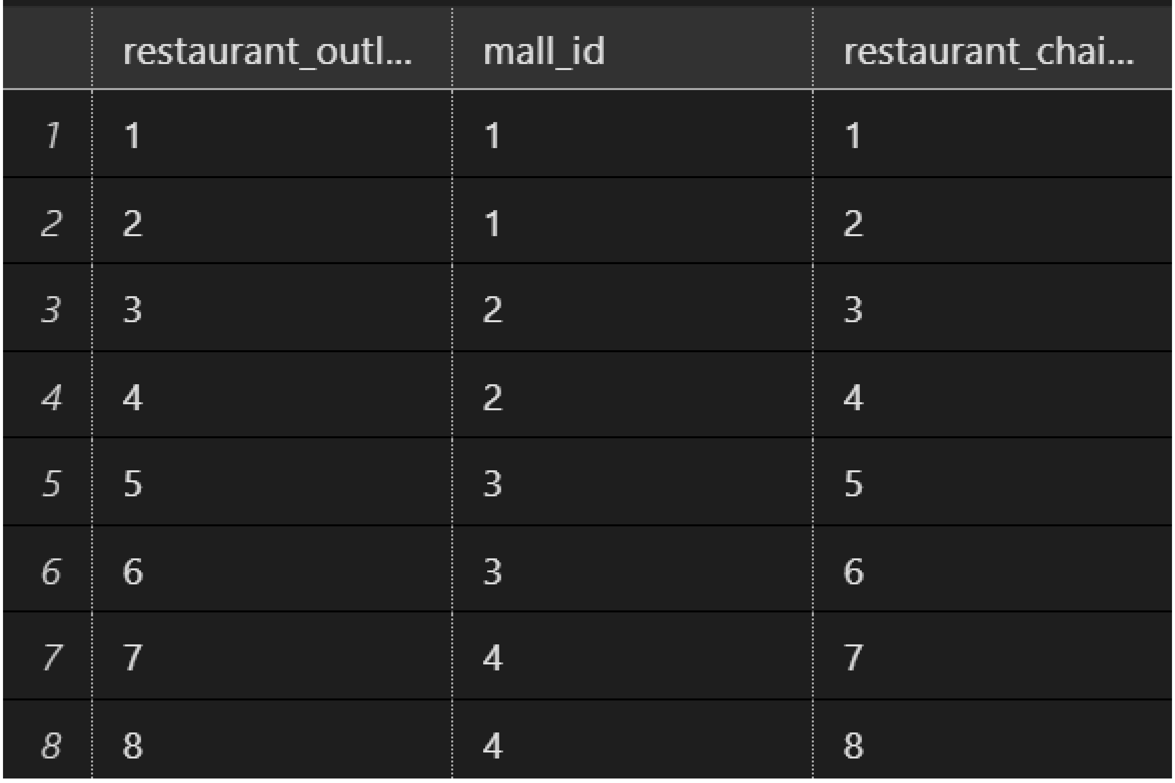
	voucher_id	group_size	group_discount	date_time	user_account_id
1	31	5	10	2024-04-01 12:...	31
2	32	10	15	2024-04-02 13:...	32
3	33	15	20	2024-04-03 14:...	33
4	34	20	25	2024-04-04 15:...	34
5	35	25	30	2024-04-05 16:...	35
6	36	30	35	2024-04-06 17:...	36
7	37	35	40	2024-04-07 18:...	37
8	38	40	45	2024-04-08 19:...	38



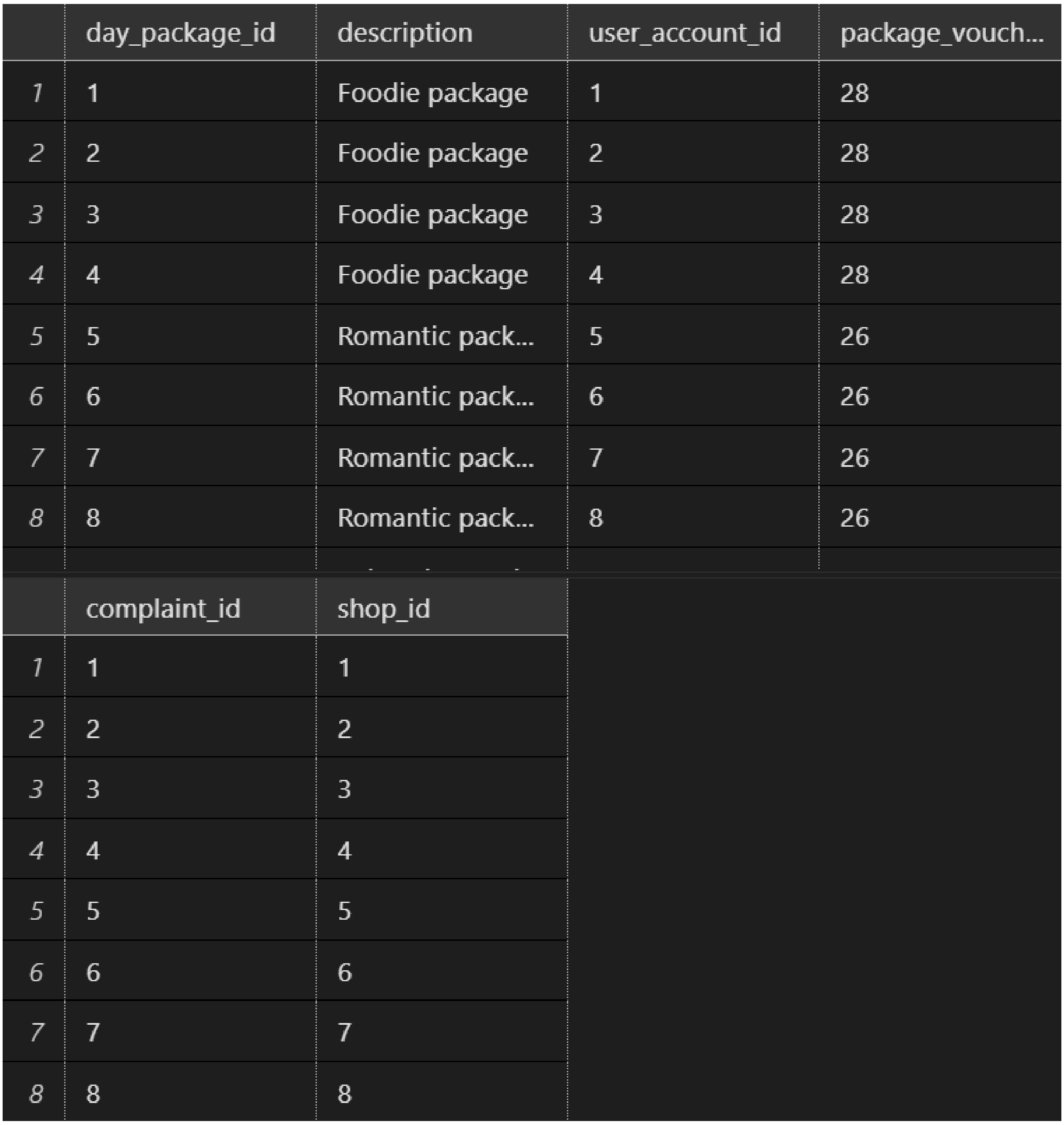
	complaint_id	text	status	filed_date_time	user_account_id
1	1	Slow service	Pending	2024-04-01 12:...	1
2	2	Incorrect order ...	Pending	2024-04-02 13:...	2
3	3	Rude staff beh...	Pending	2024-04-03 14:...	3
4	4	Food quality w...	Pending	2024-04-04 15:...	4
5	5	Unhygienic con...	Pending	2024-04-05 16:...	5
6	6	Long waiting ti...	Pending	2024-04-06 17:...	6
7	7	Billing discrepa...	Pending	2024-04-07 18:...	7
8	8	Table not clean	Pending	2024-04-08 19:...	8



	shop_id	type	mall_id
1	1	Clothing	1
2	2	Electronics	2
3	3	Jewelry	3
4	4	Books	4
5	5	Grocery	5
6	6	Sporting Goods	6
7	7	Cosmetics	7
8	8	Home Decor	8

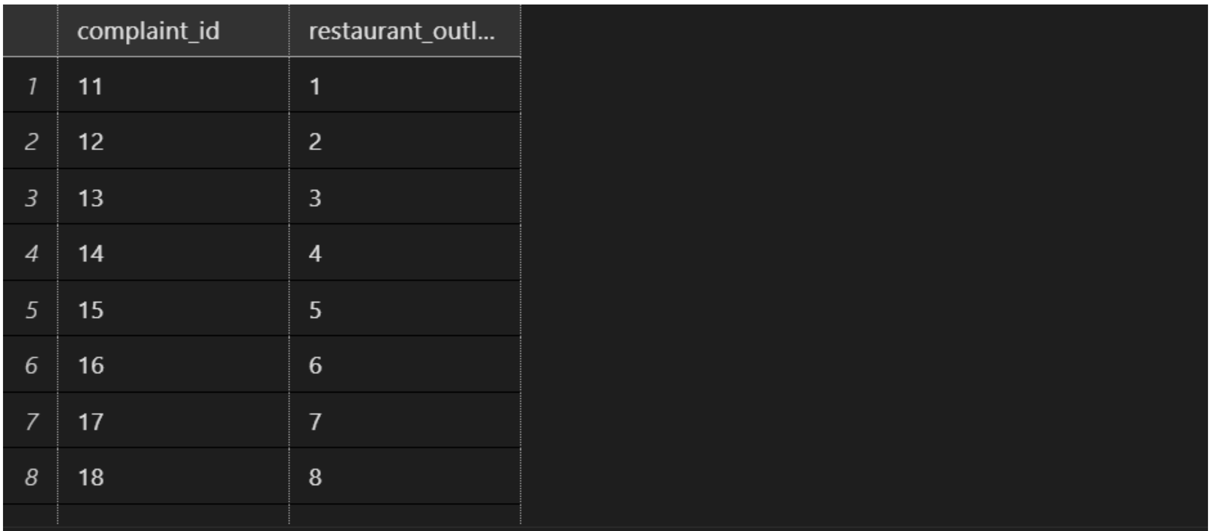


	restaurant_outl...	mall_id	restaurant_chai...
1	1	1	1
2	2	1	2
3	3	2	3
4	4	2	4
5	5	3	5
6	6	3	6
7	7	4	7
8	8	4	8

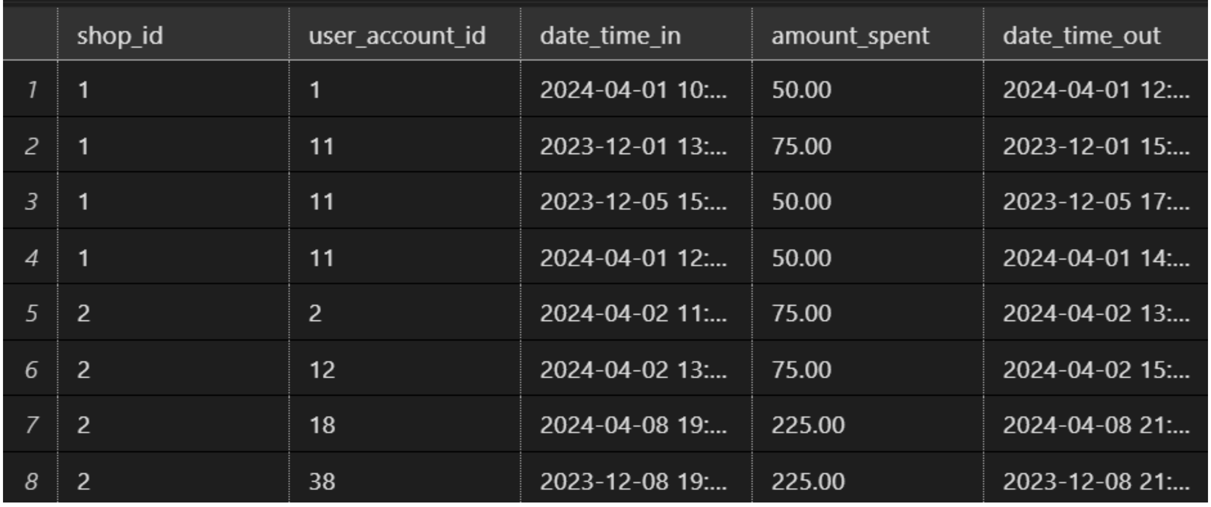


	day_package_id	description	user_account_id	package_vouch...
1	1	Foodie package	1	28
2	2	Foodie package	2	28
3	3	Foodie package	3	28
4	4	Foodie package	4	28
5	5	Romantic pack...	5	26
6	6	Romantic pack...	6	26
7	7	Romantic pack...	7	26
8	8	Romantic pack...	8	26

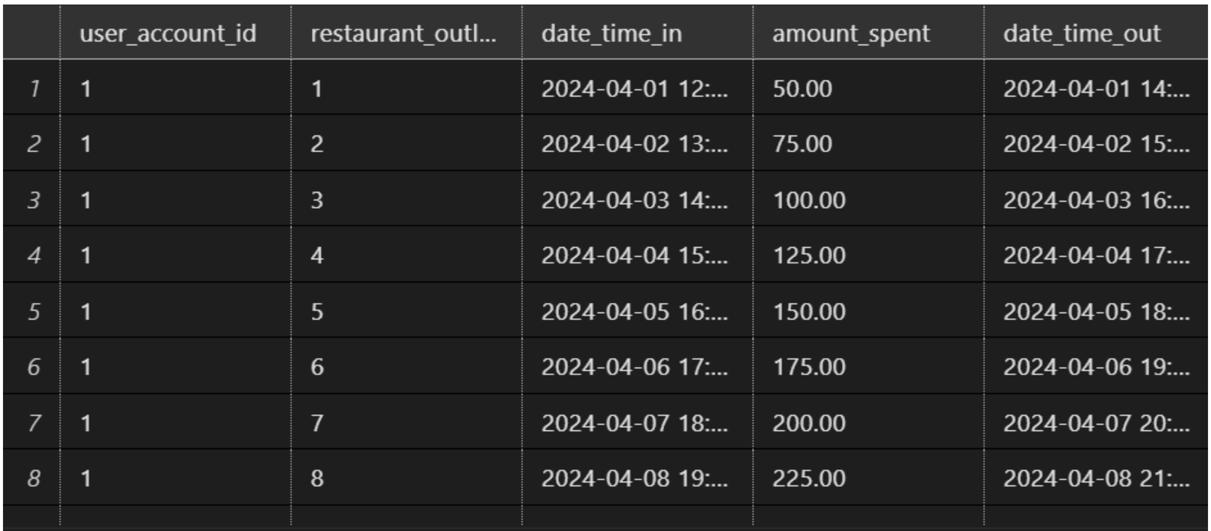
	complaint_id	shop_id
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8



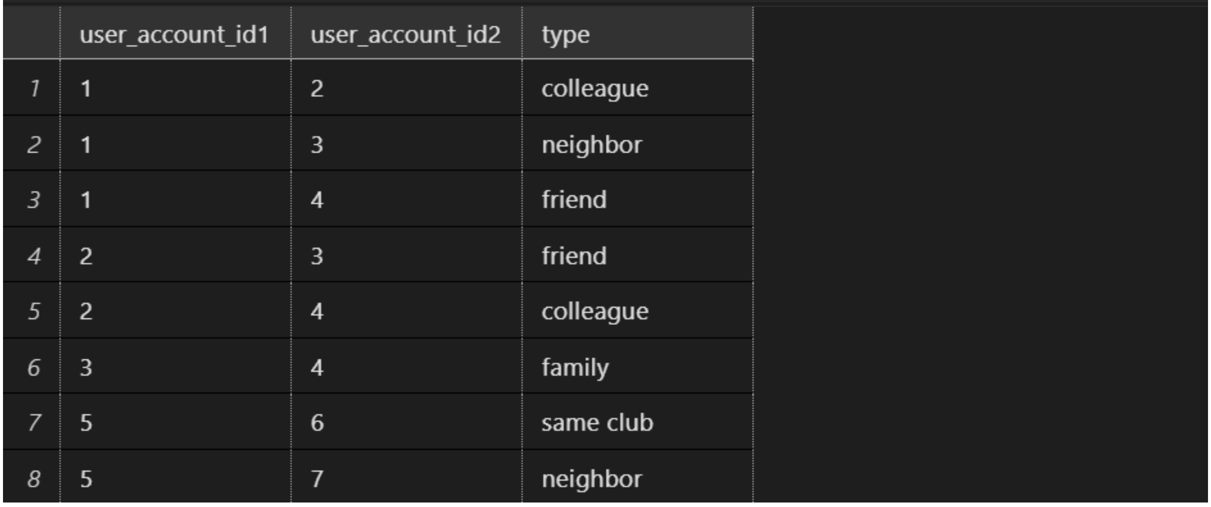
	complaint_id	restaurant_outl...
1	11	1
2	12	2
3	13	3
4	14	4
5	15	5
6	16	6
7	17	7
8	18	8



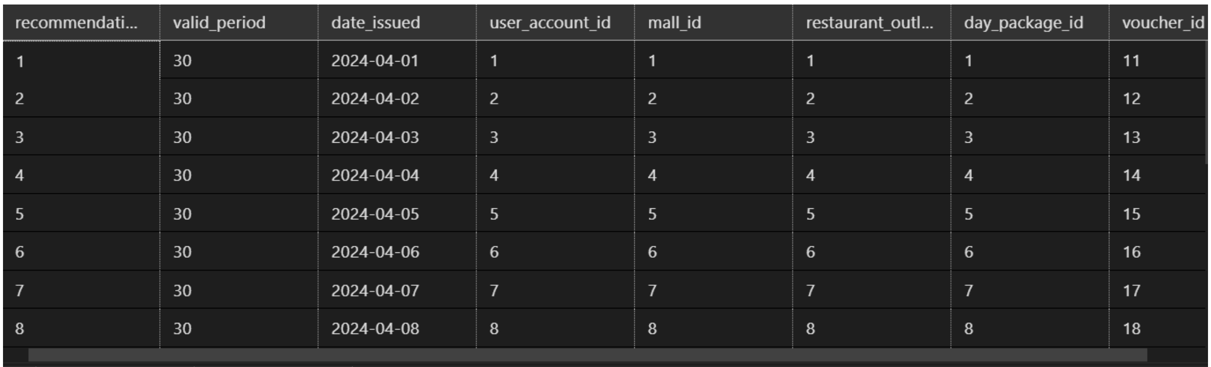
	shop_id	user_account_id	date_time_in	amount_spent	date_time_out
1	1	1	2024-04-01 10:...	50.00	2024-04-01 12:...
2	1	11	2023-12-01 13:...	75.00	2023-12-01 15:...
3	1	11	2023-12-05 15:...	50.00	2023-12-05 17:...
4	1	11	2024-04-01 12:...	50.00	2024-04-01 14:...
5	2	2	2024-04-02 11:...	75.00	2024-04-02 13:...
6	2	12	2024-04-02 13:...	75.00	2024-04-02 15:...
7	2	18	2024-04-08 19:...	225.00	2024-04-08 21:...
8	2	38	2023-12-08 19:...	225.00	2023-12-08 21:...



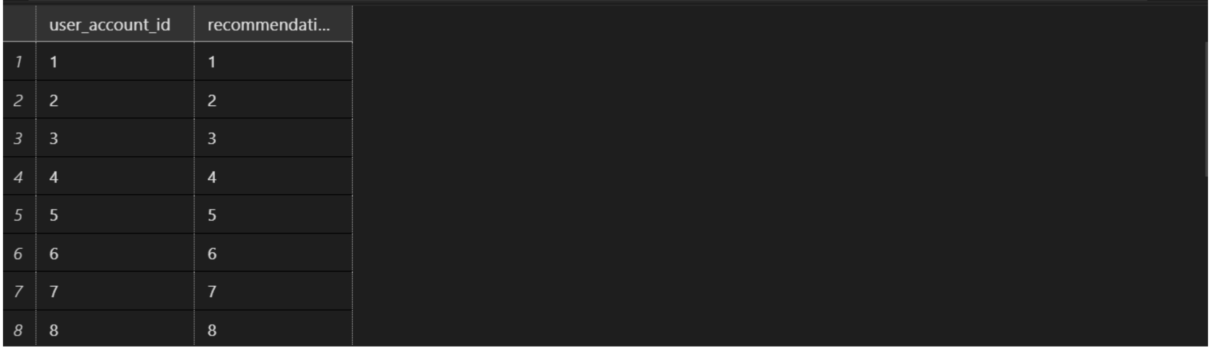
	user_account_id	restaurant_outl...	date_time_in	amount_spent	date_time_out
1	1	1	2024-04-01 12:...	50.00	2024-04-01 14:...
2	1	2	2024-04-02 13:...	75.00	2024-04-02 15:...
3	1	3	2024-04-03 14:...	100.00	2024-04-03 16:...
4	1	4	2024-04-04 15:...	125.00	2024-04-04 17:...
5	1	5	2024-04-05 16:...	150.00	2024-04-05 18:...
6	1	6	2024-04-06 17:...	175.00	2024-04-06 19:...
7	1	7	2024-04-07 18:...	200.00	2024-04-07 20:...
8	1	8	2024-04-08 19:...	225.00	2024-04-08 21:...



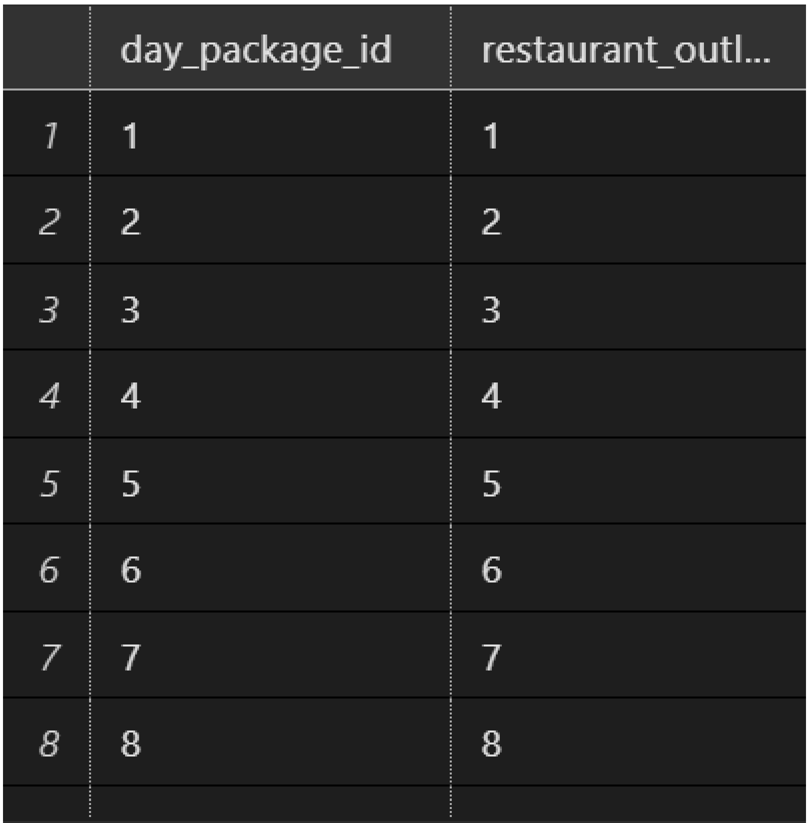
	user_account_id1	user_account_id2	type
1	1	2	colleague
2	1	3	neighbor
3	1	4	friend
4	2	3	friend
5	2	4	colleague
6	3	4	family
7	5	6	same club
8	5	7	neighbor



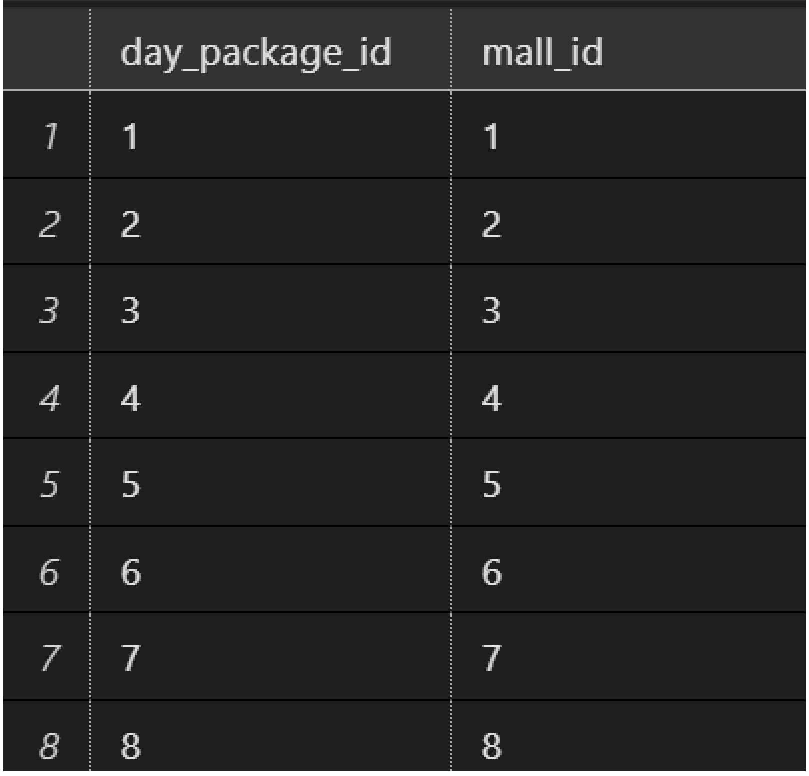
recommendati...	valid_period	date_issued	user_account_id	mall_id	restaurant_outl...	day_package_id	voucher_id
1	30	2024-04-01	1	1	1	1	11
2	30	2024-04-02	2	2	2	2	12
3	30	2024-04-03	3	3	3	3	13
4	30	2024-04-04	4	4	4	4	14
5	30	2024-04-05	5	5	5	5	15
6	30	2024-04-06	6	6	6	6	16
7	30	2024-04-07	7	7	7	7	17
8	30	2024-04-08	8	8	8	8	18



	user_account_id	recommendati...
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8



	day_package_id	restaurant_outl...
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8



	day_package_id	mall_id
1	1	1
2	2	2
3	3	3
4	4	4
5	5	5
6	6	6
7	7	7
8	8	8

SQL Queries

Query 1

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

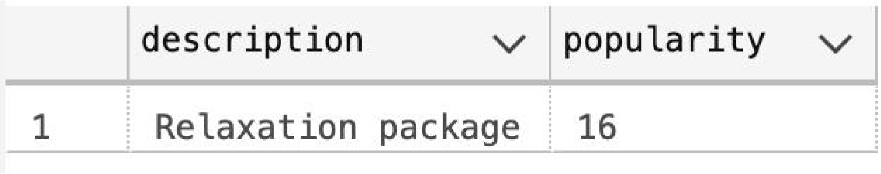
Explanation:

- The query starts by selecting the top 1 record (the most popular) and retrieves the description of the day package and the count of participants (popularity). It uses a subquery with the EXISTS operator to filter the day packages based on the existence of certain conditions.
- The subquery joins the day_package table with itself twice (dp1 and dp2) on the condition that the descriptions are the same but the day_package_id is different, to avoid the same record being joined. It also joins the user_relationship table to ensure that the participants (user_account_id) in dp1 and dp2 are related to each other either as family members or members of the same club.
- The query groups the results by the description of dp1 and then groups the outer query results by the description of the day_package table to calculate the total popularity. Finally, it orders the results by popularity in descending order to find the most popular day package.

Final Code:

```
USE TestDB
GO

-- Q1. What is the most popular day packages where all participants are
related to one another as either family members or members of the same
club?

SELECT TOP 1 dp.description, COUNT(*) AS popularity
FROM day_package dp
WHERE EXISTS (
    SELECT 1
    FROM day_package dp1
    JOIN day_package dp2 ON dp1.description = dp2.description AND
dp1.day_package_id <> dp2.day_package_id
    JOIN user_relationship ur ON (ur.user_account_id1 =
dp1.user_account_id AND ur.user_account_id2 = dp2.user_account_id)
    WHERE dp1.description = dp.description AND ur.type IN ('family',
'same club')
    GROUP BY dp1.description
)
GROUP BY dp.description
ORDER BY popularity DESC;
```

Output:



	description	popularity
1	Relaxation package	16

Query 2

Find families who frequently shopped and dined together, with or without day packages. As part of your output, indicate whether these families use day packages or not. “frequently” means at least 50% of the time.

Assumptions:

- We define a “**family**” to be a **pair** of people whose relationship is *family*
- We define the **frequency** as the number of the pair going out **together** divided by the total number of times **at least one** person goes out.
- If a *family* appears in a shop or restaurant at **exactly the same time**, we will define them **together**.
- For one visit of a family, if **one of the people** used a day package at the same time and the same shop/restaurant they visited, we will define they are using a day package when visiting.
- If a “family” used a day package **at least one time** when they visited a place **together**, we will define this family used the day package.

Explanations:

- Firstly, we extracted the family as *FamilyPairs* for easier management

```
WITH FamilyPairs AS (
    SELECT user_account_id1, user_account_id2
    FROM user_relationship
    WHERE type = 'family'
),
```

- Next, we group the records by FamilyPairs, with the restriction that they appear in the shop at the same time. And we count how many day packages they used at that time.
- We first use JOIN to get the shopping record for every family. And we use a LEFT JOIN to attach the user_use_day_package data to the shopping record.
- Then we use the same pipeline to get the dining data.

```
-- find shopping activities where family members shopped together and
count day package usage
ShoppingTogether AS (
    SELECT
        fp.user_account_id1,
        fp.user_account_id2,
        sr.date_time_in,
        COUNT(DISTINCT uudp.day_package_id) as num_day_packages
    FROM FamilyPairs fp
    JOIN shop_record sr ON
        fp.user_account_id1 = sr.user_account_id
    JOIN shop_record sr2 ON
        fp.user_account_id2 = sr2.user_account_id AND
        sr.date_time_in = sr2.date_time_in AND
        sr.shop_id = sr2.shop_id
    LEFT JOIN user_use_day_package uudp ON (
            fp.user_account_id1 = uudp.user_account_id OR
            fp.user_account_id2 = uudp.user_account_id
        ) AND sr.date_time_in = uudp.date_time_in
    GROUP BY fp.user_account_id1, fp.user_account_id2, sr.date_time_in
),

-- find dining activities where family members dined together and count
day package usage
DiningTogether AS (
    SELECT
        fp.user_account_id1,
        fp.user_account_id2,
        dr.date_time_in,
        COUNT(DISTINCT uudp.day_package_id) as num_day_packages
    FROM FamilyPairs fp
    JOIN dine_record dr ON
        fp.user_account_id1 = dr.user_account_id
    JOIN dine_record dr2 ON
        fp.user_account_id2 = dr2.user_account_id AND
        dr.date_time_in = dr2.date_time_in AND
        dr.restaurant_outlet_id = dr2.restaurant_outlet_id
    LEFT JOIN user_use_day_package uudp ON (
            fp.user_account_id1 = uudp.user_account_id OR
            fp.user_account_id2 = uudp.user_account_id
        ) AND dr.date_time_in = uudp.date_time_in
    GROUP BY fp.user_account_id1, fp.user_account_id2, dr.date_time_in
),
```

- After that, we use UNION to group the shop and dining activities together.

```
-- combine shopping and dining activities to calculate total activities
together
TogetherActivity AS (
    SELECT
        user_account_id1,
        user_account_id2,
        date_time_in,
        SUM(num_day_packages) AS num_day_packages
    FROM (
        SELECT
            user_account_id1,
            user_account_id2,
            date_time_in,
            num_day_packages
        FROM ShoppingTogether
        UNION
        SELECT
            user_account_id1,
            user_account_id2,
            date_time_in,
            num_day_packages
        FROM DiningTogether
    ) AS CombinedActivities
    GROUP BY user_account_id1, user_account_id2, date_time_in
),
```

- Then, we group the shop_record and dine_record using UNION, and group by the account as individual activity, counting the number of times they went out.

```
-- count individual activities for each user
IndividualActivity AS (
    SELECT user_account_id, date_time_in
    FROM shop_record
    UNION
    SELECT user_account_id, date_time_in
    FROM dine_record
),
IndividualActivityCount AS (
    SELECT
        user_account_id,
        COUNT(date_time_in) AS num_times
    FROM IndividualActivity
    GROUP BY user_account_id
),
```

- Then we grouped all family records to get how many day packages they totally used and how many times they went to the shop/restaurant together.

```
-- summarize family activities together
FamilyTogether AS (
    SELECT
        user_account_id1,
        user_account_id2,
        SUM(num_day_packages) AS num_day_packages,
        COUNT(date_time_in) AS num_times
    FROM TogetherActivity
    GROUP BY user_account_id1, user_account_id2
),
```

- After that, we grouped every piece of information together and calculated the frequency.

- We let n_1 be the number of times the first one goes out, n_2 be the number of times the second one goes out, and n_0. Using inclusion-exclusion principle, we can know that the total number of time at least one person go out is n_1+n_2-n_0. Then we can successfully calculate the frequency.

```
FamilyRecord AS (
    SELECT
        ft.user_account_id1,
        ft.user_account_id2,
        ft.num_day_packages,
        ft.num_times AS family_times,
        iac.num_times AS user1_times,
        iac2.num_times AS user2_times,
        CAST(ft.num_times AS FLOAT) / iac.num_times * 100 AS
user1_percentage,
        CAST(ft.num_times AS FLOAT) / iac2.num_times * 100 AS
user2_percentage,
        CAST(ft.num_times AS FLOAT) / (iac.num_times + iac2.num_times -
ft.num_times) * 100 AS family_percentage
    FROM FamilyTogether ft
    JOIN IndividualActivityCount iac ON ft.user_account_id1 =
iac.user_account_id
    JOIN IndividualActivityCount iac2 ON ft.user_account_id2 =
iac2.user_account_id
)
```

- And finally, we select the record with the family_percentage greater or equal to 50. Then format the result and output.

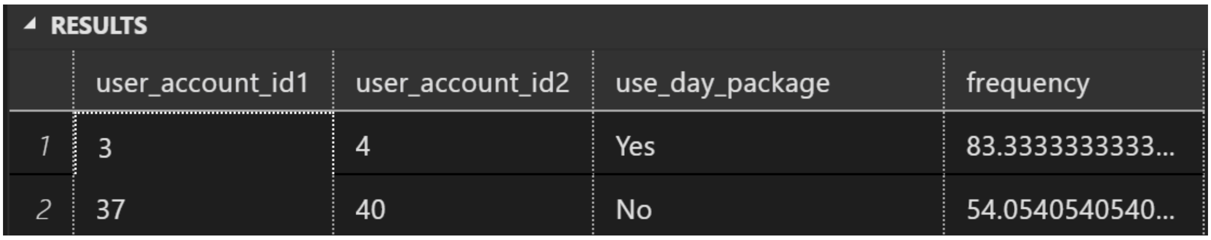
```
SELECT
    user_account_id1,
    user_account_id2,
    CASE
```

```sql
        WHEN num_day_packages > 0 THEN 'Yes'
        ELSE 'No'
    END AS use_day_package,
    family_percentage AS frequency
FROM FamilyRecord
WHERE family_percentage >= 50;
```

Final Code:

```sql
USE TestDB
GO

-- Q2: Find families who frequently shopped and dined together, with or
without day packages. As part of your output, indicate whether these
families use day packages or not. "frequently" means at least 50% of the
time.


-- identify pairs of users who are family members
WITH FamilyPairs AS (
    SELECT user_account_id1, user_account_id2
    FROM user_relationship
    WHERE type = 'family'
),

-- find shopping activities where family members shopped together and
count day package usage
ShoppingTogether AS (
    SELECT
        fp.user_account_id1,
        fp.user_account_id2,
        sr.date_time_in,
        COUNT(DISTINCT uudp.day_package_id) as num_day_packages
    FROM FamilyPairs fp
    JOIN shop_record sr ON
        fp.user_account_id1 = sr.user_account_id
    JOIN shop_record sr2 ON
        fp.user_account_id2 = sr2.user_account_id AND
        sr.date_time_in = sr2.date_time_in AND
        sr.shop_id = sr2.shop_id
    LEFT JOIN user_use_day_package uudp ON (
            fp.user_account_id1 = uudp.user_account_id OR
            fp.user_account_id2 = uudp.user_account_id
        ) AND sr.date_time_in = uudp.date_time_in
    GROUP BY fp.user_account_id1, fp.user_account_id2, sr.date_time_in
),

-- find dining activities where family members dined together and count day package usage
DiningTogether AS (
    SELECT
        fp.user_account_id1,
        fp.user_account_id2,
        dr.date_time_in,
        COUNT(DISTINCT uudp.day_package_id) as num_day_packages
    FROM FamilyPairs fp
    JOIN dine_record dr ON
        fp.user_account_id1 = dr.user_account_id
    JOIN dine_record dr2 ON
        fp.user_account_id2 = dr2.user_account_id AND
        dr.date_time_in = dr2.date_time_in AND
        dr.restaurant_outlet_id = dr2.restaurant_outlet_id
    LEFT JOIN user_use_day_package uudp ON (
            fp.user_account_id1 = uudp.user_account_id OR
            fp.user_account_id2 = uudp.user_account_id
        ) AND dr.date_time_in = uudp.date_time_in
    GROUP BY fp.user_account_id1, fp.user_account_id2, dr.date_time_in
),

-- combine shopping and dining activities to calculate total activities together
TogetherActivity AS (
    SELECT
        user_account_id1,
        user_account_id2,
        date_time_in,
        SUM(num_day_packages) AS num_day_packages
    FROM (
        SELECT
            user_account_id1,
            user_account_id2,
            date_time_in,
            num_day_packages
        FROM ShoppingTogether
        UNION
        SELECT
            user_account_id1,
            user_account_id2,
            date_time_in,
            num_day_packages
        FROM DiningTogether
    ) AS CombinedActivities
    GROUP BY user_account_id1, user_account_id2, date_time_in
),

-- count individual activities for each user
IndividualActivity AS (
    SELECT user_account_id, date_time_in
    FROM shop_record
    UNION
    SELECT user_account_id, date_time_in
    FROM dine_record
),
IndividualActivityCount AS (
    SELECT
        user_account_id,
        COUNT(date_time_in) AS num_times
    FROM IndividualActivity
    GROUP BY user_account_id
),

-- summarize family activities together
FamilyTogether AS (
    SELECT
        user_account_id1,
        user_account_id2,
        SUM(num_day_packages) AS num_day_packages,
        COUNT(date_time_in) AS num_times
    FROM TogetherActivity
    GROUP BY user_account_id1, user_account_id2
),

-- calculate the percentage of activities done together by each family pair
FamilyRecord AS (
    SELECT
        ft.user_account_id1,
        ft.user_account_id2,
        ft.num_day_packages,
        ft.num_times AS family_times,
        iac.num_times AS user1_times,
        iac2.num_times AS user2_times,
        CAST(ft.num_times AS FLOAT) / iac.num_times * 100 AS user1_percentage,
        CAST(ft.num_times AS FLOAT) / iac2.num_times * 100 AS user2_percentage,
        CAST(ft.num_times AS FLOAT) / (iac.num_times + iac2.num_times -
ft.num_times) * 100 AS family_percentage
    FROM FamilyTogether ft
    JOIN IndividualActivityCount iac ON ft.user_account_id1 =
iac.user_account_id
    JOIN IndividualActivityCount iac2 ON ft.user_account_id2 =
iac2.user_account_id
)

-- output family pairs who frequently shopped and dined together, with
or without day packages
SELECT
    user_account_id1,
    user_account_id2,
    CASE
        WHEN num_day_packages > 0 THEN 'Yes'
        ELSE 'No'
    END AS use_day_package,
    family_percentage AS frequency
FROM FamilyRecord
WHERE family_percentage > 50;
```

Output:



RESULTS

	user_account_id1	user_account_id2	use_day_package	frequency
1	3	4	Yes	83.3333333333...
2	37	40	No	54.0540540540...

Query 3

Find the most popular recommendations from the app regarding malls

Assumption:

- The most popular recommendations are as most favored by people, where we should mainly focus on user recommendations in the recommendation table.

Explanation:

Step 1:

- Select the recommendation_id and mall_id from the recommendation table.
- Filter the results to only include rows where the mall_id matches the mall_id from the subquery result, which is the most popular mall recommendation.

```
USE TestDB
GO

-- Q3. What are the most popular recommendations from the app regarding
malls?
SELECT recommendation_id, mall_id
FROM recommendation
```

Step 2:

- Calculate the total number of times each mall has been recommended by joining the recommendation table with the user_use_recommendation table on the recommendation_id.
- Group the results by mall_id.
- Order the results by the total number of recommendations in descending order.
- Select the top 1 result.

```
FROM recommendation
INNER JOIN user_use_recommendation
ON recommendation.recommendation_id =
user_use_recommendation.recommendation_id
```

```
GROUP BY recommendation.mall_id
ORDER BY total_recommendations DESC) AS most_popular_mall);
```

```
FROM (SELECT TOP 1
      recommendation.mall_id,
      COUNT(recommendation.recommendation_id) AS total_recommendations
```

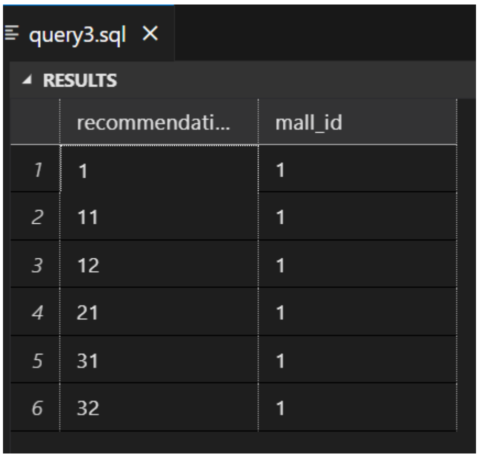
Step 3:

- Show the result by presenting mall_id and recommendation_id

Final Code:

```
USE TestDB
GO

-- Q3. What are the most popular recommendations from the app regarding
malls?
SELECT recommendation_id, mall_id
FROM recommendation
WHERE mall_id = (
  SELECT most_popular_mall.mall_id
  FROM (
    SELECT TOP 1
    recommendation.mall_id,
    COUNT(recommendation.recommendation_id) AS total_recommendations
    FROM recommendation
    INNER JOIN user_use_recommendation
          ON recommendation.recommendation_id =
user_use_recommendation.recommendation_id
    GROUP BY recommendation.mall_id
    ORDER BY total_recommendations DESC
  )
  AS most_popular_mall
);

GO
```

Output:



```
   recommendati...  mall_id
1  1                1
2  11               1
3  12               1
4  21               1
5  31               1
6  32               1
```

Query 4

Compulsive shoppers are those who have visited a certain mall more than 5 times within a certain period of time. Find the youngest compulsive shoppers and the amount they spent in total during December 2023.

Assumptions:

- Visits to shops with different shop IDs but the same mall ID counts towards the same mall.
- Compulsive shoppers may visit multiple malls but have visited at least one mall more than 5 times. However, we do not include the amount spent at malls they did not visit compulsively.
- We do not include dining records in shopping.

Explanation:

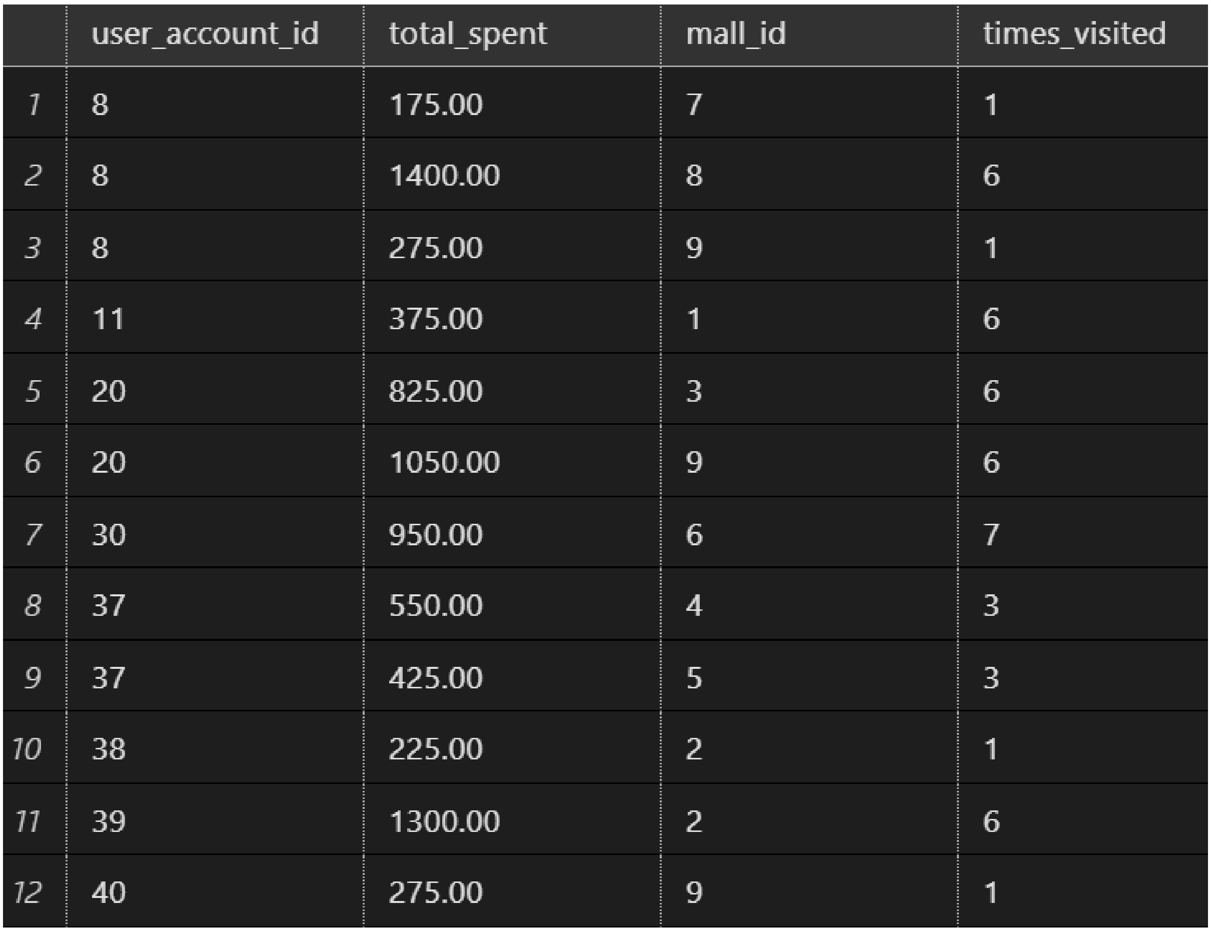
- First, let's show all the shopping records and the malls visited in December 2023.

```
use TestDB
GO
-- Show all shopping records and the malls visited in Dec 2023
SELECT X.user_account_id, SUM(X.amount_spent) AS total_spent,
shop.mall_id, COUNT(*) AS times_visited
FROM shop INNER JOIN (
  SELECT user_account_id, amount_spent, shop_id
  FROM shop_record
  WHERE shop_record.date_time_in BETWEEN '2023-12-01' AND '2023-12-31'
) AS X
ON shop.shop_id = X.shop_id
GROUP BY X.user_account_id, shop.mall_id
ORDER BY X.user_account_id, shop.mall_id;
```

clarification:

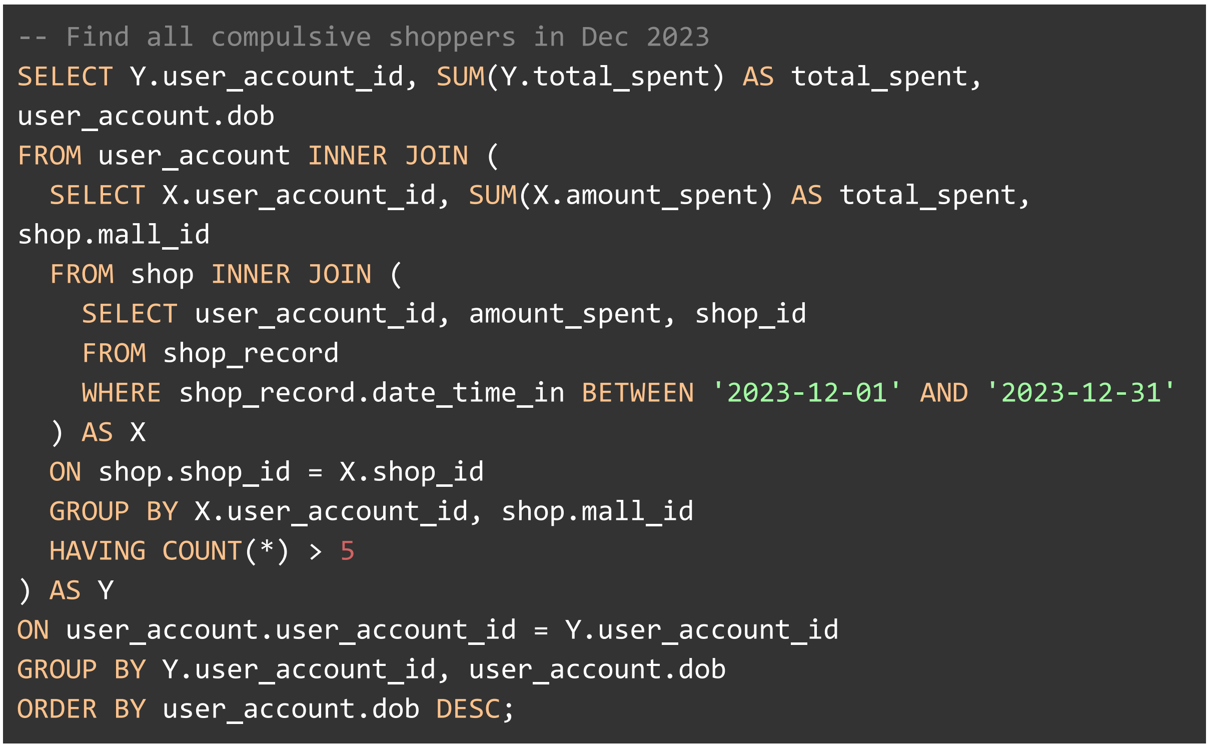
- First, we take the user ID, amount spent, and the shop ID of every shopping instance in December 2023. Then, we create an inner join with the "shop" table which allows us to group the shopping records by user ID and mall ID. We then add up the total amount spent by each user and the number of times they visited a particular mall.

Output:



	user_account_id	total_spent	mall_id	times_visited
1	8	175.00	7	1
2	8	1400.00	8	6
3	8	275.00	9	1
4	11	375.00	1	6
5	20	825.00	3	6
6	20	1050.00	9	6
7	30	950.00	6	7
8	37	550.00	4	3
9	37	425.00	5	3
10	38	225.00	2	1
11	39	1300.00	2	6
12	40	275.00	9	1

- Next, we will identify all of the compulsive shoppers and arrange them from youngest to oldest.



```
-- Find all compulsive shoppers in Dec 2023
SELECT Y.user_account_id, SUM(Y.total_spent) AS total_spent,
user_account.dob
FROM user_account INNER JOIN (
  SELECT X.user_account_id, SUM(X.amount_spent) AS total_spent,
shop.mall_id
  FROM shop INNER JOIN (
    SELECT user_account_id, amount_spent, shop_id
    FROM shop_record
    WHERE shop_record.date_time_in BETWEEN '2023-12-01' AND '2023-12-31'
  ) AS X
  ON shop.shop_id = X.shop_id
  GROUP BY X.user_account_id, shop.mall_id
  HAVING COUNT(*) > 5
) AS Y
ON user_account.user_account_id = Y.user_account_id
GROUP BY Y.user_account_id, user_account.dob
ORDER BY user_account.dob DESC;
```

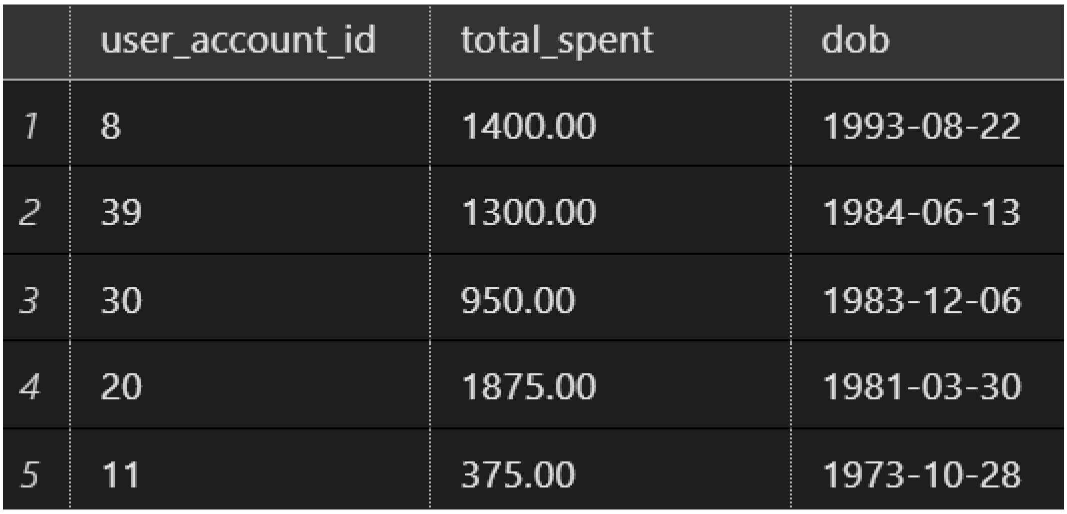
clarification:

- Again, we start by selecting the shop records from December 2023 and group them by user ID and mall. However, this time we will only keep records where the user has visited a mall more than 5 times. We join the table with the “user account” table to obtain the shoppers’ date of birth, and sum up the total amount spent once more to account for cases where a user has compulsively shopped at more than one mall. Finally, we order the compulsive shoppers in decreasing order of their date of birth.

Final Code:

```sql
-- Show all shopping records and the malls visited in Dec 2023
SELECT X.user_account_id, SUM(X.amount_spent) AS total_spent,
shop.mall_id, COUNT(*) AS times_visited
FROM shop INNER JOIN (
  SELECT user_account_id, amount_spent, shop_id
  FROM shop_record
  WHERE shop_record.date_time_in BETWEEN '2023-12-01' AND '2023-12-31'
) AS X
ON shop.shop_id = X.shop_id
GROUP BY X.user_account_id, shop.mall_id
ORDER BY X.user_account_id, shop.mall_id;

GO
-- Find all compulsive shoppers in Dec 2023
SELECT Y.user_account_id, SUM(Y.total_spent) AS total_spent,
user_account.dob
FROM user_account INNER JOIN (
  SELECT X.user_account_id, SUM(X.amount_spent) AS total_spent,
shop.mall_id
  FROM shop INNER JOIN (
    SELECT user_account_id, amount_spent, shop_id
    FROM shop_record
    WHERE shop_record.date_time_in BETWEEN '2023-12-01' AND '2023-12-31'
  ) AS X
  ON shop.shop_id = X.shop_id
  GROUP BY X.user_account_id, shop.mall_id
  HAVING COUNT(*) > 5
) AS Y
ON user_account.user_account_id = Y.user_account_id
GROUP BY Y.user_account_id, user_account.dob
ORDER BY user_account.dob DESC;

GO
```

Final output:



	user_account_id	total_spent	dob
1	8	1400.00	1993-08-22
2	39	1300.00	1984-06-13
3	30	950.00	1983-12-06
4	20	1875.00	1981-03-30
5	11	375.00	1973-10-28

- From the given results, we can infer that the 3 youngest compulsive shoppers are the users with IDs 8, 39 and 30.

Query 5

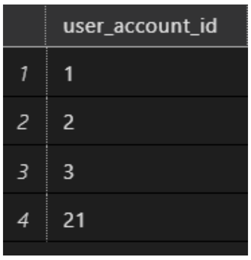
Find users who have dined in all the restaurants in some malls, but have never dined in any restaurants in some other malls.

Explanation:

Step 1: Find the users who have dined in all the restaurants in some malls.

```
-- users who have dined in all the restaurants in some malls, A
SELECT dr.user_account_id
FROM (dine_record AS dr JOIN restaurant_outlet AS ro ON
dr.restaurant_outlet_id = ro.restaurant_outlet_id)
GROUP BY dr.user_account_id, ro.mall_id
-- the number of restaurants that the user has dined in a mall is equal to
the total number of restaurants in the mall
HAVING count(DISTINCT ro.restaurant_outlet_id) = (SELECT count(DISTINCT
restaurant_outlet_id)
FROM restaurant_outlet
WHERE mall_id = ro.mall_id)
```

Output:



	user_account_id
1	1
2	2
3	3
4	21

clarification:

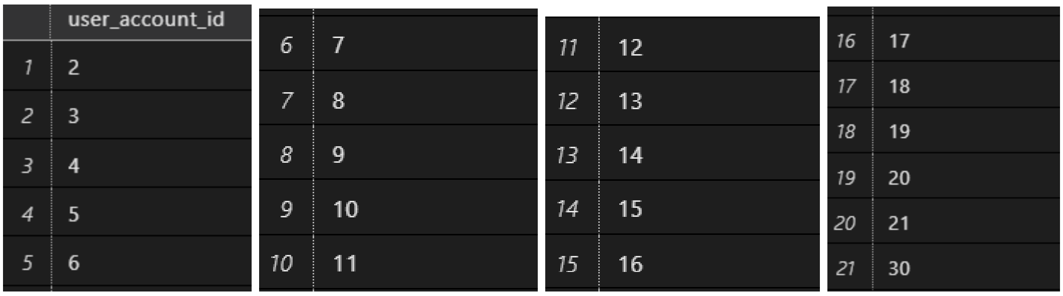
- We first **join** the tables 1. *dine_record* , 2. *restaurant_outlet* on the rows where the *restaurant_outlet_id* from both tables are the same. Then, we **group by** *user_account_id* & *mall_id*, and further filter the group **having** the **count of distinct** *restaurant_outlet_id* equals to the **count of distinct** *restaurant_outlet_id* in the mall referenced in the main query.

- The result is the *user_account_id* where they have dined in all the restaurants in some malls.

Step 2: Find the users who have never dined in any restaurants in some malls.

```
    -- users who have never dined in any restaurants in some other malls, B
    SELECT dr.user_account_id
    FROM dine_record AS dr JOIN restaurant_outlet AS ro ON
dr.restaurant_outlet_id = ro.restaurant_outlet_id
    GROUP BY dr.user_account_id
    HAVING count(DISTINCT ro.mall_id) <> (SELECT count(DISTINCT mall_id)
    FROM mall)
```

Output:



	user_account_id
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
10	11
11	12
12	13
13	14
14	15
15	16
16	17
17	18
18	19
19	20
20	21
21	30

Clarification:

- We first implement a subquery returning the *user_acount_id*. We **join** the tables 1. *dine_record* , 2. *restaurant_outlet* on the rows where the *restaurant_outlet_id* from both tables are the same. Then, we **group by** *user_account_id* and further filter the group **having** the **count of distinct** *mall_id* **NOT equals** to the total **count of distinct** *mall_id* in the database. Then, we find the *users_account_id* which is in the set returned by the subquery above, meaning that these users have never dined in any restaurants in some mall(s).

- The result is the *user_account_id* where they have never dined in any restaurants in some mall(s).

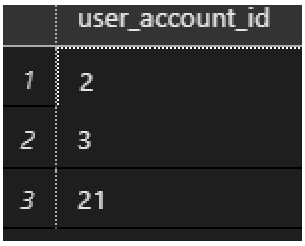
Step 3: Finally we **intercept** the result of the 2 queries above to get the final result as follow.

Final Code:

```
USE TestDB
GO

-- Q5. Find users who have dined in all the restaurants in some malls, but
have never dined in any restaurants
-- in some other malls.

-- users who have dined in all the restaurants in some malls, A
SELECT id dr.user_account_id
FROM (dine_record AS dr JOIN restaurant_outlet AS ro ON
dr.restaurant_outlet_id = ro.restaurant_outlet_id)
GROUP BY dr.user_account_id, ro.mall_id
```

```
HAVING count(DISTINCT ro.restaurant_outlet_id) = (SELECT count(DISTINCT
restaurant_outlet_id)
FROM restaurant_outlet
WHERE mall_id = ro.mall_id)

INTERSECT -- A n B gives the answer

-- users who have never dined in any restaurants in some other malls, B
SELECT ua.user_account_id
FROM user_account AS ua
WHERE ua.user_account_id IN
    (SELECT dr.user_account_id
    FROM dine_record AS dr JOIN restaurant_outlet AS ro ON
dr.restaurant_outlet_id = ro.restaurant_outlet_id
    GROUP BY dr.user_account_id
    HAVING count(DISTINCT ro.mall_id) <> (SELECT count(DISTINCT mall_id)
    FROM mall))
```

Output: Final result of Query 5



	user_account_id
1	2
2	3
3	21

Query 6

Find the top 3 highest earning malls and restaurants

Explanation:

Step 1: Find the top 3 highest earning restaurants.

```sql
-- What are the top 3 highest earning restaurants?
SELECT TOP 3
  restaurant_outlet_id,
  SUM(amount_spent) AS total_spent
FROM dine_record
GROUP BY restaurant_outlet_id
ORDER BY total_spent DESC;
```

- Our query focuses on identifying the top three highest-earning restaurants. We select the restaurant outlet IDs and calculate the total amount spent at each outlet. Then, we rank them based on the total amount spent, selecting the top three.

Step 2: Find the top 3 highest earning malls

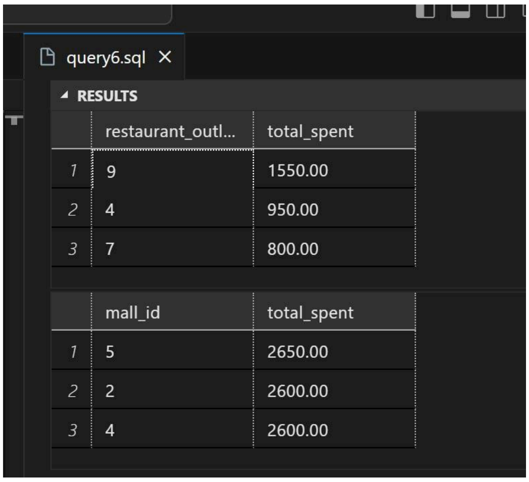
```sql
-- What are the top 3 highest earning malls?

SELECT TOP 3
  combined.mall_id,
  SUM(combined.total_spent) AS total_spent
FROM
  -- Join shop and shop_record tables to get the total amount spent in each shop
  (SELECT mall_id, SUM(amount_spent) AS total_spent
  FROM shop
      INNER JOIN shop_record ON shop.shop_id = shop_record.shop_id
  GROUP BY mall_id
  UNION ALL -- Union the results of the two queries below to get the total amount spent in each mall
  -- Join restaurant_outlet and dine_record tables to get the total amount spent in each restaurant
  SELECT mall_id, SUM(amount_spent) AS total_spent
  FROM restaurant_outlet
      INNER JOIN dine_record ON restaurant_outlet.restaurant_outlet_id = dine_record.restaurant_outlet_id
  GROUP BY mall_id) AS combined
GROUP BY combined.mall_id
ORDER BY total_spent DESC;
```

- We aggregate the total amount spent in each shop and restaurant within each mall. By combining the expenditures from both shops and restaurants, we can identify the top earners among malls.

Final Code:

```
USE TestDB
GO

-- Q6. What are the top 3 highest earning malls and restaurants?

-- What are the top 3 highest earning restaurants?
SELECT TOP 3
  restaurant_outlet_id,
  SUM(amount_spent) AS total_spent
FROM dine_record
GROUP BY restaurant_outlet_id
ORDER BY total_spent DESC;
GO

-- What are the top 3 highest earning malls?

SELECT TOP 3
  combined.mall_id,
  SUM(combined.total_spent) AS total_spent
FROM
  -- Join shop and shop_record tables to get the total amount
spent in each shop
  (SELECT mall_id, SUM(amount_spent) AS total_spent
  FROM shop
      INNER JOIN shop_record ON shop.shop_id = shop_record.shop_id
  GROUP BY mall_id
  UNION ALL -- Union the results of the two queries below to get
the total amount spent in each mall
  -- Join restaurant_outlet and dine_record tables to get the
total amount spent in each restaurant
  SELECT mall_id, SUM(amount_spent) AS total_spent
  FROM restaurant_outlet
      INNER JOIN dine_record ON
restaurant_outlet.restaurant_outlet_id =
dine_record.restaurant_outlet_id
  GROUP BY mall_id) AS combined
GROUP BY combined.mall_id
ORDER BY total_spent DESC;

GO
```

Output:



```
   restaurant_outl...  total_spent
1  9                   1550.00
2  4                   950.00
3  7                   800.00

   mall_id             total_spent
1  5                   2650.00
2  2                   2600.00
3  4                   2600.00
```

Additional Queries

Query 7

Select the shops with top 3 total complaints and lowest 3 total complaints and compare the amount spent in each shop.

Explanation:

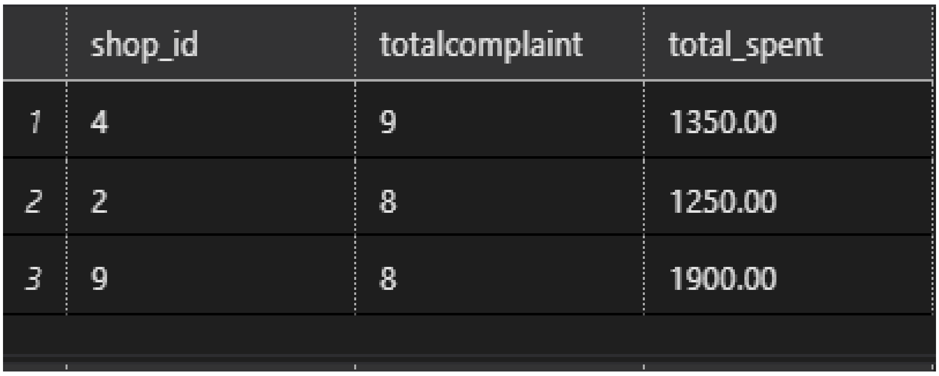
- Firstly, the code joins the shop_record table with the complaints_on_shop table using the shop_id column as the common key. The code groups the joined data by the shop_id column, so that the aggregation functions (COUNT and SUM) can be applied to each shop.
- For each shop, the code calculates the total number of complaints by using the COUNT(complaints_on_shop.shop_id) function, which counts the number of rows in the complaints_on_shop table that match the shop_id of the current shop. Also For each shop, the code calculates the total amount spent by summing up the amount_spent column from the shop_record table using the SUM(shop_record.amount_spent) function.
- The code orders the results by the total number of complaints (totalcomplaint) in descending order using ORDER BY totalcomplaint DESC, to get the shops with the highest number of complaints at the top.Then the code selects the top 3 shops with the highest number of complaints using SELECT TOP 3.
- The same way as above, the code orders the results by the total number of complaints in ascending order using ORDER BY totalcomplaint ASC, to get the shops with the lowest number of complaints at the top. Then it selects the top 3 shops with the lowest number of complaints using SELECT TOP 3.

Final Code:

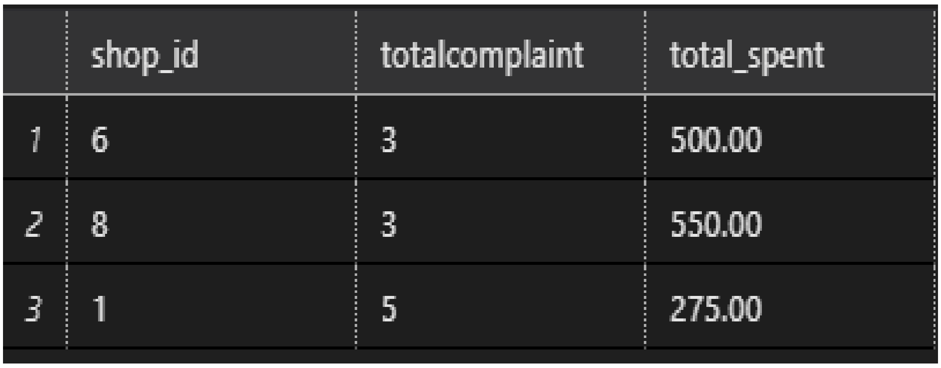
```
SELECT TOP 3
    shop_record.shop_id,
    COUNT(complaints_on_shop.shop_id) AS totalcomplaint,
    SUM(shop_record.amount_spent) AS total_spent
FROM shop_record
    INNER JOIN complaints_on_shop ON shop_record.shop_id =
complaints_on_shop.shop_id
GROUP BY shop_record.shop_id
ORDER BY totalcomplaint DESC;

SELECT TOP 3
    shop_record.shop_id,
    COUNT(complaints_on_shop.shop_id) AS totalcomplaint,
    SUM(shop_record.amount_spent) AS total_spent
FROM shop_record
    INNER JOIN complaints_on_shop ON shop_record.shop_id =
complaints_on_shop.shop_id
```

```
GROUP BY shop_record.shop_id
ORDER BY totalcomplaint ASC;
```

Output:



	shop_id	totalcomplaint	total_spent
1	4	9	1350.00
2	2	8	1250.00
3	9	8	1900.00



	shop_id	totalcomplaint	total_spent
1	6	3	500.00
2	8	3	550.00
3	1	5	275.00

Query 8

Find the top 3 restaurants with the most friend gatherings in December 2023.

Explanation:

- Firstly, we define friends as pairs with the user_relationship as “friends”

```
WITH FriendPairs AS (
    SELECT user_account_id1, user_account_id2
    FROM user_relationship
    WHERE type = 'friend'
),
```

- Then we use JOIN to create a table of records where friends ate together, with restaurant_outlet_id attached.

```
DiningTogether AS (
    SELECT
        fp.user_account_id1,
        fp.user_account_id2,
        sr.restaurant_outlet_id,
        sr.date_time_in
    FROM FriendPairs fp
    JOIN dine_record sr ON fp.user_account_id1 = sr.user_account_id AND
sr.date_time_in >= '2023-12-01' AND sr.date_time_in < '2024-01-01'
    JOIN dine_record sr2 ON fp.user_account_id2 = sr2.user_account_id
AND sr.date_time_in = sr2.date_time_in AND sr.restaurant_outlet_id =
sr2.restaurant_outlet_id
),
```

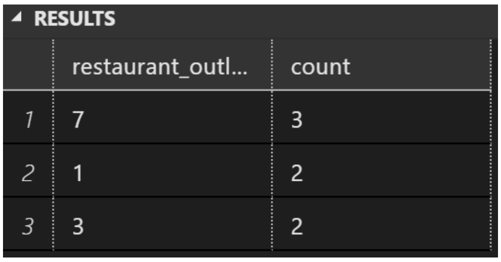
- Finally, we combined the records by restaurant_outlet_id and selected the top 3 with descending order.

```
DiningTogetherCount AS (
    SELECT
        restaurant_outlet_id,
        COUNT(*) AS count
    FROM DiningTogether
    GROUP BY restaurant_outlet_id
)
SELECT TOP 3
    *
FROM DiningTogetherCount
ORDER BY count DESC, restaurant_outlet_id;
```

Final Code:

```
USE TestDB
GO

WITH FriendPairs AS (
    SELECT user_account_id1, user_account_id2
    FROM user_relationship
    WHERE type = 'friend'
),
DiningTogether AS (
    SELECT
        fp.user_account_id1,
        fp.user_account_id2,
        sr.restaurant_outlet_id,
        sr.date_time_in
    FROM FriendPairs fp
    JOIN dine_record sr ON fp.user_account_id1 = sr.user_account_id AND
sr.date_time_in >= '2023-12-01' AND sr.date_time_in < '2024-01-01'
    JOIN dine_record sr2 ON fp.user_account_id2 = sr2.user_account_id
AND sr.date_time_in = sr2.date_time_in AND sr.restaurant_outlet_id =
sr2.restaurant_outlet_id
),
DiningTogetherCount AS (
    SELECT
        restaurant_outlet_id,
        COUNT(*) AS count
    FROM DiningTogether
    GROUP BY restaurant_outlet_id
)
SELECT TOP 3
    *
FROM DiningTogetherCount
ORDER BY count DESC, restaurant_outlet_id;
```

Output:



RESULTS

	restaurant_outl...	count
1	7	3
2	1	2
3	3	2

Query 9 (Assignment query)

Find shops that received the most complaints in December 2023.

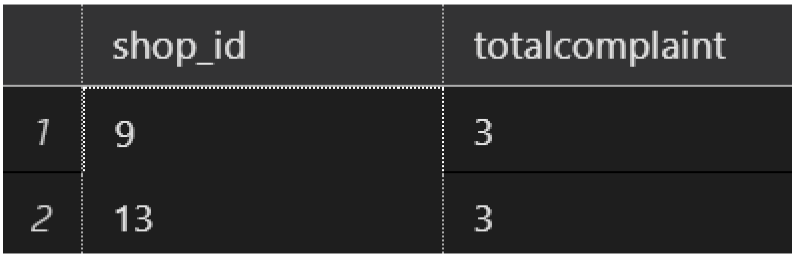
Explanation:

- Query 8 starts by selecting the top 1 record with ties to display all the shops that have the highest amount of complaints. It selects shop_id from complaints_on_shop table and counts the number of complaints. Then it joins the complaint table by the same complaint_id, and uses the WHERE statement to filter out those complaints made in December 2023. Finally it groups by shop_id and sorts the totalcomplaint in descending order to identify the shop that received the most complaints.

Final Code:

```
USE TestDB
GO

-- Q8. Find shops that received the most complaints in December 2023

SELECT TOP 1 WITH TIES
    complaints_on_shop.shop_id,
    COUNT(*) AS totalcomplaint
FROM complaints_on_shop
    INNER JOIN complaint ON complaint.complaint_id =
complaints_on_shop.complaint_id
WHERE complaint.filed_date_time BETWEEN '2023-12-01' AND '2023-12-31'
GROUP BY complaints_on_shop.shop_id
ORDER BY totalcomplaint DESC;

GO
```

Output:



	shop_id	totalcomplaint
1	9	3
2	13	3

Appendixes

Appendix A: Individual Contribution Form

Full Name	**Individual Contribution to Lab 5 Submission**	**Percentage of Contribution**	**Signature**
Pu Fanyi	SQL	14.29%	濮凡轶
Jin Qingyang			晋清扬
Qian Jianheng Oscar			Qianjh
Soo Ying Xi			
Ting Ruo Chee			Phube Ting
Ye Yuhan			
Tang Yutong			Tang

Appendix B: Individual Contribution Form

Team member	Signature	Date	A or B*
Pu Fanyi	濮凡轶	17/Apr/2024	A
Jin Qingyang	晋清扬		
Qian Jianheng Oscar	Qianjh		
Soo Ying Xi			
Ting Ruo Chee	[illegible] Ting		
Ye Yuhan			
Tang Yutong	Tang [illegible]		

* Each team member should indicate either A or B:

A. I affirm that my contribution(s) to the lab work is my own, produced without help from any AI tool(s).
B. I affirm that my contribution(s) to the lab work has been produced with the use of AI tool(s).