NANYANG TECHNOLOGICAL UNIVERSITY SINGAPORE

Lab 5 Final Report

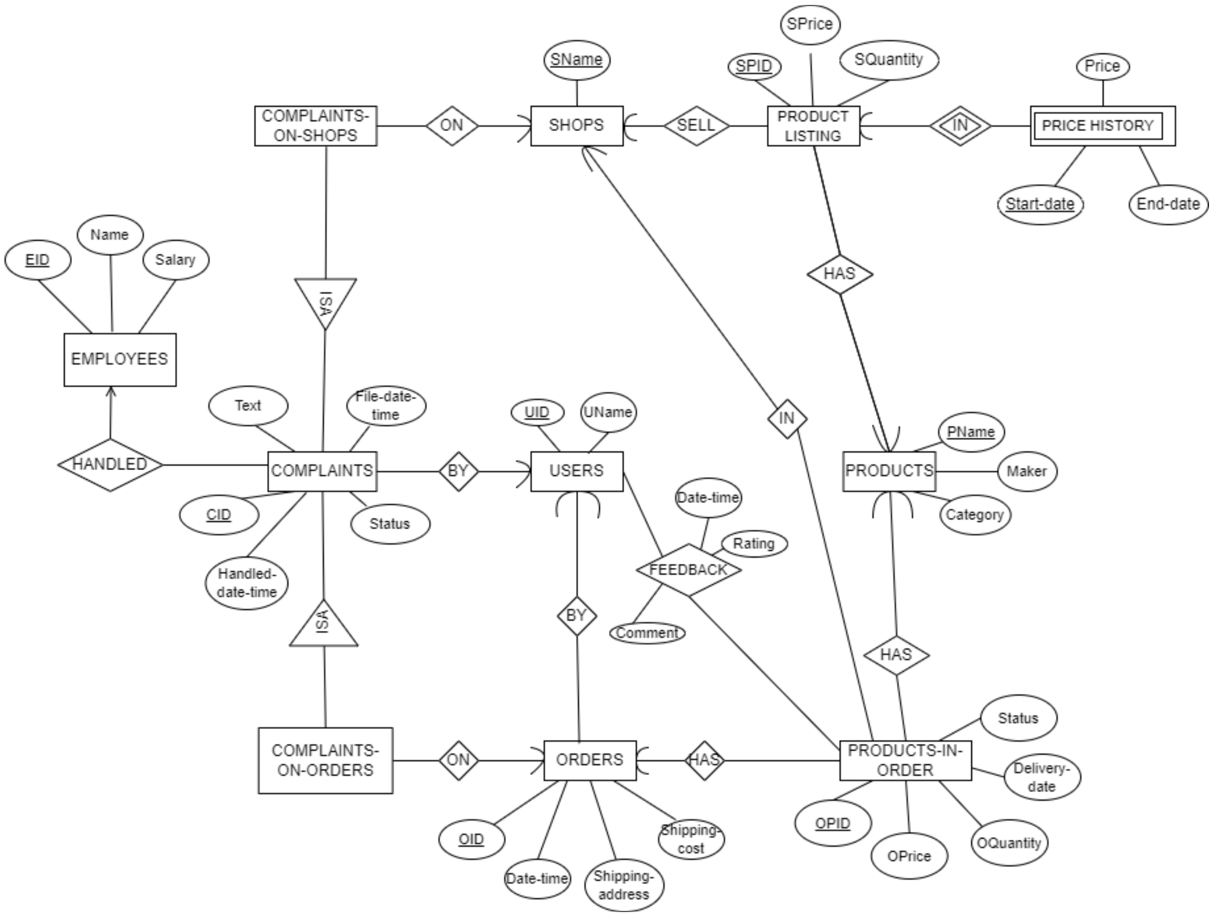
SS8 Team 2

Members
Nathanael Axel Wibisono
Wang Qianteng
Shreya Ramasubramanian
Tan Ye Quan
Bansal Arushi
Wang Dian

Table Contents

1. Final ER Diagram and Schema

1.1 ER Diagram



1.2 Schema

Entity set -> Relation

Employees (EID, Name, Salary)

Shops (SName)

Orders (OID, DateTime, Shipping-address, Shipping-cost, UID)

Users (UID, Uname)

Products (PName, Maker, Category)

Product Listing (SPID, SPrice, SQuantity, Pname, Sname)

Products in Order (OPID, OPrice, OQuantity, Delivery-date, Status, SName, PName, OID)

Many-to-Many Relationship -> Relation

Feedback (UID, OPID, Comment, Date-Time, Rating)

Weak Entity Set -> Relation

Price History (SPID, Start-Date, End-Date, Price)

Subclass / Superclass -> Relation (ER approach)

Complaints (CID, Filed-date-time, Text, Status, Handled-date-time, EID, UID)

Complaints on shops (CID, Sname)

Complaints on orders (CID, OID)

2. Commands to create database

```
CREATE DATABASE SS8G2DB
```

3. Commands to create tables

3.1 Entity Set -> Relation

Employees

```
CREATE TABLE Employees
(
  EID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
  Name VARCHAR(50) NOT NULL,
  SALARY INT,
  CHECK(salary >= 0)
);
```

Shops

```
CREATE TABLE Shops
(
 Sname VARCHAR(50) PRIMARY KEY NOT NULL
);
```

Orders

```
CREATE TABLE Orders
(
 OID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
 DateTime DATETIME NOT NULL,
 ShippingAddress VARCHAR(100) NOT NULL,
 ShippingCost DECIMAL(18,2) NOT NULL,
 UID INT,
 FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE SET DEFAULT ON UPDATE CASCADE,
 CHECK(ShippingCost >= 0)
);
```

Users

```
CREATE TABLE Users
(
 UID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
 UName VARCHAR(50) NOT NULL
);
```

Products

```
CREATE TABLE Products
(
 Pname VARCHAR(50) PRIMARY KEY NOT NULL,
 Maker VARCHAR(50) NOT NULL,
 Category VARCHAR(50) NOT NULL
);
```

ProductListings

```
CREATE TABLE ProductListings
(
SPID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
SPrice DECIMAL(18,2) NOT NULL,
SQuantity INT NOT NULL,
PName VARCHAR(50),
SName VARCHAR(50),
FOREIGN KEY (Pname) REFERENCES Products(Pname) ON DELETE SET DEFAULT ON UPDATE
CASCADE,
FOREIGN KEY (Sname) REFERENCES Shops(Sname) ON DELETE SET DEFAULT ON UPDATE CASCADE,
CHECK(SPrice >= 0),
CHECK(SQuantity >= 0),
UNIQUE(Pname,Sname)
);
```

ProductsInOrders

```
CREATE TABLE ProductsInOrders
(
 OPID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
 OPrice DECIMAL(18,2) NOT NULL,
 OQuantity INT NOT NULL,
 DeliveryDate DATETIME NOT NULL,
 Status VARCHAR(50),
 PName VARCHAR(50),
 Sname VARCHAR(50),
 OID INT,
 FOREIGN KEY (Pname) REFERENCES Products(Pname) ON DELETE SET DEFAULT ON UPDATE
CASCADE,
 FOREIGN KEY (Sname) REFERENCES Shops(Sname) ON DELETE SET DEFAULT ON UPDATE CASCADE,
 FOREIGN KEY (OID) REFERENCES Orders(OID) ON DELETE SET DEFAULT ON UPDATE CASCADE,
 CHECK(OPrice >= 0),
 CHECK(OQuantity >= 0),
 CHECK(Status = 'Being Processed' or Status = 'Shipped' or Status = 'Delivered' or
 Status = 'Returned'),
 UNIQUE(OID, Sname, Pname)
);
```

3.2 Many-to-Many Relationship -> Relation

Feedback

```
CREATE TABLE Feedback
(
 UID INT NOT NULL,
 OPID INT NOT NULL,
 Comment VARCHAR(100),
 DateTime DATETIME NOT NULL,
 Rating INT NOT NULL,
 PRIMARY KEY (UID, OPID),
 FOREIGN KEY (UID) REFERENCES Users (UID) ON DELETE NO ACTION ON UPDATE NO ACTION,
 FOREIGN KEY (OPID) REFERENCES ProductsInOrders(OPID) ON DELETE NO ACTION ON UPDATE NO
ACTION,
 CHECK(RATING >= 1 and Rating <= 5)
);
```

3.3 Weak Entity Set -> Relation

PriceHistory

```
CREATE TABLE PriceHistory
(
 SPID INT NOT NULL,
 StartDate DATETIME NOT NULL,
 EndDate DATETIME NOT NULL,
 Price DECIMAL(18,2),
 PRIMARY KEY(SPID, StartDate),
 FOREIGN KEY (SPID) REFERENCES ProductListings(SPID) ON DELETE NO ACTION ON UPDATE
CASCADE,
 CHECK(StartDate <= EndDate),
 CHECK(Price >= 0)
);
```

3.4 Subclass / Superclass -> Relation (ER approach)

Complaints

```
CREATE TABLE Complaints
(
 CID INT PRIMARY KEY NOT NULL IDENTITY(1,1),
 FilledDateTime DATETIME NOT NULL,
 HandledDateTime DATETIME,
 Text VARCHAR(50) NOT NULL,
 Status VARCHAR(50) NOT NULL,
 EID INT,
 UID INT,
 FOREIGN KEY (EID) REFERENCES Employees(EID) ON DELETE SET NULL ON UPDATE CASCADE,
 FOREIGN KEY (UID) REFERENCES Users(UID) ON DELETE SET NULL ON UPDATE CASCADE,
 CHECK(FilledDateTime <= HandledDateTime),
 CHECK(Status = 'Pending' or Status = 'Being Handled' or Status = 'Addressed')
);
```

ComplaintsOnShops

```
CREATE TABLE ComplaintsOnShops
(
 CID INT PRIMARY KEY NOT NULL,
 Sname VARCHAR(50) NOT NULL,
 FOREIGN KEY (CID) REFERENCES Complaints(CID) ON DELETE NO ACTION ON UPDATE NO ACTION,
 FOREIGN KEY (Sname) REFERENCES Shops(Sname) ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

ComplaintsOnOrders

```
CREATE TABLE ComplaintsOnOrders
(
 CID INT PRIMARY KEY NOT NULL,
 OID INT NOT NULL,
 FOREIGN KEY (CID) REFERENCES Complaints(CID) ON DELETE NO ACTION ON UPDATE NO ACTION,
 FOREIGN KEY (OID) REFERENCES Orders(OID) ON DELETE NO ACTION ON UPDATE NO ACTION
);
```

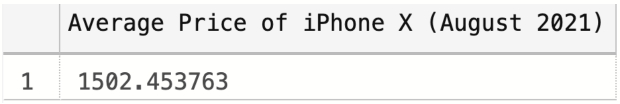
4. Commands for queries

4.1 Find the average price of “iPhone Xs” on Shiokee from 1 August 2021 to 31 August 2021.

Query Input

```
SELECT AVG(Price) AS 'Average Price of iPhone X (August 2021)'
FROM PriceHistory, ProductListings
WHERE ProductListings.PName = 'iPhone X'
AND StartDate >= '2021-08-01'
AND EndDate <= '2021-08-31'
AND ProductListings.SPID = PriceHistory.SPID;
```

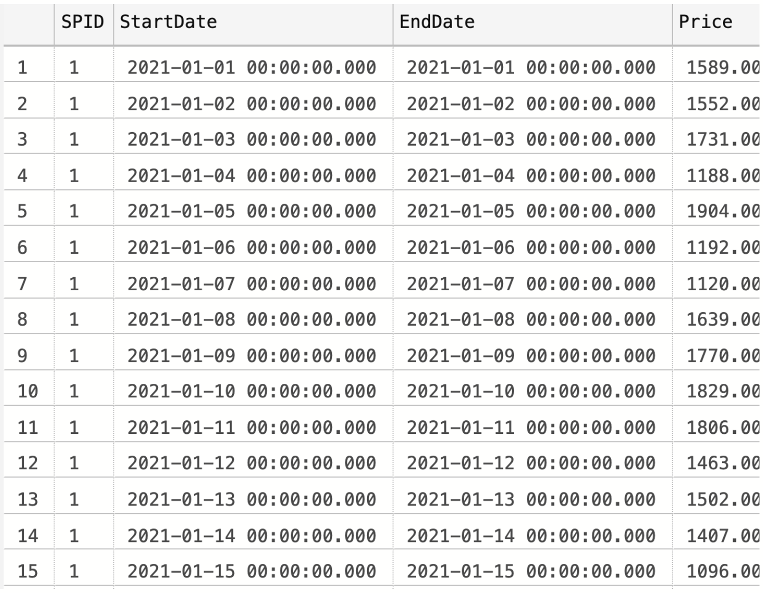
Query Output



	Average Price of iPhone X (August 2021)
1	1502.453763

Explanation

In PriceHistory, the price is recorded on a daily basis. As such, we can easily determine the average price of all iPhone X.



	SPID	StartDate	EndDate	Price
1	1	2021-01-01 00:00:00.000	2021-01-01 00:00:00.000	1589.00
2	1	2021-01-02 00:00:00.000	2021-01-02 00:00:00.000	1552.00
3	1	2021-01-03 00:00:00.000	2021-01-03 00:00:00.000	1731.00
4	1	2021-01-04 00:00:00.000	2021-01-04 00:00:00.000	1188.00
5	1	2021-01-05 00:00:00.000	2021-01-05 00:00:00.000	1904.00
6	1	2021-01-06 00:00:00.000	2021-01-06 00:00:00.000	1192.00
7	1	2021-01-07 00:00:00.000	2021-01-07 00:00:00.000	1120.00
8	1	2021-01-08 00:00:00.000	2021-01-08 00:00:00.000	1639.00
9	1	2021-01-09 00:00:00.000	2021-01-09 00:00:00.000	1770.00
10	1	2021-01-10 00:00:00.000	2021-01-10 00:00:00.000	1829.00
11	1	2021-01-11 00:00:00.000	2021-01-11 00:00:00.000	1806.00
12	1	2021-01-12 00:00:00.000	2021-01-12 00:00:00.000	1463.00
13	1	2021-01-13 00:00:00.000	2021-01-13 00:00:00.000	1502.00
14	1	2021-01-14 00:00:00.000	2021-01-14 00:00:00.000	1407.00
15	1	2021-01-15 00:00:00.000	2021-01-15 00:00:00.000	1096.00

We select PriceHistory table and ProductListings table, then join the tuples with the same SPID, which uniquely identifies the products in shops. We then select ‘iPhone X’ product with price history in August 2021. The average price is then obtained using aggregate function.

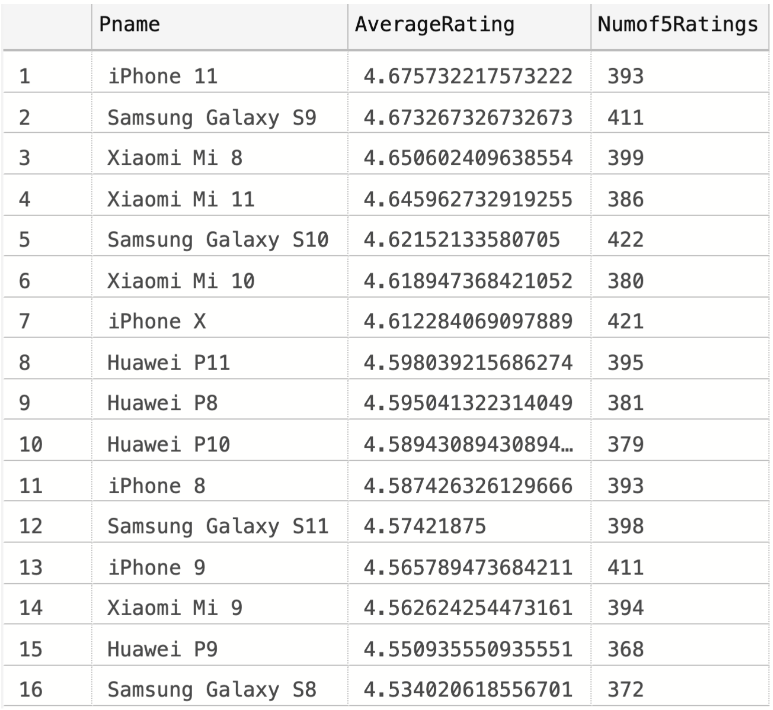
4.2 Find products that received at least 100 ratings of “5” in August 2021, and order them by their average ratings.

Scenario 1: Feedback is given for each ProductsInOrders, regardless of Oquantity ordered.

Query Input

```
SELECT Pname, AVG(CAST(rating AS FLOAT)) AS AverageRating, SUM(CASE WHEN rating = 5
THEN 1 ELSE 0 END) AS Numof5Ratings
FROM  Feedback, ProductsInOrders
WHERE Feedback.OPID = ProductsInOrders.OPID
AND Feedback.DateTime >= '2021-08-01'
AND Feedback.DateTime <= '2021-08-31'
GROUP BY Pname
Having SUM(CASE WHEN rating = 5 THEN 1 ELSE 0 END) >= 100
ORDER BY AVG(CAST(rating AS FLOAT)) DESC
```

Query Output



	Pname	AverageRating	Numof5Ratings
1	iPhone 11	4.675732217573222	393
2	Samsung Galaxy S9	4.673267326732673	411
3	Xiaomi Mi 8	4.650602409638554	399
4	Xiaomi Mi 11	4.645962732919255	386
5	Samsung Galaxy S10	4.62152133580705	422
6	Xiaomi Mi 10	4.618947368421052	380
7	iPhone X	4.612284069097889	421
8	Huawei P11	4.598039215686274	395
9	Huawei P8	4.595041322314049	381
10	Huawei P10	4.58943089430894…	379
11	iPhone 8	4.587426326129666	393
12	Samsung Galaxy S11	4.57421875	398
13	iPhone 9	4.565789473684211	411
14	Xiaomi Mi 9	4.562624254473161	394
15	Huawei P9	4.550935550935551	368
16	Samsung Galaxy S8	4.534020618556701	372

Explanation

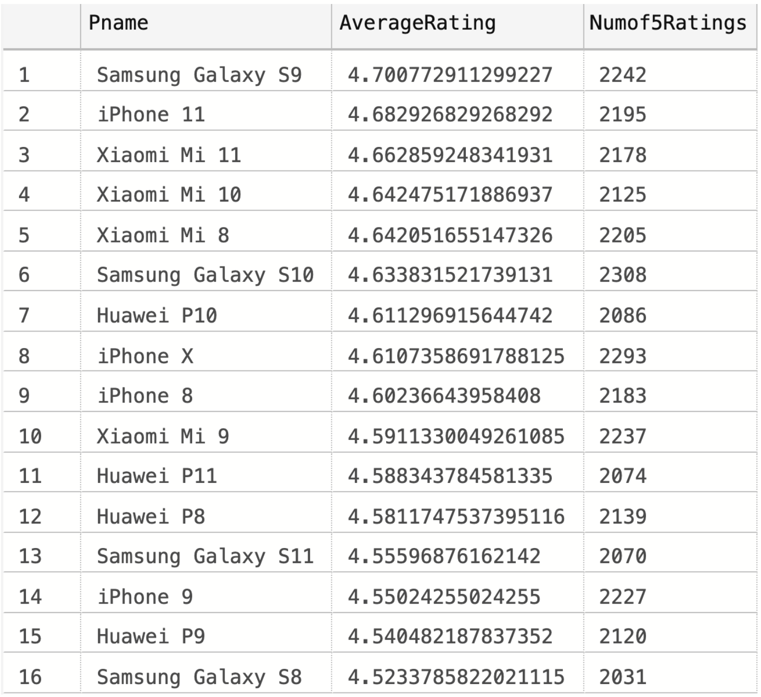
Firstly, we join the Feedback table and ProductsInOrders table by OPID, which uniquely identifies products in orders, and select feedback filed in August 2021 in WHERE clause. Secondly, we group the products with feedback tuples by PName, i.e. the name of the products. Thirdly, among the groups, we select only products that received a “5” rating in at least 100 orders. Finally, the output is the PName, average rating of each product and the number of “5” ratings, ordered by descending average rating.

Scenario 2: Feedback is given for each Oquantity ordered in ProductsInOrders

Query Input

```sql
SELECT Pname,
CAST(SUM(Rating * OQuantity)AS FLOAT)/CAST(SUM(OQuantity) AS FLOAT) AS AverageRating,
SUM(CASE WHEN rating = 5 THEN OQuantity ELSE 0 END) AS Numof5Ratings
FROM  Feedback, ProductsInOrders
WHERE Feedback.OPID = ProductsInOrders.OPID
AND Feedback.DateTime >= '2021-08-01'
AND Feedback.DateTime <= '2021-08-31'
GROUP BY Pname
Having SUM(CASE WHEN rating = 5 THEN OQuantity ELSE 0 END) >= 100
ORDER BY AverageRating DESC;
```

Query Output



	Pname	AverageRating	Numof5Ratings
1	Samsung Galaxy S9	4.700772911299227	2242
2	iPhone 11	4.682926829268292	2195
3	Xiaomi Mi 11	4.662859248341931	2178
4	Xiaomi Mi 10	4.642475171886937	2125
5	Xiaomi Mi 8	4.642051655147326	2205
6	Samsung Galaxy S10	4.633831521739131	2308
7	Huawei P10	4.611296915644742	2086
8	iPhone X	4.6107358691788125	2293
9	iPhone 8	4.60236643958408	2183
10	Xiaomi Mi 9	4.5911330049261085	2237
11	Huawei P11	4.588343784581335	2074
12	Huawei P8	4.5811747537395116	2139
13	Samsung Galaxy S11	4.55596876162142	2070
14	iPhone 9	4.55024255024255	2227
15	Huawei P9	4.540482187837352	2120
16	Samsung Galaxy S8	4.5233785822021115	2031

Explanation

Firstly, we join the Feedback table and ProductsInOrders table by OPID, which uniquely identifies products in orders, and select feedback filed in August 2021 in WHERE clause. Secondly, we group the products with feedback tuples by PName, i.e. the name of the products. Thirdly, among the groups, we select only products that received at least 100 “5” ratings. One product with quantity N in one order is considered as having received N ratings. Finally, the output is the PName, average rating of each product and the number of “5” ratings, ordered by descending average rating.

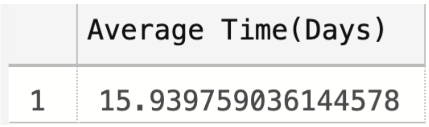
4.3 For all products purchased in June 2021 that have been delivered, find the average time from the ordering date to the delivery date.

Scenario 1: Without considering OQuantity

Query Input

```
SELECT CAST(SUM(DATEDIFF(day, Orders.DateTime, ProductsInOrders.DeliveryDate)) AS
FLOAT)/CAST(COUNT(Orders.OID) AS FLOAT) AS 'Average Time(Days)'
FROM Orders, ProductsInOrders
WHERE Orders.OID = ProductsInOrders.OID
AND status = 'Delivered'
AND Orders.DateTime >= '2021-06-01'
AND Orders.DateTime <= '2021-06-30'
```

Query Output



	Average Time(Days)
1	15.939759036144578

Explanation

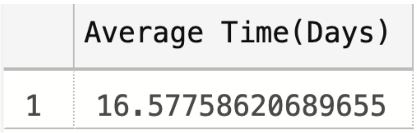
We join the Orders table and ProductsInOrders table by OID, which is the ID of each order, then select products that were purchased in June 2021 and have been delivered. Multiple pieces of the same products in one order are only considered once. The output is the average time from the ordering date to the delivery date of these products. The result is cast into float for better accuracy.

Scenario 2: Considering OQuantity

Query Input

```
SELECT CAST(SUM(OQuantity *DATEDIFF(day, Orders.DateTime,
ProductsInOrders.DeliveryDate)) AS FLOAT)/CAST(SUM(OQuantity) AS FLOAT) AS 'Average
Time(Days)'
FROM Orders, ProductsInOrders
WHERE Orders.OID = ProductsInOrders.OID
AND status = 'Delivered'
AND Orders.DateTime >= '2021-06-01'
AND Orders.DateTime <= '2021-06-30'
```

Query Output



	Average Time(Days)
1	16.57758620689655

Explanation

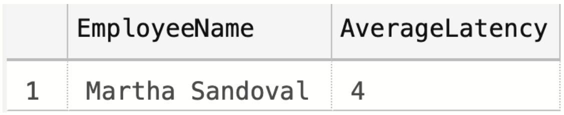
We join the Orders table and ProductsInOrders table by OID, which is the ID of each order, then select products that were purchased in June 2021 and have been delivered. Multiple pieces of the same product in one order are considered separately. The output is the average time from the ordering date to the delivery date of these products. The result is cast into float for better accuracy.

4.4 Let us define the “latency” of an employee by the average that he/she takes to process a complaint. Find the employee with the smallest latency.

Query Input

```
-- Step 2: Find the employee with the least latency
SELECT Employees.name AS EmployeeName,
AVG(DATEDIFF(day, FilledDateTime, HandledDateTime)) AS AverageLatency
FROM Complaints,Employees
WHERE Employees.EID = Complaints.EID
GROUP BY Employees.Name
HAVING AVG(DATEDIFF(day, FilledDateTime, HandledDateTime)) =
(
   -- Step 1: Find the least latency
   SELECT TOP 1 AVG(DATEDIFF(day, FilledDateTime, HandledDateTime))
   FROM Complaints,Employees
   WHERE Employees.EID = Complaints.EID
   GROUP BY Employees.Name
   ORDER BY AVG(DATEDIFF(day, FilledDateTime, HandledDateTime)) ASC
)
```

Query Output



	EmployeeName	AverageLatency
1	Martha Sandoval	4

Explanation

Step 1: Find the smallest latency.

In the subquery, we join the Employees table and Complaints table by EID, which uniquely identifies an employee. Then we group the tuples by employee names. We calculate the latency of each employee and order the result by ascending latency. We return the latency of the first row, which is the smallest latency.

Step 2: Find the employees with the smallest latency.

Similarly, in the outer query we find the latency of each employee. We select the employees whose latency equals the smallest latency that we find in step 1.

4.5 Produce a list that contains (i) all products made by Samsung, and (ii) for each of them, the number of shops on Shiokee that sell the product.

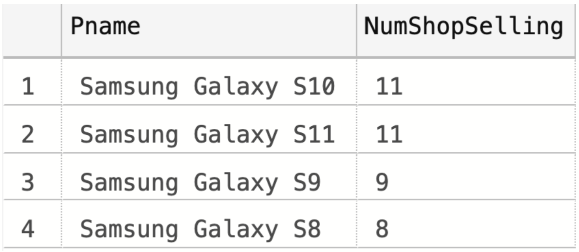
Query Input 1

```
SELECT ProductListings.Pname, COUNT(ProductListings.Sname) AS NumShopSelling
FROM ProductListings, Products
WHERE ProductListings.Pname = Products.Pname
AND Maker = 'SAMSUNG'
GROUP BY ProductListings.Pname
ORDER BY NumShopSelling DESC
```

Query Input 2

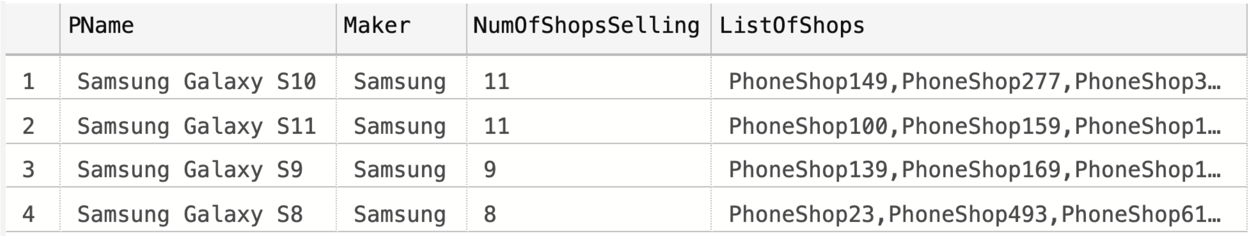
```
SELECT Products.PName,Maker,COUNT(SName) AS NumOfShopsSelling, STRING_AGG(SName,',')
AS ListOfShops
FROM
ProductListings, Products
WHERE Products.PName = ProductListings.PName
AND Maker = 'Samsung'
GROUP BY Products.PName,Maker
ORDER BY COUNT(Sname) DESC;
```

Query Output 1



	Pname	NumShopSelling
1	Samsung Galaxy S10	11
2	Samsung Galaxy S11	11
3	Samsung Galaxy S9	9
4	Samsung Galaxy S8	8

Query Output 2



	PName	Maker	NumOfShopsSelling	ListOfShops
1	Samsung Galaxy S10	Samsung	11	PhoneShop149,PhoneShop277,PhoneShop3…
2	Samsung Galaxy S11	Samsung	11	PhoneShop100,PhoneShop159,PhoneShop1…
3	Samsung Galaxy S9	Samsung	9	PhoneShop139,PhoneShop169,PhoneShop1…
4	Samsung Galaxy S8	Samsung	8	PhoneShop23,PhoneShop493,PhoneShop61…

Explanation

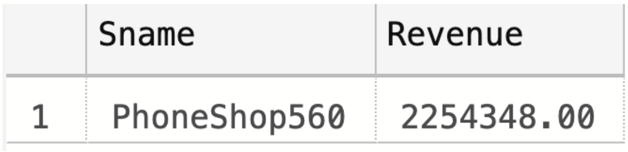
We join the ProductListings table and Products table by PName, i.e. the product name, and select products made by Samsung. Then we group the tuples by product names. The output is obtained using the aggregate function to count the number of shops for each product. We can also obtain the list of shops selling each of the Samsung products.

4.6 Find shops that made the most revenue in August 2021.

Query Input

```
-- Step 2: Find the shops with maximum revenue
SELECT Sname, SUM(OPrice * OQuantity) AS Revenue
FROM ProductsInOrders, Orders
WHERE ProductsInOrders.OID = Orders.OID
AND Orders.DateTime >= '2021-08-01'
AND Orders.DateTime <= '2021-08-31'
GROUP BY Sname
HAVING SUM(OPrice * OQuantity) =
(
   -- Step 1: Find the maximum revenue
   SELECT TOP 1 SUM(OPrice * OQuantity)
   FROM ProductsInOrders, Orders
   WHERE ProductsInOrders.OID = Orders.OID
   AND Orders.DateTime >= '2021-08-01'
   AND Orders.DateTime <= '2021-08-31'
   GROUP BY SName
   ORDER BY SUM(OPrice * OQuantity) DESC
)
```

Query Output



	Sname	Revenue
1	PhoneShop560	2254348.00

Explanation

Step 1: Find the highest revenue.

In the subquery, We join the ProductsInOrders table and Orders table by OID, the ID of each order, and select the orders made during August 2021. Then we group the tuples by the shop name SName. We calculate the revenue for each group and order the tuple by descending revenue. The revenue of the first row is the highest revenue.

Step 2: Find the shop with the highest revenue.

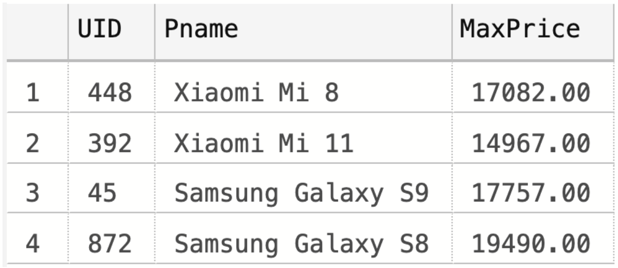
In the outer query, we find the revenue of each shop in the same way as in step 1. Then we select the shops whose revenue equals the highest revenue.

4.7 For users that made the most amount of complaints, find the most expensive products he/she has ever purchased.

Query Input

```
-- Step 4: Find the name of such product for each user
SELECT Y.UID, Pname, MaxPrice
FROM
(
  -- Step 3: Find the price of the most expensive items bought by these users
  SELECT X.UID, MAX(Oprice) as MaxPrice
  FROM
  (
      -- Step 2: Find the users with the highest number of complaint
      SELECT Users.UID
      FROM Users,Complaints
      WHERE Users.UID = Complaints.UID
      GROUP By Users.UID
      HAVING COUNT(complaints.uid) =
      (
          -- Step 1: Find the highest number of complaints
          SELECT TOP 1 COUNT(Complaints.UID)
          FROM Users,Complaints
          WHERE Users.UID = Complaints.UID
          GROUP By Complaints.UID
          ORDER By Count(CID) DESC
      )
  )X, Orders, ProductsInOrders
  WHERE Orders.UID = X.UID
  AND Orders.OID = ProductsInOrders.OID
  GROUP BY X.UID
)Y, Orders, ProductsInOrders
WHERE Orders.UID = Y.UID
AND Orders.OID = ProductsInOrders.OID
AND OPrice = MaxPrice
```

Query Output



	UID	Pname	MaxPrice
1	448	Xiaomi Mi 8	17082.00
2	392	Xiaomi Mi 11	14967.00
3	45	Samsung Galaxy S9	17757.00
4	872	Samsung Galaxy S8	19490.00

Explanation

Step 1: Find the highest number of complaints.

We join the Users table and Complaints table, group by the ID of each user UID, and calculate the number of complaints each user has made. Then the maximum number of complaints is obtained by ordering and selecting the top 1.

Step 2: Find the users with the highest number of complaints.

Similar to step 1, we find the UID and number of complaints for each user. We select the users whose number of complaints equals the highest number of complaints obtained in step 1.

Step 3: Find the price of the most expensive products bought by each of the users. We join the table of users obtained in step 2, the Order table and the ProductsInOrders table by UID and OID. Then we group by UID and find the highest price of the products a user has bought for each user.

Step 4: Find the name of the most expensive product for each user.

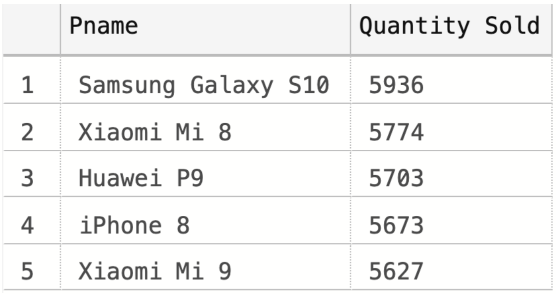
We join the Orders table, ProductsInOrders table together with the table of users and maximum prices obtained in step 3. Then we find the name of the most expensive product for each user, as required in the question.

4.8 Find products that have never been purchased by some users, but are the top 5 most purchased products-by other users in August 2021. (e.g. Suppose some users: uid = 521 & 581)

Query Input

```sql
-- Step 2:Finding top 5 most purchased products by other users
Select TOP 5 Pname, SUM(OQuantity) as 'Quantity Sold'
FROM Orders, ProductsInOrders
Where Orders.OID = ProductsInOrders.OID
AND PName NOT IN
(
   -- Step 1:Finding all products purchased by UID 521 & UID 581
   SELECT Pname
   FROM Orders, ProductsInOrders
   WHERE Orders.OID = ProductsInOrders.OID
   AND Orders.UID = 521
   UNION
   SELECT Pname
   FROM Orders, ProductsInOrders
   WHERE Orders.OID = ProductsInOrders.OID
   AND Orders.UID = 581
)
AND Orders.DateTime >= '2021-08-01'
AND Orders.DateTime <= '2021-08-31'
GROUP BY Pname
ORDER BY SUM(OQuantity) DESC
```

Query Output



	Pname	Quantity Sold
1	Samsung Galaxy S10	5936
2	Xiaomi Mi 8	5774
3	Huawei P9	5703
4	iPhone 8	5673
5	Xiaomi Mi 9	5627

Explanation

Our interpretation of the question is as such (as explained by our TA):

1. Given some user IDs or user names (E.g., User A and User B)
2. Find the products that have never been purchased by these users.
3. From these products, find the top 5 most purchased products by other users (the users except User A and User B)

Therefore, our implementation is as follows:

Step 1: Find all products purchased by UID 521 and UID 581.

We randomly choose these two users to implement this query. We use the Orders and ProductsInOders table to find the products bought by UID 521, and the products bought by 581. We union the two resulting tables to obtain all products bought by these two users.

Step 2: From products not in the table obtained in step 1, find the top 5 most purchased products.

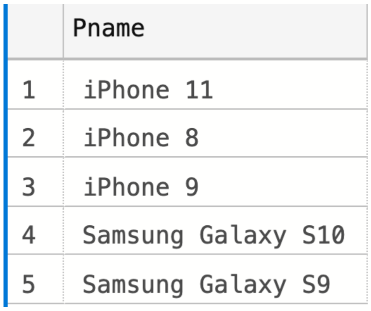
We join the Products table and the ProductsInOrders table by OID. Then we find the products not in the table obtained in step 2 (which are the products that have not been purchased by UID 521 and UID 581) and purchased in August 2021. For each product, we find the total sale quantity. We sort the sale quantity by descending order. Thus, we can obtain the top 5 sold products, as desired by the question.

4.9 Find products that are increasingly being purchased over at least 3 months.

Query Input

```
-- Step 2: Find product that increasingly being purchased over at least 3 months
SELECT DISTINCT Pname
FROM
(
   -- Step 1: Find difference between no. product sold this month compared to one month ago, two month ago, and three month ago
   SELECT Products.Pname, SUM(OQuantity) as NumSold,
   YEAR(Orders.DateTime)*100 + MONTH(Orders.DateTime) AS YearMonth,
   SUM(OQuantity) - Lag(SUM(OQuantity),1,NULL) OVER (PARTITION BY Products.Pname ORDER BY Products.Pname, YEAR(Orders.DateTime)*100 + MONTH(Orders.DateTime)) AS Diff1,
   SUM(OQuantity) - Lag(SUM(OQuantity),2,NULL) OVER (PARTITION BY Products.Pname ORDER BY Products.Pname, YEAR(Orders.DateTime)*100 + MONTH(Orders.DateTime)) AS Diff2,
   SUM(OQuantity) - Lag(SUM(OQuantity),3,NULL) OVER (PARTITION BY Products.Pname ORDER BY Products.Pname, YEAR(Orders.DateTime)*100 + MONTH(Orders.DateTime)) AS Diff3
   FROM ProductsInOrders, Products, Orders
   WHERE ProductsInOrders.Pname = Products.Pname
   AND Orders.OID = ProductsInOrders.OID
   GROUP By Products.Pname, YEAR(Orders.DateTime)*100 + MONTH(Orders.DateTime)
) AS X
WHERE Diff1 >0
AND Diff2 > 0
AND Diff3 > 0
AND Diff2 > Diff1
AND Diff3 > Diff2
AND Diff3 > Diff1
```

Query Output



	Pname
1	iPhone 11
2	iPhone 8
3	iPhone 9
4	Samsung Galaxy S10
5	Samsung Galaxy S9

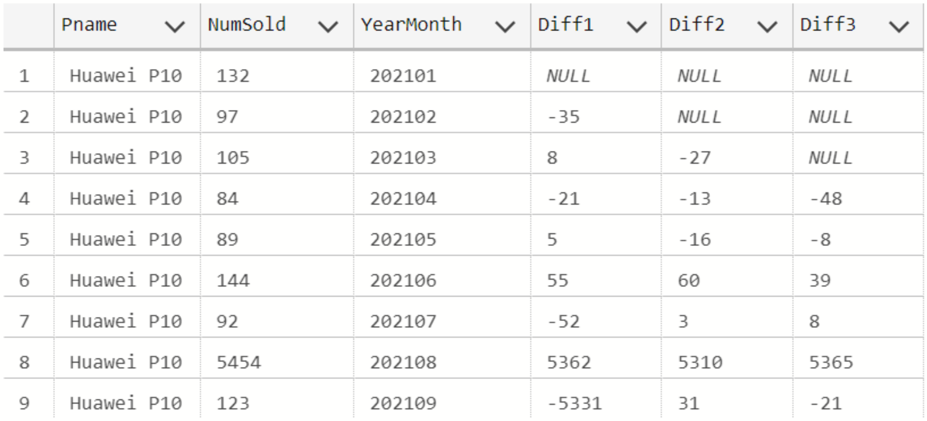
Explanation

Idea:

For each product and each month i, let Diff1 be the increase in sales between month i and month (i-1), Diff2 be the increase between month i and month (i-2), and Diff3 between month i and month (i-3). The basic idea is to find Diff1, Diff2, Diff3 for each product and each month. If there exits Diff1, Diff2, Diff3 for a product such that Diff1 > 0, Diff2 > 0, Diff3 >0, Diff2 > Diff1, Diff3 > Diff2, and Diff3 > Diff1, then this product is increasingly be purchased over the last three months.

Step 1: Find Diff1, Diff2, Diff3 for each product and each month.

We join the ProductsInOrders table, Products table and the Orders table by PName and OID. Then we group by Pname and an index YearMonth defined as “yyyymm”. To calculate Diff1, we use LAG() function to find the sales one month ago. Diff1 = sales of this month - sales one month ago. Similarly we obtain Diff2 and Diff3. Finally, we obtain the table (Pname, Numsold, YearMonth, Diff1, Diff2, Diff3).



	Pname	NumSold	YearMonth	Diff1	Diff2	Diff3
1	Huawei P10	132	202101	*NULL*	*NULL*	*NULL*
2	Huawei P10	97	202102	-35	*NULL*	*NULL*
3	Huawei P10	105	202103	8	-27	*NULL*
4	Huawei P10	84	202104	-21	-13	-48
5	Huawei P10	89	202105	5	-16	-8
6	Huawei P10	144	202106	55	60	39
7	Huawei P10	92	202107	-52	3	8
8	Huawei P10	5454	202108	5362	5310	5365
9	Huawei P10	123	202109	-5331	31	-21

Step 2: Find the products that are increasingly being purchased over the last three months.

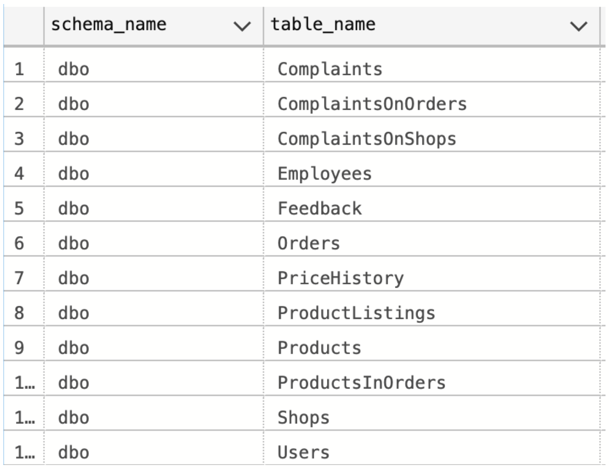
Using the table obtained in step 1, we select products where there exists a row such that Diff1 > 0, Diff2 > 0, Diff3 >0, Diff2 > Diff1, Diff3 > Diff2, and Diff3 > Diff1. These products are increasingly being purchased over the last three months.

5. Table records

5.1 Excel files of complete tables in the drive below:

https://drive.google.com/drive/folders/1x3DCEZXxLGJrQ4UE8wwWJ_IAmo4MWXjK?usp=sharing

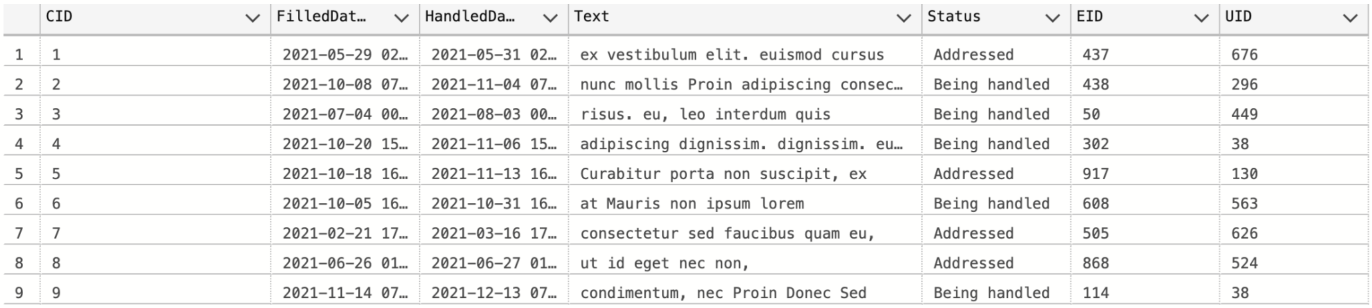
List of tables



	schema_name	table_name
1	dbo	Complaints
2	dbo	ComplaintsOnOrders
3	dbo	ComplaintsOnShops
4	dbo	Employees
5	dbo	Feedback
6	dbo	Orders
7	dbo	PriceHistory
8	dbo	ProductListings
9	dbo	Products
1…	dbo	ProductsInOrders
1…	dbo	Shops
1…	dbo	Users

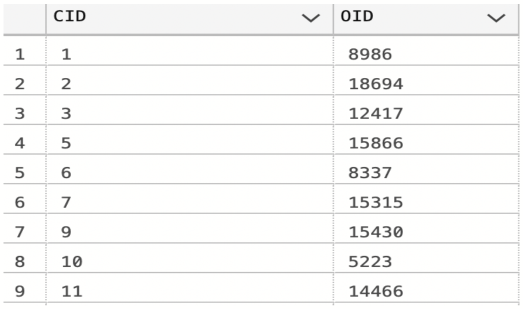
5.2 Screen captures of few records from the tables

Complaints



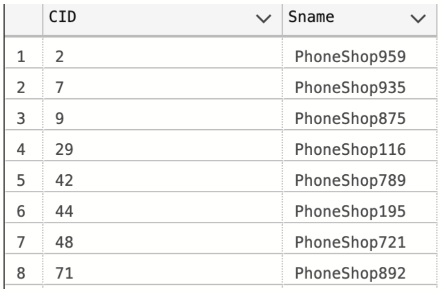
	CID	FilledDat…	HandledDa…	Text	Status	EID	UID
1	1	2021-05-29 02…	2021-05-31 02…	ex vestibulum elit. euismod cursus	Addressed	437	676
2	2	2021-10-08 07…	2021-11-04 07…	nunc mollis Proin adipiscing consec…	Being handled	438	296
3	3	2021-07-04 00…	2021-08-03 00…	risus. eu, leo interdum quis	Being handled	50	449
4	4	2021-10-20 15…	2021-11-06 15…	adipiscing dignissim. dignissim. eu…	Being handled	302	38
5	5	2021-10-18 16…	2021-11-13 16…	Curabitur porta non suscipit, ex	Addressed	917	130
6	6	2021-10-05 16…	2021-10-31 16…	at Mauris non ipsum lorem	Being handled	608	563
7	7	2021-02-21 17…	2021-03-16 17…	consectetur sed faucibus quam eu,	Addressed	505	626
8	8	2021-06-26 01…	2021-06-27 01…	ut id eget nec non,	Addressed	868	524
9	9	2021-11-14 07…	2021-12-13 07…	condimentum, nec Proin Donec Sed	Being handled	114	38

ComplaintsOnOrders



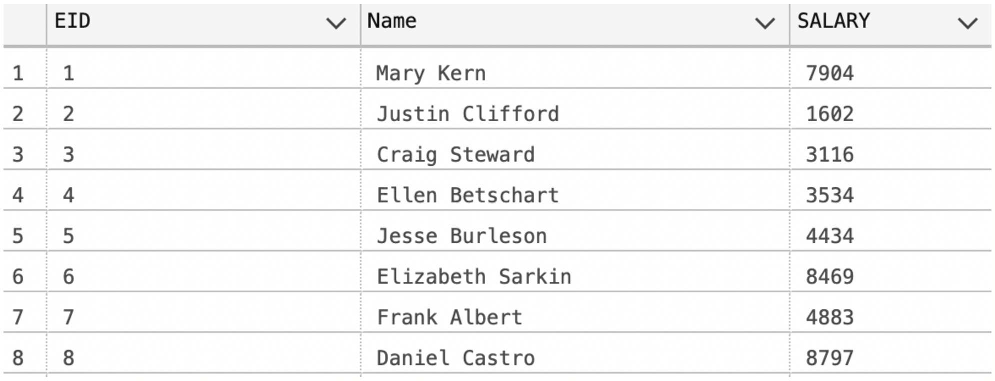
	CID	OID
1	1	8986
2	2	18694
3	3	12417
4	5	15866
5	6	8337
6	7	15315
7	9	15430
8	10	5223
9	11	14466

ComplaintsOnShops



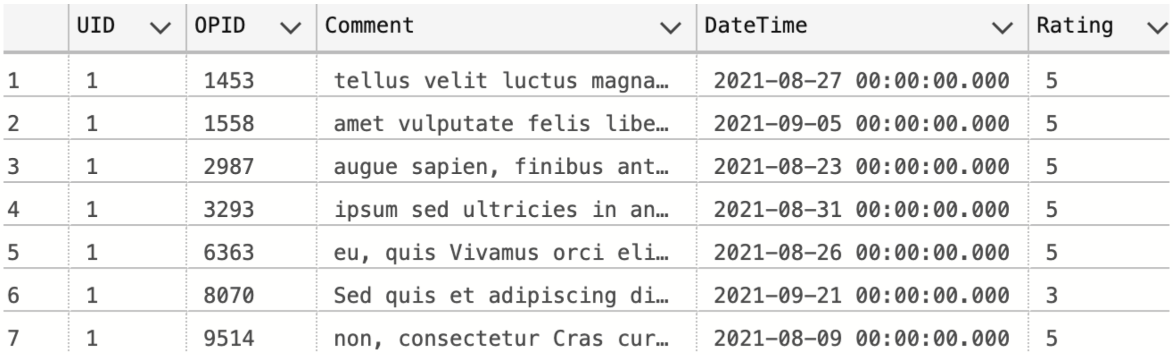
	CID	Sname
1	2	PhoneShop959
2	7	PhoneShop935
3	9	PhoneShop875
4	29	PhoneShop116
5	42	PhoneShop789
6	44	PhoneShop195
7	48	PhoneShop721
8	71	PhoneShop892

Employees



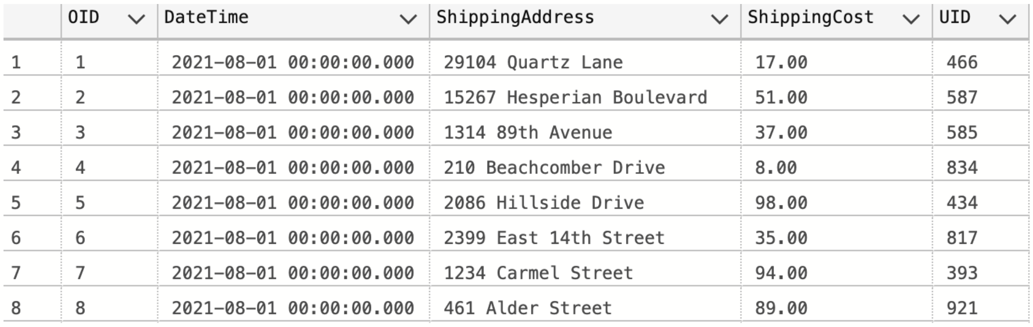
	EID	Name	SALARY
1	1	Mary Kern	7904
2	2	Justin Clifford	1602
3	3	Craig Steward	3116
4	4	Ellen Betschart	3534
5	5	Jesse Burleson	4434
6	6	Elizabeth Sarkin	8469
7	7	Frank Albert	4883
8	8	Daniel Castro	8797

Feedback



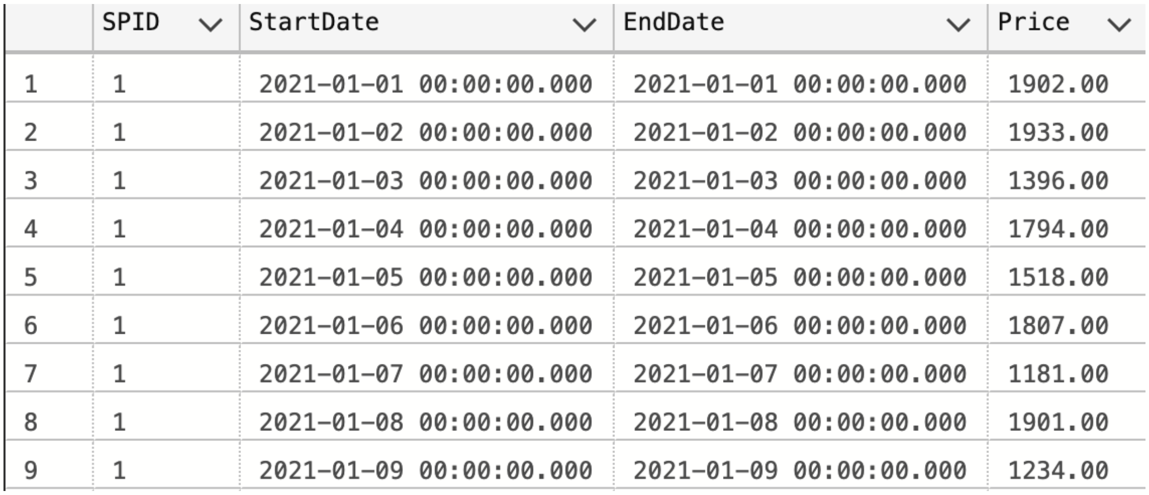
	UID	OPID	Comment	DateTime	Rating
1	1	1453	tellus velit luctus magna…	2021-08-27 00:00:00.000	5
2	1	1558	amet vulputate felis libe…	2021-09-05 00:00:00.000	5
3	1	2987	augue sapien, finibus ant…	2021-08-23 00:00:00.000	5
4	1	3293	ipsum sed ultricies in an…	2021-08-31 00:00:00.000	5
5	1	6363	eu, quis Vivamus orci eli…	2021-08-26 00:00:00.000	5
6	1	8070	Sed quis et adipiscing di…	2021-09-21 00:00:00.000	3
7	1	9514	non, consectetur Cras cur…	2021-08-09 00:00:00.000	5

Orders



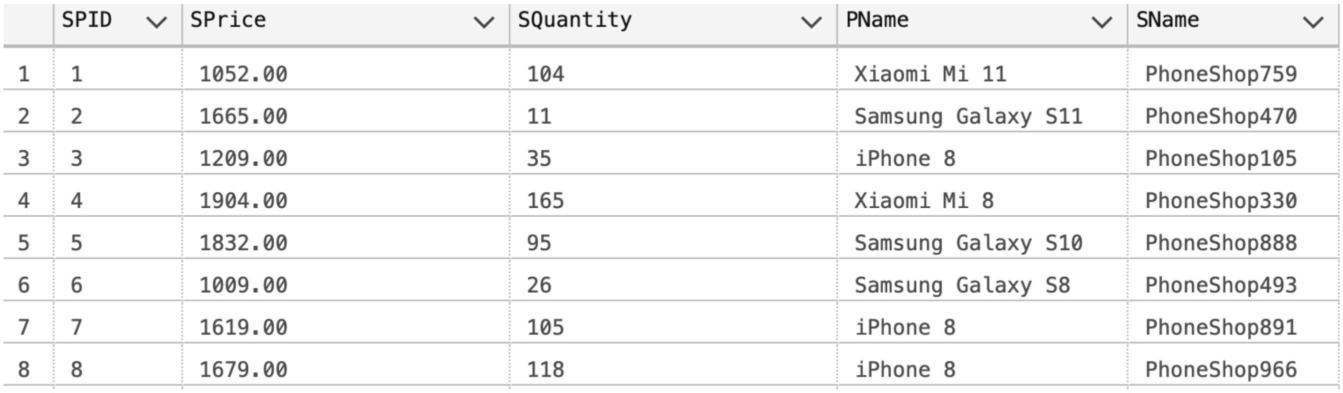
	OID	DateTime	ShippingAddress	ShippingCost	UID
1	1	2021-08-01 00:00:00.000	29104 Quartz Lane	17.00	466
2	2	2021-08-01 00:00:00.000	15267 Hesperian Boulevard	51.00	587
3	3	2021-08-01 00:00:00.000	1314 89th Avenue	37.00	585
4	4	2021-08-01 00:00:00.000	210 Beachcomber Drive	8.00	834
5	5	2021-08-01 00:00:00.000	2086 Hillside Drive	98.00	434
6	6	2021-08-01 00:00:00.000	2399 East 14th Street	35.00	817
7	7	2021-08-01 00:00:00.000	1234 Carmel Street	94.00	393
8	8	2021-08-01 00:00:00.000	461 Alder Street	89.00	921

PriceHistory



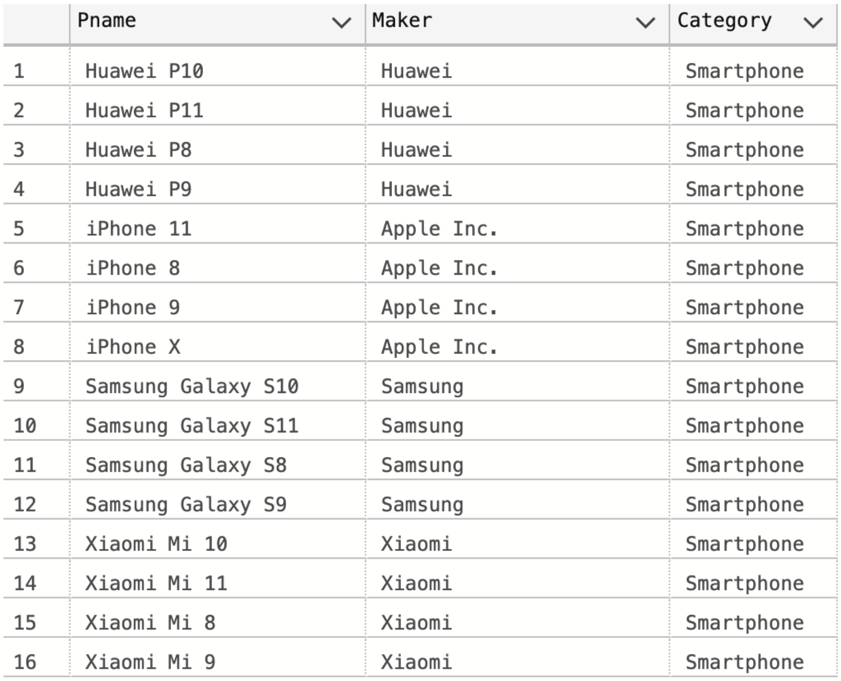
	SPID	StartDate	EndDate	Price
1	1	2021-01-01 00:00:00.000	2021-01-01 00:00:00.000	1902.00
2	1	2021-01-02 00:00:00.000	2021-01-02 00:00:00.000	1933.00
3	1	2021-01-03 00:00:00.000	2021-01-03 00:00:00.000	1396.00
4	1	2021-01-04 00:00:00.000	2021-01-04 00:00:00.000	1794.00
5	1	2021-01-05 00:00:00.000	2021-01-05 00:00:00.000	1518.00
6	1	2021-01-06 00:00:00.000	2021-01-06 00:00:00.000	1807.00
7	1	2021-01-07 00:00:00.000	2021-01-07 00:00:00.000	1181.00
8	1	2021-01-08 00:00:00.000	2021-01-08 00:00:00.000	1901.00
9	1	2021-01-09 00:00:00.000	2021-01-09 00:00:00.000	1234.00

ProductListings



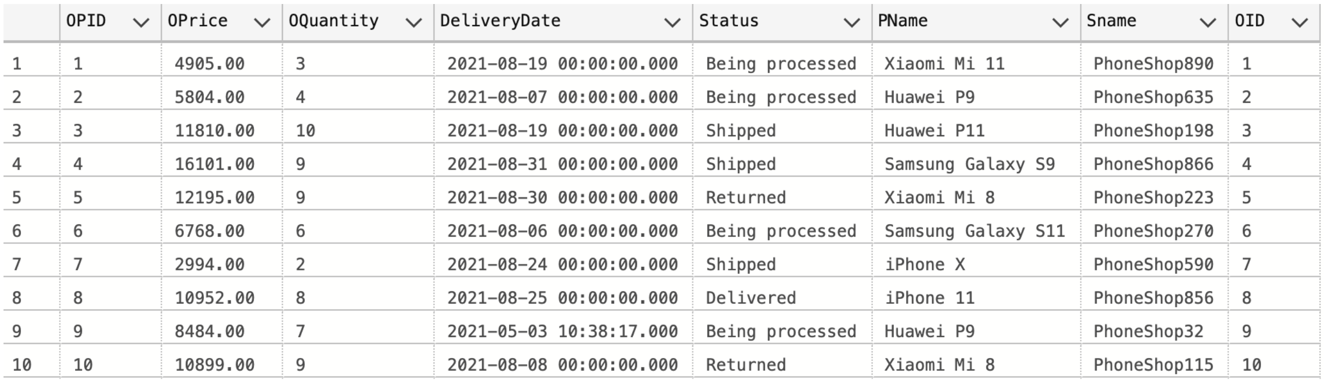
	SPID	SPrice	SQuantity	PName	SName
1	1	1052.00	104	Xiaomi Mi 11	PhoneShop759
2	2	1665.00	11	Samsung Galaxy S11	PhoneShop470
3	3	1209.00	35	iPhone 8	PhoneShop105
4	4	1904.00	165	Xiaomi Mi 8	PhoneShop330
5	5	1832.00	95	Samsung Galaxy S10	PhoneShop888
6	6	1009.00	26	Samsung Galaxy S8	PhoneShop493
7	7	1619.00	105	iPhone 8	PhoneShop891
8	8	1679.00	118	iPhone 8	PhoneShop966

Products



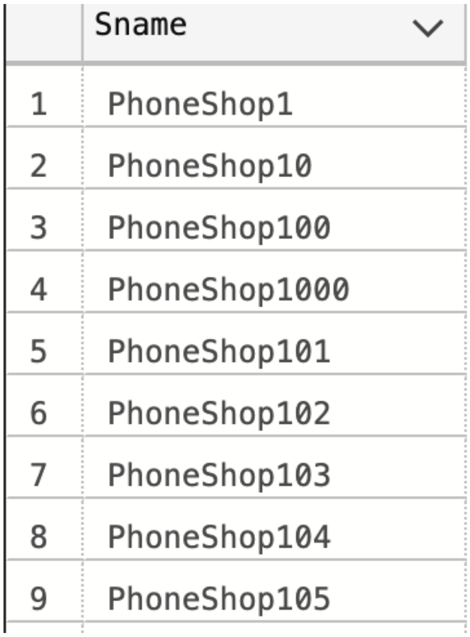
	Pname	Maker	Category
1	Huawei P10	Huawei	Smartphone
2	Huawei P11	Huawei	Smartphone
3	Huawei P8	Huawei	Smartphone
4	Huawei P9	Huawei	Smartphone
5	iPhone 11	Apple Inc.	Smartphone
6	iPhone 8	Apple Inc.	Smartphone
7	iPhone 9	Apple Inc.	Smartphone
8	iPhone X	Apple Inc.	Smartphone
9	Samsung Galaxy S10	Samsung	Smartphone
10	Samsung Galaxy S11	Samsung	Smartphone
11	Samsung Galaxy S8	Samsung	Smartphone
12	Samsung Galaxy S9	Samsung	Smartphone
13	Xiaomi Mi 10	Xiaomi	Smartphone
14	Xiaomi Mi 11	Xiaomi	Smartphone
15	Xiaomi Mi 8	Xiaomi	Smartphone
16	Xiaomi Mi 9	Xiaomi	Smartphone

ProductsInOrders



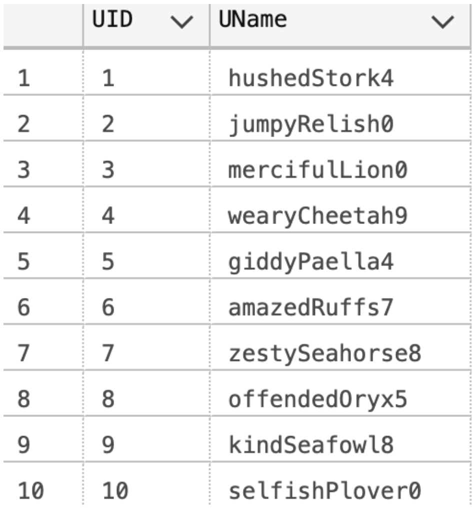
	OPID	OPrice	OQuantity	DeliveryDate	Status	PName	Sname	OID
1	1	4905.00	3	2021-08-19 00:00:00.000	Being processed	Xiaomi Mi 11	PhoneShop890	1
2	2	5804.00	4	2021-08-07 00:00:00.000	Being processed	Huawei P9	PhoneShop635	2
3	3	11810.00	10	2021-08-19 00:00:00.000	Shipped	Huawei P11	PhoneShop198	3
4	4	16101.00	9	2021-08-31 00:00:00.000	Shipped	Samsung Galaxy S9	PhoneShop866	4
5	5	12195.00	9	2021-08-30 00:00:00.000	Returned	Xiaomi Mi 8	PhoneShop223	5
6	6	6768.00	6	2021-08-06 00:00:00.000	Being processed	Samsung Galaxy S11	PhoneShop270	6
7	7	2994.00	2	2021-08-24 00:00:00.000	Shipped	iPhone X	PhoneShop590	7
8	8	10952.00	8	2021-08-25 00:00:00.000	Delivered	iPhone 11	PhoneShop856	8
9	9	8484.00	7	2021-05-03 10:38:17.000	Being processed	Huawei P9	PhoneShop32	9
10	10	10899.00	9	2021-08-08 00:00:00.000	Returned	Xiaomi Mi 8	PhoneShop115	10

Shops



	Sname
1	PhoneShop1
2	PhoneShop10
3	PhoneShop100
4	PhoneShop1000
5	PhoneShop101
6	PhoneShop102
7	PhoneShop103
8	PhoneShop104
9	PhoneShop105

Users



	UID	UName
1	1	hushedStork4
2	2	jumpyRelish0
3	3	mercifulLion0
4	4	wearyCheetah9
5	5	giddyPaella4
6	6	amazedRuffs7
7	7	zestySeahorse8
8	8	offendedOryx5
9	9	kindSeafowl8
10	10	selfishPlover0

Additional efforts

Wrote a Python script to populate the Shiokee database with fake data records to test our SQL queries. Link to the GitHub with the Python script can be found below:

https://github.com/wdwdwdwdwdwdwd/Shiokee-Fake-Data

APPENDIX C: INDIVIDUAL CONTRIBUTION FORM

Name	**Individual Contribution to Submission 1 (Lab 1)**	**Percentage of Contribution**	**Signature**
Nathanael Axel Wibisono	ER Diagram & Written Discussion	16.67%	Axel
Wang Qianteng	ER Diagram & Written Discussion	16.67%	Qianteng
Shreya Ramasubramanian	ER Diagram & Written Discussion	16.67%	Shreya
Tan Ye Quan	ER Diagram & Written Discussion	16.67%	Ye Quan
Bansal Arushi	ER Diagram & Written Discussion	16.67%	Arushi
Wang Dian	ER Diagram & Written Discussion	16.67%	wd

Name	**Individual Contribution to Submission 2 (Lab 3)**	**Percentage of Contribution**	**Signature**
Nathanael Axel Wibisono	ER Diagram & Database Schema	16.67%	Axel
Wang Qianteng	ER Diagram & Database Schema	16.67%	Qianteng
Shreya Ramasubramanian	ER Diagram & Database Schema	16.67%	Shreya
Tan Ye Quan	ER Diagram & Database Schema	16.67%	Ye Quan
Bansal Arushi	ER Diagram & Database Schema	16.67%	Arushi
Wang Dian	ER Diagram & Database Schema	16.67%	wd

Name	**Individual Contribution to Submission 3 (Lab 5)**	**Percentage of Contribution**	**Signature**
Nathanael Axel Wibisono	SQL table creation , SQL queries , documentation	16.67%	Axel
Wang Qianteng	SQL table creation , SQL queries , documentation	16.67%	Qianteng
Shreya Ramasubramanian	SQL table creation , SQL queries , documentation	16.67%	Shreya
Tan Ye Quan	SQL table creation , SQL queries , documentation	16.67%	Ye Quan
Bansal Arushi	SQL table creation , SQL queries , documentation	16.67%	Arushi
Wang Dian	SQL table creation , SQL queries , documentation	16.67%	wd