CZ2007 SS7 Team 3 Lab 5 Discussion Report

SQL DDL Commands for Table Creation

```
CREATE TABLE Shop (
        shopName VARCHAR(50) NOT NULL,
        PRIMARY KEY (shopName)
);

CREATE TABLE Users (
        userID int NOT NULL CHECK (userID>=0),
        name varchar(50) NOT NULL,
        PRIMARY KEY (userID)
);

CREATE TABLE Employee (
        EID INT NOT NULL CHECK(EID>=0),
        employeeName VARCHAR(50) NOT NULL,
        monthlySalary FLOAT NOT NULL CHECK(monthlySalary>0),
        PRIMARY KEY (EID)
);

CREATE TABLE ProductMaker(
        productName VARCHAR(50),
        maker VARCHAR(50) NOT NULL,
        category VARCHAR(50) NOT NULL,
        PRIMARY KEY (productName)
);

CREATE TABLE Product(
        shopName VARCHAR(50),
        productName VARCHAR(50),
        PID INT NOT NULL CHECK (PID>=0),
        Quantity INT NOT NULL CHECK (Quantity>=0),
        averageRating FLOAT DEFAULT NULL CHECK ((averageRating>=1 AND averageRating<=5)OR
        averageRating IS NULL),
        userCount INT NOT NULL DEFAULT 0 CHECK (userCount>=0),
        currentPrice FLOAT NOT NULL CHECK (currentPrice>=0),
        PRIMARY KEY (shopName,productName),
        UNIQUE (shopName,PID),
        FOREIGN KEY (shopName) REFERENCES Shop(shopName)
                ON UPDATE CASCADE,
        FOREIGN KEY (productName) REFERENCES ProductMaker(productName)
                ON UPDATE CASCADE
);
```

```
CREATE TABLE ComplaintR1(
        CID int NOT NULL CHECK (CID>=0),
        userID int NOT NULL,
        FOREIGN KEY (userID) REFERENCES Users(userID)
                ON UPDATE CASCADE,
        EID int,
        FOREIGN KEY (EID) REFERENCES Employee(EID)
                ON UPDATE CASCADE,
        text varChar(MAX) NOT NULL,
        complaintDate  datetime NOT NULL,
        pickUpDate datetime,
        resolveDate datetime,
        CHECK([pickUpDate] > [ComplaintDate]),
        CHECK([resolveDate] > [pickUpDate]),
        PRIMARY KEY (CID),
);

CREATE TABLE ComplaintR2(
        complaintStatus varChar(50) NOT NULL,
        pickUpDate datetime,
        resolveDate datetime,
        CHECK([resolveDate] > [pickUpDate]),
        UNIQUE(resolveDate, pickUpDate),
        CHECK ((resolveDate IS NULL AND pickUpDate IS NULL AND complaintStatus='pending' ) OR
                (resolveDate IS NULL AND pickUpDate IS NOT NULL AND complaintStatus='being
                handled') OR (resolveDate IS NOT NULL AND pickUpDate IS NOT NULL AND
                complaintStatus='addressed'))
);

CREATE TABLE ProductComplaint(
        CID int NOT NULL,
        shopName varChar(50) NOT NULL,
        productName varChar(50) NOT NULL,
        PRIMARY KEY (CID),
        FOREIGN KEY (CID) REFERENCES ComplaintR1(CID)
                ON UPDATE CASCADE,
        FOREIGN KEY (shopName, productName) REFERENCES Product(shopName,productName)
                ON UPDATE CASCADE
);

CREATE TABLE ShopComplaint(
        CID int NOT NULL,
        shopName varChar(50) NOT NULL,
        PRIMARY KEY (CID),
        FOREIGN KEY (CID) REFERENCES ComplaintR1(CID)
                ON UPDATE CASCADE,
        FOREIGN KEY (shopName) REFERENCES Shop(shopName)
                ON UPDATE CASCADE,
);
```

```
CREATE TABLE ShippingInfo(
        shippingAddress VARCHAR(50),
        orderTime DATE,
        shippingCost REAL NOT NULL CHECK (shippingCost>=0),
        PRIMARY KEY (shippingAddress, orderTime)
);

CREATE TABLE Orders(
        OID INT CHECK (OID>=0),
        shippingAddress VARCHAR(50) NOT NULL,
        orderTime DATE NOT NULL,
        userID INT NOT NULL,
        PRIMARY KEY (OID),
        FOREIGN KEY (userID) REFERENCES Users(userID)
                ON UPDATE CASCADE,
        FOREIGN KEY (shippingAddress, orderTime) REFERENCES ShippingInfo(shippingAddress,
        orderTime)
                ON UPDATE CASCADE
);

CREATE TABLE OrderlineR1(
        OID int NOT NULL,
        OLID int NOT NULL,
        shopName varchar(50) NOT NULL,
        productName varchar(50) NOT NULL,
        orderItemQuantity int CHECK (orderItemQuantity>=0),
        orderItemPrice float CHECK (orderItemPrice>=0),
        orderLineStatus varchar(50) NOT NULL,
        deliveryDate DATE,
        dateModified DATE,
        rating FLOAT CHECK (rating>=0 AND rating <=5),
        comment varchar(max),
        UNIQUE (OID, OLID),
        FOREIGN KEY (OID) REFERENCES Orders(OID)
                ON UPDATE CASCADE,
        FOREIGN KEY (shopName,productName) REFERENCES Product(shopName, productName)
                ON UPDATE CASCADE,
        PRIMARY KEY (OID, OLID)
);

CREATE TABLE OrderlineR2(
        OID int NOT NULL,
        userID int,
        UNIQUE (OID, userID),
        FOREIGN KEY (OID) REFERENCES Orders(OID),
        FOREIGN KEY (userID) REFERENCES Users(userID)
);
```

```
CREATE TABLE PriceRecord(
        startDate date NOT NULL,
        endDate date,
        price float NOT NULL CHECK (price>=0),
        shopName  varChar(50) NOT NULL,
        productName  varChar(50) NOT NULL,
        PRIMARY KEY (shopName, productName, startDate),
        FOREIGN KEY (shopName, productName) REFERENCES Product(shopName, productName)
                ON UPDATE CASCADE,
        CHECK ([startDate] <= [endDate])
);
```

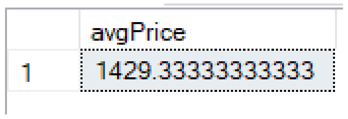
SQL statements to solve the queries in Appendix B and additional queries

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

```
WITH ValidDayPriceRecords AS ( SELECT *, (CASE
        WHEN startDate <= '2021-08-01' AND (endDate >= '2021-08-31' OR endDate IS NULL)
        THEN 31
        WHEN startDate <= '2021-08-01' AND endDate like '2021-08-%' THEN DATEDIFF(day,
        CAST('2021-07-31' AS DATE), endDate)
        WHEN startDate like '2021-08-%' AND (endDate >= '2021-08-31' OR endDate IS NULL)
        THEN DATEDIFF(day, startDate, CAST('2021-09-01' AS DATE))
        WHEN startDate like '2021-08-%' AND endDate like '2021-08-%' THEN DATEDIFF(day,
        startDate, endDate) +1
        ELSE NULL
END ) AS daysValid
FROM PriceRecord P1)

SELECT SUM(price*daysValid)/SUM(daysValid) AS avgPrice
FROM ValidDayPriceRecords
WHERE productName = 'iPhone Xs'
```

Query Output



```
   avgPrice
1  1429.33333333333
```

Explanation

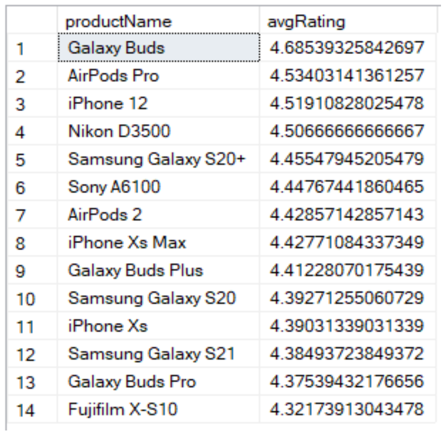
ValidDayPriceRecords is a temporary view which appends a column to the PriceRecord table. This column contains the number of days that falls within the requested time period, 1 August 2021 to 31 August 2021 during which that price record is valid, which is calculated using startDate and endDate columns of PriceRecord, and is NULL if the PriceRecord tuple is not relevant to the query. The price column is the price of the product between startDate and endDate inclusive. From there, we took the weighted average of the prices of ‘iPhone Xs’ weighted by the number of days the price was valid to calculate the average price of ‘iPhone Xs’ in the requested time period.

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

```
WITH
        A1 as (
                SELECT olr1.productName
                FROM OrderLineR1 olr1
                WHERE olr1.Rating = 5 AND olr1.dateModified >= '2021-08-01' AND olr1.dateModified <
                ='2021-08-31'
                GROUP BY olr1.productName
                HAVING COUNT(olr1.Rating) >= 100
),
        CalculatedAvgRatings as (
                SELECT productName, AVG(rating) as avgRating
                FROM OrderlineR1
                GROUP BY productName
)

SELECT DISTINCT productName, car.avgRating
FROM CalculatedAvgRatings car
WHERE car.productName IN (SELECT * FROM A1)
ORDER BY car.avgRating DESC;
```

Query Output



	productName	avgRating
1	Galaxy Buds	4.68539325842697
2	AirPods Pro	4.53403141361257
3	iPhone 12	4.51910828025478
4	Nikon D3500	4.50666666666667
5	Samsung Galaxy S20+	4.45547945205479
6	Sony A6100	4.44767441860465
7	AirPods 2	4.42857142857143
8	iPhone Xs Max	4.42771084337349
9	Galaxy Buds Plus	4.41228070175439
10	Samsung Galaxy S20	4.39271255060729
11	iPhone Xs	4.39031339031339
12	Samsung Galaxy S21	4.38493723849372
13	Galaxy Buds Pro	4.37539432176656
14	Fujifilm X-S10	4.32173913043478

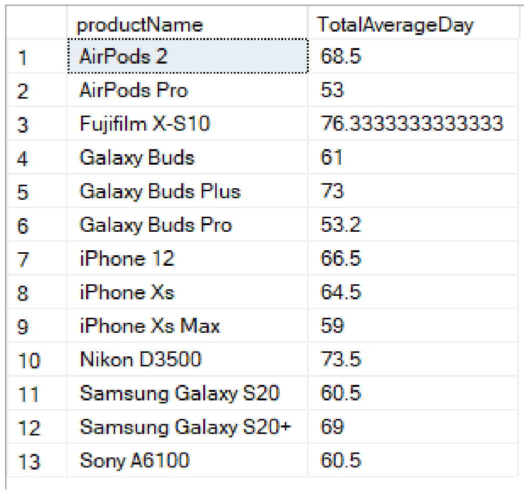
Explanation

A1 is a temporary view that contains one column, productName, and all records in A1 are selected from OrderlinerR1 such that when counting all records that have a rating of ‘5’ in August 2021 grouped by productName, that count is greater than or equal to 100, which represents that there at at least 100 ratings of ‘5’ in August 2021.This generates A1, which is a list of productNames of the products that satisfy the condition given.
CalculatedAvgRatings is a temporary view that contains the productName and average rating of each product over all records with ratings in OrderlineR1.’
Using these two views, the productName and average rating of all products whose productName is in the view A1 are found, completing the query.

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

```
SELECT DISTINCT OL1.productName, AVG(CAST(DATEDIFF(day, O1.orderTime,
OL1.deliveryDate) AS FLOAT)) AS TotalAverageDay
FROM OrderLineR1 OL1, Orders O1
WHERE OL1.orderLineStatus = 'delivered' AND O1.OID = OL1.OID AND O1.orderTime LIKE
'2021-06-%'
GROUP BY OL1.productName;
```

Query Output



	productName	TotalAverageDay
1	AirPods 2	68.5
2	AirPods Pro	53
3	Fujifilm X-S10	76.3333333333333
4	Galaxy Buds	61
5	Galaxy Buds Plus	73
6	Galaxy Buds Pro	53.2
7	iPhone 12	66.5
8	iPhone Xs	64.5
9	iPhone Xs Max	59
10	Nikon D3500	73.5
11	Samsung Galaxy S20	60.5
12	Samsung Galaxy S20+	69
13	Sony A6100	60.5

Explanation

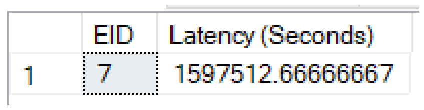
For each product that has at least one tuple in OrderlineR1 joined with Orders for which orderLineStatus is ‘delivered’, representing product has been delivered, and where orderTime is in June, representing a purchase made in June, the average difference in deliveryDate and orderTime is found in days, representing the average time from ordering date to delivery date.

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.

```
WITH LATENCY_TABLE AS (
        SELECT E.EID AS EID, (AVG(CAST(DATEDIFF(second, C1.pickUpDate,
        C1.resolveDate) AS FLOAT))) AS Latency
        FROM ComplaintR1 C1, ComplaintR2 C2, Employee E
        WHERE C1.EID = E.EID AND C1.pickUpDate = C2.pickUpDate AND C1.resolveDate =
        C2.resolveDate AND C2.complaintStatus = 'addressed'
        GROUP BY E.EID
)

SELECT L.EID, L.Latency AS "Latency (Seconds)"
FROM LATENCY_TABLE L
WHERE L.Latency = (SELECT(MIN(L2.Latency)) FROM LATENCY_TABLE L2)
```

Query Output



	EID	Latency (Seconds)
1	7	1597512.66666667

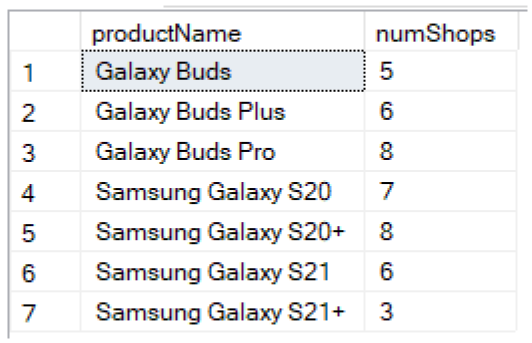
Explanation

LATENCY_TABLE is a temporary view which holds the employee id, EID and the latency of the employee, calculated by the average time taken between picking up and resolving the complaint in seconds, which we treat as the time taken to process a complaint. From this view, we isolated the record where the latency was the minimum, and displayed it in the query, showing the EID and latency of the employee with the smallest latency.

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.

```
SELECT Product.productName,COUNT(DISTINCT shopName) AS numShops
FROM ProductMaker INNER JOIN Product
ON Product.productName=ProductMaker.productName
WHERE maker='Samsung'
GROUP BY Product.productName
```

Query Output



	productName	numShops
1	Galaxy Buds	5
2	Galaxy Buds Plus	6
3	Galaxy Buds Pro	8
4	Samsung Galaxy S20	7
5	Samsung Galaxy S20+	8
6	Samsung Galaxy S21	6
7	Samsung Galaxy S21+	3

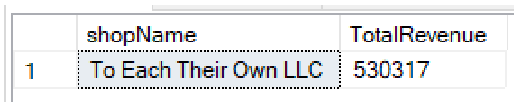
Explanation
This query selects the products where the maker of the product, as defined in ProductMaker, is Samsung, and counts the number of distinct occurrences of shopName in the Product table for each Samsung product which is the number of shops on Shiokee that sell the product.

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

```
WITH RevenueTable AS (
        SELECT DISTINCT ol2.shopName, SUM(ol2.orderItemPrice*ol2.orderItemQuantity) AS
        TotalRevenue
        FROM OrderLineR1 ol2
        WHERE ol2.orderLineStatus = 'delivered' AND ol2.deliveryDate LIKE '2021-08-%'
        GROUP BY shopName
)

SELECT *
FROM RevenueTable
WHERE TotalRevenue = (SELECT(MAX(TotalRevenue)) FROM RevenueTable)
```

Query Output



	shopName	TotalRevenue
1	To Each Their Own LLC	530317

Explanation

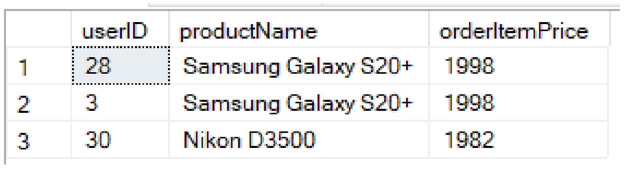
We created the temporary view RevenueTable which contains a list of shops, and their respective total revenues calculated by the sum of orderItemQuantity and orderItemPrice across all records in OrderlineR1 which have been delivered in August 2021. This assumes that the shops only receive payment for products on confirmation of delivery, and that refunds detract from revenue.

Using this view, we selected the tuple with the maximum revenue among all tuples in this view.

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

```
WITH usersComplaints AS(
        SELECT R1.UserID, COUNT(CID) AS ComplaintCount
        FROM Users R1, ComplaintR1 C1
        WHERE R1.UserID= C1.UserID
        GROUP BY R1.UserID
),
UsersWithMostComplaints AS(
        SELECT userID, ComplaintCount
        FROM usersComplaints
        WHERE complaintCount= (SELECT MAX(ComplaintCount) FROM usersComplaints)
)
SELECT u1.userID, ol1.productName, ol1.orderItemPrice
FROM OrderlineR1 ol1, Orders o1, UsersWithMostComplaints u1
WHERE ol1.OID = o1.OID AND o1.userID = u1.userID AND
ol1.orderItemPrice = (
        SELECT MAX(ol2.orderItemPrice)
        FROM OrderlineR1 ol2, Orders o2
        WHERE ol2.OID = o2.OID AND o2.userID = u1.userID
)
```

Query Output



	userID	productName	orderItemPrice
1	28	Samsung Galaxy S20+	1998
2	3	Samsung Galaxy S20+	1998
3	30	Nikon D3500	1982

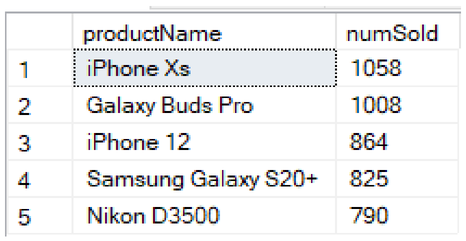
Explanation

userComplaints is a temporary view that counts the number of complaints made by each user identified by their userID by joining the Users and ComplaintR1 tables.
UsersWithMostComplaints is a temporary view which finds the users that made the most complaints using the userComplaints view.
Then, for each user in the list of users who made the most complaints in the UsersWithMostComplaints view, the product they purchased at the highest price was found from the OrderlineR1 table, using a correlated subquery, together with the price they purchased the product at.

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.

```
SET ROWCOUNT 5
SELECT productName, SUM(orderItemQuantity) as numSold
FROM OrderLineR1, Orders
WHERE orderLineR1.OID = Orders.OID AND MONTH(Orders.orderTime) = 8 AND
YEAR(Orders.orderTime) = 2021
GROUP BY productName
HAVING COUNT(DISTINCT Orders.userID) < (SELECT COUNT(*) FROM Users)
ORDER BY numSold DESC
```

Query Output



	productName	numSold
1	iPhone Xs	1058
2	Galaxy Buds Pro	1008
3	iPhone 12	864
4	Samsung Galaxy S20+	825
5	Nikon D3500	790

Explanation

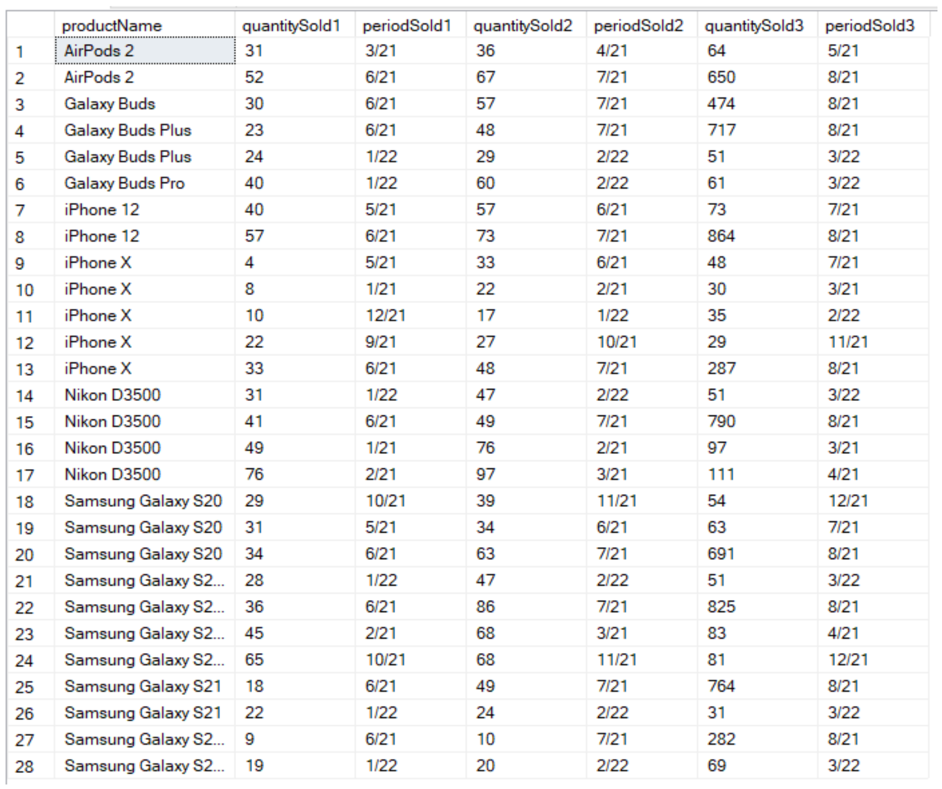
SET ROWCOUNT 5 ensures only the top 5 results show up.
From there, the products and sum of quantities purchased for each product are selected from OrderlineR1 only for orders made in August 2021, as determined by the orderTime in the Order table that was joined with OrderlineR1. This assumes that a purchase is made in August 2021 if it was ordered in August 2021. For quantities of product purchased, it is considered as part of the total if an OrderlineR1 entry was made, regardless of the status of the orderline. The only products that show up are those that have never been purchased by at least 1 user in August 2021. This was done by comparing the total number of distinct users ordering the product against the total number of users to ensure that there are fewer users ordering the product than exist in the database, implying at least 1 user did not purchase the product in August 2021.

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

```
WITH ProductsSold AS (
SELECT productName, CONVERT(DATE, CAST((YEAR(orderTime)*100+MONTH(orderTime))
AS VARCHAR(6))+'01') AS dateSold, SUM(orderItemQuantity) as numSold
FROM OrderlineR1, Orders
WHERE OrderlineR1.OID = Orders.OID
GROUP BY productName, MONTH(orderTime), YEAR(orderTime)
)

SELECT DISTINCT P1.productName, P1.numSold AS quantitySold1,
CONCAT(MONTH(P1.dateSold),'/',YEAR(P1.dateSold)-2000) AS periodSold1, P2.numSold AS
quantitySold2, CONCAT(MONTH(P2.dateSold),'/',YEAR(P2.dateSold)-2000) AS periodSold2,
P3.numSold AS quantitySold3, CONCAT(MONTH(P3.dateSold),'/',YEAR(P3.dateSold)-2000)
AS periodSold3
FROM ProductsSold P1, ProductsSold P2, ProductsSold P3
WHERE P1.productName = P2.productName AND
        P2.productName = P3.productName AND
        DATEADD(month, 1, P1.dateSold) = P2.dateSold AND
        P1.numSold < P2.numSold AND
        DATEADD(month, 1, P2.dateSold) = P3.dateSold AND
        P2.numSold < P3.numSold
```

Query Output



	productName	quantitySold1	periodSold1	quantitySold2	periodSold2	quantitySold3	periodSold3
1	AirPods 2	31	3/21	36	4/21	64	5/21
2	AirPods 2	52	6/21	67	7/21	650	8/21
3	Galaxy Buds	30	6/21	57	7/21	474	8/21
4	Galaxy Buds Plus	23	6/21	48	7/21	717	8/21
5	Galaxy Buds Plus	24	1/22	29	2/22	51	3/22
6	Galaxy Buds Pro	40	1/22	60	2/22	61	3/22
7	iPhone 12	40	5/21	57	6/21	73	7/21
8	iPhone 12	57	6/21	73	7/21	864	8/21
9	iPhone X	4	5/21	33	6/21	48	7/21
10	iPhone X	8	1/21	22	2/21	30	3/21
11	iPhone X	10	12/21	17	1/22	35	2/22
12	iPhone X	22	9/21	27	10/21	29	11/21
13	iPhone X	33	6/21	48	7/21	287	8/21
14	Nikon D3500	31	1/22	47	2/22	51	3/22
15	Nikon D3500	41	6/21	49	7/21	790	8/21
16	Nikon D3500	49	1/21	76	2/21	97	3/21
17	Nikon D3500	76	2/21	97	3/21	111	4/21
18	Samsung Galaxy S20	29	10/21	39	11/21	54	12/21
19	Samsung Galaxy S20	31	5/21	34	6/21	63	7/21
20	Samsung Galaxy S20	34	6/21	63	7/21	691	8/21
21	Samsung Galaxy S2...	28	1/22	47	2/22	51	3/22
22	Samsung Galaxy S2...	36	6/21	86	7/21	825	8/21
23	Samsung Galaxy S2...	45	2/21	68	3/21	83	4/21
24	Samsung Galaxy S2...	65	10/21	68	11/21	81	12/21
25	Samsung Galaxy S21	18	6/21	49	7/21	764	8/21
26	Samsung Galaxy S21	22	1/22	24	2/22	31	3/22
27	Samsung Galaxy S2...	9	6/21	10	7/21	282	8/21
28	Samsung Galaxy S2...	19	1/22	20	2/22	69	3/22

Explanation

The temporary view ProductsSold contains a list of quantities sold of each product, grouped by productName and month and year of orderTime, which is considered as time of purchase. The month and year are converted to a date object with day as 1 for ease of further processing.

Then, 3 such views are joined on productName and their respective periods of sale are ensured to be in consecutive months. While doing so, it is checked that the quantity sold in a preceding period is always less than the quantity sold in the following period. This ensures that the products are being increasingly purchased over at least 3 months. The results are then displayed, showing the product, the respective periods selected, and the respective quantities sold in each period.

Additional efforts

Insertion Triggers: Utilized triggers for insertion of some dependant tables with foreign keys

```
CREATE TRIGGER OrderTrig
ON Orders
INSTEAD OF INSERT
AS
BEGIN
        INSERT INTO ShippingInfo(
                shippingAddress,
                orderTime,
                shippingCost
        )
        SELECT
                i.shippingAddress,
                i.orderTime,
                2 + CASE
                        WHEN MONTH(i.orderTime) > 9 OR MONTH(i.orderTime) < 3 THEN 2
                        ELSE 0
                END + CASE
                        WHEN i.shippingAddress LIKE '%Jurong%' THEN 1
                        ELSE 0
                END
        FROM
                inserted i
        WHERE NOT EXISTS (SELECT * FROM ShippingInfo s WHERE s.orderTime =
i.orderTime AND s.shippingAddress = i.shippingAddress)

        INSERT INTO Orders(
                OID,
                shippingAddress,
                orderTime,
                userID
        )
        SELECT
                i.OID,
                i.shippingAddress,
                i.orderTime,
                i.userID
        FROM
                inserted i
END
```

```
CREATE TRIGGER OrderLineTrig
ON OrderLineR1
INSTEAD OF INSERT
AS
BEGIN
        INSERT INTO OrderLineR1(
                OID,
                OLID,
                shopName,
                productName,
                orderItemQuantity,
                orderItemPrice,
                orderLineStatus,
                deliveryDate,
                dateModified,
                rating,
                comment
        )
        SELECT
                i.OID,
                i.OLID,
                i.shopName,
                i.productName,
                i.orderItemQuantity,
                i.orderItemPrice,
                i.orderLineStatus,
                CASE
                        WHEN i.deliveryDate = '2020-01-01' THEN NULL
                        ELSE i.deliveryDate
                END,
                i.dateModified,
                i.rating,
                CASE
                        WHEN i.rating IS NULL THEN NULL
                        ELSE i.comment
                END
        FROM
                inserted i

END
```

```
CREATE TRIGGER ComplaintTrig
ON ComplaintR1
INSTEAD OF INSERT
AS
BEGIN
        INSERT INTO ComplaintR2(
                complaintStatus,
                pickUpDate,
                resolveDate
        )
        SELECT CASE
                WHEN pickUpDate IS NULL AND resolveDate IS NULL THEN 'pending'
                WHEN pickUpDate IS NOT NULL AND resolveDate IS NULL THEN 'being
                handled'
                ELSE 'addressed'
        END,
                i.pickUpDate,
                i.resolveDate
        FROM inserted i
        WHERE NOT EXISTS (SELECT * FROM ComplaintR2 c WHERE  ((i.pickUpDate IS
        NULL AND c.pickUpDate IS NULL ) OR c.pickUpDate = i.pickUpDate) AND
        ((i.resolveDate IS NULL AND c.resolveDate IS NULL ) OR c.resolveDate =
        i.resolveDate))

        INSERT INTO ComplaintR1(
                CID,
                userID,
                EID,
                text ,
                complaintDate,
                pickUpDate,
                resolveDate
        )
        SELECT
                i.CID,
                i.userID,
                i.EID,
                i.text ,
                i.complaintDate,
                i.pickUpDate,
                i.resolveDate
        FROM inserted i
END
```

Maintenance Triggers: Utilized triggers for the maintenance of the correctness of values in some tables

```
CREATE TRIGGER aveRatingDeleteTrig
ON OrderlineR1
AFTER DELETE
AS
BEGIN
UPDATE Product
SET averageRating=(SELECT AVG(rating) FROM OrderlineR1 WHERE
OrderlineR1.productName=i.productName AND OrderLineR1.shopName=i.shopName),
userCount=(SELECT COUNT(rating) FROM OrderlineR1 WHERE
OrderlineR1.productName=i.productName AND OrderLineR1.shopName=i.shopName)
FROM deleted i
WHERE Product.productName=i.productName AND Product.shopName=i.shopName
END

CREATE TRIGGER aveRatingInsertTrig
ON OrderlineR1
AFTER INSERT
AS
BEGIN
UPDATE Product
SET averageRating=(SELECT AVG(rating) FROM OrderlineR1 WHERE
OrderlineR1.productName=i.productName AND OrderLineR1.shopName=i.shopName),
userCount=(SELECT COUNT(rating) FROM OrderlineR1 WHERE
OrderlineR1.productName=i.productName AND OrderLineR1.shopName=i.shopName)
FROM inserted i
WHERE Product.productName=i.productName AND Product.shopName=i.shopName
END

CREATE TRIGGER aveRatingUpdateTrig
ON OrderlineR1
AFTER UPDATE
AS
BEGIN
UPDATE Product
SET averageRating=(SELECT AVG(rating) FROM OrderlineR1 WHERE
OrderlineR1.productName=i.productName AND OrderLineR1.shopName=i.shopName),
userCount=(SELECT COUNT(rating) FROM OrderlineR1 WHERE
OrderlineR1.productName=i.productName AND OrderLineR1.shopName=i.shopName)
FROM inserted i
WHERE Product.productName=i.productName AND Product.shopName=i.shopName
END
```