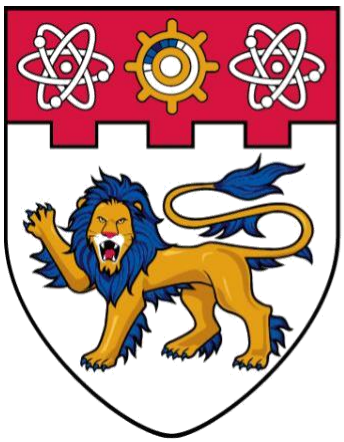

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ2007 Introduction to Databases

Lab 5 Report Submission:
Final Demonstration and Report

Name:

Aide Iskandar Bin Abdul Rahim (U1922414E)
Bryson Teo Yuan Harn (U1920640A)
Chien Yong Qiang (U1920129E)
Kondreddy Saitejareddy (U1923841F)
Pal Aratrika (U1922069F)
Ta Quynh Nga (U1920601L)

TABLE CREATION

```
CREATE TABLE Users (
    UserID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    uName varchar(100) NULL
);

CREATE TABLE Product(
    pName nvarchar(500) NOT NULL PRIMARY KEY,
    category varchar(100) NOT NULL,
    maker nvarchar(100) NOT NULL
);

CREATE TABLE Employee(
    eID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    eName nvarchar(200) NOT NULL,
    salary float(24) NOT NULL DEFAULT 0.0 CHECK(salary >= 0.0)
);

CREATE TABLE Shop(
sName varchar(100) PRIMARY KEY,
);

CREATE TABLE Complaint(
    cID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    complaintext nvarchar(3000) NOT NULL,
    filedatetime datetime NOT NULL DEFAULT getdate(),
    complainstatus varchar(15) DEFAULT 'Pending' Check(complainstatus = 'Pending' OR complainstatus =
'Being handled' OR complainstatus = 'Addressed'),
    handledatetime datetime NULL,
    UserID int FOREIGN KEY REFERENCES Users(UserID) ON DELETE SET NULL,
    eID int FOREIGN KEY REFERENCES Employee(eID) ON DELETE SET NULL
);

CREATE TABLE ComplaintOnShop(
    cID int FOREIGN KEY REFERENCES Complaint(cID) ON DELETE CASCADE,
    sName varchar(100) FOREIGN KEY REFERENCES Shop(sName) ON DELETE CASCADE ON UPDATE CASCADE
    PRIMARY KEY (cID),
);

CREATE TABLE Orders(
    oID int NOT NULL IDENTITY(1,1) PRIMARY KEY,
    shippingcost float(24) NOT NULL DEFAULT 0.0 CHECK(shippingcost>=0.0),
    shippingaddr nvarchar(500) NOT NULL,
    orderdatetime datetime DEFAULT getdate(),
    UserID int FOREIGN KEY REFERENCES Users(UserID) ON DELETE CASCADE,
);

CREATE TABLE ComplaintOnProduct(
    cID int FOREIGN KEY REFERENCES Complaint(cID) ON DELETE CASCADE,
    pName nvarchar(500) FOREIGN KEY REFERENCES Product(pName) ON DELETE CASCADE ON UPDATE CASCADE,
    sName varchar(100) FOREIGN KEY REFERENCES Shop(sName) ON DELETE CASCADE ON UPDATE CASCADE,
    oID int FOREIGN KEY REFERENCES Orders(oID) ON DELETE CASCADE,
    PRIMARY KEY (cID),
);

CREATE TABLE ProductsInShops(
    pName nvarchar(500) FOREIGN KEY REFERENCES Product(pName) ON DELETE CASCADE ON UPDATE CASCADE,
    sName varchar(100) FOREIGN KEY REFERENCES Shop(sName) ON DELETE CASCADE ON UPDATE CASCADE,
    quantity int NOT NULL DEFAULT 0 CHECK(quantity>=0),
    price float(24) NOT NULL DEFAULT 0.0 CHECK(price>=0.0),
    PRIMARY KEY (pName, sName),
);

CREATE TABLE ProductsInOrders(
    pName nvarchar(500) FOREIGN KEY REFERENCES Product(pName) ON DELETE CASCADE ON UPDATE CASCADE,
    sName varchar(100) FOREIGN KEY REFERENCES Shop(sName) ON DELETE CASCADE ON UPDATE CASCADE,
    oID int FOREIGN KEY REFERENCES Orders(oID) ON DELETE CASCADE,
    orderquantity int NOT NULL DEFAULT 0 CHECK(orderquantity>=0),
    orderprice float(24) NOT NULL DEFAULT 0.0 CHECK(orderprice>=0.0),
    deliverystatus varchar(50) NOT NULL DEFAULT 'being processed'
CHECK(deliverystatus = 'being processed' OR deliverystatus = 'shipped' OR deliverystatus = 'delivered'
 OR deliverystatus = 'returned'), deliverydate datetime DEFAULT NULL,
```

```
-- The product in order either has DeliveryStatus = 'delivered' or 'returned' and a DeliveryDateTime.
-- or DeliveryStatus 'being processed' or 'shipped' and DeliveryDateTime = NULL.
CHECK((deliverystatus='delivered' AND deliverydate<>NULL)
OR (deliverystatus='returned' AND deliverydate<>NULL)
OR (deliverystatus='being processed' AND deliverydate=NULL)
OR (deliverystatus='shipped' AND deliverydate=NULL)),
PRIMARY KEY(pName, sName, oID),
);

CREATE TABLE PriceHistory(
    pName nvarchar(500) FOREIGN KEY REFERENCES Product(pName) ON DELETE CASCADE ON UPDATE CASCADE,
    sName varchar(100) FOREIGN KEY REFERENCES Shop(sName) ON DELETE CASCADE ON UPDATE CASCADE,
    startdate datetime NOT NULL DEFAULT getdate(),
    enddate datetime NULL,
    price float(24) NOT NULL DEFAULT 0.0 CHECK(price>=0.0),
    PRIMARY KEY(pName, sName, startdate),
);

CREATE TABLE Feedback(
    pName nvarchar(500) FOREIGN KEY REFERENCES Product(pName) ON DELETE CASCADE ON UPDATE CASCADE,
    sName varchar(100) FOREIGN KEY REFERENCES Shop(sName) ON DELETE CASCADE ON UPDATE CASCADE,
    oID int FOREIGN KEY REFERENCES Orders(oID) ON DELETE CASCADE,
    rating int NOT NULL CHECK(rating >=1 AND rating <=5),
    comment nvarchar(500) NULL,
    feedbackDate datetime DEFAULT getdate(),
    PRIMARY KEY(pName, sName, oID),
);
```

TRIGGERS CREATION

```
-- Update DeliveryDateTime when DeliveryStatus changed.
-- If DeliveryStatus changed to 'Delivered', then DeliveryDateTime=GETDATE()
-- If DeliveryStatus changed to 'Pending', then DeliveryDateTime=NULL

GO
CREATE TRIGGER UpdateDelivery
ON ProductsInOrders
AFTER UPDATE
NOT FOR REPLICATION
AS
BEGIN
    UPDATE  ProductsInOrders
-- DeliveryDateTime will not be updated unless DeliveryStatus changes from 'shipped' to 'delivered'
    SET     deliverydate = CASE

-- If previous DeliveryStatus='shipped' and is changed to 'delivered',
then DeliveryDateTime=GETDATE().
                                    WHEN d.deliverystatus='shipped' AND i.deliverystatus='delivered'
                                        THEN GETDATE()
                                        -- DeliveryDateTime retains the old value
                                    ELSE
                                        d.deliverydate
                                END

-- DeliveryStatus will not be updated unless if follows the sequence: 'being processed'->
'shipped'->'delivered'->'returned'
            , deliverystatus = CASE

-- If previous DeliveryStatus='being processed'. It can only be changed to 'shipped'.
                                    WHEN d.deliverystatus='being processed' AND i.deliverystatus<>'shi
pped'
                                        THEN 'being processed'

-- If previous DeliveryStatus='shipped'. It can only be changed to 'delivered'.
                                    WHEN d.deliverystatus='shipped' AND i.deliverystatus<>'delivered'
                                        THEN 'shipped'

-- If previous DeliveryStatus='delivered'. It can only be changed to 'returned'.
                                    WHEN d.deliverystatus='delivered' AND i.deliverystatus<>'returned'
                                        THEN 'delivered'
-- DeliveryStatus retains updated value
```

```
                                    WHEN d.deliverystatus='returned'
                                        THEN 'returned'

                                    ELSE
                                        i.deliverystatus
                                END
    FROM    ProductsInOrders o, inserted i, deleted d

-- Get all the records that have just been updated, and find the previous value
(inserted gives the updated rows, and deleted gives the previous values for these rows)

    WHERE   o.SName=i.SName AND o.PName=i.PName AND o.oID=i.oID
    AND o.SName=d.SName AND o.PName=d.PName AND o.oID=d.oID;

END

-- Trigger to check the value of “complainstatus” when its value is being updated.
-- When ComplainStatus = “pending”, it can only change to “being handled”.
-- When ComplainStatus = “being handled”, it can only change to “addressed”.
-- When ComplainStatus = “addressed”, it can’t be changed to other values.
-- When ComplainStatus is being updated to “being handled” and employee ID = “null”, then
ComplaintStatus stays the same and won’t be updated.

GO
CREATE TRIGGER ComplainStatus
ON Complaint
AFTER UPDATE
NOT FOR REPLICATION
AS
BEGIN
    UPDATE  Complaint
    SET     complainstatus= CASE

-- If previous complaintstatus ='pending' and employeedID = “null”. It won’t be updated to “being
handled”.
                                WHEN d.complainstatus='pending' AND i.complainstatus='being handled' A
ND i.eID IS NULL
                                        THEN 'pending'

-- If previous complaintstatus ='pending' and employeedID isn’t “null”. It can be changed to ‘being
handled’.
                                 WHEN d.complainstatus='pending' AND i.complainstatus='being handled'
AND i.eID IS NOT NULL
                                        THEN 'being handled'

-- If previous complaintstatus ='being handled'. It can only be changed to ‘addressed’.
                                    WHEN d.complainstatus='being handled' AND i.complainstatus<>'addre
ssed'
                                        THEN 'being handled'

-- If previous complaintstatus = 'pending'. It can only be changed to ‘being handled’.
                                    WHEN d.complainstatus='pending' AND i.complainstatus<>'being handl
ed'
                                        THEN 'pending'

-- If previous complaintstatus = 'addressed'. It can’t be changed to other values.
                                    WHEN d.complainstatus='addressed'
                                        THEN 'addressed'
                                    ELSE
                                        i.complainstatus
                                END,
            handledatetime= CASE
                                    WHEN d.complainstatus='being handled' AND i.complainstatus='addres
sed'
                                        THEN getdate()
                                    ELSE
                                        d.handledatetime

                                END
    FROM    Complaint o, inserted i, deleted d
    WHERE   o.cID=i.cID AND o.cID = d.cID

END
```

```
GO
CREATE TRIGGER NoUserUpdate ON Users
AFTER UPDATE
AS
IF UPDATE(UserID)
BEGIN
   ;THROW 51000, 'You can''t update the primary key UserID', 1;
END

GO
CREATE TRIGGER NoEmployeeUpdate ON Employee
AFTER UPDATE
AS
IF UPDATE(eID)
BEGIN
   ;THROW 51000, 'You can''t update the primary key employeeID', 1;
END

GO
CREATE TRIGGER NoOrderUpdate ON Orders
AFTER UPDATE
AS
IF UPDATE(oID)
BEGIN
   ;THROW 51000, 'You can''t update the primary key orderID', 1;
END
```

SAMPLE QUERIES

```
---- Query 1 ---- DONE
-- Question: Find the average price of iPhone Xs on Sharkee from 1 August 2020 to 31 August 2020.

SELECT AVG(price) AS AvgPrice
FROM PriceHistory
WHERE pName = 'iPhone X'
AND ((startdate >= '2020.08.01 00:00:00' AND startdate < '2020.09.01 00:00:00')
    OR (enddate >= '2020.08.01 00:00:00' AND enddate < '2020.09.01 00:00:00'));

---- Query 2 ---- version 1 DONE
-- Quesition: Find products that received at least 100 ratings of 5 in August 2020,
and order them by their average ratings.

-- create temporary table which stores product name with more than 100 ratings of "5"
SELECT pName INTO Pdts
FROM Feedback
WHERE rating = 5 AND MONTH(feedbackDate) = 8 AND YEAR(feedbackDate) = 2020
GROUP BY pName
HAVING COUNT(rating)>=100;

-- printing the average ratings for these products
SELECT pName, ROUND(AVG(Cast(rating as Float)),2) AS AvgRatings
FROM Feedback
WHERE pName IN(SELECT * FROM Pdts) AND MONTH(feedbackDate) = 8 AND YEAR(feedbackDate) = 2020
GROUP BY pName
ORDER BY AVG(rating) DESC;


---- Query 2 ---- version 2 DONE
-- Question: Find products that received at least 100 ratings of 5 in August 2020,
and order them by their average ratings.

-- Only extract those rows whereby the month of the feedback is August(08) and the year is 2020

-- products that receive feedback in August 2020
WITH F0 AS (
SELECT *
FROM Feedback f
WHERE MONTH(feedbackDate) = 8 AND YEAR(feedbackDate) = 2020),
```

```
-- products that received ratings 5
F1 AS(
SELECT *
FROM F0
WHERE rating=5),

-- products with at least 100 ratings of 5
F2 AS(
SELECT pName, COUNT(rating) AS NumRatings5
FROM F1
GROUP BY pName
HAVING COUNT(rating)>=100)


-- First cast Rating to be a Float so we can get the decimal point values,
then Average all the Rating values

-- Round the average rating to be 2 decimal point

SELECT F2.pName, ROUND(AVG(Cast(F0.rating as Float)),2) AS AvgRatings
FROM F2
JOIN F0 ON F2.pName=F0.pName
GROUP BY F2.pName
ORDER BY AvgRatings DESC;


---- Query 3 ---- DONE
-- Question: For all products purchased in June 2020 that have been delivered,
find the average time from the ordering date to the delivery date.

-- print average time of delivery in hours
SELECT CAST(AVG(DATEDIFF(second,orderdatetime, deliverydate)) AS FLOAT)/3600 AS AvgTimeOnDelivery
FROM Orders, ProductsInOrders
WHERE Orders.oID= ProductsInOrders.oID
AND orderdatetime >= '2020-06-01 00:00:01' AND orderdatetime <= '2020-06-30 23:59:59'
AND (deliverystatus = 'Delivered' OR deliverystatus = 'Returned');



---- Query 4 ---- DONE
-- Question: 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.

SELECT eID, AVG(DATEDIFF(second, filedatetime, handledatetime)) AS AvgLatency INTO LatencyRecord
FROM Complaint
GROUP BY eID;

-- cross-check table
-- SELECT *
-- FROM LatencyRecord;

SELECT eID
FROM LatencyRecord
WHERE AvgLatency = (SELECT MIN(AvgLatency) FROM LatencyRecord);



---- Query 5 ---- DONE
-- Question: Produce a list that contains (i) all products made by Samsung,
and (ii) for each of them, the number of shops on Sharkee that sell the product.

-- Part(i) --
SELECT pName
FROM Product
WHERE maker = 'Samsung';

-- Part(ii) -- version 1
SELECT Product.pName, COUNT(Sname) AS noOfShops
FROM ProductsInShops RIGHT JOIN Product ON Product.pName = ProductsInShops.pName
WHERE maker = 'Samsung'
GROUP BY Product.pName;
```

```
-- Part(ii) -- version 2
SELECT Product.pName AS Product, COUNT(ProductsInShops.PName) AS noOfShops
FROM Product
LEFT JOIN ProductsInShops
ON Product.pName = ProductsInShops.pName
WHERE maker = 'Samsung'
GROUP BY Product.pName;
---- Query 6 ---- DONE
-- Question: Find shops that made the most revenue in August 2020.

WITH A1 AS
(
 SELECT t2.sName, SUM(t2.orderprice*t2.orderquantity) AS revenue
 FROM Orders as t1

 -- Left join on common attribute OrderID of both tables
 LEFT JOIN ProductsInOrders AS t2
 ON t1.oID = t2.oID

 -- OrderDateTime should fall under 2020/08
 WHERE MONTH(t1.orderdatetime) = 8 AND YEAR(t1.orderdatetime) = 2020

-- Group by Shop name with aggregate function SUM of all revenue(OrderPrice*OrderQuantity)
by this shop
 GROUP BY sName
)
-- crosscheck table
-- SELECT *
-- FROM A1;

SELECT sName
FROM A1
WHERE revenue = (SELECT MAX(revenue) FROM A1);



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

-- Counts the total number of complaints each user has made
WITH A1 AS
(
 SELECT UserID, COUNT(UserID) as noOfComplaints
 FROM Complaint
 GROUP BY UserID
),

-- Select the users in A1 that has made the most complaints and their orderID
A2 AS
(
 SELECT t1.UserID, t2.oID
 FROM A1 as t1
 LEFT JOIN Orders as t2
 ON t1.UserID = t2.UserID
 WHERE noOfComplaints = (SELECT MAX(noOfComplaints) FROM A1)
),

-- Find all products that these users in A2 has ever purchased
A3 AS
(
 SELECT t1.UserID, t2.oID, t2.pName, t2.orderprice
 FROM A2 as t1
 LEFT JOIN ProductsInOrders as t2
 ON t1.oID = t2.oID

),

-- Find the most expensive product that each user in A3 has purchased
A4 AS
(
 SELECT UserID, MAX(OrderPrice) as maxProductPrice FROM A3
 GROUP BY UserID
)
```

```
-- Get the product name by matching UserID and the Product price
SELECT t1.UserID, t2.pName, t2.orderprice
FROM A4 as t1
LEFT JOIN A3 as t2
ON t1.UserID = t2.UserID AND t1.maxProductPrice = t2.OrderPrice;
```

---- Query 8 ----
-- Question: Find products that have never been purchased by some users,
but are the top 5 most purchased products by other users in August 2020.

```
-- Create a view with all products sold in August, group by PName, and find total quantity
GO
CREATE VIEW AllProducts AS
(SELECT pName, SUM(orderquantity) AS TotalQuantity
FROM ProductsInOrders PIO
JOIN Orders O ON PIO.oID =O.oID AND
(O.OrderDateTime >= '2020.08.01 00:00:00' AND O.OrderDateTime < '2020.09.01 00:00:00')
GROUP BY PName);

GO
-- View that excludes top product in August
CREATE VIEW NonTop1Products AS
(SELECT *
FROM AllProducts AP
WHERE AP.TotalQuantity <> (SELECT MAX(TotalQuantity)
                        FROM Allproducts AP2));

GO
-- View that excludes top 2 product in August
CREATE VIEW NonTop2Products AS
(SELECT *
FROM NonTop1Products NT
WHERE NT.TotalQuantity <> (SELECT MAX(TotalQuantity)
                        FROM NonTop1Products NT1));

GO
-- View that excludes top 3 product in August
CREATE VIEW NonTop3Products AS
(SELECT *
FROM NonTop2Products NT
WHERE NT.TotalQuantity <> (SELECT MAX(TotalQuantity)
                        FROM NonTop2Products NT1));

GO
-- View that excludes top 4 product in August
CREATE VIEW NonTop4Products AS
(SELECT *
FROM NonTop3Products NT
WHERE NT.TotalQuantity <> (SELECT MAX(TotalQuantity)
                        FROM NonTop3Products NT1));

GO
-- View that excludes top 5 product in August
CREATE VIEW NonTop5Products AS
(SELECT *
FROM NonTop4Products NT
WHERE NT.TotalQuantity <> (SELECT MAX(TotalQuantity)
                        FROM NonTop4Products NT1));

GO
-- View that get top 5 product in August
-- Assume that there can be more than 5 products if products have the same order quantity.
CREATE VIEW TopProducts AS
(SELECT *
FROM AllProducts
EXCEPT
SELECT *
FROM NonTop5Products);
```

```
GO
-- View that gets the number of unique users
CREATE VIEW UserCount AS
SELECT COUNT(*) AS NumUniqueUsers
FROM Users;

GO
-- View that gets the number of unique purchases for each product
CREATE VIEW UniquePurchases AS
SELECT pName, Count(UserID) AS NumUniquePurchases
FROM (SELECT DISTINCT U.UserID, pName
      FROM Users U, Orders O ,ProductsInOrders PIO
      WHERE U.UserID=O.UserID AND O.oID=PIO.oID) AS UniquePurchase
GROUP BY pName;

GO
-- View that gets the products that are not bought by some users, but are top 5 products
SELECT DISTINCT TP.pName
FROM TopProducts TP, UserCount UC,UniquePurchases UP
WHERE TP.pName=UP.pName AND NumUniquePurchases < NumUniqueUsers;

GO
-- additional commands to visualise views : not the query answer, but additional visualisation
for clarity

-- All products
SELECT *
FROM AllProducts
ORDER BY TotalQuantity DESC;

-- Top 5 products
SELECT *
FROM TopProducts
ORDER BY TotalQuantity DESC;

-- Number of unique users
SELECT *
FROM UserCount;

-- Number of  unique purchases for each product
SELECT *
FROM UniquePurchases;

-- Number of  unique purchases for each product with number of unique users added
SELECT TP.pName, NumUniquePurchases, NumUniqueUsers
FROM TopProducts TP, UserCount UC,UniquePurchases UP
WHERE TP.pName=UP.pName;



---- Query 9 ----
-- Question: Find products that are increasingly being purchased over at least 3 months.

GO
-- create a view of the products sold, their quantities, month and year
CREATE VIEW ProductsInMonthYear AS
( SELECT pName, orderquantity, MONTH(orderdatetime) AS Month, YEAR(orderdatetime) AS Year
FROM ProductsInOrders JOIN Orders ON ProductsInOrders.OID=Orders.OID) ;

GO
-- view showing the total quantity of each product for per month, per year
CREATE VIEW PdtMonthlySales AS
(SELECT pName, Month, Year, SUM(orderquantity) AS TotalQuantity
FROM ProductsInMonthYear
GROUP BY pName, Month, Year);

GO
SELECT DISTINCT P1.pName
FROM PdtMonthlySales P1, PdtMonthlySales P2, PdtMonthlySales P3
WHERE(P1.pName=P2.pName AND P2.pName=P3.pName)
AND   ((P1.Year=P2.Year AND (P3.Year-P2.Year)=1 AND P1.Month=11 AND P2.Month=12 AND P3.Month=1)
--Eg Nov 2019, Dec 2019 and Jan 2020
```

```
OR((P2.Year-P1.Year)=1 AND P2.Year=P3.Year AND P1.Month=12 AND P2.Month=1 AND P3.Month=2)
-- Eg Dec 2019, Jan 2020 and Feb 2020

OR(P1.Year=P2.Year AND P2.Year=P3.Year AND (P3.Month-P2.Month)=1 AND (P2.Month-P1.Month)=1))
-- any 3 consecutive months in 2020.

AND (P3.TotalQuantity>P2.TotalQuantity AND P2.TotalQuantity>P1.TotalQuantity);

SELECT *
FROM PdtMonthlySales
ORDER BY pName, Month, Year;
```

TESTING THE TRIGGERS

```
-- Checking that UserID cannot be updated, due to the trigger created
GO
CREATE VIEW CheckUserID AS
(SELECT *
FROM Users
WHERE UserID=3);

GO
SELECT * FROM CheckUserID;

GO
UPDATE Users
SET UserID=1000
WHERE UserID=3;

GO
SELECT * FROM CheckUserID;

-- checking that employee ID cannot be updated due to the trigger created
GO
CREATE VIEW CheckEmpID AS
(SELECT *
FROM Employee
WHERE eID=3);

GO
SELECT * FROM CheckEmpID;

GO
UPDATE Employee
SET eID=1000
WHERE eID=3;

GO
SELECT * FROM CheckEmpID;

-- checking that order ID cannot be updated due to the trigger created
GO
CREATE VIEW CheckOrderID AS
(SELECT *
FROM Orders
WHERE oID=3);

GO
SELECT * FROM CheckOrderID;

GO
UPDATE Orders
SET oID=1000
WHERE oID=3;

GO
SELECT * FROM CheckOrderID;
```

```
-- Checking the UpdateDelivery trigger in ProductsInOrders
-- DELETE FROM ProductsInOrders
-- WHERE pName='Glass Cup' AND sName='Walmart' AND oID=257
-- INSERT [dbo].[ProductsInOrders] ([sName], [pName], [oID], [orderprice], [orderquantity],
[deliverystatus], [deliverydate]) VALUES (N'Walmart', N'Glass Cup', 257, 1, 600, N'being processed',
NULL)

GO
CREATE VIEW CheckStatusOfGlassCup AS
(SELECT * FROM ProductsInOrders
WHERE pName='Glass Cup' AND sName='Walmart' AND oID=257);

GO
SELECT * FROM CheckStatusOfGlassCup; -- check deliverystatus of glass cup
-- update should not happen because status cannot directly change from being processed to delivered
UPDATE ProductsInOrders
SET deliverystatus='delivered', deliverydate=getdate()
WHERE pName='Glass Cup'AND oID=257 AND sName='Walmart'
SELECT * FROM CheckStatusOfGlassCup; --check deliverystatus of glass cup

-- Stepwise update of delivery status from 'being processed' -> 'shipped' -> 'delivered'
and maybe, to 'returned'
UPDATE ProductsInOrders
SET deliverystatus = 'shipped'
WHERE oID=257 AND pName='Glass Cup';
SELECT * FROM CheckStatusOfGlassCup; --check deliverystatus of glass cup

-- 'shipped' -> 'delivered'. DateTime will be set to current date time by default.
Avoids scam of inputting wrong date.
UPDATE ProductsInOrders
SET deliverystatus = 'delivered', deliverydate = '2019.12.01 18:00:00'
WHERE oID=257 AND pName='Glass Cup';
SELECT * FROM CheckStatusOfGlassCup; -- check deliverystatus of glass cup

-- 'delivered' and maybe, to 'returned'. DateTime will not be changed.
UPDATE ProductsInOrders
SET deliverystatus = 'returned', deliverydate = '2019.12.01 18:00:00'
WHERE oID=257 AND pName='Glass Cup';
SELECT * FROM CheckStatusOfGlassCup; -- check deliverystatus of glass cup

-- cannot be changed to others onced DeliveryStatus is 'returned'
UPDATE ProductsInOrders
SET deliverystatus = 'being processed', deliverydate=null
WHERE oID=257 AND pName='Glass Cup';
SELECT * FROM CheckStatusOfGlassCup; -- check deliverystatus of glass cup

GO
-- Trying out the ComplaintStatus trigger
-- Add a complaint to the table just to check if trigger is working
SET IDENTITY_INSERT [dbo].[Complaint] ON
INSERT [dbo].[Complaint] ([cID], [complaintext], [complainstatus], [filedatetime], [UserID], [eID], [h
andledatetime]) VALUES (11, N'Horrible Service.', N'pending', CAST(N'2020-08-
03T00:00:00.000' AS DateTime), 4,Null,Null)
SET IDENTITY_INSERT [dbo].[Complaint] OFF

GO
-- create a view for the Complaint which is to be checked
CREATE VIEW checkempltrigger AS
(SELECT*
from Complaint
WHERE cID=11);

GO
SELECT*
from checkempltrigger
```

```
-- complainstatus cannot be changed from pending to addressed directly
GO
UPDATE Complaint
SET complainstatus='addressed'
WHERE cID=11;

GO
SELECT * FROM checkempltrigger;


-- complaint status cannot be changed to being handled without providing employee ID
GO
UPDATE Complaint
SET complainstatus='being handled'
WHERE cID=11;

GO
SELECT * FROM checkempltrigger;


-- complaint status changing to being handled along with eID
GO
UPDATE Complaint
SET complainstatus='being handled', eID=1
WHERE cID=11;

GO
SELECT * FROM checkempltrigger;

-- when status is changed to addressed then it also adds handledatetime
GO
UPDATE Complaint
SET complainstatus='addressed'
WHERE cID=11;

GO
SELECT * FROM checkempltrigger;

-- if status is addressed, cannot change status to anything else
GO
UPDATE Complaint
SET complainstatus='being handled'
WHERE cID=11;

GO
SELECT * FROM checkempltrigger;
-- delete the record that was added in only to test the trigger
DELETE FROM Complaint
WHERE cID=11;
```