~~2.1) SQL DDL commands for table creation (page 4 of lab manual).~~

2.2) SQL statements for the five queries and your own two additional queries. Each query should be immediately followed by the query output. Briefly explain how the output is obtained.

2.3) Describe how the five constraints are implemented. Show SQL statements if applicable.

2.4) Describe any additional effort done.

2.5) ~~Print all table records.~~ Less Photo

**REPORT ON *“INSERT TITLE HERE”***

**NANYANG TECHNOLOGICAL UNIVERSITY**

**SCHOOL OF COMPUTER SCIENCE AND ENGINEERING**

Logo

Description automatically generated

**By**

**Ang Guang Yao (U2021298J)**

**Chai Wen Xuan (U2020350F)**

**Reeves Chiu (U2023200F)**

**Ingale Omkar (U2020724H)**

**Ivan Pua (U2021665H)**

**In partial fulfilment of the requirements for the course**

**CZ2007: Introduction To Databases**

**Title: *“INSERT TITLE HERE”***

**Tutor: Dr. Ng Wee Keong**

**Submission Date: 2 November 2021**

**TABLE OF CONTENTS**

[**Section 1: Creation of Tables**](#_br26cu8t9nl9) **5**

[Table 1: Customer](#_em7uuv1s72r4) 5

[Table 2: Credit Card](#_d0l26gmpqm4z) 6

[Table 3: Order](#_ejbe8vievu9g) 6

[Table 4: Invoice](#_n042bdcx5n4y) 7

[Table 5: Payment](#_gil8k4hvpabq) 7

[Table 6: Shipment](#_se5yzr2hunkw) 8

[Table 7: Shop](#_pyf4vm1redhp) 8

[Table 8: Product Type](#_1p59rvcilbi) 8

[Table 9: Product](#_u30mtmk14s2f) 9

[Table 10: Photo](#_f7q5btf1z26n) 9

[Table 11: Restricted relationship between shop and product](#_wt03gxu25p73) 10

[Table 12: Order Item](#_7adke6oxwn4q) 11

[**Section 2: Queries**](#_1zfugn6clhi4) **12**

[Query 1:](#_w1ycsjsc4i06) 12

[Query 2](#_oiymmy10vzie) 13

[Query 3](#_7y3k0rfsqiyz) 15

[Query 4](#_olq6z11muihe) 16

[Query 5](#_xdcvd6s6sgt7) 18

[Query 6](#_s31psynkvp27) 18

[Query 7](#_xm07tj3j91o4) 20

[**Section 3: Triggers/Constraint**](#_3t5wl1b1tot8) **22**

[Constraint 1](#_t3wz1ay60szl) 22

[Constraint 2](#_xh3woe1k5445) 23

[Constraint 3](#_j86nvmuu7nnu) 24

[Constraint 4](#_d4hz2wcuft0c) 24

[Constraint 5](#_or4otzd04knm) 26

[**Section 4: Additional Effort**](#_uddpvptoznel) **28**

[Sequence Number](#_bwfrdvbloidq) 28

[Overpay Checking](#_3pxr2z94kkms) 29

[**Section 5: Table records**](#_cha3pho0pqho) **30**

[Record 1: Customer](#_96650xflqrsr) 30

[Record 2: CreditCard](#_k6xvsbyljmyl) 30

[Record 3: OrderTable](#_gyqwd1lg6kef) 30

[Record 4: Invoice](#_fpfhg46udk6) 31

[Record 5: Payment](#_rpxkqfm5sirq) 31

[Record 6: Shop](#_oknw6mitqz25) 31

[Record 7: Shipment](#_9rui47dq49ui) 32

[Record 8: ProductType](#_rauftzci88gj) 33

[Record 9: Product](#_xe35t5sb84ll) 34

[Record 10: RestrictedTo](#_le51whyaw7c3) 37

[Record 11: OrderItem](#_51hczyrw0as4) 38

[Record 12: Photos](#_90cscuhommtn) 40

[**Appendix A - Entity Relationship Diagram**](#_n7xgzh9pzbg3) **43**

[**Appendix D**](#_pa2h4bq7bv72) **44**

# 

# Section 1: Creation of Tables

## Table 1: Customer

| CREATE TABLE Customer(  custID int NOT NULL identity(1,1),  emailAddress VARCHAR(50) NOT NULL,  username VARCHAR(50) NOT NULL,  fullName VARCHAR(100) NOT NULL,  phoneNumber int NOT NULL,  userAddress VARCHAR(200) NOT NULL,  userPassword VARCHAR(50) NOT NULL,  PRIMARY KEY (custID),  UNIQUE (emailAddress, username)  ); |
| --- |

## 

## Table 2: Credit Card

| CREATE TABLE CreditCard(  cardNumber int NOT NULL,  custID int NOT NULL,  expiryDate DATE NOT NULL,  PRIMARY KEY (cardNumber),  FOREIGN KEY(custID) REFERENCES Customer(custID)  ON UPDATE CASCADE ON DELETE CASCADE  ); |
| --- |

## Table 3: Order

| CREATE TABLE OrderTable(  orderID int NOT NULL identity(1,1),  custID int NOT NULL,  orderDate DATE NOT NULL DEFAULT GETDATE(),  *-- 0 - processing, 1 - completed, 2 - canceled*  orderStatus SMALLINT NOT NULL DEFAULT 0,  PRIMARY KEY (orderID),  FOREIGN KEY(custID) REFERENCES Customer(custID)  ON UPDATE CASCADE ON DELETE CASCADE,  *-- itemStatus must be within 0 and 2*  CHECK(orderStatus >= 0 AND orderStatus < 3) ); |
| --- |

## 

## Table 4: Invoice

| CREATE TABLE Invoice(  invoiceNum int NOT NULL identity(1,1),  orderID int NOT NULL,  *-- 0 -not paid, 1 - partially paid, 2 - fully paid*  invoiceStatus SMALLINT NOT NULL DEFAULT 0,  invoiceDate DATE NOT NULL DEFAULT GETDATE(),  PRIMARY KEY(invoiceNum),  FOREIGN KEY(orderID) REFERENCES OrderTable(orderID)  ON UPDATE CASCADE ON DELETE CASCADE,  *-- invoiceStatus must be within 0 and 2*  CHECK(invoiceStatus >= 0 AND invoiceStatus < 3)  ); |
| --- |

## Table 5: Payment

| CREATE TABLE Payment(  paymentID int NOT NULL identity(1,1),  invoiceNumber int NOT NULL,  cardNumber int NOT NULL,  amount REAL NOT NULL DEFAULT 0,  paymentDate DATE NOT NULL DEFAULT GETDATE(),  PRIMARY KEY(paymentID),  FOREIGN KEY(invoiceNumber) REFERENCES Invoice(invoiceNum),  FOREIGN KEY(cardNumber) REFERENCES CreditCard(cardNumber)  ON UPDATE CASCADE ON DELETE CASCADE,  CHECK(amount >= 0)  ); |
| --- |

## Table 6: Shipment

| CREATE TABLE Shipment(  shipmentID int NOT NULL identity(1,1),  shipmentDate DATE NOT NULL DEFAULT GETDATE(),  trackingNum int NOT NULL,  PRIMARY KEY(shipmentID),  ); |
| --- |

## 

## Table 7: Shop

| CREATE TABLE Shop(  shopID int NOT NULL identity(1,1),  sName VARCHAR(20) NOT NULL,  PRIMARY KEY(shopID)  ); |
| --- |

## Table 8: Product Type

| CREATE TABLE ProductType(  ptDescription VARCHAR(50) NOT NULL,  productTypeID int NOT NULL,  parentID int DEFAULT NULL, -- NULL- no parent  PRIMARY KEY(productTypeID),  FOREIGN KEY (parentID) REFERENCES ProductType(productTypeID),  CHECK(productTypeID != parentID) -- productType cannot be its own parent  ); |
| --- |

## Table 9: Product

| CREATE TABLE Product(  productID int NOT NULL identity(1,1),  shopID int NOT NULL,  pName VARCHAR(30) NOT NULL,  color VARCHAR(20) NOT NULL,  price REAL NOT NULL DEFAULT 0,  pDescription VARCHAR(100) DEFAULT NULL,  productTypeID int NOT NULL,  size VARCHAR(10) NOT NULL DEFAULT NULL,  PRIMARY KEY(productID),  FOREIGN KEY (shopID) REFERENCES Shop(shopID),  FOREIGN KEY (productTypeID) REFERENCES ProductType(productTypeID)  ON UPDATE CASCADE ON DELETE CASCADE,  CHECK(price >= 0)  ); |
| --- |

## Table 10: Photo

| CREATE TABLE Photo(  photoID int NOT NULL identity(1,1),  content IMAGE NOT NULL,  productID int NOT NULL,  PRIMARY KEY(photoID),  FOREIGN KEY(productID) REFERENCES Product(productID)  ON UPDATE CASCADE ON DELETE CASCADE  ); |
| --- |

## Table 11: Restricted relationship between shop and product

| CREATE TABLE RestrictedTo(  shopID int NOT NULL,  productTypeID int NOT NULL,  PRIMARY KEY(shopID, productTypeID),  FOREIGN KEY(shopId) REFERENCES Shop(shopId),  FOREIGN KEY(productTypeID) REFERENCES ProductType(productTypeID)  ON UPDATE CASCADE ON DELETE CASCADE  ); |
| --- |

## 

## Table 12: Order Item

| CREATE TABLE OrderItem(  orderID int NOT NULL,  sequenceNum int,  shipmentId int,  productID int NOT NULL,  quantity int DEFAULT 1,  unitPrice REAL NOT NULL DEFAULT 0,  itemStatus SMALLINT DEFAULT 0, -- 0 - processing, 1 - Shipped, 2 - out of stock  PRIMARY KEY(orderID, sequenceNum),  FOREIGN KEY(orderID) REFERENCES OrderTable(orderID),  FOREIGN KEY(shipmentId) REFERENCES Shipment(shipmentID),  FOREIGN KEY(productID) REFERENCES Product(productID)  ON UPDATE CASCADE ON DELETE CASCADE,  CHECK(quantity > 0), -- must at least have 1 qty  CHECK(unitPrice > 0), -- unit price cannot be below 0  CHECK(itemStatus >= 0 AND itemStatus < 3) -- itemStatus must be within 0 and 2  ); |
| --- |

# 

# Section 2: Queries

## Query 1:

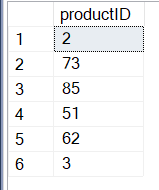
Given a customer by an email address, returns the product ids that have been ordered and paid by this customer but not yet shipped.

| SELECT o.productID FROM Customer c, OrderTable t, OrderItem o, Invoice i  WHERE c.CustID = t.CustID AND t.orderID = o.orderID AND o.orderID = i.orderID  AND i.invoiceStatus = 2  AND o.shipmentId IS NULL  AND c.emailAddress = 'EmilySoto@gmail.com’ |
| --- |

This query joins the Customer, OrderTable, OrderItem and Invoice so that we are able to filter based on the required conditions.

1. i.invoiceStatus = 2 implies that the invoice for the order is fully paid.
2. o.shipmentID will not be NULL if the orderItemis shipped.
3. Query allow us to filter based on c.emailAddress

Output:



## 

## Query 2

Find the 3 best selling product type ids in terms of product quantity sold. The products of concern must be ordered and paid. Whether they have been shipped is irrelevant.

| SELECT TOP 3 p1.productTypeID, SUM(o1.quantity) as "Quantity Sold"  FROM Product p1, Invoice I1, OrderTable or1, OrderItem o1  WHERE  o1.orderID = or1.orderID  AND or1.orderID = I1.orderID  AND or1.orderID = o1.orderID  AND or1.orderStatus <> 2  AND i1.invoiceStatus = 2  AND o1.productID = p1.productID  GROUP BY p1.productTypeID  ORDER BY SUM(o1.quantity) DESC; |
| --- |

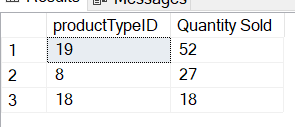
In this query, we will need data from 4 Table which are ( Product, Invoice, OrderTable, OrderItem)

Constraints:

1. We connect OrderItem, OrderTable, Invoice using orderID by writing constraints that their orderID must be the same.
2. Check the OrderTable status which is not canceled( orderStatus != 2 )
3. Check order that is fully paid by invoiceStatus (invoiceStatus == 2)
4. productID in OrderItem connect with productID in Product
5. Group By productTypeID
6. Count total quantity of orderItem of each productTypeId

Order By total quantity and get the TOP 3 product type ID.

Output:



## 

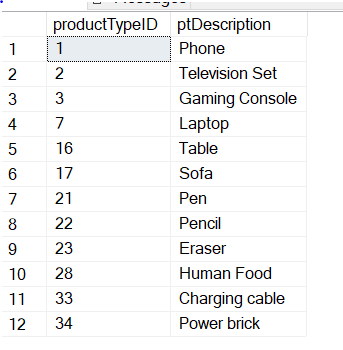
## Query 3

Return the descriptions of all the 2nd level product types. The product types with no parent will be regarded as 1st level product types and their direct child product types will be regarded as 2nd level.

| SELECT p2.productTypeID, p2.ptDescription FROM ProductType p1, ProductType p2  WHERE p1.productTypeID = p2.parentID AND p1.parentID IS NULL; |
| --- |

All 1st level productTypes will have parentID as null. This query joins 2 ProductTypeTable. P1 will contain the 1st level productType, i.e. p1.parentID IS NULL. P2 will contain the second level productType whose parent ProductType does not have a parent, i.e. p2.parentID= p1.productTypeID

Output:



## 

## Query 4

Find 2 product ids that are ordered together the most.

| ;WITH Pair AS (SELECT DISTINCT or1.orderID,o1.productID AS productID1, o2.productID AS productID2  FROM OrderTable or1, OrderItem o1, OrderItem o2  WHERE  or1.orderID = o1.orderID  AND or1.orderID = o2.orderID  AND o1.sequenceNum <> o2.sequenceNum  AND o1.productID < o2.productID)  SELECT Pair.productID1, Pair.productID2, COUNT(\*) AS 'Number Order Together'  FROM Pair  GROUP BY Pair.productID1,Pair.productID2  HAVING COUNT(\*) = (  SELECT TOP 1 COUNT(\*) AS 'Count'  FROM Pair  GROUP BY Pair.productID1,Pair.productID2  ORDER BY 'Count' DESC) |
| --- |

## 

First, we create a view “Pair” to store distinct two productID (productID1 , productID2) from OrderItem1and OrderItem2, with each orderID.

Constraint:

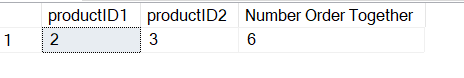
1. orderItem must come from a same orderTable. (Check using orderID)
2. Sequence number cannot be same
3. productID1 < productID2 to avoid repetition in pair.

Next, we select productID1, productID2 and Number Order Together from Pair grouping it by (productID1, productID2)

In the subquery we group Pair again and get the top Count of the number Order Together.

Whichever Group that has the same COUNT with TOP 1 COUNT will be selected from the outer Query.

Output:



## 

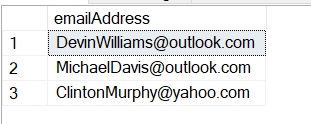
## Query 5

Get 3 random customers and return their email addresses

| SELECT TOP 3 emailAddress FROM Customer  ORDER BY NEWID(); |
| --- |

This query SELECT the top 3 emailAddress from the Customer table after randomly ordering them by NEWID()

Output:

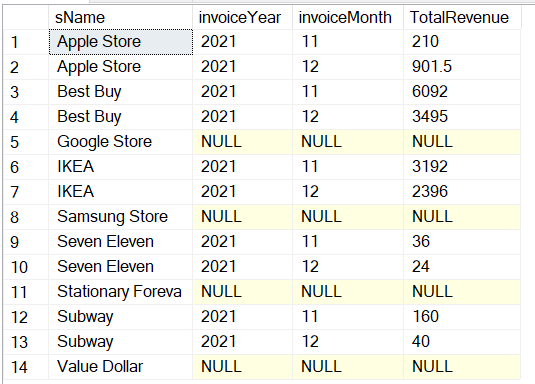


## Query 6

| SELECT sName, invoiceYear, invoiceMonth, SUM(Revenue) AS TotalRevenue  FROM (  SELECT p.shopID, YEAR(i.invoiceDate) AS invoiceYear, MONTH(i.invoiceDate) AS invoiceMonth, SUM(o.unitPrice\*o.quantity) as Revenue FROM OrderItem o, Invoice i, Product p  WHERE o.orderID = i.orderID AND i.invoiceStatus = 2 AND o.productID = p.productID  GROUP BY p.shopID, invoiceDate  ) AS MonthlyRevenue  FULL JOIN Shop s ON s.shopID = MonthlyRevenue.shopID  GROUP BY sName,invoiceYear,invoiceMonth  ; |
| --- |

The inner subquery calculates the total revenue of orderItem (unitPrice\*quantity) for each invoice by shopID and projecting only the invoiceYear and invoiceMonth. The outer query then aggregates the total revenue by shopID, invoiceYear and invoiceMonth resulting in a table with total revenue by month. Stores with no sales will appear once with NULL values.

Output:



## 

## Query 7

| ;WITH Temp AS (  SELECT \*  FROM ( SELECT oi.orderID as "orderID", SUM(oi.unitPrice\*oi.quantity) AS "Total Amount"  FROM OrderItem oi  GROUP BY oi.orderID ) AS A  LEFT JOIN ( SELECT i.orderID as "orderID1", COUNT(\*) AS "Number Of Payment"  FROM Invoice i, Payment p  WHERE i.orderID = p.invoiceNumber  GROUP BY i.orderID) AS B  ON A.orderId=B.orderId1  LEFT JOIN ( SELECT p.invoiceNumber as "orderID2", SUM(p.amount) AS "Paid Amount"  FROM Payment p  GROUP BY p.invoiceNumber) AS C  ON A.orderID=C.orderId2  )  SELECT orderID, ISNULL([Number Of Payment], 0 ) AS "Number Of Payment", ISNULL([Total Amount], 0 ) AS "Total Amount", ISNULL([Paid Amount], 0 ) AS "Paid Amount", ISNULL([Total Amount], 0 ) - ISNULL([Paid Amount], 0 ) AS "Unpaid Amount"  FROM Temp |
| --- |

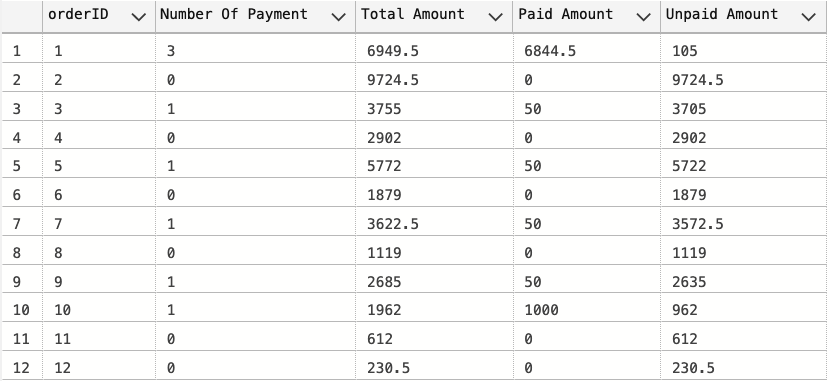
In this query we join three Select statements by Left Outer Join on orderID and store it in a temporary view.

Select Statements:

1. Get Total Amount by SUM(unitPrice\*quantity) from OrderItem and group by orderID
2. Get Number of Payment by counting the row of each row from Payment that is grouped by orderId
3. Get Paid Amount by SUM(amount) from Payment that is grouped by orderID.

Select orderID,Number of Payment, Total Amount, Paid Amount, and calculate a new row (“Unpaid Amount) whose value is Total Amount - Paid Amount from the temporary view.

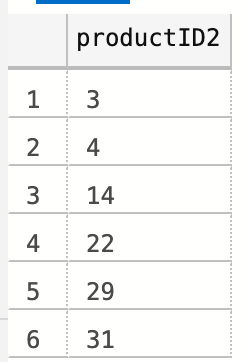
Output:



## Query 8

| ;WITH Pair AS (SELECT DISTINCT or1.orderID AS orderID1, or2.orderID AS orderID2 ,o1.productID AS productID1, o2.productID AS productID2, or1.custID  FROM OrderTable or1, OrderTable or2, OrderItem o1, OrderItem o2  WHERE  or1.orderID = o1.orderID  AND or2.orderID = o2.orderID  AND or1.custID = or2.custID  AND o1.productID < o2.productID  AND o1.productID = 2)  SELECT DISTINCT p.productID2  FROM Pair p, ProductType pt1, ProductType pt2, product pro1, product pro2  WHERE p.productID1 = pro1.productID  AND p.productID2 = pro2.productID  AND (pro1.productTypeID = pro2.productTypeID  OR (pt1.productTypeID = pro1.productTypeID  AND pt2.productTypeID = pro2.productTypeID  AND pt2.parentID = pt1.parentID) ) |
| --- |

**Output:**

****

# Section 3: Triggers/Constraint

## Constraint 1

When the full payment to an invoice is made, the invoice status is changed from ‘issued’ to ‘paid’.

| CREATE TRIGGER FullPayment  ON Payment  AFTER INSERT  AS  BEGIN  SELECT invoiceNum, SUM(OrderItem.unitPrice\*OrderItem.quantity) AS totalCost INTO #TotalCost FROM Invoice, OrderItem  WHERE OrderItem.orderID=Invoice.orderID  GROUP BY invoiceNum;  SELECT invoiceNumber, SUM(amount) AS TotalPaid INTO #Paid FROM Payment GROUP BY invoiceNumber;    UPDATE Invoice SET invoiceStatus = 1  WHERE invoiceNum IN (  SELECT invoiceNum FROM #TotalCost,#Paid  WHERE #Paid.invoiceNumber = #TotalCost.invoiceNum  AND #TotalCost.totalCost > #Paid.TotalPaid);  UPDATE Invoice SET invoiceStatus = 2  WHERE invoiceNum IN (  SELECT invoiceNum FROM #TotalCost,#Paid  WHERE #Paid.invoiceNumber = #TotalCost.invoiceNum  AND #TotalCost.totalCost <= #Paid.TotalPaid);  END |
| --- |

Created a trigger after insertion on Payment, i.e payment is being made.The first SELECT statement retrieves the total cost payable for each invoice. The second SELECT statement retrieves the total payment made for each invoice. The third statement set the invoiceStatus=1 (partially paid) for invoice with total cost > total payment, i.e full payment have not been made. The fourth statement set invoiceStatus=2 (fully paid) for invoice with total payment >= total cost, i.e. full payment has been made.

## Constraint 2

When an order item is shipped, its status is changed from ‘processing’ to ‘shipped’.

Extra: OrderItem cannot be shipped when invoice is not fully paid

| CREATE TRIGGER ShipOrderItem  ON OrderItem  AFTER UPDATE  AS  BEGIN  IF UPDATE(shipmentId)  BEGIN  UPDATE o SET shipmentId=NULL FROM OrderItem o WHERE EXISTS(  SELECT \* FROM inserted  LEFT JOIN Invoice ON Invoice.orderID = inserted.orderID  WHERE (invoiceStatus <> 2 OR invoiceStatus IS NULL) AND inserted.shipmentId IS NOT NULL AND inserted.sequenceNum = o.sequenceNum  AND inserted.orderID = o.orderID  );  UPDATE OrderItem SET itemStatus = 1 WHERE shipmentId IS NOT NULL;  IF EXISTS (SELECT \* FROM inserted  LEFT JOIN Invoice ON Invoice.orderID = inserted.orderID  WHERE (invoiceStatus <> 2 OR invoiceStatus IS NULL) AND inserted.shipmentId IS NOT NULL)  BEGIN  ;THROW 51000, 'Cannot ship orderItem that are not fully paid',1  END  END    END |
| --- |

Create trigger after update on OrderItem. If shipmentID is being updated, i.e. orderItem is shipped, the trigger will check if any of the updated shipmentID are not fully paid. If there are any orderItem that are not fully paid, it will set the shipmentID to NULL. For those orderItem that are successfully shipped, it will update the itemStatus to 1 (shipped).

## Constraint 3

When all the products in an order have been shipped, the order status is changed from ‘processing’ to ‘completed’.

| CREATE TRIGGER ChangeOrderStatus  ON OrderItem  AFTER UPDATE  AS  BEGIN  UPDATE OrderTable SET orderStatus = 1 WHERE orderID NOT IN(SELECT DISTINCT orderID FROM orderItem WHERE itemStatus!=1);  END |
| --- |

Create trigger after update of OrderItem to set orderStatus to 1 (completed) for orders that have all orderItem shipped.

## Constraint 4

There can be at most 3 payments to an invoice, i.e., if the customer chooses to perform partial payments, the 3rd payment must complete the full amount.

| CREATE TRIGGER ThirdPayment  ON Payment  INSTEAD OF INSERT  AS  BEGIN  DECLARE @in INT  DECLARE @cn INT  DECLARE @am REAL  DECLARE @pd DATE  DECLARE @ta REAL -- Total Amount  DECLARE @pa REAL -- Paid Amount  DECLARE @paTemp REAL -- Paid Amount  DECLARE @pn INT -- Payment Number  SELECT @in=invoiceNumber FROM INSERTED  SELECT @cn=cardNumber FROM INSERTED  SELECT @am=amount FROM INSERTED  SELECT @pd=paymentDate FROM INSERTED  – Payment Number  SET @pn = (SELECT COUNT(\*) FROM Payment WHERE Payment.invoiceNumber = @in)  -- Total Amount  SET @ta = (SELECT SUM(OrderItem.unitPrice\*OrderItem.quantity)  FROM OrderItem  WHERE OrderItem.orderID=@in)  -- Paid Amount  SET @paTemp = (SELECT SUM(amount) FROM Payment WHERE Payment.invoiceNumber=@in)  IF (@paTemp > 0)  SET @pa = @paTemp  ELSE  SET @pa = 0  -- Check if payment exceed total amount and check if third payment is full  IF ((@am > (@ta-@pa)) OR (((@pn)=2) AND @am < (@ta-@pa)))  ROLLBACK TRANSACTION  ELSE  INSERT INTO dbo.Payment(invoiceNumber,cardNumber,amount,paymentDate)  VALUES (@in,@cn,@am,@pd);  END |
| --- |

Created a trigger to check if this is the third payment of the order, if it is, check if the amount inserted can pay the order fully or not.

Declared @in (Invoice Number) ,@cn (Card Number), @am (Amonut), @pd (PaymentDate) and select them from INSERTED.

Set Payment Numbers by counting total number of payment made on one Invoice Number

Set Total Amount by SUM(unitPrice\*quantity) from orderItem with the same order

Set Paid Amount by computing SUM(amount) from payment with same Invoice Number

If Payment Number == 2 and if (Amount Inserted < Total Amount - Paid Amount) , insertion will be rollback.

An extra checking also done by checking if (amount inserted > Total Amount - Paid Amount), insertion will also be rollback. This is to prevent overpaying from customers on the order.

## Constraint 5

If an order has been paid, either fully or partially, it can no longer be cancelled, i.e., its status cannot be changed to ‘cancelled’.

| CREATE TRIGGER CancelPaid  ON OrderTable  AFTER UPDATE  AS  BEGIN  IF UPDATE(orderStatus)  IF ((SELECT orderStatus From Inserted) = 2)  IF ((SELECT SUM(p.amount) FROM Payment p WHERE p.invoiceNumber = (SELECT orderID FROM Inserted)) > 0)  BEGIN  ;THROW 51000, 'Cannot cancel order that is paid (partially/fully)',1  UPDATE OrderTable SET orderStatus = (SELECT orderStatus FROM Deleted)  END  END |
| --- |

Created a trigger on OrderTable. Check After Update.

If the orderStatus is updated in the last update, check if the inserted orderStatus.

If the inserted orderStatus is 2, check total paid amount from payment in the same invoice number(orderID)

If the total paid amount is larger than 0, it means that it was paid before and it cannot be canceled.

Set the current orderStatus back to the deleted orderStatus and throw a message.

# Section 4: Additional Effort

## Sequence Number

| CREATE TRIGGER TRG  ON OrderItem  INSTEAD OF INSERT  AS  DECLARE @sid INT  DECLARE @iid INT  DECLARE @shipid INT  DECLARE @pid INT  DECLARE @qty INT  DECLARE @up REAL  DECLARE @istatus SMALLINT  SELECT @iid = orderID FROM INSERTED  SELECT @sid = sequenceNum FROM INSERTED  SELECT @shipid = shipmentId FROM INSERTED  SELECT @pid = productID FROM INSERTED  SELECT @qty = quantity FROM INSERTED  SELECT @up = unitPrice FROM INSERTED  SELECT @istatus = itemStatus FROM INSERTED  --check if inserted AreaID exists in table -for setting SurfaceID  IF NOT EXISTS (SELECT \* FROM OrderItem WHERE orderID = @iid)  SET @sid = 1  ELSE  SET @sid = (SELECT MAX(O.sequenceNum)+1  FROM OrderItem O  WHERE O.orderID = @Iid  )  INSERT INTO OrderItem (orderID, sequenceNum, shipmentId, productID, quantity, unitPrice, itemStatus)  VALUES (@iid, @sid, @shipid, @pid, @qty, @up, @istatus) |
| --- |

For this trigger, we are implementing a constraint on the sequence number attribute in the table OrderItem. Whenever a new OrderItem with a new orderID which does not already exist in the table is inserted, the sequence number will start counting from 1. Otherwise, the sequence number will continue from the previous sequence number respective to its orderID.

## Overpay Checking

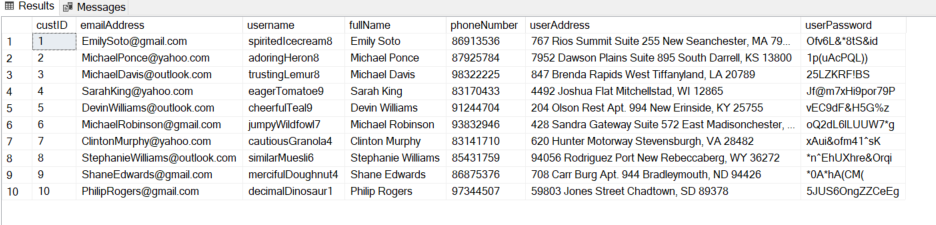
| – Payment Number  SET @pn = (SELECT COUNT(\*) FROM Payment WHERE Payment.invoiceNumber = @in)  -- Total Amount  SET @ta = (SELECT SUM(OrderItem.unitPrice\*OrderItem.quantity)  FROM OrderItem  WHERE OrderItem.orderID=@in)  -- Paid Amount  SET @paTemp = (SELECT SUM(amount) FROM Payment WHERE Payment.invoiceNumber=@in)  IF (@paTemp > 0)  SET @pa = @paTemp  ELSE  SET @pa = 0  -- Check if payment exceed total amount and check if third payment is full  IF ((@am > (@ta-@pa)) OR (((@pn)=2) AND @am < (@ta-@pa)))  ROLLBACK TRANSACTION  ELSE  INSERT INTO dbo.Payment(invoiceNumber,cardNumber,amount,paymentDate)  VALUES (@in,@cn,@am,@pd); |
| --- |

In constraint 4, an additional effort is done by checking if @am (Amount paid in the inserted payment) is larger than the Unpaid Amount @ta - @pa (Total Amount - Paid Amount).

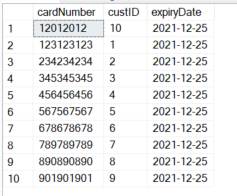
If it is larger than the unpaid amount, it will rollback the transaction.

# Section 5: Table records

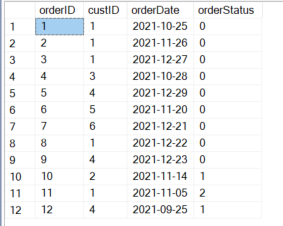
## Record 1: Customer



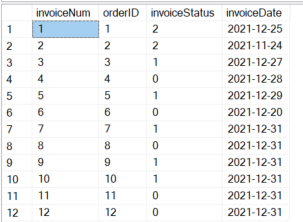
## Record 2: CreditCard



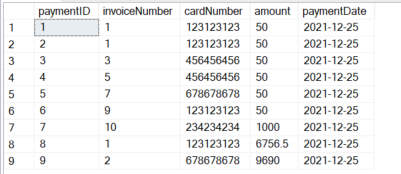
## Record 3: OrderTable



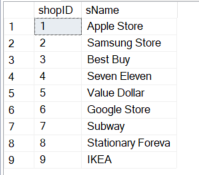
## Record 4: Invoice



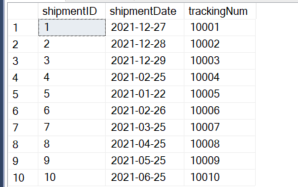
## Record 5: Payment



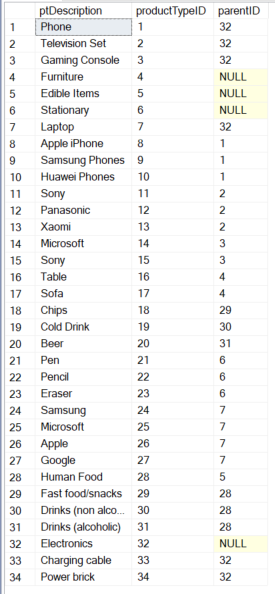
## Record 6: Shop



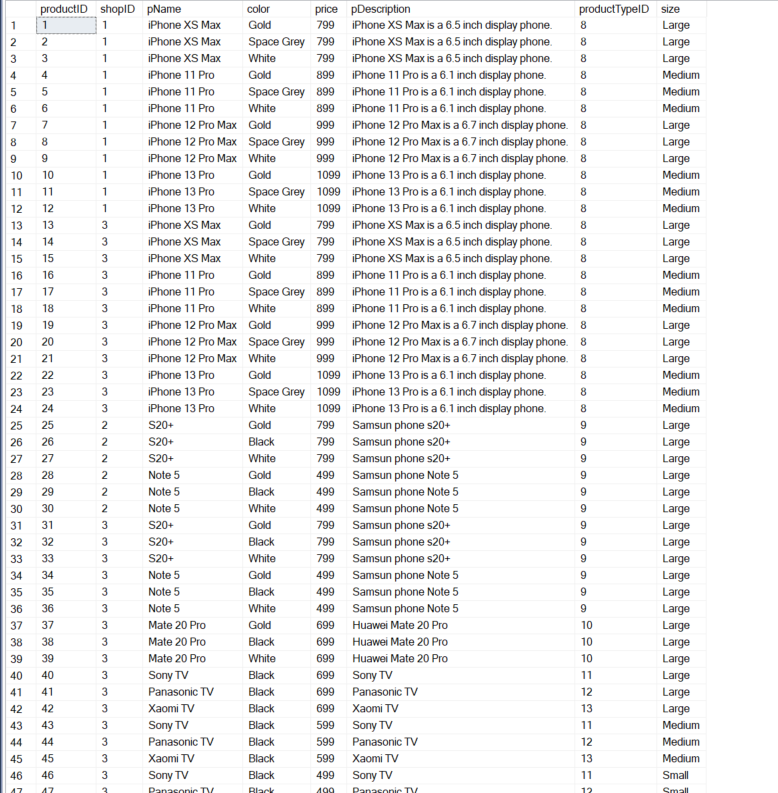
## Record 7: Shipment

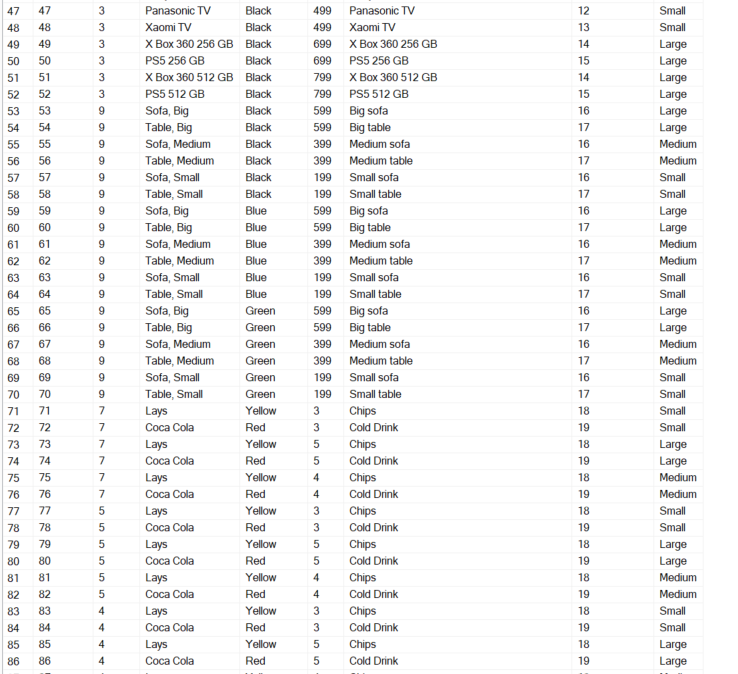


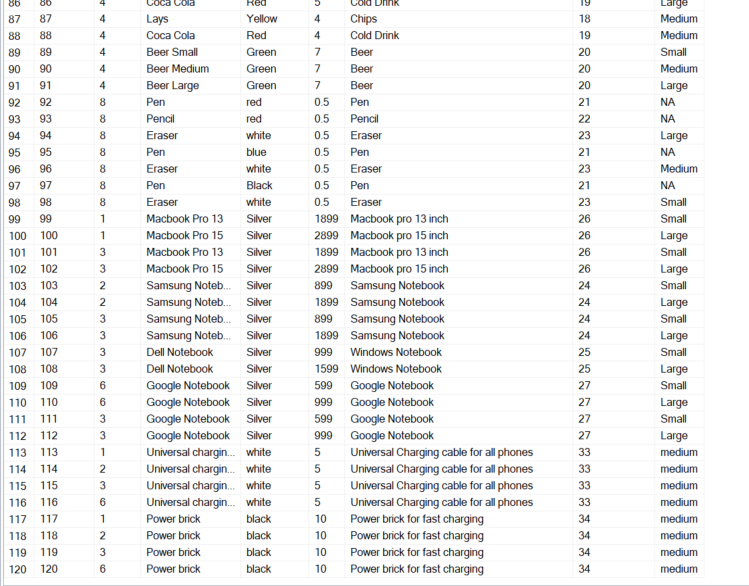
## Record 8: ProductType



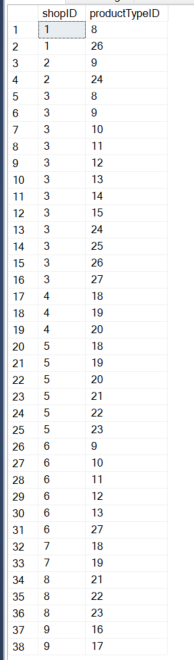
## Record 9: Product



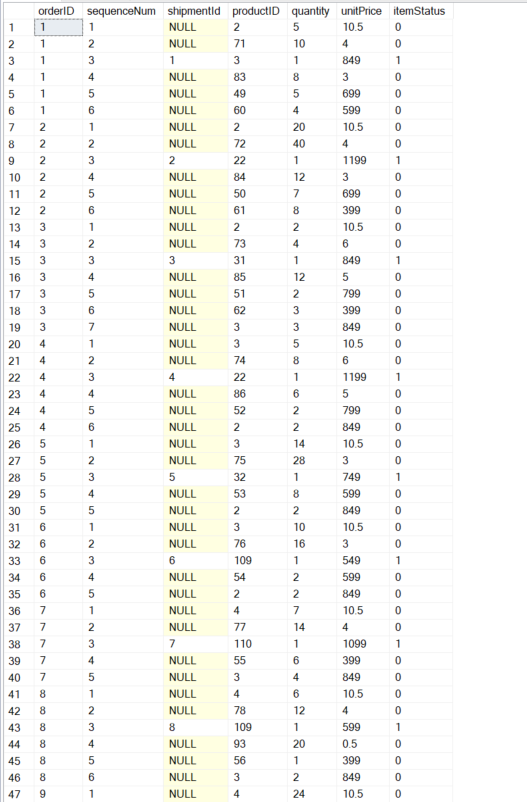


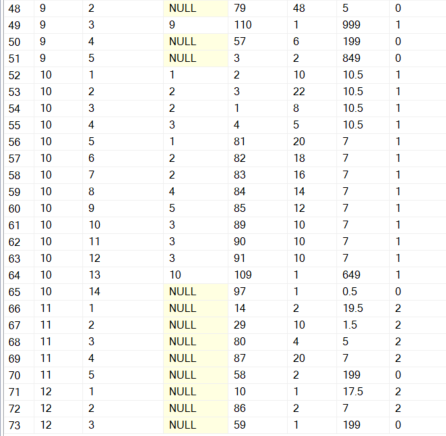


## Record 10: RestrictedTo



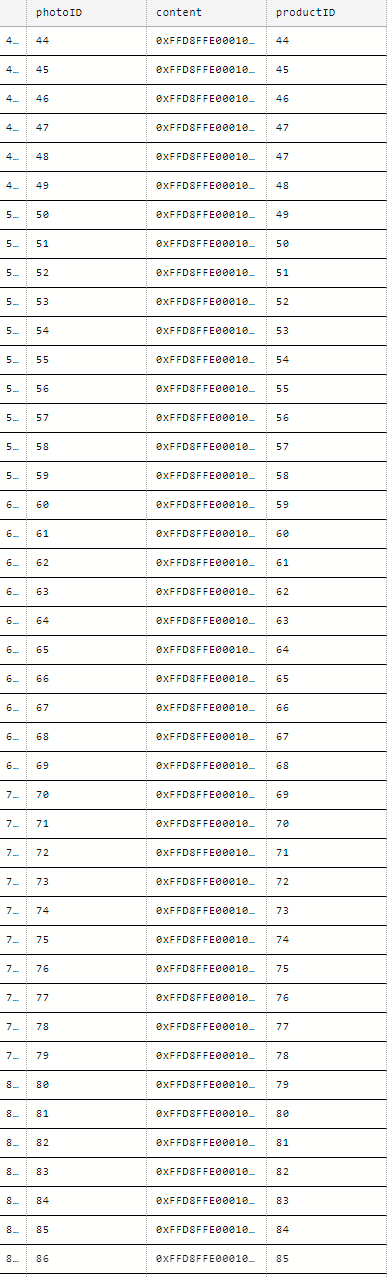
## Record 11: OrderItem

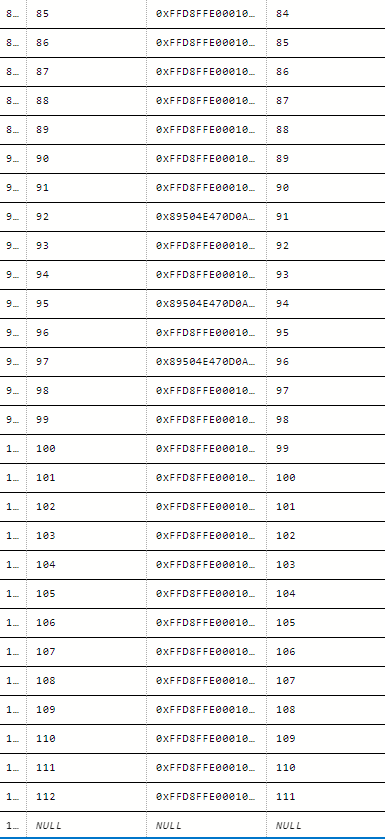




## Record 12: Photos

## 





# Appendix A - Entity Relationship Diagram

## O

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

## 

# Appendix D

| Name | Individual Contribution for Lab 5 | Percentage of Contribution (100% total) |
| --- | --- | --- |
| Ang Guang Yao | Crafted SQL query 1,3,5,6 and constraint 1,2,3. Double check on table creation. Tested the database | 20% |
| Chai Wen Xuan | Crafted SQL query 2,4,7 and constraint 4,5. Test database. | 20% |
| Reeves Chiu | Create table in DB and populate the tables with contents | 20% |
| Ivan Pua | Created tables for the database and also populated them with relevant information. | 20% |
| Ingale Omkar | Help create tables in the database. Also helped populate the database. | 20% |