

## **CZ2007 - Introduction to Databases**



### **Lab 5 Report - SSP3 Group 3**

#### **Group members:**

Lee Yew Chuan Michael, U2021372J

Lau Chen Yi Wynne, U2020016B

Chua Wen Rong Jonathan, U2021875L

Li Ziyang, U1821317D

## Table Of Contents

|   |           |
|---|-----------|
| <b>Queries to CREATE all the tables</b>       | <b>3</b>  |
| <b>Queries to SELECT all the tables</b>       | <b>8</b>  |
| <b>SQL to insert data into tables</b>         | <b>10</b> |
| <b>SQL Queries</b>                            | <b>16</b> |
| APPENDIX B                                    | 16        |
| Query 1                                       | 16        |
| Query 2                                       | 17        |
| Query 3                                       | 18        |
| Query 4                                       | 19        |
| Query 5                                       | 20        |
| Self-Query 1                                  | 21        |
| Self-Query 2                                  | 22        |
| <b>SQL Constraints</b>                        | <b>23</b> |
| APPENDIX C                                    | 23        |
| Required trigger 1: UpdateInvoice             | 23        |
| Required trigger 2: updateOrderItemStatus     | 24        |
| Required trigger 3: updateShipped(Order_item) | 25        |
| Required trigger 4: OnlyThreePayments         | 26        |
| Required trigger 5: NoCancellation            | 28        |
| Trigger 6: noDuplicateProduct(Order_Item)     | 29        |
| Trigger 7: UpdatePaymentStatus(Payment)       | 30        |
| Trigger 8: maintainPartialPayment             | 32        |
| Trigger 9: maintainFullPayment                | 33        |
| Trigger 10: UpdateParentTypeID                | 34        |
| Trigger 11: CheckPaymentBeforeShipment        | 35        |
| Trigger 12: CheckPartialPaymentID             | 36        |
| Trigger 13: CheckFullPaymentID                | 36        |
| Trigger 14: UpdateParentTypeID_2              | 37        |

## Queries to CREATE all the tables

```
DROP TABLE IF EXISTS Customer;
```

```
CREATE TABLE Customer
```

```
(
    customerID INT,
    Email VARCHAR(50) NOT NULL UNIQUE,
    Username VARCHAR(50) NOT NULL UNIQUE,
    fullName VARCHAR(50) NOT NULL,
    Phone# INT NOT NULL,
    Address VARCHAR(50) NOT NULL,
    Password VARCHAR(50) NOT NULL,
    PRIMARY KEY(customerID)
);
```

```
DROP TABLE IF EXISTS Product_Type;
```

```
CREATE TABLE Product_Type
```

```
(
    typeID INT,
    parentTypeID INT,
    typeDescription VARCHAR(255) NOT NULL,
    PRIMARY KEY(typeID),
);
```

```
DROP TABLE IF EXISTS Orders;
```

```
CREATE TABLE Orders
```

```
(
    orderID INT,
    paymentStatus VARCHAR(50) NOT NULL DEFAULT 'notFull' CHECK(paymentStatus IN
('Full', 'notFull') ),
    orderDate DATE NOT NULL,
    orderStatus VARCHAR(50) NOT NULL DEFAULT 'processing' CHECK(orderStatus IN
('processing', 'completed', 'cancelled') ),
    customerID INT NOT NULL,
    PRIMARY KEY(orderID),
    FOREIGN KEY (customerID) REFERENCES Customer(customerID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
DROP TABLE IF EXISTS CreditCard;
CREATE TABLE CreditCard
(
    Card# VARCHAR(50),
    customerID INT NOT NULL,
    expiryDate DATE NOT NULL,
    PRIMARY KEY(Card#),
    FOREIGN KEY (customerID) REFERENCES Customer(customerID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
DROP TABLE IF EXISTS Photo;
CREATE TABLE Photo
(
    photoID INT,
    productID INT NOT NULL,
    PRIMARY KEY(photoID),
    FOREIGN KEY(productID) REFERENCES Product(productID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
DROP TABLE IF EXISTS Shops;
CREATE TABLE Shops
(
    shopID INT,
    shopName varchar(50) NOT NULL,
    PRIMARY KEY(shopID)
);
```

```
DROP TABLE IF EXISTS Can_Sell;
CREATE TABLE Can_Sell
(
    shopID INT NOT NULL,
    typeID INT NOT NULL,
    PRIMARY KEY(shopID, typeID),
    FOREIGN KEY(shopID) REFERENCES Shops(shopID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY(typeID) REFERENCES Product_Type(typeID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
DROP TABLE IF EXISTS Shipment;
CREATE TABLE Shipment
(
    shipmentID INT,
    Tracking# INT NOT NULL UNIQUE,
    shipmentDate DATE NOT NULL,
    PRIMARY KEY(shipmentID)
);
```

```
DROP TABLE IF EXISTS Payment;
CREATE TABLE Payment
(
    paymentID INT,
    paymentDate DATE NOT NULL,
    amount FLOAT NOT NULL CHECK ( amount >= 0 ),
    PRIMARY KEY(paymentID)
);
```

```
DROP TABLE IF EXISTS Full_Payment;
CREATE TABLE Full_Payment
(
    paymentID INT,
    invoice# INT NOT NULL,
    PRIMARY KEY(paymentID),
    FOREIGN KEY(invoice#) REFERENCES Invoice(invoice#)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
DROP TABLE IF EXISTS Partial_Payment;
CREATE TABLE Partial_Payment
(
    paymentID INT,
    invoice# INT NOT NULL ,
    PRIMARY KEY(paymentID),
    FOREIGN KEY(invoice#) REFERENCES Invoice(invoice#)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```
DROP TABLE IF EXISTS Invoice;
CREATE TABLE Invoice
(
    invoice# INT,
    orderID INT NOT NULL UNIQUE,
    invoiceStatus VARCHAR(50) NOT NULL DEFAULT 'issued' CHECK(invoiceStatus IN
('issued', 'paid')),
    invoiceDate DATE NOT NULL,
    PRIMARY KEY(invoice#),
    FOREIGN KEY(orderID) REFERENCES Orders(orderID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

```

DROP TABLE IF EXISTS Product;
CREATE TABLE Product(
    productID INT,
    typeID INT NOT NULL,
    shopID INT NOT NULL,
    Colour VARCHAR(50) NOT NULL,
    productName VARCHAR(50) NOT NULL,
    Description VARCHAR(255) NOT NULL,
    productPrice FLOAT NOT NULL CHECK ( productPrice > 0 ),
    Size VARCHAR(50) NOT NULL,
    PRIMARY KEY(productID),
    FOREIGN KEY(typeID) REFERENCES Product_Type(typeID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY(shopID) REFERENCES Shops(shopID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

```

```

DROP TABLE IF EXISTS Order_Item;
CREATE TABLE Order_Item
(
    sequence# INT,
    orderID INT,
    shipmentID INT,
    productID INT NOT NULL,
    unitPrice FLOAT NOT NULL CHECK ( unitPrice > 0 ),
    orderItemStatus VARCHAR(50) NOT NULL DEFAULT 'processing' CHECK
(orderItemStatus IN ('processing', 'shipped', 'out of stock') ) ,
    Qty INT NOT NULL CHECK ( Qty > 0 ),
    PRIMARY KEY(sequence#, orderID),
    FOREIGN KEY(orderID) REFERENCES Orders(orderID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY(shipmentID) REFERENCES Shipment(shipmentID)
    ON DELETE CASCADE
    ON UPDATE CASCADE,
    FOREIGN KEY(productID) REFERENCES Product(productID)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);

```

## Queries to SELECT all the tables

```
select * from Customer
select * from Invoice
select * from CreditCard
select * from Can_Sell
select * from Full_Payment
select * from Partial_Payment
select * from Payment
select * from Order_Item order by orderID, sequence#
select * from Orders
select * from Photo
select * from Product
select * from Product_Type
select * from Shipment
select * from Shops
```



## Relational Schemas

**Customer**(customerID, Email, Username, fullName, Phone#, Address, Password)

**Product\_Type**(typeID, parentTypeID, typeDescription)

**Product**(productID, typeID, shopID, Colour, productName, Description, productPrice, Size)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Orders**(orderID, paymentStatus, orderDate, orderStatus, customerID)

**Invoice**(Invoice#, orderID, invoiceStatus, invoiceDate)

**Shipment**(shipmentID, Tracking#, shipmentDate)

**CreditCard**(Card#, customerID, expiryDate)

**Photo**(photoID, productID)

**Shops**(shopID, shopName)

**Can\_Sell**(shopID, typeID)

**Payment**(paymentID, paymentDate, amount)

**Full\_Payment**(paymentID, Invoice#)

**Partial\_Payment**(paymentID, Invoice#)

## SQL to insert data into tables

```
INSERT INTO Customer (customerID, Email, Username, fullName, Phone#, Address, Password)
VALUES
(1, 'abc@somemail.com', 'abc123', 'John Doe', 82321832, '222 rainbow ave 5 #06-05', '7c6a180b3689'),
(2, 'nly4562@somemail.com', 'nly4562', 'Ng Liew Ying', 91234234, '435 rainbow street 25, #05-06', '7c6a180b3690' ),
(3, 'ahmh5432@somemail.com', 'ahmh5432', 'Andrew Ho Man Hong', 88382828, '354 jurong street 22, #04-05', '7c6a180b3691'),
(4, 'lyx1234@somemail.com', 'lyx1234', 'Lee Yu Xiang', 90128324, '223 balmoral ave 4, #03-02', '7c6a180b3692'),
(5, 'cltt2341@somemail.com', 'cltt2341', 'Calista Ling Ting Ting', 92104234, '120 balmoral ave 3, #10-02', '7c6a180b3693'),
(6, 'tag5432@somemail.com', 'tag5432', 'Tan Ah Gao', 88452828, '220 jurong street 22, #04-05', '7c6a180b3691'),
(7, 'txt1023@somemail.com', 'txt1023', 'Tan Xiao Tong', 82321832, '222 rainbow ave 5, #06-05', '7c6a180b3688'),
(8, 'cleopang@somemail.com', 'ytf22451', 'Yun tong fong', 93637237, '134 queenstown ave 3, #06-03', '736a127b3695'),
(9, 'JohnTan@somemail.com', 'jtx4567', 'Tan ling John', 93628336, '242 clementi ave 8, #01-08', '7c1a580j3687'),
(10, 'chanyiting@somemail.com', 'cyt3251', 'Chan Yi Tiing', 93341235, '123 Street 32, #06-2A', '736a124d3653');
```

```
INSERT INTO Product_Type (typeID, parentTypeID, typeDescription)
VALUES
(1, NULL, 'These products are consumed regularly and purchased very frequently. '),
(2, NULL, 'Frequency of purchase is comparatively less. These are generally expensive products. '),
(3, 1, 'Very expensive products, thus customers are very selective and picky. '),
(4, 2, 'These are the products consumers buy without any planning and in case of some urgency. '),
(5, NULL, 'Consumers purchase decision gets affected by the promotional activity, deep discounts and it forces customers to indulge in impulse buying. '),
(6, 5, 'This product can cure all illnesses and diseases. '),
(7, 6, 'This product can cure fever and headache'),
(8, 7, 'This product can alleviate headache'),
(9, 4, 'This product smells great. It has a lasting scent for up to 72hrs'),
(10, 4, 'Freshly baked out of the oven. Guaranteed to be crispy and succulent'),
(11, 7, 'A super powered battery guaranteed to last for up to 5 yrs'),
(12, 11, 'Sticky notepads for quick and easy notes. Great for students');
```

```
INSERT INTO Shops (shopID, shopName)
VALUES
(1, 'A'),
(2, 'B'),
(3, 'C'),
(4, 'D'),
(5, 'D');
```

```
INSERT INTO Can_Sell (shopID, typeID)
VALUES
(1, 1),
(2, 4),
(4, 2),
(3, 3),
(5, 5),
(3, 6),
(3, 7),
(1, 8),
(4, 9),
(5, 10),
(5, 11),
(4, 12);
```

```

INSERT INTO Product (productID, typeID, shopID, Colour, productName, Description,
productPrice, Size)
VALUES
(1, 1, 1, 'Green', 'Hydra Bebe Facial Cream', 'Provides an immediate, long-lasting
moisturizing effect. Strengthens the skin barrier and leaves the skin feeling silky, supple
and even softer', 14.4, '40'),
(2, 4, 2, 'Black', 'Men Long Sleeve Loose Shirts', 'Mens spring and autumn shirt', 12.55,
'3X,4XL,5XL'),
(3, 2, 4, 'Rainbow', 'Mercedes Benz Class A Sedan', 'Has a maximum top speed of 134
mph (215 km/h), a curb weight of 2811 lbs (1275 kgs), the Class A Sedan 180 Auto has
a turbocharged Inline 4 cylinder engine, Petrol motor.', 256888, '466.7, 179.6, 144.6'),
(4, 3, 3, 'Black', 'Classic Door Lever', 'A pair of zinc door handles, reversible levers for
right or left hand use. Match matte black finish.', 34.69, '7.5, 5.7, 1.95'),
(5, 4, 2, 'Blue', 'Lays Potato Chips, Barbecue', 'Cooked and seasoned to perfection
before adding the spicy sweet flavour of barbecue sauce', 4.75, 'L'),
(6, 5, 5, 'Red', 'PeppaPig', 'Soft toy.', 7, '466.7, 179.6, 144.6'),
(7, 5, 5, 'Red', 'Classic', 'Match matte black finish.', 34.69, '7.5, 5.7, 1.95'),
(8, 6, 3, 'Blue', 'Chips, Barbecue', 'barbecue sauce', 4.75, 'M'),
(9, 7, 3, 'Red', 'Door', 'A pair of zinc door handles', 80, '7.5, 5.7, 1.95'),
(10, 8, 1, 'Blue', 'Lays', 'spicy sweet flavour of barbecue sauce', 4.75, 'S');

```

```

INSERT INTO Orders (orderID, paymentStatus, orderDate, orderStatus, customerID)
VALUES
(1, 'notFull', '2020-01-30', 'processing', 1),
(2, 'notFull', '2020-01-25', 'processing', 2),
(3, 'notFull', '2020-03-15', 'processing', 3),
(4, 'notFull', '2020-03-10', 'processing', 4),
(5, 'notFull', '2020-05-12', 'processing', 5),
(6, 'notFull', '2020-05-11', 'processing', 6),
(7, 'notFull', '2020-05-20', 'processing', 7),
(8, 'notFull', '2020-05-15', 'processing', 8),
(9, 'notFull', '2020-05-14', 'processing', 9),
(10, 'notFull', '2020-05-10', 'processing', 10);

```

```
INSERT INTO Order_Item (sequence#, orderID, shipmentID, productID, unitPrice,
orderItemStatus, Qty)
```

```
VALUES
```

```
(1, 1, NULL, 1, 2.5, 'processing', 5),
(2, 1, NULL, 2, 3.5, 'processing', 1),
(3, 1, NULL, 3, 20, 'processing', 3),
(1, 2, NULL, 4, 50, 'processing', 1),
(2, 2, NULL, 5, 40, 'processing', 1),
(1, 3, NULL, 3, 20, 'processing', 4),
(2, 3, NULL, 2, 3.5, 'processing', 5),
(1, 4, NULL, 1, 2.5, 'processing', 2),
(1, 5, NULL, 3, 20, 'processing', 1),
(2, 5, NULL, 2, 3.5, 'processing', 3),
(1, 6, NULL, 2, 3.5, 'processing', 5),
(2, 6, NULL, 5, 40, 'processing', 5),
(1, 7, NULL, 6, 12.5, 'processing', 2),
(2, 7, NULL, 7, 20, 'processing', 2),
(1, 8, NULL, 10, 25, 'processing', 1),
(1, 9, NULL, 9, 23.60, 'processing', 5),
(2, 9, NULL, 1, 2.5, 'processing', 5),
(3, 9, NULL, 2, 3.5, 'processing', 5),
(1, 10, NULL, 10, 25, 'processing', 3);
```

```
INSERT INTO Invoice (Invoice#, orderID, invoiceStatus, invoiceDate)
```

```
VALUES
```

```
(1, 1, 'issued', '2020-02-01'),
(2, 2, 'issued', '2020-01-26'),
(3, 3, 'issued', '2020-03-20'),
(4, 4, 'issued', '2020-03-12'),
(5, 5, 'issued', '2020-05-15'),
(6, 6, 'issued', '2020-06-16'),
(7, 7, 'issued', '2020-06-27'),
(8, 8, 'issued', '2020-07-18'),
(9, 9, 'issued', '2020-08-19'),
(10, 10, 'issued', '2020-09-10');
```

```
INSERT INTO CreditCard (Card#, customerID, expiryDate)
VALUES
(1, 1, '2025-03-02'),
(2, 1, '2026-08-01'),
(3, 2, '2024-04-05'),
(4, 3, '2023-05-06'),
(5, 3, '2024-12-06'),
(6, 4, '2025-05-06'),
(7, 5, '2023-06-27'),
(8, 5, '2022-06-27'),
(9, 6, '2025-07-26'),
(10, 7, '2023-02-08'),
(11, 8, '2024-02-01'),
(12, 9, '2022-02-15'),
(13, 10, '2021-06-18'),
(14, 10, '2021-05-06');
```

```
INSERT INTO Photo (photoID, productID)
VALUES
(1, 1),
(2, 1),
(3, 1),
(4, 2),
(5, 3),
(6, 4),
(7, 4),
(8, 5),
(9, 5),
(10, 6),
(11, 7),
(12, 8),
(13, 9),
(14, 9);
```

```
INSERT INTO Shipment (shipmentID, Tracking#, shipmentDate)
VALUES
(1, 1, '2020-01-02'),
(2, 2, '2020-01-02'),
(3, 3, '2020-02-03'),
(4, 4, '2020-02-04'),
(5, 5, '2020-04-05'),
(6, 6, '2020-02-03'),
(7, 7, '2020-05-04'),
(8, 8, '2020-12-05');
```

```
INSERT INTO Full_payment (paymentID, Invoice#)
VALUES
(1, 3),
(2, 4);
```

```
INSERT INTO Partial_Payment (paymentID, Invoice#)
VALUES
(3, 1),
(4, 1);
```

```
UPDATE Payment
SET    amount = [enter the amount here]
WHERE paymentID = [enter payment ID here]
```

# SQL Queries

## APPENDIX B

### 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.

Relevant Schemas:

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Orders**(orderID, paymentStatus, orderDate, orderStatus, customerID)

**Customer**(customerID, Email, Username, fullName, Phone#, Address, Password)

Query:

```
SELECT DISTINCT productID
FROM Order_Item
WHERE orderItemStatus <> 'shipped'
AND orderID in (
    SELECT orderID
    FROM Orders
    WHERE paymentStatus = 'Full'
    AND customerID = (
        SELECT customerID
        FROM Customer
        WHERE Email = 'abc@somemail.com'
    )
);
```



## Query 2

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

Assumption:

- We assume paid means fully paid order.

Relevant Schemas:

**Product**(productID, typeID, shopID, Colour, productName, Description, productPrice, Size)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Orders**(orderID, paymentStatus, orderDate, orderStatus, customerID)

Query:

```
SELECT TOP 3 typeID, SUM(Qty) AS TotalQty
FROM Product, Order_Item, Orders
WHERE paymentStatus = 'Full'
AND orderStatus <> 'cancelled'
AND Order_Item.orderID = Orders.orderID
AND Order_Item.productID = Product.productID
GROUP BY typeID
ORDER BY TotalQty DESC;
```

### 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.

Relevant Schemas:

**Product\_Type**(typeID, parentTypeID, typeDescription)

Query:

```
SELECT typeDescription
FROM Product_Type
WHERE parentTypeID IN (SELECT typeID
                       FROM Product_Type
                       WHERE parentTypeID IS NULL
                       );
```

## Query 4

- Find 2 product ids that are ordered together the most.

Assumption:

- We assume that even if a customer has cancelled the order, the products are still considered ordered together.

Relevant Schemas:

**Order\_Item**(Sequence#, orderId, shipmentID, productID, unitPrice, orderItemStatus, Qty)

Query:

```
SELECT  OI1.productID, OI2.productID
FROM    Order_Item as OI1, Order_Item as OI2
WHERE   OI1.orderID = OI2.orderID
AND OI2.orderID = OI1.orderID
AND OI1.orderID = OI2.orderID
AND OI1.productID < OI2.productID
GROUP BY OI1.productID, OI2.productID
HAVING COUNT(*) >= ALL (
        SELECT  COUNT(*)
        FROM    Order_Item as OI3, Order_Item as OI4
        WHERE   OI3.orderID = OI4.orderID
        AND OI3.productID < OI4.productID
        GROUP BY OI3.productID, OI4.productID);
```

## Query 5

- Get 3 random customers and return their email addresses.

Relevant Schemas:

**Customer**(customerID, Email, Username, fullName, Phone#, Address, Password)

Query:

```
SELECT TOP 3 Email  
FROM Customer  
ORDER BY NEWID();
```

## Self-Query 1

- Find the customer who has spent the most on this platform (qty \* unitPrice)

Relevant Schemas:

**Customer**(customerID, Email, Username, fullName, Phone#, Address, Password)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Order**(orderID, paymentStatus, orderDate, orderStatus, customerID)

Query:

```
SELECT      C.customerID
FROM        Customer as C, Order_item As OI, Orders as O
WHERE       C.customerID = O.customerID AND O.orderID = OI.orderID
GROUP BY    C.customerID
HAVING      SUM(OI.Qty * OI.unitPrice) >= ALL (
            SELECT  SUM(OI2.Qty * OI2.unitPrice) AS TotalSpentInDuration
            FROM      Customer as C2, Order_item As OI2, Orders as O2
            WHERE     C2.customerID = O2.customerID
            AND O2.orderID = OI2.orderID
            GROUP BY  C2.customerID
            );
```

## Self-Query 2

- Find the most shipped order item of the month

Relevant Schemas:

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Shipment**(shipmentID, Tracking#, shipmentDate)

Query:

```
SELECT OI.productID
FROM   Order_item as OI, shipment as S
WHERE  S.shipmentID = OI.shipmentID AND S.shipmentDate >= [start of month] AND
S.shipmentDate <= [end of month]
GROUP BY OI.productID
HAVING SUM(Qty) >= ALL (SELECT SUM(Qty) AS TotalQty
                        FROM   Order_item as OI, shipment as S
                        WHERE  S.shipmentID = OI.shipmentID AND
                        S.shipmentDate >= [start of month] AND S.shipmentDate
                        <= [end of month]
                        GROUP BY OI.productID );
```

# SQL Constraints

## APPENDIX C

### Required trigger 1: UpdateInvoice

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

Relevant Schemas:

**Invoice**(Invoice#, orderID, invoiceStatus, invoiceDate)

**Payment**(paymentID, paymentDate, amount)

**Full\_Payment**(paymentID, Invoice#)

**Partial\_Payment**(paymentID, Invoice#)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Order**(orderID, paymentStatus, orderDate, orderStatus, customerID)

Query:

```
CREATE TRIGGER updateInvoice ON Order
```

```
AFTER UPDATE
```

```
AS
```

```
IF ((SELECT inserted.paymentStatus
```

```
    FROM inserted , Orders
```

```
    WHERE inserted.orderID = orders.orderID) = 'Full'
```

```
)
```

```
BEGIN
```

```
    UPDATE Invoice
```

```
    SET Invoice.invoiceStatus = 'paid'
```

```
    FROM inserted, Orders, Invoice
```

```
    WHERE Orders.orderID = inserted.orderID
```

```
    AND Invoice.orderID = Orders.orderID
```

```
END;
```

## Required trigger 2: updateOrderItemStatus

- When an order item is shipped, its status is changed from 'processing' to 'shipped'.

Relevant Schemas:

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

Query:

```
CREATE TRIGGER updateOrderItemStatus ON Order_Item
AFTER UPDATE
AS
IF EXISTS (
    SELECT Order_Item.orderID
    FROM inserted, Order_Item
    WHERE Order_Item.shipmentID IS NOT NULL
    AND inserted.sequence# = Order_Item.sequence#
    AND inserted.orderID = Order_Item.orderID
    AND Order_Item.orderItemStatus = 'processing'
)
BEGIN
    UPDATE Order_Item
    SET Order_Item.orderItemStatus = 'shipped'
    FROM inserted
    WHERE inserted.orderID = Order_Item.orderID
    AND inserted.sequence# = Order_Item.sequence#
END;
```



### Required trigger 3: updateShipped(Order\_item)

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

Relevant Schemas:

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

Query:

```
CREATE TRIGGER updateShipped ON Order_item
AFTER UPDATE
AS
IF NOT EXISTS (
    SELECT Order_item.orderID
    FROM   INSERTED AS I, Order_item
    WHERE  I.orderID = Order_item.orderID AND
           Order_item.orderItemStatus <> 'shipped'
)
BEGIN
    UPDATE      Orders
    SET         orderStatus = 'completed'
    FROM        inserted
    WHERE       Orders.orderID = inserted.orderID
END;
```

## Required trigger 4: OnlyThreePayments

- 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.

**Partial\_Payment**(paymentID, Invoice#)

**Invoice**(Invoice#, orderID, invoiceStatus, invoiceDate)

**Order**(orderID, paymentStatus, orderDate, orderStatus, customerID)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Full\_Payment**(paymentID, Invoice#)

**Payment**(paymentID, paymentDate, amount)

Query:

AFTER UPDATE

AS

```
IF (      ( (SELECT COUNT(DISTINCT PP2.paymentID)
              FROM INSERTED AS INS, Partial_Payment AS PP, Orders AS O, Invoice AS I,
              Partial_Payment AS PP2
              WHERE INS.paymentID = PP.paymentID
              AND I.Invoice# = PP.Invoice#
              AND I.invoice# = PP2.invoice#) = 3 )
      AND
      ( (SELECT SUM(Payment.amount)
        FROM inserted, Payment, Invoice, Partial_Payment
        WHERE Partial_Payment.paymentID = Payment.paymentID
        AND Partial_Payment.Invoice# = Invoice.Invoice#
        AND Invoice.Invoice# = (   SELECT I.Invoice#
                                  FROM Invoice AS I, Partial_Payment AS PP
                                  WHERE inserted.paymentID = PP.paymentID
                                  AND I.Invoice# = PP.Invoice#)
      ) < (SELECT SUM(Order_Item.Qty * Order_Item.unitPrice)
        FROM Order_Item, Partial_Payment, Orders, Invoice, inserted
        WHERE Orders.orderID = Order_Item.orderID
        AND inserted.paymentID = Partial_Payment.paymentID
        AND Invoice.orderID = Order_Item.orderID
        AND Invoice.Invoice# = ( SELECT I.Invoice#
                                  FROM Invoice AS I, Partial_Payment AS PP
                                  WHERE inserted.paymentID = PP.paymentID
                                  AND i.Invoice# = PP.Invoice#)
      )
    )
```

```
BEGIN  
ROLLBACK TRANSACTION  
END;
```

## Required trigger 5: NoCancellation

- 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'.

**Invoice**(Invoice#, orderID, invoiceStatus, invoiceDate)

**Order**(orderID, paymentStatus, orderDate, orderStatus, customerID)

Query:

CREATE TRIGGER NoCancellation ON Orders

AFTER UPDATE

AS

IF EXISTS (

    SELECT \*

    FROM Orders as O, Invoice as I

    WHERE O.orderStatus = 'cancelled' AND O.orderID = I.orderID

    AND (I.invoice# IN (

        (SELECT Invoice#

        FROM Full\_Payment AS FP)

        UNION

        (SELECT Invoice#

        FROM Partial\_Payment AS P)

    )

    )

)

ROLLBACK TRANSACTION;

### Trigger 6: noDuplicateProduct(Order\_Item)

Ensure that when a customer orders more of the same productID in the same order, it gets added to the existing row instead of creating a new entry.

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

Query:

```
CREATE TRIGGER noDuplicateProduct ON Order_Item  
AFTER INSERT
```

```
AS
```

```
DECLARE @sequenceNum INT;
```

```
DECLARE @orderNum INT;
```

```
DECLARE @Qty INT;
```

```
DECLARE @productID INT;
```

```
SELECT @sequenceNum = inserted.sequence#, @orderNum = inserted.orderID, @Qty =  
        inserted.Qty, @productID = inserted.productID
```

```
FROM inserted
```

```
IF EXISTS (
```

```
    (SELECT *
```

```
    FROM Order_Item
```

```
    WHERE Order_item.orderID = @orderNum
```

```
    AND Order_item.productID = @productID
```

```
    AND Order_Item.sequence# <> @sequenceNum) )
```

```
BEGIN
```

```
    DELETE FROM Order_Item
```

```
    WHERE Order_Item.orderID = @orderNum
```

```
    AND Order_Item.sequence# = @sequenceNum
```

```
    UPDATE Order_Item
```

```
    SET Order_Item.Qty = Order_Item.Qty+ @Qty
```

```
    FROM Order_Item
```

```
    WHERE Order_item.orderID= @orderNum
```

```
    AND Order_item.productID = @productID
```

```
END;
```

## Trigger 7: UpdatePaymentStatus(Payment)

Update paymentStatus in Order table to full when the total amount paid for the order is equal to the full amount

**Partial\_Payment**(paymentID, Invoice#)

**Invoice**(Invoice#, orderID, invoiceStatus, invoiceDate)

**Order**(orderID, paymentStatus, orderDate, orderStatus, customerID)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Full\_Payment**(paymentID, Invoice#)

**Payment**(paymentID, paymentDate, amount)

Query:

CREATE TRIGGER UpdatePaymentStatus ON Payment

AFTER INSERT, UPDATE

AS

IF(

```
(SELECT SUM(Payment.amount)
  FROM inserted, Payment, Invoice, Partial_Payment
 WHERE Partial_Payment.paymentID = Payment.paymentID
 AND Partial_Payment.Invoice# = Invoice.Invoice#
 AND Invoice.Invoice# = (
   SELECT I.Invoice#
   FROM Invoice AS I, Partial_Payment AS PP
  WHERE inserted.paymentID = PP.paymentID
  AND I.Invoice# = PP.Invoice#
 )
)=
```

```
(SELECT SUM(Order_Item.Qty * Order_Item.unitPrice)
  FROM Order_Item, Partial_Payment, Orders, Invoice, inserted
 WHERE Orders.orderID = Order_Item.orderID
 AND inserted.paymentID = Partial_Payment.paymentID
 AND Invoice.orderID = Order_Item.orderID
 AND Invoice.Invoice# = (SELECT I.Invoice#
   FROM Invoice AS I, Partial_Payment AS PP
  WHERE inserted.paymentID = PP.paymentID
  AND i.Invoice# = PP.Invoice#))
)
```

BEGIN

```
  UPDATE Orders
  SET Orders.paymentStatus = 'Full'
  FROM Partial_Payment, Orders, Invoice, inserted
```

```
WHERE Invoice.orderID = Orders.orderID  
AND inserted.paymentID = Partial_Payment.paymentID  
AND Invoice.Invoice# = Partial_Payment.Invoice#  
END;
```

## Trigger 8: maintainPartialPayment

After a partial payment is created in Partial\_Payment, an entry with amount 0 into Payment table.

### Schema:

**Payment**(paymentID, paymentDate, amount)

**Partial\_Payment**(paymentID, Invoice#)

```
CREATE TRIGGER maintainPartialPayment ON Partial_Payment
AFTER INSERT
AS
IF EXISTS(
SELECT inserted.paymentID
FROM inserted
WHERE inserted.paymentID NOT IN (      SELECT paymentID
                                      FROM Payment)
)
BEGIN

    INSERT INTO Payment
    SELECT inserted.paymentID, CONVERT(date, GETDATE()), 0
    FROM inserted

END;
```



## Trigger 9: maintainFullPayment

After a full payment is created in Full\_Payment, an entry is inserted into Payment table, with amount automatically calculated and equal to total amount payable in the order.

**Invoice**(Invoice#, orderID, invoiceStatus, invoiceDate)

**Order**(orderID, paymentStatus, orderDate, orderStatus, customerID)

**Order\_Item**(Sequence#, orderID, shipmentID, productID, unitPrice, orderItemStatus, Qty)

**Full\_Payment**(paymentID, Invoice#)

**Payment**(paymentID, paymentDate, amount)

Query:

CREATE TRIGGER maintainFullPayment ON Full\_Payment

AFTER INSERT

AS

IF ( (SELECT INSERTED.paymentID  
FROM INSERTED) NOT IN

(  
SELECT paymentID  
FROM Payment  
)

)  
BEGIN

INSERT INTO Payment

Values((SELECT inserted.paymentID FROM inserted), CONVERT(date, GETDATE()),

0)

UPDATE Payment

SET Payment.amount = (SELECT SUM(Order\_Item.Qty \* Order\_Item.unitPrice)  
FROM Order\_Item, Invoice, inserted  
WHERE inserted.invoice# = Invoice.orderID  
AND Invoice.orderID = Order\_Item.orderID  
)

FROM inserted

WHERE Payment.paymentID = inserted.paymentID

UPDATE Orders

SET Orders.paymentStatus = 'Full'

FROM Full\_Payment, Orders, Invoice, inserted

WHERE Invoice.orderID = Orders.orderID

AND inserted.paymentID = Full\_Payment.paymentID

AND Invoice.Invoice# = Full\_Payment.Invoice#

END;

## Trigger 10: UpdateParentTypeID

When a product\_typeID is deleted, all the parentTypeID of its child type is set to null

**Product\_Type**(typeID, parentTypeID, typeDescription)

Query:

```
CREATE TRIGGER UpdateParentTypeID ON Product_Type
AFTER DELETE
AS
IF EXISTS(
SELECT *
FROM deleted
WHERE deleted.typeID IN (
    SELECT parentTypeID
    FROM Product_Type
)
)
BEGIN
    UPDATE Product_Type
    SET parentTypeID = NULL
    FROM deleted
    WHERE Product_Type.parentTypeID = deleted.typeID
END;
```

## Trigger 11: CheckPaymentBeforeShipment

**Orders**(orderId, paymentStatus, orderDate, orderStatus, customerID)

Query:

CREATE TRIGGER CheckPaymentBeforeShip ON Order\_Item

AFTER UPDATE

AS

```
IF      (UPDATE(shipmentID)
        AND EXISTS (
            SELECT *
            FROM Orders, inserted
            WHERE Orders.paymentStatus <> 'Full'
            AND Orders.OrderID = inserted.OrderID
            AND inserted.shipmentID IN (SELECT shipmentID FROM Shipment)
        ))
```

BEGIN

ROLLBACK TRANSACTION

END;

## Trigger 12: CheckPartialPaymentID

**Payment**(paymentID, paymentDate, amount)

Query:

```
CREATE TRIGGER CheckPartialPaymentID ON Partial_Payment
AFTER INSERT
AS
IF      (SELECT inserted.paymentID
        FROM inserted)
      IN (SELECT paymentID FROM Payment)
BEGIN
ROLLBACK TRANSACTION
END;
```

## Trigger 13: CheckFullPaymentID

**Payment**(paymentID, paymentDate, amount)

Query:

```
CREATE TRIGGER CheckPartialPaymentID ON Full_Payment
AFTER INSERT
AS
IF      (SELECT inserted.paymentID
        FROM inserted)
      IN (SELECT paymentID FROM Payment)
BEGIN
ROLLBACK TRANSACTION
END;
```

## Trigger 14: UpdateParentTypeID\_2

When a product\_typeID is updated, all the parentTypeID of its child type is updated accordingly

**Product\_Type**(typeID, parentTypeID, typeDescription)

```
CREATE TRIGGER UpdateParentTypeID_2 ON product_type
AFTER UPDATE
AS
IF EXISTS(
    SELECT *
    FROM deleted
    WHERE deleted.typeID IN (
        SELECT parentTypeID
        FROM Product_Type
    )
)
BEGIN
    UPDATE Product_Type
    SET parentTypeID = inserted.typeID
    FROM inserted, deleted
    WHERE Product_Type.parentTypeID = deleted.typeID
END;
```

## Appendix D

### Individual Contribution Form

| Name                   | Individual Contribution for Submission 1 (Lab 1) | Percentage of Contribution (100% in total) |
|------------------------|--|--|
| Lee Yew Chuan Michael  | Analyzed requirements and created ER diagram.    | 25   |
| Lau Chen Yi Wynne      | Analyzed requirements and created ER diagram.    | 25   |
| Chua Wen Rong Jonathan | Analyzed requirements and created ER diagram.    | 25   |
| Li Ziyan g             | Analyzed requirements and created ER diagram.    | 25   |

| Name                  | Individual Contribution for Submission 2 (Lab 3) | Percentage of Contribution (100% in total) |
|-----------------------|--|--|
| Lee Yew Chuan Michael | Analyzed ER diagram and created schemas.         | 25   |

|                                     |  |    |
|-------------------------------------|--|----|
| Lau<br>Chen<br>Yi<br>Wynne          | Analyzed ER diagram and created schemas. | 25 |
| Chua<br>Wen<br>Rong<br>Jonath<br>an | Analyzed ER diagram and created schemas. | 25 |
| Li<br>Ziyang                        | Analyzed ER diagram and created schemas. | 25 |

| Name                                | Individual Contribution for Submission 3 (Lab 5) | Percentag<br>e of<br>Contributio<br>n<br>(100% in<br>total) |
|-------------------------------------|--|---|
| Lee<br>Yew<br>Chuan<br>Michae<br>l  | Analyzed schemas and created SQL queries.        | 25  |
| Lau<br>Chen<br>Yi<br>Wynne          | Analyzed schemas and created SQL queries.        | 25  |
| Chua<br>Wen<br>Rong<br>Jonath<br>an | Analyzed schemas and created SQL queries.        | 25  |
| Li<br>Ziyang                        | Analyzed schemas and created SQL queries.        | 25  |

**Name and Signature from all group members**

Name and Signature of  
Member 1      Name and  
Signature of Member 2 Name  
and Signature of Member 3  
Name and Signature of  
Member 4      Name and  
Signature of Member 5