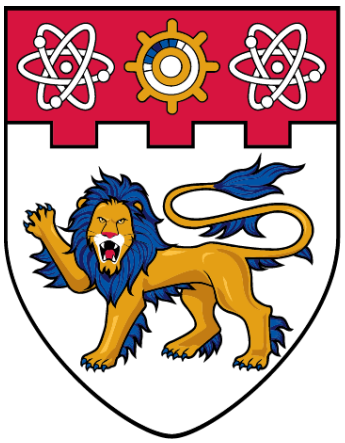

NANYANG TECHNOLOGICAL UNIVERSITY
SINGAPORE

CZ2007_Lab5_SSP1G7

SQL DDL commands for table creation

```
CREATE TABLE [CUSTOMER] (
  [Id] INT NOT NULL,
  [Phone_num] VARCHAR(8) NOT NULL,
  [Username] VARCHAR(30) UNIQUE NOT NULL,
  [Email] VARCHAR(50) UNIQUE NOT NULL,
  [Password] VARCHAR(50) NOT NULL,
  [Full_name] VARCHAR(30) NOT NULL,
  [Address] VARCHAR(50) NOT NULL,
  PRIMARY KEY ([Id]),
);
```

```
CREATE TABLE [CREDIT_CARD] (
  [Card_num] VARCHAR(16) NOT NULL,
  [Customer_id] INT NOT NULL,
  [Bank] VARCHAR(20) NOT NULL,
  [Date_valid_to] DATE NOT NULL,
  [Date_valid_from] DATE NOT NULL,
  PRIMARY KEY ([Card_num]),
  FOREIGN KEY (Customer_id) REFERENCES CUSTOMER(Id)
  ON DELETE NO ACTION
  ON UPDATE CASCADE
);
```

```
CREATE TABLE [SHOP] (
  [Id] INT NOT NULL,
  [Name] VARCHAR(50) NOT NULL,
  PRIMARY KEY ([Id]),
);

CREATE TABLE [PRODUCT_TYPE] (
  [Id] INT NOT NULL,
  [Parent_id] INT ,
  [Description] VARCHAR(300) NOT NULL,
  PRIMARY KEY ([Id]),
  FOREIGN KEY (Parent_id) REFERENCES PRODUCT_TYPE(Id)
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
);

CREATE TABLE [RESTRICTED_TO] (
  [Shop_id] INT NOT NULL,
  [Product_type_id] INT NOT NULL,
  PRIMARY KEY ([Shop_id], [Product_type_id]),
  FOREIGN KEY (Shop_id) REFERENCES SHOP(Id)
  ON DELETE CASCADE
  ON UPDATE CASCADE,
  FOREIGN KEY (Product_type_id) REFERENCES PRODUCT_TYPE(Id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);

CREATE TABLE [PRODUCT] (
  [Id] INT NOT NULL,
  [Shop_id] INT NOT NULL,
  [Product_type_id] INT NOT NULL,
  [Name] VARCHAR(50) NOT NULL,
  [Colour] VARCHAR(10) NOT NULL,
  [Size] VARCHAR(5) NOT NULL,
  [Price] FLOAT(7)NOT NULL CHECK(Price > 0),
  [Description] VARCHAR(300) NOT NULL,
  PRIMARY KEY ([Id]),
  FOREIGN KEY (Shop_id) REFERENCES SHOP(Id)
  ON DELETE NO ACTION       -- A product must have a shop_id, we set NO ACTION to
prevent from deleting shop_id this action
  ON UPDATE CASCADE,
  FOREIGN KEY (Product_type_id) REFERENCES PRODUCT_TYPE(Id)
```

```
  ON DELETE NO ACTION       -- A product must have a Product_type_id, we set NO
ACTION to prevent from deleting Product_type_id this action
  ON UPDATE CASCADE
);

CREATE TABLE [PHOTO] (
  [Seq] INT NOT NULL,
  [Product_id] INT NOT NULL,
  [Url] VARCHAR(50) NOT NULL,
  PRIMARY KEY ([Seq], [Product_id]),
  FOREIGN KEY (Product_id) REFERENCES PRODUCT(Id)
  ON DELETE CASCADE
  ON UPDATE CASCADE
);

CREATE TABLE [SHIPMENT] (
  [Id] INT NOT NULL,
  [Date] DATE NOT NULL,
  PRIMARY KEY ([Id]),
  [Tracking_num] VARCHAR(30) UNIQUE,
);

CREATE TABLE [ORDERS] (
  [Id] INT NOT NULL,
  [Customer_id] INT NOT NULL,
  [Date] DATE NOT NULL,
  [Status] VARCHAR(10) DEFAULT 'processing',
  PRIMARY KEY ([Id]),
  FOREIGN KEY (Customer_id) REFERENCES CUSTOMER(Id)
  ON DELETE NO ACTION  --  Prevent from deleting customer_id this action to trace the
order record
  ON UPDATE CASCADE
);

CREATE TABLE [INVOICE] (
  [Number] VARCHAR(10) NOT NULL,
  [Order_id] INT NOT NULL,
  [Date] DATE NOT NULL,
  [Status] VARCHAR(10) DEFAULT 'issued',
  PRIMARY KEY ([Number]),
  FOREIGN KEY (Order_id) REFERENCES ORDERS(Id)
  ON DELETE NO ACTION
  ON UPDATE CASCADE
);
```

```
CREATE TABLE [PAYMENT] (
  [Id] INT,
  [Invoice_number] VARCHAR(10) NOT NULL,
  [Credit_card_num] VARCHAR(16) NOT NULL,
  [Amount] FLOAT(10) NOT NULL CHECK(Amount > 0),
  PRIMARY KEY ([Id]),
  FOREIGN KEY (Invoice_number) REFERENCES INVOICE(Number)          -- prevent
from changes to trace the record
  ON DELETE NO ACTION
  ON UPDATE NO ACTION,
  FOREIGN KEY (Credit_card_num) REFERENCES CREDIT_CARD(Card_num)   --
prevent from changes to trace the record
  ON DELETE NO ACTION
  ON UPDATE NO ACTION
);

CREATE TABLE [ORDER_ITEM] (
  [Sequence_num] INT,
  [Order_id] INT,
  [Product_id] INT NOT NULL,
  [Shipment_id] INT ,
  [Quantity] INT NOT NULL CHECK(Quantity>0),
  [Status] VARCHAR(20) DEFAULT 'processing',
  [Product_unit_price] FLOAT(7) not null CHECK(Product_unit_price>0),
  PRIMARY KEY ([Sequence_num], [Order_id]),

  FOREIGN KEY (Order_id) REFERENCES ORDERS(Id)
  ON DELETE NO ACTION
  ON UPDATE CASCADE,
  FOREIGN KEY (Product_id) REFERENCES PRODUCT(Id)
  ON DELETE NO ACTION
  ON UPDATE CASCADE,
  FOREIGN KEY (Shipment_id) REFERENCES SHIPMENT(Id)
  ON DELETE NO ACTION
  ON UPDATE CASCADE
);




-- Most of time, we set ON DELETE NO ACTION to prevent from losing the record
```

Query 1

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

MSSQL Code

```
SELECT DISTINCT Product_id
FROM CUSTOMER
JOIN ORDERS ON CUSTOMER.Id = ORDERS.Customer_id
JOIN ORDER_ITEM ON ORDERS.Id = ORDER_ITEM.Order_id
JOIN INVOICE ON INVOICE.Order_id = ORDERS.id
WHERE CUSTOMER.Email = 'Fullname - 100@gmail.com'
AND INVOICE.status = 'paid'
AND ORDER_ITEM.Status = 'processing';
```

Query 2

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

MSSQL Code

```
SELECT TOP 3*
FROM(
	SELECT Product_type_id,sum(quantity) as totalSales
	FROM ORDER_ITEM oi,PRODUCT p
	WHERE oi.Product_id = p.Id
	AND oi.Order_id IN (
		SELECT O.id
		FROM ORDERS O, INVOICE I
		WHERE I.status = 'paid'
		AND I.Order_id = O.id
		)
	GROUP BY Product_type_id
)as productSalesTable
ORDER BY totalSales DESC;
```

Query 3

Query

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.

MSSQL Code

```
SELECT PT2.description
FROM PRODUCT_TYPE PT2
WHERE PT2.Parent_id IN (
   SELECT PT1.Id
   FROM PRODUCT_TYPE PT1
   WHERE PT1.Parent_id is NULL);
);
```

Query 4

Query

Find 2 product ids that are ordered together the most.

MSSQL Code

```
WITH [togetherTable] AS (
        SELECT O1.Product_id as P1, O2.Product_id as P2, COUNT(*) AS togetherTimes
        FROM ORDER_ITEM O1, ORDER_ITEM O2
        WHERE O1.Order_id = O2.ORDER_id
        AND   O1.Product_id <> O2.Product_id
        AND O1.Product_id < O2.Product_id
        GROUP BY O1.Product_id, O2.Product_id
        )

SELECT P1, P2
FROM togetherTable
WHERE togetherTimes IN (
   SELECT MAX(togetherTimes)
   FROM togetherTable T2
);
```

Query 5

Query
Get 3 random customers and return their email addresses.

MSSQL Code

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

Extra 1

Query
Given a customer id, find ids of 5 most similar customers. The invoice for the orders must be paid. (Similar customers are customers that purchase exactly the same products. Quantity of the products do not matter)

MSSQL Code

```
SELECT TOP 5 Customer_id
FROM (SELECT O2.Customer_id, COUNT(DISTINCT OI1.Product_id) AS similarity
FROM ORDERS O1, ORDERS O2, ORDER_ITEM OI1, ORDER_ITEM OI2, INVOICE
I1, INVOICE I2
WHERE O1.Customer_id=1876 AND O2.Customer_id!=1876
AND OI1.Order_id=O1.Id AND OI2.Order_id = O2.Id AND I1.Order_id=O1.Id AND
I2.Order_id = O2.Id
AND I1.status='paid' AND I2.Status = 'paid'
AND OI1.Product_id=OI2.Product_id
GROUP BY O2.Customer_id) AS similarCustomer
ORDER BY similarity DESC
```

Extra 2

Query
Find the top 10 shops that have the highest sales and return their ids

MSSQL Code

```
SELECT TOP 10*
FROM(
SELECT S.Id, SUM(OI.Quantity) AS SALES
FROM INVOICE I, ORDER_ITEM OI, PRODUCT P, SHOP S
WHERE I.Status = 'paid'  AND I.Order_id = OI.Order_id AND OI.Product_id = P.Id AND
P.Shop_id = S.Id
GROUP BY S.Id) AS shopSale
ORDER BY SALES DESC;
```

Constraint 1

Query

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

MSSQL Code

```
CREATE TRIGGER [dbo].[invoiceStatusTGR1]
ON [dbo].[PAYMENT] AFTER INSERT,UPDATE
AS
BEGIN

	DECLARE @orderPrice float
	SET @orderPrice = (SELECT SUM(Product_unit_price * quantity)
		FROM ORDER_ITEM OT, INVOICE I, inserted
		WHERE inserted.Invoice_number = I.Number
		AND I.Order_id = OT.Order_id)

	DECLARE @payAmount float
	SET @payAmount = (SELECT SUM(PAYMENT.Amount)
		FROM PAYMENT, inserted
		WHERE PAYMENT.Invoice_number = inserted.Invoice_number)

	IF @payAmount = @orderPrice
	BEGIN
		UPDATE INVOICE
		SET Status = 'paid'
		FROM INVOICE I
		INNER JOIN inserted ON I.Number = inserted.Invoice_number
		Print 'The full payment to the invoice is made, the invoice status changes to
paid'
	END
END;
```

Constraint 2

Query

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

MSSQL Code

```
CREATE TRIGGER [dbo].[TR2_changeToShipped]
ON [dbo].[ORDER_ITEM] AFTER UPDATE,INSERT
AS
BEGIN
	DECLARE @shipmentID int
	SET @shipmentID = (SELECT Shipment_id From inserted)
	DECLARE @status varchar(20)
	SET @status = (SELECT Status FROM inserted)
	IF @shipmentID is not NULL AND @status != 'shipped'
	BEGIN
		UPDATE ORDER_ITEM
		SET Status = 'shipped'
		FROM ORDER_ITEM OT INNER JOIN inserted ON OT.Order_id
=inserted.Order_id AND OT.Sequence_num = inserted.Sequence_num
		print('the order is shipped. status changed')
	END
END
```

Constraint 3

Query

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

MSSQL Code

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
CREATE TRIGGER [dbo].[tr3_completeOrder_afterUpdate]
  ON  [dbo].[ORDER_ITEM]
  AFTER UPDATE
AS
BEGIN
	DECLARE @orderID int
	SET @orderID = (SELECT Order_id FROM INSERTED)

	IF  NOT EXISTS(
		SELECT *
		FROM ORDER_ITEM OT
		WHERE OT.Order_id = @orderID AND OT.Status!='shipped')
	BEGIN
		UPDATE ORDERS
		SET Status='completed'
		WHERE ORDERS.Id = @orderID
		print('all order items in this order is shipped, the order status changes to
completed')
	END
End;
```

Constraint 4

Query

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.

MSSQL Code

```
CREATE TRIGGER [dbo].[TR4_thirdPaymentFull]
  ON  [dbo].[PAYMENT]
  AFTER INSERT
AS
BEGIN
	SET NOCOUNT ON;

	Declare @paymentInvoiceNum varchar(10);
	Declare @orderID int;
	Declare @total float;
	Declare @paid float;
	Declare @remainder float;
	Declare @paymentNum int;

	set @orderID = (select distinct i.Order_id
	from inserted , INVOICE i
	where inserted.Invoice_number = i.Number)

	Set @paymentNum =  (select count(*)
	from INVOICE i,ORDERS o,PAYMENT p
	where p.Invoice_number = i.Number and
	i.Order_id = o.Id and
	o.Id = @orderID)
	begin

		set @total =(select sum(ORDER_ITEM.Quantity *
ORDER_ITEM.Product_unit_price)
		from ORDER_ITEM
		where ORDER_ITEM.Order_id = @orderID)

		set @paymentInvoiceNum = (select inserted.Invoice_number from inserted)


		set @paid = (select sum(payment.amount)
					from PAYMENT
```

```
                                        where PAYMENT.Invoice_number =
@paymentInvoiceNum
                )
                --current remainder after 3rd insert
                set @remainder = @total - @paid

                IF (@remainder != 0) AND (@paymentNum = 3)
                BEGIN
                        RAISERROR('Unsuccessful payment. The third payment must
complete the full amount',16,2);
                        ROLLBACK TRANSACTION
                        RETURN;
                END
        END
END
```

Constraint 5

Query

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

MSSQL Code

```
CREATE TRIGGER [dbo].[tr5_canclePrecentTGR]
   ON [dbo].[ORDERS]
   After UPDATE
AS
BEGIN
 -- SET NOCOUNT ON added to prevent extra result sets from
 -- interfering with SELECT statements.
 SET NOCOUNT ON;
 DECLARE @newStatus varchar(20)
            SET @newStatus = (SELECT Status from inserted)

 IF(EXISTS(
 --check updated table
  select *
  from INVOICE,PAYMENT,deleted
  where PAYMENT.Invoice_number = INVOICE.Number and
  INVOICE.Order_id = deleted.Id
 ) AND @newStatus = 'cancelled')
 begin
 --revert the changes as not allow to be cancel
 --since have invoice payment existence
  RAISERROR('Cannot be cancel as payment has been maid',16,2);
   ROLLBACK TRANSACTION
 end
END;
```

Additional constraints that we come up with

Trigger 1: If there is an order_item in order is out of stock, the order will be cancelled

```
ALTER TRIGGER [dbo].[tr_orderOutOfStock_onInsertUpdate]
ON [dbo].[ORDER_ITEM] AFTER INSERT, UPDATE
AS
BEGIN
	DECLARE @status varchar(20)
	SET @status = (SELECT Status FROM inserted)

	DECLARE @orderId int
	SET @orderId = (SELECT Order_id FROM inserted)

	IF(@status = 'out of stock')
	BEGIN
		UPDATE ORDERS
		SET Status='cancelled'
		WHERE ORDERS.Id = @orderId
		print('the order item is out of stock, this order will be cancelled')
	END

END;
```

Trigger 2: cannot pay for a cancelled order

```
CREATE TRIGGER tr_noPaymentOnCancelledOrder_onUpdateInsert
ON PAYMENT AFTER INSERT
AS
BEGIN


	DECLARE @orderStatus varchar(10)
	SET @orderStatus = (SELECT O.Status FROM ORDERS O, inserted i, INVOICE
inv WHERE i.Invoice_number=inv.Order_id AND inv.Order_id=O.Id)

	IF @orderStatus = 'cancelled'
	BEGIN
		RAISERROR('Cannot pay for a cancelled order',16,1);
```

```
                ROLLBACK TRANSACTION
                RETURN;
        END
END;
```

Appendix D

Individual Contribution Form

Name	Individual Contribution for Submission 1 (Lab 1)	Percentage of Contribution (100% in total)
Zeren	Analyze the usage of weak entity sets.	20%
Xunyi	Analyze the choice of entity sets	20%
Mulder	Compare compare entity sets with alternative solutions	20%
Bryan	Construct a suitable ER diagram	20%
Peilun	Finalise ER diagram and written discussion	20%

Name	Individual Contribution for Submission 2 (Lab 3)	Percentage of Contribution (100% in total)
Zeren	Finalize the database design	20%
Xunyi	Produce suitable normalized relations	20%
Mulder	Decomposed normalized relations	20%
Bryan	Decomposed normalized relations	20%
Peilun	Normalized database schema and FDs	20%

Name	Individual Contribution for Submission 3 (Lab 5)	Percentage of Contribution (100% in total)
Zeren	Implement DB with SQL DDL commands with SSMS and populate data	20%
Xunyi	Formulate the SQL statements with necessary constraints	20%
Mulder	Implementation and execution of additional queries	20%
Bryan	Additional Queries, triggers and database testing	20%
Peilun	Generate database SQL and documentation	20%

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