1. PAYMENT DATABASE

For the Payment.aspx (page and logic):

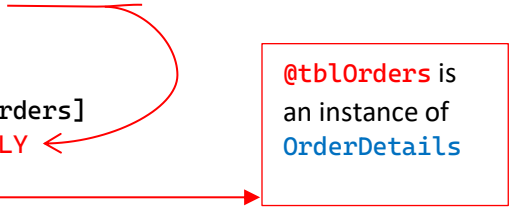
1. Firstly, we create OrderDetails; a TYPE in SQL works as a TABLE (basically it is a temporary table to store data temporary in it before we insert them to the real table -- like the transaction)

```
1.  -- 1. Create a table type
2.  CREATE TYPE OrderDetails AS TABLE
3.  (
4.      OrderNo NVARCHAR(50),
5.      ProductId INT,
6.      Quantity INT,
7.      UserId INT,
8.      Status NVARCHAR(20),
9.      PaymentId INT,
10.     OrderDate DATETIME
11. );
```

2. Secondly, we create Save_Orders; a STORED PROCEDURE in SQL that takes the data from the OrderDetails temporary table using a parameter of the table (basically this parameter is an instance of the temporary table, so it is a TABLE too).

   We use an INSERT query merged with SELECT statement together, to insert to Orders table, by selecting the data from the instance of OrderDetails

***NOTE THAT any instance of a TYPE tables should be *READONLY* when using it in a procedure for example.**




```
1.  -- 2. Execute this one first
2.  CREATE PROCEDURE [dbo].[Save_Orders]
3.  @tblOrders OrderDetails READONLY
4.  AS
5.  BEGIN
6.      SET NOCOUNT ON;
7.      INSERT INTO Orders(OrderNo, ProductId, Quantity, UserId, Status,
        PaymentId, OrderDate)
8.      SELECT OrderNo, ProductId, Quantity, UserId, Status, PaymentId,
        OrderDate FROM @tblOrders
9.  END
```

3. Finally, we create Save_Payment; a STORED PROCEDURE in SQL that adds the data in Payment table.

*NOTE THAT we need to return the inserted PaymentId to the backend to use it in some logic, so we have to do two things

I. AT BACKEND SIDE:
We define the InsertedId as an SQL database type of int, this parameter is assigned automatically by the database, and we have to get its value back to Backend side, so we call it as an ***Output***

```
cmd.Parameters.Add("@InsertedId", SqlDbType.Int);
cmd.Parameters["@InsertedId"].Direction = ParameterDirection.Output;
```

II. AT SQL SERVER SIDE:
We define its type as an **OUTPUT** too, then we use a **SCOPE_IDENTITY()** function to return its value to Backend side.

```
1.  -- 3. Then execute this at the end
2.  CREATE PROCEDURE [dbo].[Save_Payment]
3.         @Name VARCHAR(100) = NULL,
4.         @CardNo VARCHAR(50) = NULL,
5.         @ExpiryDate VARCHAR(50) = NULL,
6.         @Cvv INT = NULL,
7.         @Address VARCHAR(max) = NULL,
8.         @PaymentMode VARCHAR(10) = 'card',
9.         @InsertedId INT OUTPUT
10. AS
11. BEGIN
12.        SET NOCOUNT ON;
13.
14.        -- INSERT
15.        BEGIN
16.               INSERT INTO dbo.Payment(Name, CardNo, ExpiryDate, CvvNo,
    Address, PaymentMode)
17.               VALUES (@Name, @CardNo, @ExpiryDate, @Cvv, @Address,
    @PaymentMode)
18.
19.               SELECT @InsertedId = SCOPE_IDENTITY();
20.        END
21. END
```

SCOPE_IDENTITY()

returns the **last identity value** (auto-increment value) **inserted in the current session and current scope**.

2. PAYMENT LOGIC

1. Payment table:

We should keep in mind that: **"EACH Payment has multiple Orders; Each Order belongs to only one Payment"**

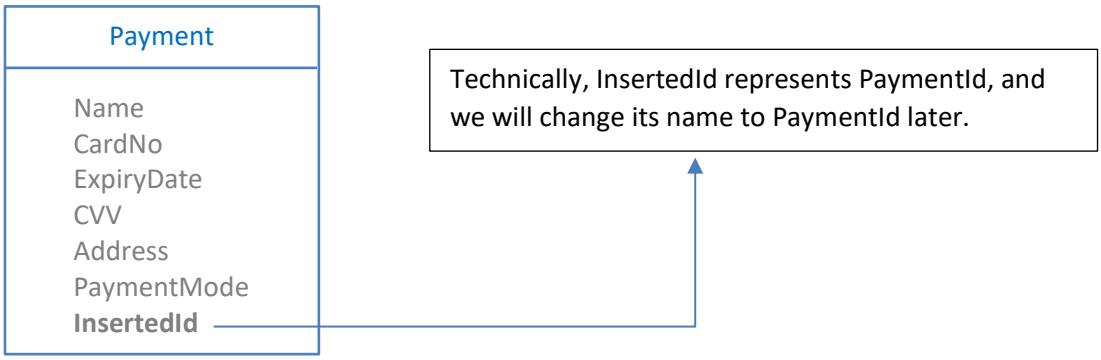
This means, that we have 1-n relationship between **Payment** and **Orders** tables.

So, we need to get the PaymentId first, to add it to each Order in database, this means we need to return the PaymentId once it's been generated from the database, and that is the benefit of SCOPE_IDENTITY() function

SCOPE_IDENTITY()

returns the **last identity value** (auto-increment value) **inserted in the current session and current scope**.

Payment Table:



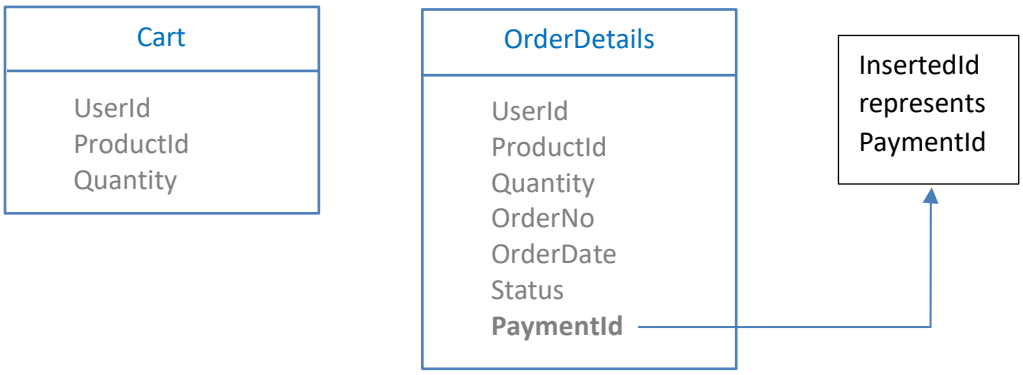
2. Payment table:

Now we need to read each row in Cart, so we use **SqlDataReader** class, because it's more efficient to read data that we just want to read it, not manipulating it. (typically, all we have to do is to read two columns **ProductId** and **Quantity** to pass them, so we don't need the whole data from the table)

For each row in Cart table, take ProductId and Quantity and pass them to two separated functions:

ii. UpdateQuantity(productId, quantity)
iii. DeleteItemFromCart(productId)

Then we save the data inside a temporary table "OrderDetails"



Now:
For each *Product* in *Cart*, create a new *Order* record, this means if we have 10 items in *Cart*, each item has its own *Order*, and each *Order* at the same payment time has the same ***PaymentId***

Notice:
Now we have inserted multiple records at the OrderDetails table, which is typically a temporary table. So, the whole table with its data is a Variable let's define it as **@tblOrders**

3. Order table:

Now after finishing of adding records to @tblOrders (a variable of type OrderDetails), all we have to do is to transfer data from @tblOrders to Order table, and for each record inside @tblOrders add a new column PaymentId, and fill it with the InsertedId.