Lab 10: Programming with Transact-SQL

Writing a Script to Insert an Order Header

You want to create reusable scripts that make it easy to insert sales orders. You plan to create a script to insert the order header record, and a separate script to insert order detail records for a specified order header.

Both scripts will make use of variables to make them easy to reuse. Your script to insert an order header must enable users to specify values for the order date, due date, and customer ID.

You are asked to include the following sales order:

Order Date	Due Date	Customer ID
Today's date	7 days from now	1

Note: to assist error checking, please just modify the existing code in the editor.

Instructions

- Fill in the variable names to complete the `DECLARE` statements. You can infer these names from the `INSERT` statement further down the script.
- Finish the `INSERT` query. Because `SalesOrderID` is an `IDENTITY`column, this ID will automatically be generated for you. You can use the hardcoded value `'CARGO TRANSPORT 5'` for the `ShipMethod` field.
- Use `SCOPE_IDENTITY()` to print out the ID of the new sales order header.

```
DECLARE @OrderDate datetime = GETDATE();

DECLARE @DueDate datetime = DATEADD(dd, 7, GETDATE());

DECLARE @CustomerID int = 1;


INSERT INTO SalesLT.SalesOrderHeader (OrderDate, DueDate, CustomerID, ShipMethod)

VALUES (@OrderDate, @DueDate, @CustomerID, 'CARGO TRANSPORT 5');
```

```
PRINT SCOPE_IDENTITY();
```

Extend Script to Insert an Order Detail

As a next step, you want to insert an order detail. The script for this must enable users to specify a sales order ID, a product ID, a quantity, and a unit price.

The script should check if the specified sales order ID exists in the `SalesLT.SalesOrderHeader` table. This can be done with the `EXISTS` predicate.
If it does, the code should insert the order details into the `SalesLT.SalesOrderDetail` table (using default values or `NULL` for unspecified columns).
If the sales order ID does not exist in the `SalesLT.SalesOrderHeader` table, the code should print the message 'The order does not exist'.
Note: to assist error checking, please just modify the existing code in the editor.

Instructions

- Slightly adapted code from the previous exercise is available; it defines the `OrderID` with `SCOPE_IDENTITY()`.
- Complete the `IF-ELSE` block:
  - The test should check to see if there is a `SalesOrderDetail` with a `SalesOrderID` that is equal to the `OrderID` exists in the `SalesLT.SalesOrderHeader` table.
  - Finish the statement to insert a record in the `SalesOrderDetail`table when this is the case.
  - Print out `'The order does not exist'` when this is not the case.

```
DECLARE @OrderDate datetime = GETDATE();

DECLARE @DueDate datetime = DATEADD(dd, 7, GETDATE());

DECLARE @CustomerID int = 1;

INSERT INTO SalesLT.SalesOrderHeader (OrderDate, DueDate, CustomerID, ShipMethod)

VALUES (@OrderDate, @DueDate, @CustomerID, 'CARGO TRANSPORT 5');

DECLARE @OrderID int = SCOPE_IDENTITY();


-- Additional script to complete

DECLARE @ProductID int = 760;

DECLARE @Quantity int = 1;

DECLARE @UnitPrice money = 782.99;
```

```
IF EXISTS (SELECT * FROM SalesLT.SalesOrderHeader WHERE SalesOrderID = @OrderID)
BEGIN
        INSERT INTO SalesLT.SalesOrderDetail (SalesOrderID, OrderQty, ProductID, UnitPrice)
        VALUES
        (@OrderID, @Quantity, @ProductID, @UnitPrice)
END
ELSE
BEGIN
        PRINT 'The order does not exist'
END
```

Updating Bike Prices

Adventure Works has determined that the market average price for a bike is `$2,000`, and consumer research has indicated that the maximum price any customer would be likely to pay for a bike is `$5,000`.
You must write some Transact-SQL logic that incrementally increases the list price for all bike products by 10% until the average list price for a bike is at least the same as the market average, or until the most expensive bike is priced above the acceptable maximum indicated by the consumer research.

The product categories in the Bikes parent category can be determined from the `SalesLT.vGetAllCategories` view.
Note: to assist error checking, please just modify the existing code in the editor.

Instructions

- The loop should execute only if the average list price of a product in the 'Bikes' parent category is less than the market average.
- Update all products that are in the 'Bikes' parent category, increasing the list price by 10%.
- Determine the new average and maximum selling price for products that are in the 'Bikes' parent category.
- If the new maximum price is greater than or equal to the maximum acceptable price, exit the loop; otherwise continue.

```
DECLARE @MarketAverage money = 2000;
DECLARE @MarketMax money = 5000;
DECLARE @AWMax money;
```

```
DECLARE @AWAverage money;

SELECT @AWAverage = AVG(ListPrice), @AWMax = MAX(ListPrice)
FROM SalesLT.Product
WHERE ProductCategoryID IN
        (SELECT DISTINCT ProductCategoryID
         FROM SalesLT.vGetAllCategories
         WHERE ParentProductCategoryName = 'Bikes');

WHILE @AWAverage < @MarketAverage
BEGIN
  UPDATE SalesLT.Product
  SET ListPrice = ListPrice * 1.1
  WHERE ProductCategoryID IN
        (SELECT DISTINCT ProductCategoryID
         FROM SalesLT.vGetAllCategories
         WHERE ParentProductCategoryName = 'Bikes');

        SELECT @AWAverage = AVG(ListPrice), @AWMax = MAX(ListPrice)
        FROM SalesLT.Product
        WHERE ProductCategoryID IN
        (SELECT DISTINCT ProductCategoryID
         FROM SalesLT.vGetAllCategories
         WHERE ParentProductCategoryName = 'Bikes');

  IF @AWMax >= @MarketMax
    BREAK
  ELSE
    CONTINUE
```

```
END

PRINT 'New average bike price:' + CONVERT(VARCHAR, @AWAverage);
PRINT 'New maximum bike price:' + CONVERT(VARCHAR, @AWMax);
```