```sql
/*
Chris Singleton                                                   02/26/2017
PROG 140               Module 6 Assignment

*/

/*
1.  Using the uspInsertOrder stored procedure from our demo file,
Sprocs_UDFS.sql, as a model, write a stored procedure to UPDATE an order in the
Northwind database. Your sproc should update only one row each time it is called.
Add approriate documentation at the top of your stored procedure as you've done
in earlier assignments and any additional comments inside the sproc as you feel
are needed for clarity.

      Also include the following statements AFTER your sproc:
      * write the call to the stored procedure
      * include a drop statement
      * include a statement to retrieve and display the row updated by the sproc
*/

--================== First create entries in the Northwind DB ==================--
--Please run this Stored Procedure several time to create new rows.
CREATE PROCEDURE uspInsertOrder
     @CustomerID nchar(5),
     @EmployeeID [int],
     @OrderDate datetime = NULL,
     @RequiredDate datetime = NULL,
     @ShippedDate datetime = NULL,
     @ShipVia int = 1,  -- may have a primary shipper
     @Freight money,
     @ShipName nvarchar(40) = NULL,
     @ShipAddress nvarchar(60) =NULL,
     @ShipCity nvarchar(15) = NULL,
     @ShipRegion nvarchar(15) =NULL,
     @ShipPostalCode nvarchar(10) =NULL,
     @ShipCountry nvarchar(15) =NULL
AS
/*
Created By: Chris Singleton
Date: 02/26/2017
Creates a new row for Northwind.Orders database.
Please initiate this procedure several of times.
*/
     if not exists (select 1 from Customers where customerid = @Customerid)
          throw 50001, 'Invalid Customer Provided', 1;
     if not exists (select 1 from Employees where Employeeid = @EmployeeID)
          throw 50002, 'Invalid Employee Provided', 1;
     if not exists (select 1 from Shippers where ShipperId = @ShipVia)
          throw 50003, 'Invalid Shipper Provided', 1;
      INSERT Orders (CustomerID, EmployeeID, OrderDate, RequiredDate, ShippedDate
     , ShipVia, Freight, ShipName, ShipAddress, ShipCity, ShipRegion
     , ShipPostalCode, ShipCountry)

      VALUES (@CustomerID, @EmployeeID, @OrderDate, @RequiredDate, @ShippedDate
     , @ShipVia, @Freight, @ShipName, @ShipAddress, @ShipCity, @ShipRegion
     , @ShipPostalCode, @ShipCountry)
      RETURN @@Identity
-- end sproc
--DROP PROCEDURE uspInsertOrder




--========== Call the Stored Procedure uspInsertOrder =======--
-- the call:
DECLARE @Ret int
EXEC  @Ret = uspInsertOrder 'alfki', 7, '1/1/2013', null, null, 1, 25.99;
If @ret = 0
      print 'error!';
else
      print 'OrderId entered: ' + cast(@ret as varchar);
GO

--=================== Update Only One Row ================--

CREATE PROCEDURE uspUpdateOrder
      @Freight money --Testing.
AS
/*
Created by: Chris Singleton
02/26/2017
Updates each row that has 25.99 where the customer's name is 'alfki'.
*/
      BEGIN
          --DECLARE @Freight

           UPDATE top (1) Northwind.dbo.Orders
           SET Freight = @Freight + .1
           WHERE CustomerID = 'alfki' AND Freight = 25.99
      END;
GO
--=================== Calls Only the PROC ================--
-- the call:
Declare @Ret int

EXEC  @Ret = uspUpdateOrder 25.99;
If @ret = 0
      print 'error!';
else
      print 'OrderId entered: ' + cast(@ret as varchar);
GO
DROP PROCEDURE uspUpdateOrder
```

```
GO

SELECT * FROM [Northwind].[dbo].[Orders] WHERE CustomerID = 'alfki'

2.  Using the uspInsertOrder stored procedure from our demo file,
Sprocs_UDFS.sql, as a model, write a stored procedure to Insert a new row into a
child table, ie., the "many" table in a 1-many relationship, in YOUR database,
ie., the db you've submitted in previous exercises.

        Also include the following statements AFTER your sproc:
        * write the call to the stored procedure
        * include a drop statement
        * include a statement to retrieve and display the row updated by the sproc

Note: please include a "use databasename" statement so that I can identify your
database from a previous exercise.  If I can use your sproc easily in your last
submission of your database, then you can send only the code for the Insert sproc
you will create above. If not, please contact me and/or send a separate file with
a script of your database that I can use in order to test this sproc.

*/

USE CustomerTransact;

--============= Insert Rows CustomerTransact SalesOrder ==============--
GO
CREATE PROCEDURE uspCTInsertOrder
      --@SalesOrderID int, --identity
      @CustomerID int,
      @SalesOrderDate datetime,
      @SaleAmount decimal = NULL
AS
/*
Created By: Chris Singleton
Date: 02/26/2017
Adds a row to the SalesOrder table each time it is called.
*/
      Insert SalesOrder (CustomerID, SalesOrderDate, SaleAmount)
      values (@CustomerID, @SalesOrderDate, @SaleAmount)
      Return @@Identity
-- end sproc
GO
--================ Call the uspCTInsertOrder with Data ==========--

-- SELECT * FROM [CustomerTransact].[dbo].[SalesOrder]
-- the call:
Declare @Ret int
EXEC  @Ret = uspCTInsertOrder 1, '1/1/2017', 26.00;
If @ret = 0
      print 'error!';
else
      print 'OrderId entered: ' + cast(@ret as varchar);
GO
--DROP PROCEDURE uspCTInsertOrder
```

```sql
--============== Update only one row CustomerTransact DB =============--
--Checking...
--SELECT * FROM CustomerTransact.dbo.SalesOrder
GO
CREATE PROCEDURE spCTUpateSalesOrder

     @SaleAmount decimal(8,2) NULL
AS
/*
Created By: Chris Singleton
Date: 02/26/2017
About: Updates one row only when Stored Procedure is called.
Adds 10 cents on the Sale Amount of 26.00 for customerID 1.
*/
     BEGIN

          UPDATE top (1) CustomerTransact.dbo.SalesOrder
          SET SaleAmount = @SaleAmount + .1
          WHERE CustomerID = 1 AND SaleAmount = 26.00

      END;


Declare @Ret int
EXEC  @Ret = spCTUpateSalesOrder 26.00;
If @ret = 0
     print 'error!';
else
     print 'OrderId entered: ' + cast(@ret as varchar);


GO
DROP PROCEDURE spCTUpateSalesOrder

/*
3. Write a Scalar UDF, called fnYearMonth that will take any date as an input
parameter and return that same date in the following format:  YYYY-MMM  example:
2012-Dec  (4 digits for the year, a hyphen, and 3 characters for the Month)
     Note:  the Return will be in varchar format, NOT date format
     Hint: there is a similar example in our module's demo file

     Include at least 3 statements to test this new UDF with different dates
*/

GO

--=============== My Code ===================--

-- write your SQL statements here:
IF EXISTS (
    SELECT * FROM sysobjects WHERE id = object_id(N'fnYearMonth')
    AND xtype IN (N'FN', N'IF',N'TF')
)
    DROP FUNCTION fnYearMonth
GO
CREATE FUNCTION fnYearMonth
(@InputDate date) --Input parameter.
RETURNS varchar(10)
AS
BEGIN
     RETURN CAST(YEAR(@InputDate) AS varchar(4)) + ' - '
     + CAST(LEFT(DATENAME (MONTH , @InputDate ),3) AS varchar(3))
END;
GO
--Testing... Looking for dates to use.
--SELECT * FROM [Orders]
--SELECT * FROM [Employees]
--SELECT * FROM [Shippers]
--SELECT * FROM [Order Details]
--SELECT * FROM [Products]

SELECT OrderID, CustomerID, EmployeeID, dbo.fnYearMonth(OrderDate) AS OrderDate
FROM [Northwind].[dbo].[Orders]
ORDER BY OrderDate DESC



SELECT OrderID, CustomerID, EmployeeID, dbo.fnYearMonth(RequiredDate) AS
RequiredDate
FROM [Northwind].[dbo].[Orders]
ORDER BY RequiredDate DESC

SELECT OrderID, CustomerID, EmployeeID, dbo.fnYearMonth(ShippedDate) AS
ShippedDate
FROM [Northwind].[dbo].[Orders]
ORDER BY ShippedDate DESC

GO

--Testing AdventureWorks2012 (another database)
--SELECT *  FROM [AdventureWorks2012].[Sales].[SalesOrderDetail]
USE AdventureWorks2012;
GO
SELECT SalesOrderID, RevisionNumber, dbo.fnYearMonth(OrderDate) AS OrderDate
FROM [AdventureWorks2012].[Sales].[SalesOrderHeader]
ORDER BY OrderDate DESC
--Works with another database also.

GO
```

```
/*
4. Consider the following actual view in the Northwind database:
 */
/*
CREATE VIEW [dbo].[Sales by Category] AS
SELECT Categories.CategoryID, Categories.CategoryName, Products.ProductName,
      Sum("Order Details Extended".ExtendedPrice) AS ProductSales
FROM  Categories INNER JOIN
           (Products INNER JOIN
                 (Orders INNER JOIN "Order Details Extended"
                 ON Orders.OrderID = "Order Details Extended".OrderID)
           ON Products.ProductID = "Order Details Extended".ProductID)
      ON Categories.CategoryID = Products.CategoryID
WHERE Orders.OrderDate BETWEEN '19970101' And '19971231'
GROUP BY Categories.CategoryID, Categories.CategoryName, Products.ProductName
GO
*/

/*
a) Describe what this view does as though you are speaking to a technical member
   of your team. Include a description of the "Order details Extended" object.
*/
-- write your SQL statements here:
/*
This view shows two columns from Categories, a product column [ProductName] and a
joined view's (View called: Order Details Extended) which creates a derived
calculated column called [ProductSales] that sums the total amount in product
sales as a calculated field, at the same time the calculation is being converted
into the same datatype (money) during the calculation process using the formula
during conversion (CONVERT(money,(UnitPrice*Quantity*(1-Discount)/100))*100)])
with getting its data from the [Order Details] table in the Northwind database.
Basically, were showing the total amount in sales by category between the dates
of 19970101' AND '19971231 using the VIEW.

Result of the first three rows: Note: total amount in product sales for each
product [ProductSales].
CategoryID CategoryName                      ProductName      ProductSales
2             Condiments                     Gula Malacca        6737.95
6             Meat/Poultry                   Alice Mutton        17604.60
3             Confections   Sir Rodney's     Marmalade            7314.30
*//*
b) Rewrite this view as a simple table-valued function (NOT a multi-statement)
and give it the
      name fnSalesbyCategory.  This function will have a single parameter to
provide a Category that
    will allow you to parameterize the call like this:
           Select * from dbo.fnSalesbyCategory('Seafood');
c) Include at least two statements that call this function.
*/
```

(Next Page)

```sql
USE Northwind
GO

IF EXISTS (
    SELECT * FROM sysobjects WHERE id = object_id(N'fnSalesbyCategory')
    AND xtype IN (N'FN', N'IF',N'TF')
)
    DROP FUNCTION fnSalesbyCategory
GO
CREATE FUNCTION dbo.fnSalesbyCategory
(@CategoryName nvarchar(15))
/*
Created By: Chris Singleton
Date: 02/25/2017
Returns the total product sales
of all products in the Seafood category
Between the dates of '19970101' AND '19971231'.

Note: parameter 'Seafood' doesn't make sense unless your
      trying to get only that category, which can be filtered
       using the where clause.
*/
RETURNS TABLE
RETURN (
    SELECT Categories.CategoryID, Categories.CategoryName, Products.ProductName,
     SUM("Order Details Extended".ExtendedPrice) AS ProductSales

FROM  Categories INNER JOIN
           (Products INNER JOIN
                (Orders INNER JOIN "Order Details Extended"
                ON Orders.OrderID = "Order Details Extended".OrderID)
           ON Products.ProductID = "Order Details Extended".ProductID)
     ON Categories.CategoryID = Products.CategoryID
WHERE Orders.OrderDate BETWEEN '19970101' AND '19971231' --AND
Categories.CategoryName = 'Seafood'
GROUP BY Categories.CategoryID, Categories.CategoryName, Products.ProductName)
GO
SELECT * FROM dbo.fnSalesbyCategory('Seafood');

--Finding the units on order (descending) with also showing selected columns from
the function.
SELECT p.ProductID, QuantityPerUnit, UnitsOnOrder, sbc.ProductName,
sbc.CategoryName
FROM [dbo].[Products] AS p, dbo.fnSalesbyCategory('Seafood') AS sbc
ORDER BY UnitsOnOrder DESC
```

(Next Page)

```
--Total quantity of seafood product ordered between '19970101' AND '19970131'
Descending...
SELECT c.CompanyName, sbc.CategoryName, COUNT(od.Quantity) AS TotalQuantity
FROM [dbo].[Customers] AS c, dbo.fnSalesbyCategory('Seafood') AS sbc,
[dbo].[Order Details] AS od, [dbo].[Orders] AS o
WHERE o.OrderDate BETWEEN '19970101' AND '19970131' AND sbc.CategoryName =
'Seafood'
GROUP BY c.CompanyName, sbc.CategoryName, od.Quantity
ORDER BY TotalQuantity DESC
GO

5.  Review and then execute the following query:

      select lastname, count(*) TotalOrders
      from employees join orders
      on employees.employeeid = orders.employeeid
      group by lastname

      select count (*) from orders
      select top 100 * from orders
*/

SELECT *
FROM (
    SELECT lastname, COUNT(*) TotalOrders
    FROM employees JOIN orders
    ON employees.employeeid = orders.employeeid
    GROUP BY lastname
) AS t
PIVOT
(
  MAX(TotalOrders)
  FOR lastname IN ([Buchanan], [Callahan], [Davolio], [Dodsworth]
                  ,[Fuller], [King], [Leverling], [Peacock], [Suyama])

) AS p;

/* OPTIONAL extra credit:
    from #2 above add both uspDeleteXXXX and uspUpdateXXXX routines for 3 points
each to YOUR database.
       (where XXXX represents a table in your database).

      Attempt this extra credit ONLY if you are sure that all of the #1-5 answers
you've provided are correct
      to the best of your ability and you have tested them to the best of your
ability.
      Extra credit will not be given if you have not fully tested the first 5 to
ensure they work correctly.

*/
-- write your SQL statements here:
```

(Next Page)

```
USE CustomerTransact;

--=========== First Add to the CustomerTransact DB ============--
GO
CREATE PROCEDURE uspCTInsertOrder
      --@SalesOrderID int, --identity
      @CustomerID int,
      @SalesOrderDate datetime,
      @SaleAmount decimal = NULL
AS
/*
Created By: Chris Singleton
Date: 02/26/2017
Adds a row to the SalesOrder table each time it is called.
*/

      Insert SalesOrder (CustomerID, SalesOrderDate, SaleAmount)
      values (@CustomerID, @SalesOrderDate, @SaleAmount)
      Return @@Identity
-- end sproc
GO
--=============== Call the uspCTInsertOrder with Data ==========--

-- SELECT * FROM [CustomerTransact].[dbo].[SalesOrder]
--Please run several times to make several rows with data.
-- the call:
Declare @Ret int
EXEC  @Ret = uspCTInsertOrder 1, '1/1/2017', 26.00;
If @ret = 0
      print 'error!';
else
      print 'OrderId entered: ' + cast(@ret as varchar);
GO

DROP PROCEDURE uspCTInsertOrder
GO
```

(Next Page)

```
--============================================================--
SELECT * FROM [dbo].[SalesOrder];
GO
CREATE PROCEDURE uspUpdateCustomerTransact

     @SaleAmount decimal(8,2) NULL
AS
/*
Created By: Chris Singleton
Date: 02/26/2017
About: Updates one row only when Stored Procedure is called.
Adds 10 cents on the SaleAmount of 26.00 for customerID 1.
*/
     BEGIN

          UPDATE TOP (1) CustomerTransact.dbo.SalesOrder
          SET SaleAmount = @SaleAmount + .1
          WHERE CustomerID = 1 AND SaleAmount = 26.00

      END;


DECLARE @Ret int
EXEC  @Ret = uspUpdateCustomerTransact 26.00;
If @ret = 0
     PRINT 'error!';
else
     PRINT 'OrderId entered: ' + cast(@ret as varchar);


GO
DROP PROCEDURE uspUpdateCustomerTransact
```

(Next Page)

```sql
--=========== Deletes SalesOrder ==============--
GO
CREATE PROCEDURE uspDeleteCustomerTransact

      @SaleAmount decimal(8,2) NULL
AS
/*
Created By: Chris Singleton
Date: 02/26/2017
About: Deletes one row only when Stored Procedure is called.
*/
      BEGIN

            DELETE TOP (1) CustomerTransact.dbo.SalesOrder
            WHERE CustomerID = 1 AND SaleAmount = 26.00

       END;


Declare @Ret int
EXEC  @Ret = uspDeleteCustomerTransact 26.00;
If @ret = 0
      print 'error!';
else
      print 'OrderId entered: ' + cast(@ret as varchar);


GO
DROP PROCEDURE uspUpdateCustomerTransact

SELECT * FROM [dbo].[SalesOrder];
```