/*
PROG 140 Module 6 Assignment
POINTS 50 DUE DATE : Consult module

Angelique Refahiyat

05/26/2017

Prog 140

Develop a SQL statement for each task listed. This exercise uses the Northwind database.

Please type your SQL statement under each task below. Ensure your statement works by testing it prior to turning it in
When writing answers for your select statements please keep in mind that there is more than one way to write a query.
Please do your best to write a query that not only returns the information the questions asks for but to the best of your ability,
in the best way possible to suit their needs. For example, consider the best way to sort, to name columns, etc. These are things you must think about on the job!

***As always, if you have any questions about this assignment please double-check the discussion areas first! Please send an e-mail
or call the instructor if you still need clarification. If you are unable to get something working after trying your best and attempting
to get help, then please note your difficulty and indicate that you know your answer doesn't work and what your issues were.***

Turn In:

For this exercise you will submit one WORD documents (instead of a .sql file) in which you have copied and pasted
your entire work from your .sql file including all assignment questions and your SQL queries.
The document should contain your name at the top, assignment title, the date and any difficulties encountered.

Submit your file to the instructor using the Canvas Assignment tool.
*/

```
-- Tasks:

/*
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
*/
-- write your SQL statements here:
use [Northwind]
go

--Creating the stored procedure using the model provided to update one order in a table

Create Procedure uspdatedOrder
        @CustomerID nchar(5),
        @EmployeeID [int],
        @ShipVia int = 1,
        @OrderID int =10432
as
begin
        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;
        UPDATE [dbo].[Orders]
   SET [CustomerID] = 'SPLIR'
      ,[EmployeeID] = 3
      ,[OrderDate] = '02/15/2014'
```

```
      ,[RequiredDate] = '02/16/2014'
      ,[ShippedDate] = '02/22/2014'
      ,[ShipVia] = 3
      ,[Freight] = 54.25
      ,[ShipName] = 'Cafe du palee'
      ,[ShipAddress] = '56th Ave North'
      ,[ShipCity] = 'Marseille'
      ,[ShipRegion] = Null
      ,[ShipPostalCode] = 13010
      ,[ShipCountry] = 'France'
 WHERE [OrderID]=@OrderID
 return @@Identity
 end
 go
 drop Procedure uspdatedOrder
 go
-- Creating a stored procedure to update only one row
create procedure uspOneRow
 @ShipName nvarchar(40) = 'Toms Spezialitäten INC'

 as
 begin
 UPDATE top (3) [dbo].[Orders]
  SET [ShipName] = @ShipName
  WHERE CustomerID = 'TOMSP'
 end
  go
 select top (3) * from [dbo].[Orders]
--executing the store procedure
exec uspOneRow @ShipName = 'Toms Spezialitäten INC'
--Calling this store procedure
Declare @Ret int

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

```
--Calling the stored procedure
exec [dbo].[updatedOrder] @CustomerID ='SPLIR',@EmployeeID =3,@ShipVia =3, @OrderID =10432
--verify if the change is made through
select * from [dbo].[Orders]
where [OrderID] =10432
--An statement to retrieve and get the data in which is updated
Declare @getUpdate int
EXEC  @getUpdate = updatedOrder 'SPLIR', 3, 1,10432;
If @getUpdate = 0
        print 'error!';
else
        print 'OrderId entered: ' + cast(@getUpdate as varchar);

DROP PROCEDURE updatedOrder;
GO

/*
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.

*/
-- write your SQL statements here:

--Create a new Procedure to store all data from order table into a new temp table
--called the TempOrder from one data base to another
```

```
use [NorthWestDining]
go
create procedure uspnew_Orders
        @CustomerID nchar(5),
        @EmployeeID [int],
        @ShipVia int = 3,
        @OrderID int =10432
as
begin

   DECLARE @TempOrder TABLE
        (
        CustomerID nchar(5),
        EmployeeID [int],
    ShipVia int ,
        OrderID Int

        )
   insert into @TempOrder (CustomerID,EmployeeID,ShipVia,OrderID)
        --or using the previous stored procedure
        --EXEC [Northwind].[dbo].[updatedOrder] @CustomerID,@EmployeeID ,@ShipVia , @OrderID
        select CustomerID,EmployeeID ,ShipVia , OrderID from [Northwind].[dbo].[Orders]
        where OrderID=10432

        select * from @TempOrder
        end
  go
--dropping statement
  use [NorthWestDining]
  drop procedure uspnew_Orders
  go

--calling the store procedure created
use [NorthWestDining]
exec uspnew_Orders  @CustomerID ='SPLIR',@EmployeeID =3,@ShipVia =3, @OrderID =10432

--retriving data from temp table
Declare @retreiveData int
EXEC  @retreiveData = uspnew_Orders 'SPLIR', 3, 3,10432;
```

```
If @retreiveData= 0
	print 'error!';
else
	print 'OrderId entered: ' + cast(@retreiveData as varchar);

--Well here is another way to just add a row to my dtatbase's table:
use NorthWestDining
go
--select * from [dbo].[Ingredients]
CREATE PROCEDURE uspAdd_row
	@IngredientID int,
	@IngredientName varchar(40),
	@RecipeCost money,
	@RecipeID int
AS
BEGIN
	INSERT INTO [dbo].[Ingredients] (IngredientID,IngredientName,RecipeCost,RecipeID )
	VALUES  (@IngredientID,@IngredientName,@RecipeCost,@RecipeID)
END

GO
EXEC uspAdd_row @IngredientID= 403, @IngredientName ='fish',@RecipeCost=48,@RecipeID =200
drop procedure uspAdd_row
select * from [dbo].[Ingredients]
select *from [dbo].[Recipes]
--calling it
Declare @retreiveData int
EXEC  @retreiveData = uspAdd_row 403,'fish', 48, 200;
If @retreiveData= 0
	print 'error!';
else
	print 'OrderId entered: ' + cast(@retreiveData as varchar);
/*
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
*/
-- write your SQL statements here:

use Northwind
go
--We can use the if,else statement to make sure if the function is already exist if it's not exist then create it

CREATE FUNCTION fnYearMonth
(@pramYearMonth date)
RETURNS varchar(10)
 AS
 BEGIN
  RETURN convert(varchar(4),YEAR(@pramYearMonth)) + '-' + convert(varchar(3),left(DATENAME (MONTH ,
@pramYearMonth),3))
  END;
   go
drop function fnYearMonthModified
 select OrderID, dbo.fnYearMonth([OrderDate]) AS Order_Date
 from [dbo].[Orders]
 --Another way to create this function
 go
 Create function fnYearMonthTwo
        (@OrderDate DATETIME)
        returns int
Begin
        Return (Select OrderID  from [dbo].[Orders] where OrderDate =FORMAT(@OrderDate, 'yyyy-MMM'))
End;
go
drop function fnYearMonthTwo
--test 1
select [dbo].[fnYearMonthTwo]('1996-OCT') as Order_Date
--test 2:
select [dbo].[fnYearMonthTwo]('2013-JAN') as Order_Date
-- test 3
select [dbo].[fnYearMonthTwo]('1997-DEC') as Order_Date


/*
```

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
------------------------
```

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.

--The whole query is about creating a view that gives us 4 columns as CategoryID , CategoryName, ProductName, ProductSales
that on the last column we'll get the extract data in which shows the sum of the total amount from the productSales that is stored as an object. Parallel of doing that, what the query does
is calcualting the total amount by converting the data into money and showing us also the orders placed by customers between cetain period of time like between
1997/01/01 and 1997/12/31 for the whole year of 19997 from the beginning at the end.
In order to get the result of what the employer pursued for, he required to use three different join statements to make a relationship between those three table through their primary keys. After he joined all the required tables, he used the grouping statements to
group by the fields that he used at the beginning of the query to remove any duplicates.

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.

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

use Northwind
go
Create Function fnSalesbyCategory
        (@CategoryName nvarchar(40))
        Returns Table
Return (
        Select  p.ProductID,c.CategoryName, p.ProductName, sum(od.[ExtendedPrice]) as TotalDue
        from Products p join [dbo].[Order Details Extended] od
        on od.ProductID  = p.ProductID
        join [dbo].[Orders]o
        on o.OrderID = od.OrderID
        join  [dbo].[Categories]c
        on c.CategoryID = p.CategoryID

        where @CategoryName = c.CategoryName
        and o.OrderDate between '19970101' and '19971231'
        Group by p.ProductName, p.ProductID,c.CategoryName
)
drop function fnSalesbyCategory
--12 rows apears once I called the function usiing the parameter 'Seafood'
Select * from dbo.fnSalesbyCategory('Seafood');
SELECT
 distinct p.ProductID,sc.ProductName,sc.[CategoryName]
 FROM [dbo].[Products] p,
  dbo.fnSalesbyCategory('Seafood') as sc
-- select 2 other statements to call this function

SELECT *FROM dbo.fnSalesbyCategory('Beverages')
order by [ProductID]
SELECT *FROM dbo.fnSalesbyCategory('Dairy Products')
order by [ProductName]

/*
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

        Using the Pivot example from the end of our module's demo sql file as a guide,
        pivot this result set so that the lastnames are the columns
*/
-- write your SQL statements here:
        select o.[OrderID],e.[LastName], count(convert(varchar(10) ,[OrderID]))
        from [dbo].[Employees] e join [dbo].[Orders] o
        on e.employeeid = o.employeeid
        group by o.[OrderID],e.[LastName]

        --select count (*) from orders
        --select top 100 * from orders
        ----Creating a pivot table

--I was attempting to create a store Precodure to store the data to use them in a temp table
--CREATE PROCEDURE EmployeeOrders

-- @EmployeeName nvarchar(20)
--AS

--SELECT [EmployeeID] = t1.EmployeeID
--FROM [dbo].[Employees]t1
--JOIN [dbo].[Orders] t2
--ON t1.EmployeeID=t2.EmployeeID;



--GO
--drop procedure EmployeeOrders
--exec EmployeeOrders @EmployeeName ='Buchanan'

declare @TempEmp_orders as table
(
    --orderID int,
```

```
            lastName nvarchar(20),
            total_Orders int

            )
    insert into @TempEmp_orders ([LastName],total_Orders)
            --select o.[OrderID],e.[LastName], count(*) TotalOrders
            --from [dbo].[Employees] e join [dbo].[Orders] o
            --on e.employeeid = o.employeeid
            --group by o.[OrderID],e.[LastName]
                    select lastname, count(*) TotalOrders
            from employees join orders
            on employees.employeeid = orders.employeeid
            group by lastname

select * from  @TempEmp_orders

pivot
(
sum(total_Orders)
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:

use NorthWestDining
go
```

```
select * from [dbo].[RecipeOrder]
go
create procedure uspUpdateOrder
@OrderQuanity smallint
as begin
UPDATE [NorthWestDining].[dbo].[RecipeOrder]
 set [OrderQuanity]= @OrderQuanity
 where OrderID =1001
 end
 go
  exec uspUpdateOrder @OrderQuanity =35
  --calling it
  DECLARE @Ret int
  EXEC  @Ret = uspUpdateOrder 35
   If @ret = 0   print 'error!'
   else
   print 'OrderId entered: ' + cast(@ret as varchar);
   go
   --Deleting it
   drop procedure uspUpdateOrder
   go
   ---------------------
   --deleting a row by creating a store precudure
create procedure uspDeleteOrder
@OrderQuanity smallint
as begin
DELETE [NorthWestDining].[dbo].[RecipeOrder]
where OrderID =1001
 end
 go

 exec uspDeleteOrder @OrderQuanity =35
 --test it
 select * from [dbo].[RecipeOrder]
 --or by calling it
 DECLARE @Ret int
  EXEC  @Ret = uspDeleteOrder 35
   If @ret = 0   print 'error!'
   else
```

```
print 'OrderId entered: ' + cast(@ret as varchar);
go
--test it
select * from [dbo].[RecipeOrder]
```