CSIS05I

Database Systems II

Lab (9)

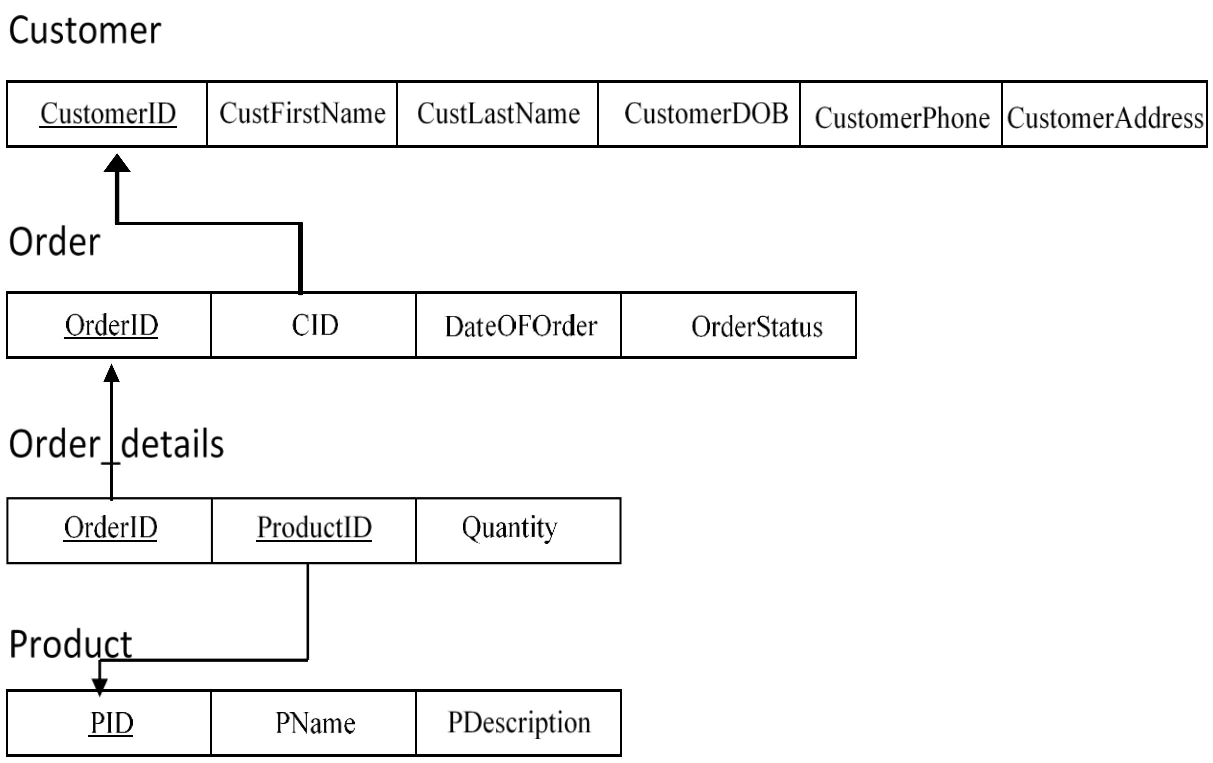
INTRODUCTION TO FUNCTIONS

Lab (9)

The Overview

In the previous lab, we mentioned the stored procedure, its creation syntax and how to use parameters within it.Through this lab, we will learn about functions and how they are used, as well as how to declare and call a function to obtain the output.

Store Schema:



Functions:

SQL functions are functions implemented completely with SQL that can be used to encapsulate logic that can be invoked like a programming sub-routine. SQL Server user-defined functions are routines that accept parameters, perform an action, such as a complex calculation, and return the result of that action as a value. The return value can either be a single scalar value or a result set. You can create SQL scalar functions and SQL table functions.

Function Creation:

To define our Function, we use the CREATE FUNCTION command. Functions are used to compute or return a certain value. It is noted that both functions and procedures can be passed arguments, yet only functions must return a value. The general format for the command is CREATE FUNCTION Function-name.

There are two types of function:

- ✓ ***Scalar user function:*** this function returns a single value and can accept one or more parameters.
- ✓ ***Table-valued function:*** this function can take parameter and return table; It has two types:
  - ➢ Inline Table Valued Function
  - ➢ Multi-Statement Table Valued Function

- ✓ **The scalar Function:** Scalar functions accept one or more parameters but return only one value; therefore, they must include a RETURN statement.

  The basic syntax of the scalar function is as follows:

```
CREATE FUNCTION Function_Name
                    ( @Parameter_Name Data_type,
                      @Parameter_Name Data_type, .... )
RETURNS Data_Type
AS
    BEGIN
        -- Function Body

        RETURN Data

    END
```

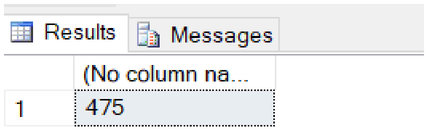
Example 1:

Create a function that calculates the sum of any two numbers the user provides.

```
CREATE FUNCTION AddTwoNumbers (@Num1 int, @Num2 int )
RETURNS INT
AS
Begin
       DECLARE @Sum INT
       Select  @Sum=(@Num1+@Num2)
       RETURN @Sum
END
```

Calling the function:

```
SELECT dbo.AddTwoNumbers  (200,275)
```



```
Results   Messages
     (No column na...
1    475
```

<u>Note:</u> Scalar-valued functions must be invoked / called by using two-part name of the function (SchemaName.TheUserDefinedFunctionName) to allow the database engine differentiate between User created functions and the built-in system functions like getdate(). Also, to specify the schema to which the function belongs.

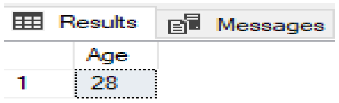
Example 2:

Create a function that calculates the age using the current year and the year of birth given by the user.

```
CREATE FUNCTION calc_age (@current_year int, @DOBYear int)
RETURNS INT

AS
BEGIN

   Declare  @Age  int

       SELECT @Age= @current_year - @DOBYear

   RETURN  @Age;

END
```

Calling the function:

```
SELECT  dbo.calc_age(2021,1993)  as  Age;
```



```
Results   Messages
     Age
1    28
```

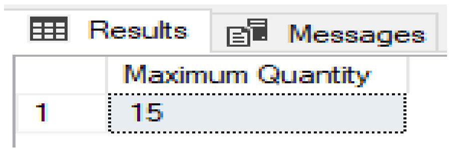
Example 3:

Create a function that retrieves the maximum quantity of a product.

```
CREATE FUNCTION Get_Maximum_Quantity()
RETURNS int
AS
BEGIN

   Declare @Max_Quantity int

       select @Max_Quantity = MAX (Quantity)
       From Order_Details

   RETURN @Max_Quantity

END
```

Calling the function:

```
SELECT dbo.Get_Maximum_Quantity() AS 'Maximum Quantity'
```



```
   Maximum Quantity
1  15
```

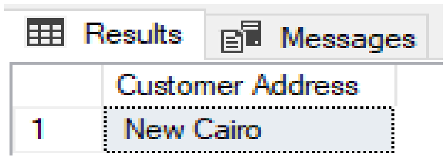
Example 4:

Create a function that retrieves the customer address using customer's ID.

```
CREATE FUNCTION customerADDRESS
(@CustomerID int ) RETURNS varchar(100)
AS
BEGIN
  Declare @address varchar(100)
       Select @address=CustomerAddress
       From Customer
       Where CustomerID= @CustomerID
  RETURN @address
END
```

Calling the function:

```
SELECT dbo.customerADDRESS(1)as 'Customer Address'
```



```
   Customer Address
1  New Cairo
```

Note: *If you need to print a function result, you should save the function result in a variable to be able to print it because 'PRINT' operator is Invalid within SQL Function body. These lines should be written outside the function body.*

```
Declare @address varchar(100)
SELECT @address = dbo.customerADDRESS(1)
Print @address
```

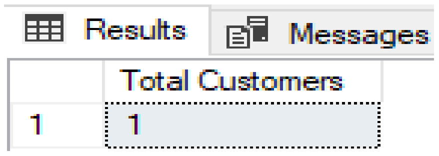
Example 5:

Retrieve the total number of customers that have their orders delivered and the quantity is greater than 5.

```
CREATE FUNCTION Customers_Number()
RETURNS int
AS
BEGIN

  Declare @NumberOfCustomers INT

       SELECT @NumberOfCustomers = COUNT (CID)
       FROM Orderr, Order_Details
       WHERE Orderr.OrderID = Order_Details.OrderID
       And OrderStatus = 'Delivered' And Quantity > 5;

  Return @NumberOfCustomers

END
```

Calling the function:

```
SELECT  dbo.Customers_Number()  as 'Total Customers';
```



	Total Customers
1	1

- ✓ **The inline table-valued Function:** is a user-defined function that returns data of a table type. The return type of a table-valued function is a table; therefore, you can use the table-valued function if you want to retrieve more than one value.

```
CREATE FUNCTION Function_Name
                    (@Parameter_Name Data_type,
                    @Parameter_Name Data_type, ……)
RETURNS table
AS
RETURN
          -- Function Body
```

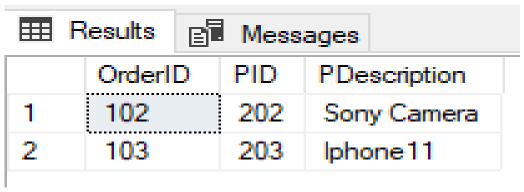
Example 6:

Create a function that retrieves all the orders along with the product ID and Product description where the product description includes 'on' and product quantity is more than 2.

```
CREATE FUNCTION OrderdetailsInformation()
RETURNS table
AS
RETURN
       SELECT OrderID, PID, PDescription
       FROM Order_details join Product
       ON ProductID=PID
       WHERE Pdescription like '%on%' and Quantity > 2;
```

Calling the function:

```
SELECT * from OrderdetailsInformation()
```



	OrderID	PID	PDescription
1	102	202	Sony Camera
2	103	203	Iphone11

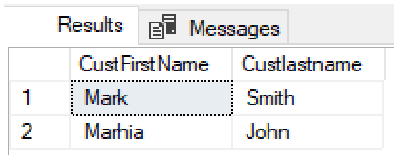
Example 7:

Create a function that retrieves the customers first and last name for the customers who have their orders delivered using complex queries.

```
CREATE FUNCTION CustomerNameFunc()
RETURNS table
AS
RETURN
       SELECT CustFirstName, Custlastname
       FROM Customer
       WHERE CustomerID IN (SELECT CID
                            FROM Orderr
                            WHERE OrderStatus = 'Delivered');
```

Calling the function:

```
SELECT * from CustomerNameFunc()
```



	CustFirstName	Custlastname
1	Mark	Smith
2	Marhia	John

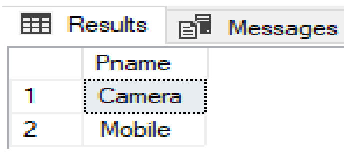
Example 8:

Create a function that retrieves the product names included in all orders except the orders that have made on '26/7/2019' using complex queries.

```
CREATE Function Pname_Order()
Returns table
AS
Return
       Select Pname
       from Product
       where PID not in (Select ProductID
                         from order_details join Orderr
                         on order_details.OrderID = orderr.OrderID
                         where orderr.DateOFOrder = '26/7/2019')
```

Calling the function:

```
SELECT * FROM Pname_Order()
```



	Pname
1	Camera
2	Mobile

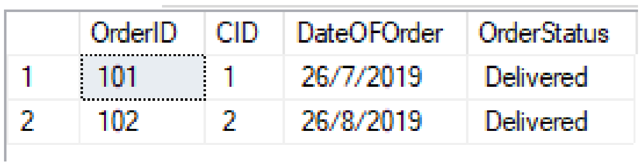
Example 9:

Create a function that retrieves all order information of all customers whose first names starts with 'M' and their total products count is equal to a certain value entered by the user; using complex queries.

```
CREATE Function OrderdetailsFunc(@ProductsCount int)
Returns table
AS
Return
       SELECT *
       FROM Orderr
       WHERE EXISTS (SELECT *
                     FROM Customer
                     WHERE  CID = CustomerID and CustFirstName like 'M%')
              AND

              EXISTS (SELECT * FROM Order_Details
              WHERE Order_Details.OrderID = Orderr.OrderID
              GROUP BY ProductID
              HAVING COUNT(ProductID) = @ProductsCount)
```

Calling the function:

```
SELECT * FROM OrderdetailsFunc(1)
```



	OrderID	CID	DateOFOrder	OrderStatus
1	101	1	26/7/2019	Delivered
2	102	2	26/8/2019	Delivered

✓ **The Multi-statement table-valued Function:** is very similar to inline table valued function as it returns a table as output. But it has different syntax, and it enables the user to specify a certain structure to returned output table.

```
CREATE FUNCTION Function_Name
                (@Parameter_Name Data_type,
                @Parameter_Name Data_type,…)
RETURNS @TABLE table ( columnName1  datatype,
columnName2  datatype)
AS
BEGIN
-- Function Body
  RETURN
END
```

Example 10:

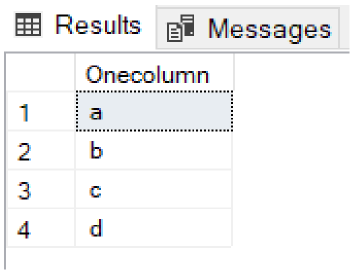
Create a function that splits the values entered by a user into separate words in a table. If the user enter a,b,c,d the result should be a column in a table that has rows, in each row there should be a letter as shown:

A
B
C
D

```
CREATE FUNCTION SpliteTheValue (@String  varchar(25),
                                @delemeter varchar(1))
RETURNS  @table  table  (Onecolumn  varchar(20))
AS
Begin
       Insert into @table
       Select  value  from  String_split (@String  ,  @delemeter)
RETURN
END
```

Calling the function:

```
SELECT  *  FROM  SpliteTheValue('a,b,c,d',',')
```


Results Messages

```
   Onecolumn
1  a
2  b
3  c
4  d
```

String_split: A built-in function that splits a string into rows of substrings, based on a specified separator character. In other words, it splits the string in the first argument by the separator in the second argument.

```
String_Split (@StringVariable, @delemeterVariable)
```

Cast: *A built-in function that converts a value from one data type to another.*

```
CAST (AttributeName AS NewDatatype)
```

To Drop a Function:

```
DROP FUNCTION functionName
```

Exercise

1. Create a function that retrieves the total number of orders.

2. Create a function that calculates the multiplication of two numbers.

3. Create a function that retrieves the first name of a customer giving the customer id entered by the user.

4. Create a function that returns all the customers’ first names and their order ids giving the customer id entered by the user.

5. Create a function that returns all product names along with their order IDs.