Department of Software Engineering Mehran University of Engineering and Technology, Jamshoro

Course: SW222 – Database Management & Administration			
Instructor	Ms Shafiya Qadeer	**Practical/Lab No.**	14
Date	2021	**CLOs**	3
Signature		**Assessment Score**	2 Marks

Topic	**To become familiar with stored procedures and stored functions.**
Objectives	- To become familiar with Creating Procedures and Functions

Lab Discussion: Theoretical concepts and Procedural steps

Stored Procedure:

A stored procedure is a prepared SQL code that you can save, so the code can be reused over and over again.

So if you have an SQL query that you write over and over again, save it as a stored procedure, and then just call it to execute it.

You can also pass parameters to a stored procedure, so that the stored procedure can act based on the parameter value(s) that is passed.

Stored Procedure Syntax:

```
CREATE PROCEDURE procedure_name
AS
sql_statement
GO;
```

Execute a Stored Procedure

```
EXEC procedure_name;
```

Demo Database

Below is a selection from the "Customers" table in the Northwind sample database:

CustomerName	ContactName	Address	City	PostalCode	Country
Alfreds Futterkiste	Maria Anders	Obere Str. 57	Berlin	12209	Germany
Ana Trujillo Emparedados y helados	Ana Trujillo	Avda. de la Constitución 2222	México D.F.	05021	Mexico
Antonio Moreno Taquería	Antonio Moreno	Mataderos 2312	México D.F.	05023	Mexico
Around the Horn	Thomas Hardy	120 Hanover Sq.	London	WA1 1DP	UK
Berglunds snabbköp	Christina Berglund	Berguvsvägen 8	Luleå	S-958 22	Sweden

Stored Procedure Example:

The following SQL statement creates a stored procedure named "SelectAllCustomers" that selects all records from the "Customers" table:

Example:

```
CREATE PROCEDURE SelectAllCustomers
AS
SELECT * FROM Customers
GO;
```

Execute the stored procedure above as follows:

Example:

```
EXEC SelectAllCustomers;
```

Stored Procedure With One Parameter

The following SQL statement creates a stored procedure that selects Customers from a particular City from the "Customers" table:

Example:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30)
AS
SELECT * FROM Customers WHERE City = @City
GO;
```

Execute the stored procedure above as follows:

Example:

```
EXEC SelectAllCustomers City = "London";
```

Stored Procedure With Multiple Parameters

Setting up multiple parameters is very easy. Just list each parameter and the data type separated by a comma as shown below.

The following SQL statement creates a stored procedure that selects Customers from a particular City with a particular PostalCode from the "Customers" table:

Example:

```
CREATE PROCEDURE SelectAllCustomers @City nvarchar(30), @PostalCode
nvarchar(10)
AS
SELECT * FROM Customers WHERE City = @City AND PostalCode = @PostalCode
GO;
```

Execute the stored procedure above as follows:

Example:

```
EXEC SelectAllCustomers City = "London", PostalCode = "WA1 1DP";
```

Stored Function:

A stored function is a special kind stored program that returns a single value.

You use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

Different from a stored procedure, you can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.

MySQL stored function syntax

The following illustrates the simplest syntax for creating a new stored function:

```
CREATE FUNCTION function_name(param1,param2,…)
    RETURNS datatype
   [NOT] DETERMINISTIC
 statements
```

First, you specify the name of the stored function after CREATE FUNCTION clause.

Second, you list all parameters of the stored function inside the parentheses. By default, all parameters are the IN parameters. You cannot specify IN , OUT or INOUT modifiers to the parameters.

Third, you must specify the data type of the return value in the RETURNS statement. It can be any valid MySQL data types.

Fourth, for the same input parameters, if the stored function returns the same result, it is considered deterministic; otherwise, the stored function is not deterministic. You have to decide whether a stored function is deterministic or

not. If you declare it incorrectly, the stored function may produce an unexpected result, or the available optimization is not used which degrades the performance. Fifth, you write the code in the body of the stored function. It can be a single statement or a compound statement. Inside the body section, you have to specify at least one RETURN statement. The RETURNstatement returns a value to the caller. Whenever the RETURN statement is reached, the stored function's execution is terminated immediately.

MySQL stored function example

Let's take a look at an example of using the stored function. We will use the customers table in the sample database for the demonstration.

The following example is a function that returns the level of a customer based on credit limit. We use the IF statement to determine the credit limit.

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
    DETERMINISTIC
BEGIN
    DECLARE lvl varchar(10);

    IF p_creditLimit > 50000 THEN
 SET lvl = 'PLATINUM';
    ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
        SET lvl = 'GOLD';
    ELSEIF p_creditLimit < 10000 THEN
        SET lvl = 'SILVER';
    END IF;

 RETURN (lvl);
END
```

customerName	CustomerLevel(creditLimit)
Alpha Cognac	PLATINUM
American Souvenirs Inc	SILVER
Amica Models & Co.	PLATINUM
ANG Resellers	SILVER
Anna's Decorations, Ltd	PLATINUM
Anton Designs, Ltd.	SILVER
Asian Shopping Network, Co	SILVER
Asian Treasures, Inc.	SILVER
Atelier graphique	GOLD
Australian Collectables, Ltd	PLATINUM
Australian Collectors, Co.	PLATINUM

Now, we can call the CustomerLevel() in a SELECT statement as follows:

```
SELECT
    customerName,
    CustomerLevel(creditLimit)
FROM
    customers
ORDER BY
    customerName;
```

We also rewrite the GetCustomerLevel() stored procedure that we developed in the MySQL IF statement tutorial as follows:

```
DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(
    IN  p_customerNumber INT(11),
    OUT p_customerLevel  varchar(10)
)
BEGIN
    DECLARE creditlim DOUBLE;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    SELECT CUSTOMERLEVEL(creditlim)
    INTO p_customerLevel;

END
```

As you can see, the GetCustomerLevel() stored procedure is much more readable when using the CustomerLevel() stored function.

Notice that a stored function returns a single value only. If you include a SELECT statement without the INTO clause, you will get an error.

In addition, if a stored function contains SQL statements, you should not use it inside other SQL statements; otherwise, the stored function will slow down the speed of the query.

In this tutorial, you have learned how to create a stored function to encapsulate the common formula or business rules.

Lab Tasks

1. Run the examples given above
2. Create your Stored procedure and function using tables of your dbs