Lab 8 – Creating Functions and Stored Procedures

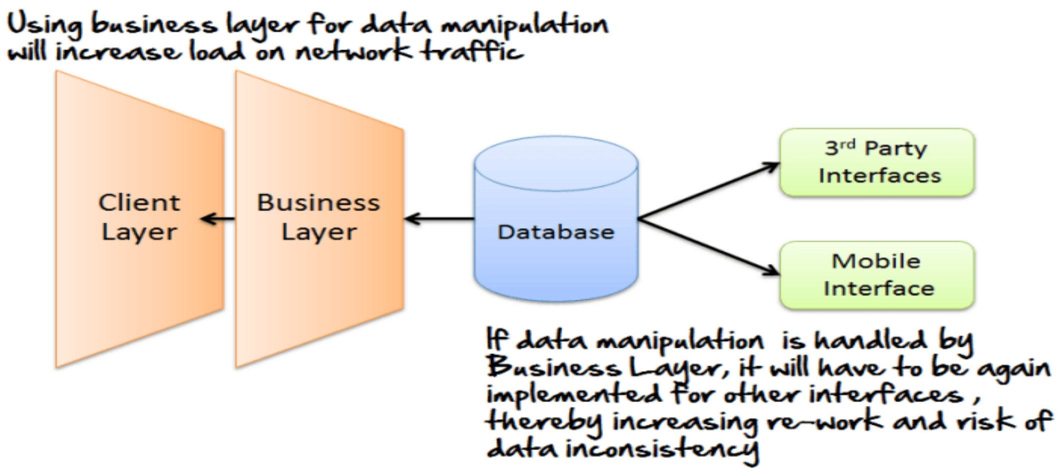
What are functions?

MySQL can do much more than just store and retrieve data. We can also **perform manipulations on the data** before retrieving or saving it. That's where MySQL Functions come in. Functions are simply pieces of code that perform some operations and then return a result. Some functions accept parameters while other functions do not accept parameters.

What are Stored procedures?

A stored procedure is a subroutine available to applications that access a relational database management system. Such procedures are stored in the database data dictionary. Uses for stored procedures include data-validation or access-control mechanisms.

Why do we need functions?


Using business layer for data manipulation will increase load on network traffic
Client Layer
Business Layer
Database
3rd Party Interfaces
Mobile Interface
If data manipulation is handled by Business Layer, it will have to be again implemented for other interfaces, thereby increasing re-work and risk of data inconsistency

Let' briefly look at an example of MySQL function. By default, MySQL saves date data types in the format "YYYY-MM-DD". Suppose we have built an application and our users want the date to be returned in the format "DD-MM-YYYY", we can use MySQL built in function DATE_FORMAT to achieve this. DATE_FORMAT is one of the most used functions in MySQL. We will look at it in more details as we unfold the lesson.

"Why bother MySQL Functions? The same effect can be achieved with scripting/programming language?" It's true we can achieve that by writing some procedures/function in the application program.

Getting back to our DATE example in the introduction, for our users to get the data in the desired format, business layer will have to do necessary processing.

This becomes a problem when the application has to integrate with other systems. This **reduces re-work in the business logic and reduce data inconsistencies.**
Another reason why we should consider using **MySQL functions is the fact that it can help reducing network traffic in client/server applications**. Business Layer will only need to make call to the stored functions without the need manipulate data. On average, the use of functions can help greatly improve overall system performance.

Why do we need Stored procedures?

The main purpose of stored procedures in SQL is to ***hide direct SQL queries from the code and improve the performance of database operations*** such as select, update, and delete data. Other advantages of procedure in SQL are: Reduces the amount of information sent to the database server.

1. Reduce the Network Traffic: Multiple SQL Statements are encapsulated in a stored procedure. When you execute it, instead of sending multiple queries, we are sending only the name and the parameters of the stored procedure
2. Easy to maintain: The stored procedure are reusable. We can implement the business logic within an SP, and it can be used by applications multiple times, or different modules of an application can use the same procedure. This way, a stored procedure makes the database more consistent. If any change is required, you need to make a change in the stored procedure only
3. Secure: The stored procedures are more secure than the AdHoc queries. The permission can be granted to the user to execute the stored procedure without giving permission to the tables used in the stored procedure. The stored procedure helps to prevent the database from SQL Injection

Difference between functions and Stored procedures

Stored procedures and functions can be used to accomplish the same task. Both can be custom-defined as part of any application, but functions are designed to send their output to a query or SQL statement. Stored procedures are designed to return outputs to the application, while a user-defined function returns table variables and cannot change the server environment or operating system environment.

Following are the main differences between functions and procedures:

Functions	Procedures
A function has a return type and returns a value.	A procedure does not have a return type. But it returns values using the OUT parameters.
You cannot use a function with Data Manipulation queries. Only Select queries are allowed in functions.	You can use DML queries such as insert, update, select etc... with procedures.
A function does not allow output parameters	A procedure allows both input and output parameters.
You cannot manage transactions inside a function.	You can manage transactions inside a procedure.
You cannot call stored procedures from a function	You can call a function from a stored procedure.
You can call a function using a select statement.	You cannot call a procedure using select statements.

MySQL Functions:

Built-in Function:

MySQL comes bundled with a number of built-in functions. Built in functions are simply functions come already implemented in the MySQL server. These functions allow us to perform different types of manipulations on the data. The built-in functions can be basically categorized into the following most used categories.

- **Strings functions** – operate on string data types
- **Numeric functions** – operate on numeric data types
- **Date functions** – operate on date data types
- **Aggregate functions** – operate on all of the above data types and produce summarized result sets.

String functions:

- CHARACTER_LENGTH(***str***)

CHARACTER_LENGTH() is a synonym for CHAR_LENGTH().

- CONCAT(***str1***,***str2***,...)
  Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.

  CONCAT() returns NULL if any argument is NULL.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

- INSTR(***str***,***substr***)
  Returns the position of the first occurrence of substring ***substr*** in string ***str***. This is the same as the two-argument form of LOCATE(), except that the order of the arguments is reversed.

```
mysql> SELECT INSTR('foobarbar', 'bar');
        -> 4
mysql> SELECT INSTR('xbar', 'foobar');
        -> 0
```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string. If either argument is NULL, this functions returns NULL.

- LCASE(***str***)
  LCASE() is a synonym for LOWER().
  LCASE() used in a view is rewritten as LOWER() when storing the view's definition
  LOWER(***str***)

Returns the string ***str*** with all characters changed to lowercase according to the current character set mapping, or NULL if ***str*** is NULL. The default character set is utf8mb4.

```
  mysql> SELECT LOWER('QUADRATICALLY');
-> 'quadratically'
```

- REPLACE(***str***,***from_str***,***to_str***)
  Returns the string ***str*** with all occurrences of the string ***from_str*** replaced by the string ***to_str***. REPLACE() performs a case-sensitive match when searching for ***from_str***.

```
mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

  This function is multibyte safe. It returns NULL if any of its arguments are NULL.

- REVERSE(***str***)
  Returns the string ***str*** with the order of the characters reversed, or NULL if ***str*** is NULL.

```
mysql> SELECT REVERSE('abc');
        -> 'cba'
```

- UCASE function to do that. It takes a string as a parameter and converts all the letters to upper case. The script shown below demonstrates the use of the “UCASE” function.

SELECT `movie_id`,`title`, UCASE(`title`) FROM `movies`;
HERE

- *UCASE(`title`) is the built in function that takes the title as a parameter and returns it in upper case letters with the alias name `upper_case_title`.*

Numeric functions:

these functions operate on numeric data types. We can perform mathematic computations on numeric data in the SQL statements.
MySQL supports the following arithmetic operators that can be used to perform computations in the SQL statements.

Name	Description
DIV	Integer division
/	Division
–	Subtraction
+	Addition
*	Multiplication
% or MOD	Modulus

Let's now look at examples of each of the above operator

Integer Division (DIV)

```
SELECT 23 DIV 6 ;
```

Executing the above script gives us the following results.

```
3
```

Division operator (/)

Let's now look at the division operator example. We will modify the DIV example.

```
SELECT 23 / 6 ;
```

Executing the above script gives us the following results.

```
3.8333
```

Date Functions:

Functions	Description
ADDDATE()	MySQL ADDDATE() adds a time value with a date.

CURDATE()	In MySQL the CURDATE() returns the current date in 'YYYY-MM-DD' format or 'YYYYMMDD' format depending on whether numeric or string is used in the function.
CURRENT_DATE()	In MySQL the CURRENT_DATE returns the current date in 'YYYY-MM-DD' format or YYYYMMDD format depending on whether numeric or string is used in the function.
DATE_ADD()	MySQL DATE_ADD() adds time values (as intervals) to a date value. The ADDDATE() is the synonym of DATE_ADD().
DATE_FORMAT()	MySQL DATE_FORMAT() formats a date as specified in the argument. A list of format specifiers given bellow may be used to format a date.
DATE_SUB()	MySql date_sub() function subtract a time value (as interval) from a date.
DATE()	MySQL DATE() takes the date part out from a datetime expression.
DATEDIFF()	MySQL DATEDIFF() returns the number of days between two dates or datetimes.
DAY()	MySQL DAY() returns the day of the month for a specified date.
DAYNAME()	MySQL DAYNAME() returns the name of the week day of a date specified in the argument.
DAY OF MONTH()	MySQL DAYOFMONTH() returns the day of the month for a given date.
DAY OF WEEK()	MySQL DAYOFWEEK() returns the week day number (1 for Sunday,2 for Monday …… 7 for Saturday) for a date specified as an argument.

DAY OF YEAR()	MySQL DAYOFYEAR() returns day of the year for a date. The return value is within the range of 1 to 366.
EXTRACT()	MySQL EXTRACT() extracts a part of a given date.
FROM_DAYS()	MySQL FROM_DAYS() returns a date against a datevalue.
FROM_UNIXTIME()	MySQL FROM_UNIXTIME() returns a date /datetime from a version of unix_timestamp.
GET_FORMAT()	MySQL GET_FORMAT() converts a date or time or datetime in a formatted manner as specified in the argument.
LAST_DAY()	MySQL LAST_DAY() returns the last day of the corresponding month for a date or datetime value.
MONTH()	MySQL MONTH() returns the month for the date within a range of 1 to 12 (January to December).
MONTHNAME()	MySQL MONTHNAME() returns the full name of the month for a given date.
NOW()	MySQL NOW() returns the value of current date and time in 'YYYY-MM-DD HH:MM:SS' format or YYYYMMDDHHMMSS.uuuuuu format depending on the context (numeric or string) of the function.
STR_TO_DATE()	MySQL STR_TO_DATE() returns a datetime value by taking a string and a specific format string as arguments.
SUBDATE()	MySQL SUBDATAE() subtracts a time value (as interval) from a given date.
SYSDATE()	MySQL SYSDATE() returns the current date and time in YYYY-MM-DD HH:MM:SS or YYYYMMDDHHMMSS.uuuuuu format depending on the context of the function.

TO_DAYS()	MySQL TO_DAYS() returns number of days between a given date and year 0.
WEEK()	MySQL WEEK() returns the week number for a given date.
WEEKDAY()	MySQL WEEKDAY() returns the index of the day in a week for a given date (0 for Monday, 1 for Tuesday and ......6 for Sunday).
WEEK OF YEAR()	MySQL WEEKOFYEAR() returns the calender week (as a number) of a given date.
YEAR()	MySQL YEAR() returns the year for a given date.
YEARWEEK()	MySQL YEARWEEK() returns year and week number for a given date.

Stored functions:

Stored functions are just like built in functions except that you have to define the stored function yourself. Once a stored function has been created, it can be used in SQL statements just like any other function. The basic syntax for creating a stored function is as shown below

```
CREATE FUNCTION sf_name ([parameter(s)])
  RETURNS data type
  DETERMINISTIC
  STATEMENTS
```

HERE

- **"CREATE FUNCTION sf_name ([parameter(s)]) "** is mandatory and tells MySQL server to create a function named `sf_name' with optional parameters defined in the parenthesis.
- **"RETURNS data type"** is mandatory and specifies the data type that the function should return.
- **"DETERMINISTIC"** means the function will return the same values if the same arguments are supplied to it.
- **"STATEMENTS"** is the procedural code that the function executes.

Demo Problem for Functions:

Let’s now look at a practical example that implements a built-in function. **Suppose we want to know which bike’s service are past the delivery date(check if any bikes service is due wrt current date).** We can create a stored function that accepts the delivery date as the parameter and then compares it with the current date in MySQL server. If the current date is less than the delivery date, then we return “No” else we return “Yes”. The script shown below helps us to achieve that.

```
DELIMITER $$

CREATE FUNCTION sf_delivery_due_date(delivery_date DATE)
RETURNS VARCHAR(50)

 BEGIN

   DECLARE sf_value VARCHAR(50);

   IF CURRENT_DATE() > delivery_date THEN
            SET sf_value = 'Bike Delivered';

   ELSEIF CURRENT_DATE() <= delivery_date THEN
            SET sf_value = 'Bike not Delivered';

          ELSE
            SET sf_value= 'Delivered but Customer didnt pay';

   END IF;

          RETURN sf_value;

END; $$

DELIMITER ;
```

Executing the above script created the stored function ` sf_delivery_due_date `.

Let’s now test our stored function.

```
SELECT
Cust_Id,service_comments,Date_Rec,CURDATE(),sf_delivery_due_date(Date_del)
FROM  service_ticket;
```

Syntax:

```
DELIMITER $$

CREATE FUNCTION function_name(
   param1,
   param2,...
)
RETURNS datatype
[NOT] DETERMINISTIC
BEGIN
 -- statements
END $$

DELIMITER ;
```

Assignment:

Write a function to find the number of tickets booked by a customer. If no of tickets is more than 3 for the current month then display error message as “cannot purchase tickets current limit is over”

Demo Problem for Stored Procedures:

Syntax:

```
Create Procedure [Procedure Name] ([Parameter 1], [Parameter 2], [Parameter 3] )
Begin
SQL Queries..
End
```

In the syntax:

1. The name of the procedure must be specified after the **Create Procedure** keyword
2. After the name of the procedure, the list of parameters must be specified in the parenthesis. The parameter list must be comma-separated
3. The SQL Queries and code must be written between **BEGIN** and **END** keywords

To execute the store procedure, you can use the CALL keyword. Below is syntax:

```
CALL [Procedure Name] ([Parameters]..)
```

In the syntax:

1. The procedure name must be specified after the CALL keyword
2. If the procedure has the parameters, then the parameter values must be specified in the parenthesis

Demo:

Write a procedure to update quantity of parts once the service ticket is raised for the part replacement also display mechanic who provided replacement service.

```
DELIMITER $$
CREATE procedure part_replacement(
IN SID int,IN M_ID int,IN PID int, OUT msg varchar(30))
BEGIN
DECLARE cnt int;
set cnt= (SELECT Qty from parts where P_ID=PID);

IF cnt = 0 THEN
  set msg= 'NO PARTS AVAILABLE';
ELSE
  update parts
  set Qty=Qty-1
  where P_ID= PID;

 set msg=(select M.F_Name
  from works w,mechanic m
  where w.Mech_ID=m.Mech_ID and w.S_ID=SID and w.P_ID=PID and
  w.Mech_ID=M_ID);



END IF;
```

```
END;$$
DELIMITER ;
```

Calling the stored procedure:

```
CALL part_replacement(113,12,67,@M);
SELECT @M;
```

Assignment:

Write a stored procedure to calculate the age of the customer when the date of birth is given. Update the column named age in the customer table.

Submission of Guildelines:

Create a pdf with function and procedure code and add appropriate output screenshot and submit pdf along with .sql files of function and procedures.
1 pdf file+2 sql files.