MySQL Functions: String, Numeric, User-Defined, Stored

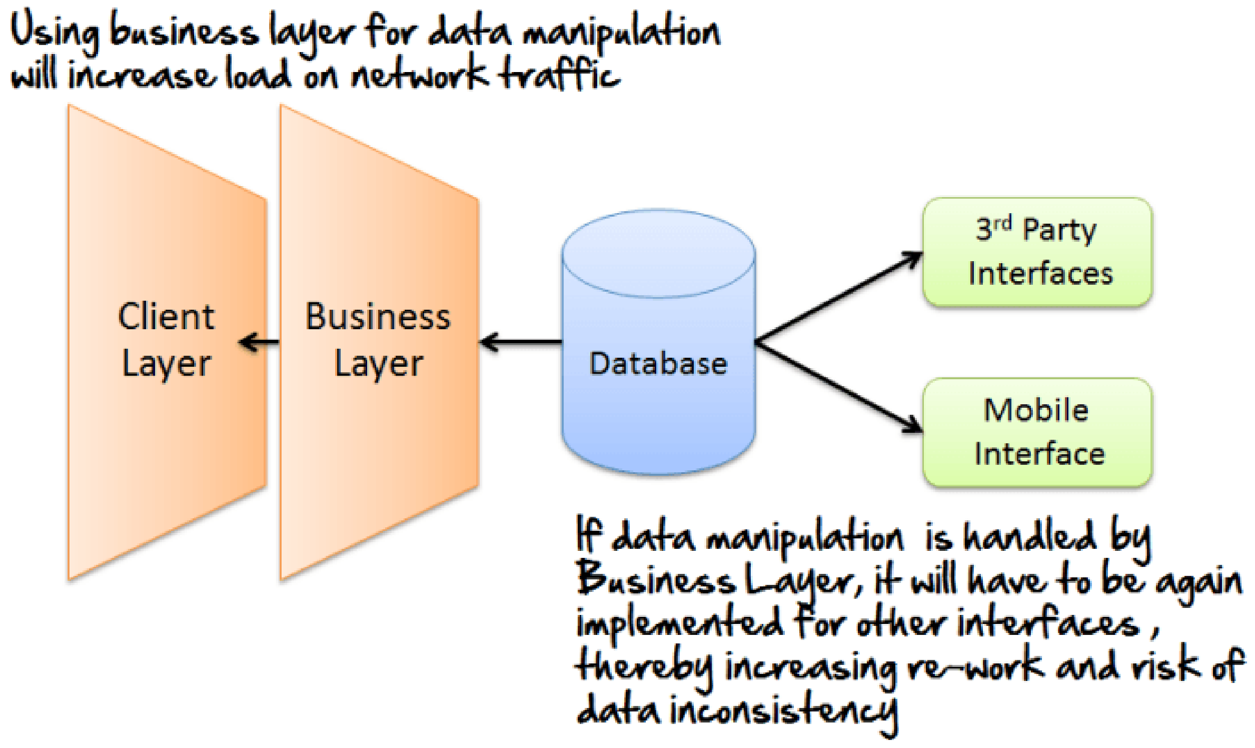
What are functions?

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.

Let's briefly look at an example of a MySQL function. By default, MySQL saves 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 detail as we unfold the lesson.

Why use functions?

Why Use 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

Based on the example given in the introduction, people with experience in computer programming may be thinking *"Why bother MySQL Functions? Can the same effect be achieved with scripting/programming language?"* It's true we can achieve that by writing some procedures/functions in the application program.

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

This becomes a problem when the application has to integrate with other systems. When we use MySQL functions such as the DATE_FORMAT, then we can have that functionality embedded into the database and any application that needs the data gets it in the required format. This **reduces re-work in the business logic and reduces data inconsistencies.**

Another reason why we should consider using **MySQL functions is the fact that it can help reduce network traffic in client/server applications**.

Business Layers will only need to make calls to the stored functions without the need to manipulate data .On average, the use of functions can help greatly improve overall system performance.

Types of functions

Built-in functions

MySQL comes bundled with a number of built-in functions. Built in functions are simply functions 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.
- **Other functions** - MySQL also supports other types of built in functions but we will limit our lesson to the above named functions only.

Let's now look at each of the functions mentioned above in detail. We will be explaining the most used functions using our "Myflixdb".

String functions

We already looked at what string functions do. We will look at a practical example that uses them. In our movies table, the movie titles are stored using combinations of lower and upper case letters. Suppose we want to get a query list that returns the movie titles in upper case letters. We can use the "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`.

Executing the above script in MySQL workbench against the Myflixdb gives us the following results shown below.

movie_id	**title**	**UCASE('title')**
16	67% Guilty	67% GUILTY
6	Angels and Demons	ANGELS AND DEMONS
4	Code Name Black	CODE NAME BLACK
5	Daddy's Little Girls	DADDY'S LITTLE GIRLS
7	Davinci Code	DAVINCI CODE
2	Forgetting Sarah Marshall	FORGETTING SARAH MARSHAL
9	Honey mooners	HONEY MOONERS
19	movie 3	MOVIE 3
1	Pirates of the Caribbean 4	PIRATES OF THE CARIBEAN 4
18	sample movie	SAMPLE MOVIE
17	The Great Dictator	THE GREAT DICTATOR

3	X-Men	X-MEN

MySQL supports a number of string functions.

Numeric functions

As earlier mentioned, these functions operate on numeric data types. We can perform mathematical computations on numeric data in the SQL statements.

Arithmetic operators

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

Subtraction operator (-)

Let's now look at the subtraction operator example. We will use the same values as in the previous two examples

```
SELECT 23 - 6 ;
```

Executing the above script gives us 17

Addition operator (+)

Let's now look at the addition operator example. We will modify the previous example.

```
SELECT 23 + 6 ;
```

Executing the above script gives us 29

Multiplication operator (*)

Let's now look at the multiplication operator example. We will use the same values as in the previous examples.

```
SELECT 23 * 6 AS `multiplication_result`;
```

Executing the above script gives us the following results.

multiplication_result
138

Modulo operator (-)

The modulo operator divides N by M and gives us the remainder. Let's now look at the modulo operator example. We will use the same values as in the previous examples.

```
SELECT 23 % 6 ;
```

OR

```
SELECT 23 MOD 6 ;
```

Executing the above script gives us 5

Let's now look at some of the common numeric functions in MySQL.

Floor - this function removes decimals places from a number and rounds it to the nearest lowest number. The script shown below demonstrates its usage.

```
SELECT FLOOR(23 / 6) AS `floor_result`;
```

Executing the above script gives us the following results.

Floor_result
3

Round - this function rounds a number with decimal places to the nearest whole number. The script shown below demonstrates its usage.

```
SELECT ROUND(23 / 6) AS `round_result`;
```

Executing the above script gives us the following results.

Round_result
4

Rand - this function is used to generate a random number, its value changes every time that the function is called. The script shown below demonstrates its usage.

```
SELECT RAND() AS `random_result`;
```

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.

Let's now look at a practical example that implements a built in function. Suppose we want to know which rented movies are past the return date. We can create a stored function that accepts the return date as the parameter and then compares it with the current date in MySQL server. If the current date is less than the return movie date, then we return "No" else we return "Yes". The script shown below helps us to achieve that.

```
DELIMITER |
CREATE FUNCTION sf_past_movie_return_date (return_date
DATE)
  RETURNS VARCHAR(3)
   DETERMINISTIC
    BEGIN
     DECLARE sf_value VARCHAR(3);
        IF curdate() > return_date
            THEN SET sf_value = 'Yes';
        ELSEIF  curdate() <= return_date
            THEN SET sf_value = 'No';
        END IF;
     RETURN sf_value;
    END|
```

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

Let's now test our stored function.

```
SELECT
`movie_id`,`membership_number`,`return_date`,CURDATE()
,sf_past_movie_return_date(`return_date`)  FROM
`movierentals`;
```

Executing the above script in MySQL workbench against the myflixdb gives us the following results.

movie_id	**membership_number**	**return_date**	**CURDATE()**	**sf_past_movie_return_date('return_date')**
1	1	NULL	04-08-2012	NULL
2	1	25-06-2012	04-08-2012	yes
2	3	25-06-2012	04-08-2012	yes
2	2	25-06-2012	04-08-2012	yes
3	3	NULL	04-08-2012	NULL

User-defined functions

MySQL also supports user defined functions that extend MySQL. User defined functions are functions that you can create using a programming language such as C, C++ etc. and then add them to MySQL server. Once added, they can be used just like any other function.

Summary

- Functions allow us to enhance the capabilities of MySQL.
- Functions always return a value and can optionally accept parameters.
- Built in functions are functions that are shipped with MySQL. They can be categorized according to the data types that they operate on i.e. strings, date and numeric built in functions.
- Stored functions are created by the user within MySQL server and can be used in SQL statements.
- User defined functions are created outside MySQL and can be incorporated into MySQL server