Lesson 10-Stored Functions

Lesson 10 Concepts
1. Gain working knowledge of how to create and use functions in MySQL
2. Understand the limitations of what a function can do

Functions perform a specific task and returns a value.
A function is a block of organized, reusable code that performs a specific task such as a computation and the returns the result of that task as a value. It has a unique name and acts as an independent block of code. The stored function can be called within SQL statements.

SYNTAX:

```
CREATE FUNCTION function_name(parameters)
RETURNS output_datatype -- DETERMINISTIC
RETURN parameter;
```

In MySQL, a function is defined by four main components. The function_name is the unique name you give to the function. The **parameters** are variables (i.e. a named placeholder) that we use to represent the value that will get input into the function. The **output_datatype** is the type of value that the function will return after it's done processing. Lastly, the **function body** contains the SQL statements that define what the function does with the input parameters to produce the output.

2 Types of functions
(1) System defined functions
(2) User defined function

System defined functions are native built in function that provided by mySQL. We have already learned some system functions such as CONCAT(), COUNT(), SUM(), AVG() etc.

User-defined functions (UDFs) in MySQL are custom functions that users create to perform specific tasks. There are three main types of UDFs:

1. **Functions that return a single value**: These functions take one or more inputs, perform some operations, and then return a single result. For example, a function could take two numbers as input and return their sum.
2. **Functions that return a table as a result**: These functions perform operations that result in a table of values. For example, a function could take a category name as input and return a table of all products in that category.
3. **Functions that perform calculations** on a set of values and return a single value: These functions take multiple inputs, perform calculations on them, and then return a single result. For example, a function could take a list of numbers as input and return their average.

Creating user-defined functions can make your SQL code more efficient and easier to read, especially when you find yourself writing the same queries or calculations over and over again. It's like creating your own custom toolset for working with your data

Example 1:

```
CREATE FUNCTION hello (s CHAR(20))
   RETURNS CHAR(50) DETERMINISTIC
   RETURN CONCAT('Hello, ',s,'!');
```

The table : Names

id	name	age
1	Bob	70
2	John	50
3	Paul	40

Select a column as a parameter:

```
SELECT hello(name) FROM names;
```

Result:

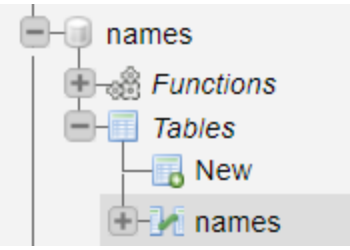
hello(name)
Hello, Bob!
Hello, John!
Hello, Paul!

You can locate a function in MySQL in 2 ways:

1. Execute the SHOW FUNCTION STATUS statement. Here's an example:

```
SHOW FUNCTION STATUS
WHERE db = 'petstore';
```
-- replace 'petstore9a' with your database name

2. Alternatively, once the function is created, you can view it in the left menu under "Functions" in the PHPMyAdmin interface. To do this, select the database and you should see a "Functions" option with a plus sign next to it.



Multiple parameters for stored functions:

In MySQL, you can create a function that takes more than one parameter. For example, you can create a function to concatenate a first name and a last name. Here's how you can do it:

```
CREATE FUNCTION full_name(first_name CHAR(20),
last_name CHAR(30))
RETURNS CHAR(50)
RETURN CONCAT(first_name, ' ', last_name);
```

Let's break down the components of this function:

- CREATE FUNCTION full_name: This line begins the creation of a stored function named full_name.
- (first_name CHAR(20), last_name CHAR(30)): These are the parameters of the function. It accepts two parameters: first_name of type CHAR with a length of 20, and last_name of type CHAR with a length of 30.
- RETURNS CHAR(50): This line declares that the function will return a value of type CHAR with a length of 50.
- RETURN CONCAT(first_name, ' ', last_name);: This is the body of the function. It uses the CONCAT function to concatenate the first_name, a space, and the last_name.

Where to use stored functions in SQL statements?
Once you've created a custom function in MySQL, you can use it in various ways:

1. **Function for Individual Value**: For example, you can use the full_name function to concatenate the first name "Jas" and the last name "Kaur":

```
SELECT full_name("Jas", "Kaur") AS "Name";
```

2. **Function in Computed Results**: You can use the function in a SELECT statement to operate on the values in a table. For example, you can use the full_name function to concatenate the first_name and last_name of each record in the employees table :

```
SELECT full_name(first_name, last_name) AS "Name" from
employees;
```

3. **Function in WHERE Clause**: You can use the function in a WHERE clause to filter records. For example, you can use the full_name function to find all employees whose first or last name contains an “L”.

```
SELECT *  FROM employees WHERE full_name(first_name, last_name) LIKE "%L%";
```

DECLARE

Suppose you want to create a function that calculates an employee’s annual salary based on their hourly pay. Here’s how you can do it:

```
DELIMITER //
CREATE FUNCTION CalculateAnnualSalary(pay_per_hour DECIMAL(10, 2))
RETURNS DECIMAL(10, 2)
BEGIN
   DECLARE annual_salary DECIMAL(10, 2);
   SET annual_salary = pay_per_hour * 2000;
   RETURN annual_salary;
END; //
DELIMITER ;
```

This function is composed of two parts: the function declaration and the function body.

1. **Function Declaration:** The function is declared with the name CalculateAnnualSalary. It takes one parameter, pay_per_hour, of type DECIMAL(10,2). The function is designed to return a DECIMAL value with a precision of 10 digits and a scale of 2.
2. **Function Body:** The body of the function is enclosed within the BEGIN and END keywords. Inside the function body, a local variable annual_salary of type DECIMAL is declared to store the calculated annual salary. The SET statement assigns the value of pay_per_hour multiplied by 2000 to the annual_salary variable, assuming a 40-hour workweek and 52 weeks in a year. The RETURN statement then returns the calculated annual_salary value. So, when you call this function with a specific pay_per_hour value, it will return the corresponding annual salary based on the calculation provided in the function body.

Now, you can use this function in a SELECT statement to get the annual salary of each employee:

```
SELECT CalculateAnnualSalary(25.00) AS AnnualSalary;
```

This will return the annual salary for an employee who earns $25.00 per hour.

SELECT …. INTO

Is it possible to assign the result of a query to a variable using the SELECT ... INTO syntax.

```
DELIMITER  //

CREATE FUNCTION `item_sales`(item_id INT)
RETURNS int(10)
BEGIN
    DECLARE sales_count INT;

    SELECT COUNT(*) INTO sales_count
    FROM sales
    WHERE sales.item = item_id;

    RETURN sales_count;
END;  //

DELIMITER ;
```

In this function, item_id is a parameter, which is a placeholder for a value that you pass to the function when you call it. Parameters make your function more versatile by allowing dynamic input.
You can call this function in a SELECT statement to find the number of sales for a specific item. For example, to find the number of sales for the item with ID 1014, you can do:

```
SELECT item_sales(1014) AS "1014 sales_count";
```

IF STATEMENT
IF statements can be used in stored functions. The syntax is as follows:

```
IF condition THEN
    statements when condition is true
ELSE
    statements when condition is false
END IF;
```

You can use IF statements in your triggers, custom functions, and stored procedures. Here is an example for the use of the IF statement in a stored function:

```
DELIMITER //
CREATE FUNCTION `GetStockStatus`(item_id INT)
RETURNS varchar(20)
BEGIN
   DECLARE quantity INT;
   SELECT inventory INTO quantity FROM Stock_Items
WHERE id = item_id;
   IF quantity > 0 THEN
      RETURN 'In Stock';
   ELSE
      RETURN 'Out of Stock';
   END IF;
END //
DELIMITER ;
```

MODIFY or DROP EXISTING FUNCTION

Once created, it is not possible to directly modify a function. Instead, you have to drop the function and then recreate it with a new definition.

```
DROP FUNCTION  function_name;
```
or
```
DROP FUNCTION IF EXISTS function_name;
```

IF EXISTS: This optional clause ensures that the database will not throw an error if the specified function does not exist. If the function exists, it will be dropped; if it doesn't, no error will be raised.