# Discussion-6 MySQL- difference between stored procedure and functions

**Discussion Topic:**

Discuss the difference between stored procedure and functions.
Give an example of when you would use one instead of the other.
If you were a new database administrator, would it be easier to develop stored procedures or functions?

**My Post:**

Hello Class,

MySQL supports stored routines (procedures and functions) (MySQL, n.d.). Stored routines are a set of SQL statements that are compiled and stored on the server database, allowing them to be accessed by users. This article provides an overview of MySQL stored procedures and functions, and compares and evaluates their characteristics, differences, and use cases.

**Store Procedures**

Stored procedures can be defined as a set of SQL statements that can be invoked with the CALL keyword (Musgrave, 2024). Applications that have access to the database can CALL procedures stored on the database server. In other words, a stored procedure is a database object containing procedural SQL code that can be invoked at any time by an application or user to fetch and/or manipulate data. (Murach, 2019).

Syntax and creation of stored procedures:

```
CREATE PROCEDURE procedure_name (
     [parameter_name_1 data_type]
    [, parameter_name_2 data_type]...
)
-- sql_block -- This represents the body of the procedure
```

Example of creating/calling/dropping a procedure:
The example provided is from “Chapter 15: How to create stored procedures and functions. Murach’s MySQL (3rd ed.)” By Murrah (2019). In this example, the procedure updates the credit_total column of the invoices table.

```
-- Change the standard delimiter from semicolon (;) to double slash (//)
-- Semicolons are used within the procedure body,
-- and we need a way to tell MySQL where the entire CREATE PROCEDURE statement ends,
-- therefore, we need to define a delimiter for the procedure itself
DELIMITER //

-- Create a new stored procedure named 'update_invoices_credit_total'.
-- This procedure takes two input parameters.
CREATE PROCEDURE update_invoices_credit_total (
    invoice_id_param INT,
    credit_total_param DECIMAL(9,2)
)
-- Start of the procedure's body
BEGIN
    -- Declare a local variable to track SQL errors.
    -- It's a TINYINT (often used for boolean flags) and defaults to FALSE (0).
```

```
    DECLARE sql_error TINYINT DEFAULT FALSE;
       SET sql_error = TRUE;

    -- Start a new transaction. This allows us to group SQL statements together
    START TRANSACTION;

    -- MySQL/SQL statements
    UPDATE invoices
    SET credit_total = credit_total_param
    WHERE invoice_id = invoice_id_param;

    -- Check if an SQL error occurred during the UPDATE statement
    IF sql_error = FALSE THEN
        COMMIT;
    ELSE
        ROLLBACK;
    END IF;
-- End of the procedure's body.
END //
-- Reset the delimiter back to the standard semicolon (;).
DELIMITER ;
```

Invoking Procedure:

```
CALL update_invoices_credit_total(56, 504);
```

Dropping the procedure

```
DROP PROCEDURE IF EXISTS update_invoices_credit_total;
```

**Store Functions**

Stored functions can be defined as a block of SQL code that can perform computations or transformations, and return a single scalar value (200ok Solutions 2024).
MySQL provides its own set of stored functions referred to as built-in functions, such as CONCAT(), NOW(), or SUM(). For a list of all the MySQL built-in functions, see: MySQL Functions by W$^3$ School (n.d.).
Users can also create their own stored functions, which are referred to as User-defined functions (UDFs).

Syntax and creation of stored functions:

```
CREATE FUNCTION function_name (
       [parameter_name_1 data_type]
[, parameter_name_2 data_type]...
 )
RETURNS data_type -- Specifies the data type of the value the function will return
-- Indicates if the function always produces the same result for the same input parameters
[NOT] DETERMINISTIC
-- Specifies the nature of SQL usage within the function
{CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA}
sql_block -- This represents the body of the function
```

Example of creating/calling/dropping a UDF function:
The example provided is from “Chapter 15: How to create stored procedures and functions. Murach’s MySQL (3rd ed.)” By Murrah (2019). In this example, the function returns the vendor ID that matches a vendor’s name.

```
-- Change the standard delimiter from semicolon (;) to double slash (//).
```

```
-- This is necessary because semicolons are used within the function body,
-- and we need a way to tell MySQL where the entire CREATE FUNCTION statement ends.
DELIMITER //

-- Create a new stored function named 'get_vendor_id'.
CREATE FUNCTION get_vendor_id (
    vendor_name_param VARCHAR(50)
)
-- return an INT (integer) value.
RETURNS INT
-- DETERMINISTIC: Indicates that this function will always return the same result
-- for the same input 'vendor_name_param', if data in the 'vendors' table doesn't change.
DETERMINISTIC
-- Indicates that the function contains SQL statements that read data
READS SQL DATA
-- start of the function body
BEGIN
    DECLARE vendor_id_var INT;

    SELECT vendor_id
    INTO vendor_id_var
    FROM vendors
    WHERE vendor_name = vendor_name_param;

    -- Return the value stored in 'vendor_id_var'.
    RETURN(vendor_id_var);
-- end of the function's body
END //
-- Reset the delimiter back to the standard semicolon (;).
DELIMITER ;
```

Invoking Function:
This was done in the same way MySQL functions are called within MySQL statements, often as part of an expression, for example, in a WHERE clause like:

```
SELECT invoice_number, invoice_total
FROM invoices
WHERE vendor_id = get_vendor_id('IBM'); -- Invoke/call/uses the function
```

Dropping the procedure

```
DROP FUNCTION IF EXISTS get_vendor_id;
```

**Stored Procedures vs. Functions**

The two stored routines are similar, but they differ in key aspects such as how they are called, what they can return, and how they interact with data. The table below provides a side-by-side comparison of their main differences.

**Table 1**
*Stored Procedures and Functions Differences*

| | Stored Procedure | Function |
|---|---|---|
| **Invocation** | Can be called from an application that has access to the database using the `CALL` statement. | Can be called from within a SQL statement, like MySQL's built-in functions. |

| | | |
|---|---|---|
| **Return Values** | Do not return a direct value like functions do. However, they can use `OUT` or `INOUT` parameters to pass values to the calling program. They can also pass result sets to the calling program using `SELECT` statements. | Are designed to always return a single value (a single piece of data, not a result set). The data type of the return value is specified and needs to be defined using the `RETURNS` keyword. |
| **Data Modification** | Can be used to modify data within a database, for example, by executing `INSERT`, `UPDATE`, or `DELETE` statements. They can manage transactions, including COMMIT and ROLLBACK operations. | It is technically possible to use functions to modify data when marked with the `MODIFIES SQL DATA` characteristic; however, functions are more commonly used to store procedures for data modification. Functions are generally used to read data or perform calculations, not to modify data. |
| **Parameters** | Can utilize `IN` (input), `OUT` (output), and INOUT (input/output) parameters. | Primarily use input parameters. Output parameters are technically possible, but they rarely make sense for the typical use case of a function. |
| **Creation Syntax** | Created using the `CREATE PROCEDURE` statement. | Created using the `CREATE FUNCTION` statement, which includes a `RETURNS` clause to define the data type of the output and can include characteristics like `DETERMINISTIC` or `READS SQL DATA`. |

*Note:* The table provides a side-by-side comparison of the main differences between stored procedures and functions in MySQL.

Another aspect to consider is when to use one instead of the other; the choice depends heavily on the specific task's purpose. The table below provides a side-by-side comparison of the common scenarios or use cases for each routine.

**Table 2**

*When to Choose a Stored Procedure vs. a Stored UDF Function*

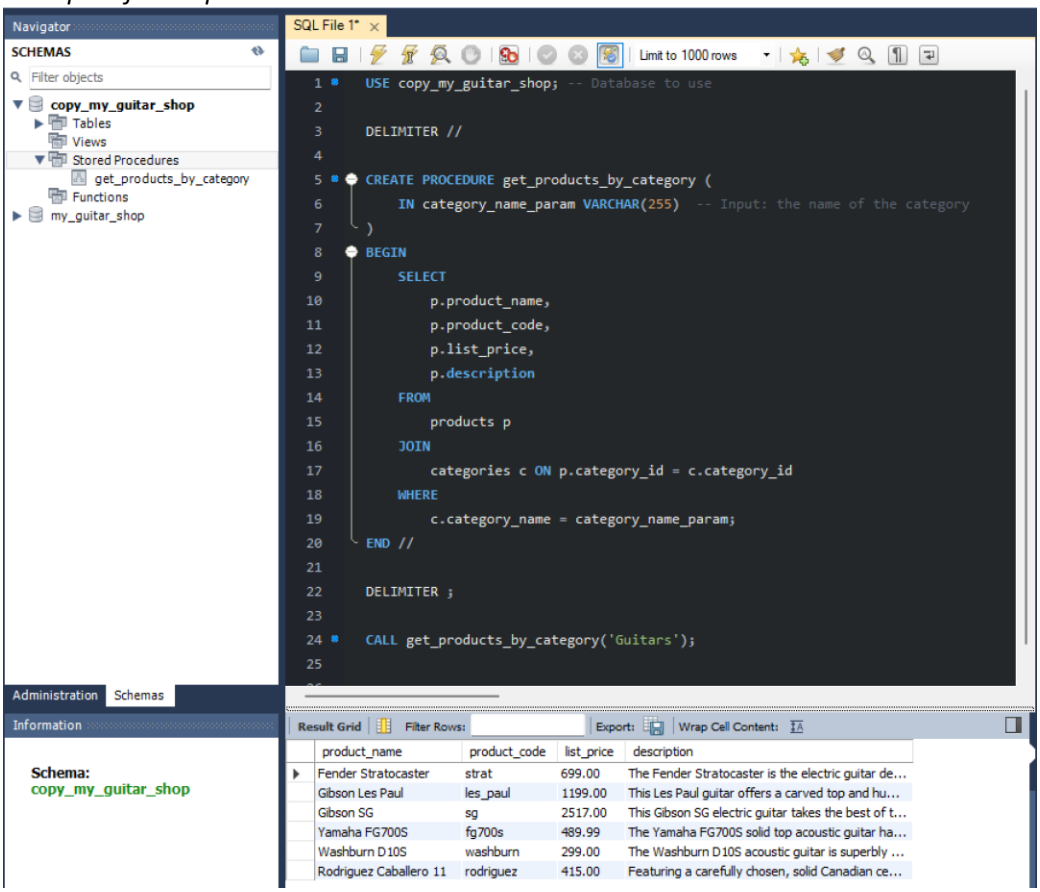
| Choose Stored Procedure When: | Choose Stored Function When: |
|---|---|
| The task involves more than just returning a single value. | The task involves specific, well-defined computations or transformations that need to return a single value. |
| The task requires multiple SQL statements, conditional logic (`IF`, `CASE`), loops, and several sequences of execution. When orchestrating multi-step logical transactions or business processes is needed | The task needs to perform calculations within multiple queries, and when these calculations need to be reused. |
| The task's primary goal is to modify data, for example, when using statements such as `INSERT`, `UPDATE`, or `DELETE` data in one or more tables. | The task consists of custom date formatting, string manipulation. |
| The task consists of atomic operations (all changes succeed or all fail together), it requires `START TRANSACTION`, `COMMIT`, and `ROLLBACK` statements. | The task uses conditional logic that returns a boolean or comparable value. |

| | |
|---|---|
| The task needs to return several distinct pieces of data (using `OUT` or `INOUT` parameters) or one or more entire tables of data. | The task results will be used as part of a larger SQL statement, for example, within `SELECT` list, `WHERE` clause, `ORDER BY`, `INSERT/UPDATE`. |
| The task consists of DDL and DML statements for database maintenance tasks (archiving data, rebuilding indexes). | The task consists of read-only and is guaranteed not to modify any database data. |

*Note:* The table provides a side-by-side comparison of common scenarios or use cases for stored procedure vs stored UDF function.

Below is an example of creating and using a simple procedure that gets all products belonging to a specific category, from the copy_my_guitar_shop schema (database).
In this example, using a stored procedure rather than a stored function is better because the task involves more than just returning a single value; it returns to an application, for example, an entire result set (a table of data) consisting of multiple rows and columns. On the other hand, stored functions cannot return data tables, they can only return a single scalar value, making them unsuitable for this kind of task.

**Figure 1**
*Example of a Simple Stored Procedure*



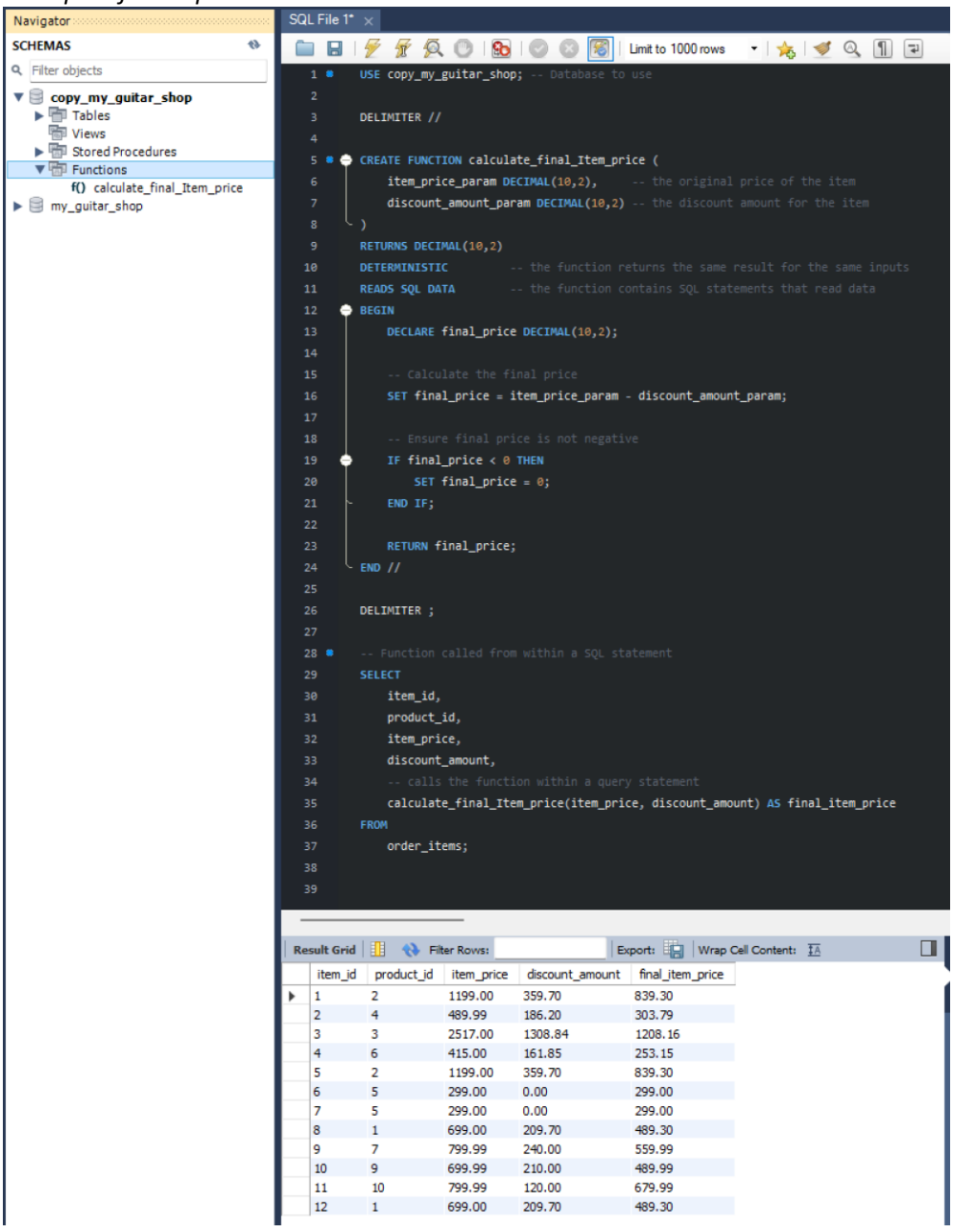

*Note:* The figure illustrates the MySQL code to create and use a simple stored Procedure that gets product information based on the given category.

Below is an example of creating and using a simple UDF function that calculates the final price of an order item after applying its discount, from the copy_my_guitar_shop schema (database).

In this example, using a stored function rather than a stored procedure is better suited as the task performs a computation (calculating the final price) that returns a single value that can be used by the SELECT query, see code below. On the other hand, a stored procedure would be overkill for this task, as its underlying design introduces unnecessary overhead for this particular computation when compared to a stored function.

**Figure 2**

*Example of a Simple UDF Function*




*Note:* The figure illustrates the MySQL code to create and use a simple stored UDF function that calculates the final price of an order item after applying its discount based on the given category.

Considering the routine differences and complexities as shown in Tables 1 and 2, and in Figures 1 and 2. New database administrators may find stored UDF functions easier to develop. Because they return a single value and have limited data modification capabilities, their development can be more straightforward/simpler, and they are safer to use than stored procedures. In other words, their limited scope and syntax can feel familiar to developers with programming experience, and their limited scope and data modification capacity make them safer to use.
Stored procedures, on the other hand, can quickly become extremely complex as they have a wider scope than stored functions. They can incorporate control flow, error handling, transaction management, and data modifications. Consequently, for a new database administrator, stored procedures may be more challenging and have a steeper learning curve compared to stored UDF functions.
As a new database administrator is best to start with functions and then gradually progress from procedures performing basic operations to procedures with more complex logic.

To summarize, stored procedures can be defined as a set of SQL statements that can be invoked using the CALL keyword. They are database objects containing procedural SQL code that can be invoked at any time by an application or user to fetch and/or manipulate data. On the other hand, a stored function is a block of SQL code that can perform computations or transformations and return a single scalar value MySQL provides its own set of stored functions, also called built-in functions, such as CONCAT(), NOW(), or SUM(). Users can create their own stored functions, called User-defined functions (UDFs). Although both stored procedures and user-defined functions provide code modularity, improved performance, and encapsulate business logic, their use cases differ. Stored procedures are used for complex data manipulation, transaction control, or when operations need to return multiple values or result sets; on the other hand, stored functions are used for calculations and transformations that return a single scalar value. Therefore, selecting the right routine for a specific task is critical for the performance and functionality of relational databases.

-Alex

**References:**

200ok Solutions (2024, September 20). MySQL stored procedures vs. functions: When and how to use them. 200ok Solutions. https://200oksolutions.com/blog/mysql-stored-procedures-vs-functions/

Murach, J. (2019). Chapter 15: How to create stored procedures and functions. *Murach's MySQL (3rd ed.)*. Murach Books. ISBN: 9781943872367

Musgrave, Z. (2024, January 17). *MySQL stored procedures: How and why with examples*. DoltHub Blog. https://www.dolthub.com/blog/2024-01-17-writing-mysql-procedures/

MySQL (n.d.). 27.2 Using stored routines. *MySQL 8.4 reference manual.* https://dev.mysql.com/doc/refman/8.4/en/stored-routines.html

W$^3$ School (n.d.). MySQL functions. *SQL reference*. W$^3$ School. https://www.w3schools.com/SQL/sql_ref_mysql.asp