LESSON 7

Stored Procedures in MySQL

Ihor Liutak

Content

1. Understand the concept and purpose of stored procedures.
2. Learn how to create, execute, and manage stored procedures in MySQL.
3. Practice building simple and advanced stored procedures.

What is a Stored Procedure?

Definition:

A stored procedure is a set of SQL statements stored in the database that can be executed as a single unit.

Purpose:

Encapsulate reusable logic.

Improve performance by reducing network traffic.

Enhance security by abstracting database operations.

Example:

A stored procedure to retrieve all users:

```
CREATE PROCEDURE GetAllUsers()
BEGIN
    SELECT * FROM users;
END;
```

Creating a Simple Stored Procedure

Steps:

Use `CREATE PROCEDURE` to define the procedure.

Write SQL logic inside the `BEGIN` and `END` block.

Use parameters for dynamic inputs if needed.

Execution:

```
CALL GetFruitsBySeason('Summer');
```

Example:

```
DROP PROCEDURE IF EXISTS GetFruitsBySeason;

CREATE PROCEDURE GetFruitsBySeason(IN season VARCHAR(255))
BEGIN
    SELECT *
    FROM fruits
    WHERE seasonality = season COLLATE utf8mb4_unicode_ci;
END;
```

Verify Session Collation

Corrected collation in the database heidenheim.site/datadbai.sql

Ensure the session's collation is compatible with the table's collation

```
SHOW VARIABLES LIKE 'collation_connection';
```

If necessary, set the session collation to `utf8mb4_unicode_ci`

```
SET collation_connection = 'utf8mb4_unicode_ci';
```

Managing Stored Procedures

Viewing Procedures:

```
SHOW PROCEDURE STATUS WHERE Db = 'database_name';
```

Describing a Procedure:

```
SHOW CREATE PROCEDURE procedure_name;
```

Deleting a Procedure:

```
DROP PROCEDURE IF EXISTS procedure_name;
```

Example:

```
DROP PROCEDURE IF EXISTS GetFruitsBySeason;
```

Advanced Stored Procedure with Parameters

Purpose: Pass multiple parameters for more flexible operations:

```
CREATE PROCEDURE UpdateFruitDescription(
    IN fruit_id INT,
    IN new_description TEXT
)
BEGIN
    UPDATE fruits
    SET description = new_description
    WHERE id = fruit_id;
END;
```

Execution

```
CALL UpdateFruitDescription(1, 'A tropical fruit with a unique flavor.');
```

Control Structures in Stored Procedures

Purpose: Add logic using loops and conditions.

Example: A procedure with conditional logic:

```
CREATE PROCEDURE CheckFruitSeason(IN fruit_id INT, OUT result VARCHAR(50))
BEGIN
    DECLARE season VARCHAR(20);
    SELECT seasonality INTO season FROM fruits WHERE id = fruit_id;

    IF season = 'Summer' THEN
        SET result = 'This fruit is available in Summer.';
    ELSE
        SET result = 'This fruit is not available in Summer.';
    END IF;
END;
```

Control Structures in Stored Procedures 2

Execution:

```
CALL CheckFruitSeason(1, @output);
SELECT @output;
```

This line calls the `CheckFruitSeason` procedure, passing `1` as the `fruit_id` and storing the output message in the variable `@output`

`SELECT @output;`

This line retrieves the value stored in `@output` and displays it.

When you execute these lines, you should see an output message that indicates whether the fruit with `id` 1 is available in Summer or not.

Control Structures in Stored Procedures 3

Execution:

In summary, the procedure CheckFruitSeason takes a fruit's ID as input, checks its seasonality, and returns a message indicating whether the fruit is available in Summer or not.

Procedure Declaration:

This declares a stored procedure called `CheckFruitSeason`. It takes two parameters:

`IN fruit_id INT`: An input parameter `fruit_id` of type integer.

`OUT result VARCHAR(50)`: An output parameter `result` of type VARCHAR with a maximum length of 50 characters.

Control Structures in Stored Procedures 4

Declare Variable

DECLARE season VARCHAR(20);

A local variable `season` of type `VARCHAR` with a maximum length of 20 characters is declared. This variable will hold the seasonality information of the fruit.

Select Statement

SELECT seasonality INTO season FROM fruits WHERE id = fruit_id;

This selects the `seasonality` from the `fruits` table where the `id` matches the provided `fruit_id` and stores the result into the `season` variable.

Control Structures in Stored Procedures 5

If-Else Statement

```
IF season = 'Summer' THEN
    SET result = 'This fruit is available in Summer.';
ELSE
    SET result = 'This fruit is not available in Summer.';
END IF;
```

This conditional block checks the value of the `season` variable. If `season` is 'Summer', it sets the `result` to 'This fruit is available in Summer.'. Otherwise, it sets the `result` to 'This fruit is not available in Summer.'

Lab Setup

Ensure you have access to MySQL and a database.

Use the `dw.loc` database for practice.

Tables used in the lab:

- `fruits`
- `users`

Lab Task 1: Create a Simple Procedure

Objective: Write a stored procedure to fetch fruits based on their storage place.

Instructions:

Create the procedure GetFruitsByStoragePlace(IN storage ENUM).

Use a SELECT statement to filter fruits.

```
CREATE PROCEDURE GetFruitsByStoragePlace(IN storage ENUM('WoodBox', 'Fridge'))
BEGIN
    SELECT * FROM fruits WHERE storage_place = storage;
END;
```

Lab Task 2: Update and Delete Operations

Objective: Create procedures to update and delete fruits.

Instructions:

Write a procedure `UpdateFruitSeason` to modify seasonality.

Write a procedure `DeleteFruit` to remove a fruit by ID.

```
CREATE PROCEDURE UpdateFruitSeason(IN fruit_id INT, IN new_season ENUM('Summer', 'Winter'))
BEGIN
    UPDATE fruits SET seasonality = new_season WHERE id = fruit_id;
END;

CREATE PROCEDURE DeleteFruit(IN fruit_id INT)
BEGIN
    DELETE FROM fruits WHERE id = fruit_id;
END;
```

Lab Task 3: Advanced Procedure with Loops

Objective: Create a procedure to update multiple fruits.

Instructions:

Use a loop to iterate through a list of fruit IDs.

Update their storage place.

'[1,2,3,4]'

```
CREATE PROCEDURE UpdateMultipleFruits(IN ids TEXT, IN new_storage ENUM('Fridge', 'WoodBox'))
BEGIN
    DECLARE id INT;
    DECLARE cur CURSOR FOR SELECT id FROM JSON_TABLE(ids, "$[*]" COLUMNS(id INT PATH "$"));
                                                                                    as jt;
    OPEN cur;

    fetch_loop: LOOP
        FETCH cur INTO id;
        IF id IS NULL THEN
            LEAVE fetch_loop;
        END IF;

        UPDATE fruits SET storage_place = new_storage WHERE id = id;
    END LOOP;

    CLOSE cur;
END;
```

Lab Task 3: Explanation 1

The procedure `UpdateMultipleFruits`:

Accepts Two Parameters:

- `ids`: A JSON string containing an array of fruit IDs.
- `new_storage`: The new storage place for the fruits (e.g., 'Fridge', 'WoodBox').

Uses a Cursor:

- A cursor is used to iterate over each fruit ID in the JSON array.
- For each ID, the storage place is updated in the database.

Loops Through IDs:

- The loop fetches each ID from the cursor and performs an `UPDATE` operation.
- The loop exits when there are no more IDs to process.

Lab Task 3: Explanation - Key SQL Concepts

JSON Data Handling:

The `JSON_TABLE` function parses the `ids` JSON string into a table structure.

This allows the procedure to iterate over the IDs.

Example Input: `"[1, 2, 3, 4]"`

Cursors:

A cursor is a database object used to fetch one row at a time from a query result set.

In this procedure:

`DECLARE cur CURSOR FOR SELECT id FROM JSON_TABLE(...);` initializes the cursor.

`FETCH cur INTO id;` retrieves the next ID.

`CLOSE cur;` ensures the cursor is closed after processing.

Lab Task 3: Explanation - Key SQL Concepts - Loop

Looping:

The `LOOP` statement processes each ID from the cursor.

The `LEAVE` statement exits the loop when no more IDs are fetched.

Example Usage:

```
CALL UpdateMultipleFruits('[1, 2, 3]', 'Fridge');
```

Updates the `storage_place` of fruits with IDs 1, 2, and 3 to `Fridge`

Performance:

Using `JSON_TABLE` and cursors might not be the most efficient method for large datasets. However, it's suitable for small lists and educational purposes

Procedure Without Cursors

```
DELIMITER //
CREATE PROCEDURE UpdateMultipleFruits(
        IN ids TEXT,
        IN new_storage ENUM('Fridge', 'WoodBox')
)
BEGIN
    UPDATE fruits
    SET storage_place = new_storage
    WHERE id IN (
        SELECT id
        FROM JSON_TABLE(ids, "$[*]" COLUMNS(id INT PATH "$")) AS jt
    );
END;
//
DELIMITER ;
```

```
CALL UpdateMultipleFruits('[1,2,3,4]', 'WoodBox');
```

How Write and Call Stored Procedures Using MySQL Console

1 Open MySQL Console

mysql -u root -p

Enter your MySQL password when prompted

2 Select the Database

USE your_database_name;

3 Create a Stored Procedure

Write the `CREATE PROCEDURE` statement in the MySQL console.

Use the `DELIMITER` command to change the statement delimiter from `;` to something else (e.g., `//` or `$$`) to avoid conflicts inside the procedure body.

```
DELIMITER //

CREATE PROCEDURE GetAllFruits()
BEGIN
    SELECT * FROM fruits;
END;
//

DELIMITER ;
```

4 Call the Stored Procedure

CALL GetAllFruits();

Summary

Stored procedures encapsulate logic for database operations.

Parameters enhance flexibility.

Control structures and loops enable advanced functionality.

Always test and manage procedures for performance.