**Stored Procedures**

A stored procedure in MySQL is a precompiled collection of one or more SQL statements and procedural logic that can be stored in the database and executed as a single unit.

It is typically used to encapsulate a set of SQL statements and business logic, providing several benefits:

1. **Modularity**: Stored procedures allow you to modularize your SQL code. You can write a series of SQL statements and procedural logic once, store it in the database, and then call it multiple times from different parts of your application.
2. **Reusability**: Since stored procedures can be called multiple times, they promote code reuse. This reduces code duplication and makes it easier to maintain and update database logic.
3. **Security**: Stored procedures can control access to data by defining who has permission to execute them. This can help enhance security by limiting direct access to tables and views.
4. **Performance**: Stored procedures can be optimized and cached by the database engine, potentially improving query performance. They can also reduce the amount of data transferred between the database server and the client.
5. **Abstraction**: By encapsulating complex SQL logic in stored procedures, application developers can work with a higher-level interface, making it easier to work with the database.

Here's a basic example of a stored procedure in MySQL:

DELIMITER //

CREATE PROCEDURE GetEmployeeCount()

BEGIN

DECLARE employee\_count INT;

SELECT COUNT(\*) INTO employee\_count FROM Employees;

SELECT employee\_count;

END //

DELIMITER ;

In this example, the **GetEmployeeCount** stored procedure calculates the number of records in the **Employees** table and returns the count.

You can call this stored procedure from your MySQL client or application code to retrieve the result.

**call GetEmployeeCount();**

Here's an example of a simple stored procedure that retrieves all customers from the Customers table:

DELIMITER //

CREATE PROCEDURE GetAllCustomers()

BEGIN

SELECT \* FROM Customers;

END //

DELIMITER ;

You can call the stored procedure like a regular SQL statement. For example:

**CALL GetAllCustomers();**

If you want to remove a stored procedure, you can use the **DROP PROCEDURE** statement. For example:

**DROP PROCEDURE IF EXISTS GetAllCustomers;**

Here's a simple stored procedure using the Northwind sample database that employs an **IF** statement. In this example, we'll create a procedure to retrieve products with a specified unit price range:

DELIMITER //

CREATE PROCEDURE GetProductsByPriceRange(

IN min\_price DECIMAL(10, 2),

IN max\_price DECIMAL(10, 2)

)

BEGIN

IF min\_price < 0 OR max\_price < 0 THEN

SELECT 'Invalid input. Prices cannot be negative.';

ELSE

SELECT ProductName, UnitPrice

FROM Products

WHERE UnitPrice BETWEEN min\_price AND max\_price;

END IF;

END //

DELIMITER ;

In this stored procedure:

* It takes two input parameters, **min\_price** and **max\_price**, representing the minimum and maximum unit prices for product retrieval.
* It uses an **IF** statement to check if either of the input prices is negative. If either is negative, it returns a message stating that prices cannot be negative.
* If both input prices are non-negative, it retrieves product names and unit prices from the **Products** table where the unit price falls within the specified range.

You can call this stored procedure to retrieve products within a price range. For example:

**CALL GetProductsByPriceRange(10.00, 50.00);**

This would retrieve products with unit prices between $10.00 and $50.00 from the Northwind sample database. Adjust the price range as needed based on your requirements

**CALL GetProductsByPriceRange(10.00, 50.00);**

This would give the error message

Here's another example of a simple stored procedure using the Northwind sample database. In this example, we'll create a procedure to check if a product with a given **ProductID** exists in the **Products** table:

DELIMITER //

CREATE PROCEDURE CheckProductExists(

IN product\_id INT,

OUT product\_exists BOOLEAN

)

BEGIN

DECLARE product\_count INT;

-- Count the number of products with the given ProductID

SELECT COUNT(\*) INTO product\_count

FROM Products

WHERE ProductID = product\_id;

-- Set the product\_exists variable based on the count

IF product\_count > 0 THEN

SET product\_exists = TRUE;

ELSE

SET product\_exists = FALSE;

END IF;

END //

DELIMITER ;

In this stored procedure:

* It takes an input parameter **product\_id** representing the ProductID to be checked for existence.
* It includes an output parameter **product\_exists** of type BOOLEAN to indicate whether the product exists or not.
* It uses a **SELECT COUNT(\*)** query to count the number of products in the **Products** table with the given **ProductID**.
* It then sets the **product\_exists** variable to **TRUE** if there is at least one matching product and **FALSE** otherwise.

You can call this stored procedure to check if a product exists by providing the **product\_id** and receiving the result in the **product\_exists** output parameter:

CALL CheckProductExists(5, @productExists);

SELECT @productExists;