Introduction to Stored Programs in MySQL

Stored programs in MySQL refer to blocks of SQL code that are saved and executed on the database server. They can consist of multiple SQL statements and include variables, control structures, and error handling. Stored programs allow for more complex operations and greater control over data.

Types of Stored Programs:

1. **Stored Procedures**: A sequence of SQL statements that are stored in the database and executed as a single block of code.

2. **Stored Functions**: Similar to stored procedures but return a single value.

Stored programs can include control structures (e.g., `IF-ELSE`, `CASE`), variables, and loops, which makes them a powerful tool for automating database tasks and improving performance.

3**. Triggers** : is a set of SQL statements that are executed automatically when a specified change operation (SQL INSERT, UPDATE, or DELETE statement) is performed on a specified table. Triggers are useful for tasks such as enforcing business rules, validating input data, and keeping an audit trail.

4**. Cursor**: in MySQL is a database object that allows you to process data returned by an SQL query, one row at a time. It provides you with a way to iterate over the result set and perform operations on each row individually.

Advantages of Stored Programs

1. Reusability: Once a stored program is created, it can be used repeatedly across different applications.

2. Performance: Since stored programs are executed on the database server, they reduce the need for frequent client-server communication.

3. Security: Stored programs encapsulate business logic, making it easier to manage permissions and access control.

4. Maintainability: Changes in logic can be done at the database level without affecting the application code.

5. Transaction Control: Stored programs can manage transactions, allowing for greater consistency in data operations.

Variables in MySQL

Variables in MySQL stored programs are used to store temporary data that can be manipulated during the execution of the program.

- Local Variables: Declared within the stored program and accessible only inside it.

- User-defined Variables: Declared by the user, persist through the session, and can be accessed outside the program.

```
SET @num = 5;
SELECT @num * @num AS Square;


SELECT IF(5 > 3, 'Yes', 'No') AS Result;


SELECT name, age,
  CASE
    WHEN age < 18 THEN 'Minor'
    WHEN age BETWEEN 18 AND 65 THEN 'Adult'
    ELSE 'Senior'
  END AS AgeCategory
FROM student;
```

Stored Procedure

- What is stored Procedure ?

A stored procedure in MySQL is a set of Pre compiled SQL statements that can be stored in the database.

It is used so that no need to write same query again and again.

SELECT * from table Students

Syntax :

```
CREATE PROCEDURE procedure_name ()
BEGIN
    -- SQL statements
END;
CALL procedure_name();
```

- Example :

```
DELIMITER //

CREATE PROCEDURE find_students()
BEGIN
    SELECT * FROM Students;
END //

DELIMITER ;
```

How to call procedure :

```
CALL procedure_name();
```

- **Advantages of Using Stored Procedures:**

1. **Modularity**: Code can be reused across different parts of the application.
2. **Performance**: Stored procedures are precompiled, which can lead to faster execution times.
3. **Security**: Permissions can be granted to execute stored procedures without giving direct access to the underlying tables.
4. **Maintainability**: Logic is centralized in one place, making it easier to manage and update.

Procedure can have parameters :

IN: Input parameter (the default type).

•**OUT**: Output parameter, used to return a value.

•**INOUT**: Can be used for both input and output.

Example using IN parameter

```
CREATE PROCEDURE ‘get_students_info’ (IN age int);
BEGIN
    SELECT * from table Students where students.age = age
END;
```

```
CALL get_students_info(31);
```

Example using OUT parameter

```
CREATE PROCEDURE ‘get_students_info’ (OUT records int);
BEGIN
    SELECT  count(*) into records from table Students where students.age = 32
END;
```

```
CALL get_students_info(@records);
Select @records as total_students;
```

Using IN-OUT

```
CREATE PROCEDURE ‘get_students_info’ (IN age int , OUT records int);
BEGIN
 SELECT  count(*) into records from  Students where students.age = age
END;
```

```
CALL get_studenst_info(31, @records);
SELECT @records as total_students;
```

```
CREATE PROCEDURE GetEmployee(IN emp_id INT, OUT emp_name VARCHAR(100))
BEGIN
    SELECT name INTO emp_name FROM employees WHERE id = emp_id;
END;

CALL GetEmployee(1, @employeeName);
SELECT @employeeName;
```

Dropping a Stored Procedure

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

Question

- Write a procedure that returns detailed order information for a given customer.

Fetch order_id , order_date, product_name, order_quantity from customer_order table.

```
CREATE PROCEDURE GetCustomerOrders(IN cust_id INT)
BEGIN
    SELECT o.order_id, o.order_date, p.product_name, od.quantity
    FROM orders o
   WHERE o.customer_id = cust_id;
END;

CALL GetCustomerOrders(5);
```

Practical

```
CREATE DATABASE school;


USE school;
CREATE TABLE Faculty (
    name VARCHAR(255),
    Faculty_id INT,
    address VARCHAR(255),
    age INT,
    salary INT
);
```

```
INSERT INTO Faculty (name, Faculty_id, address, age, salary)
VALUES
('John Doe', 1, '123 Elm Street', 16, 4500),
('Jane Smith', 2, '456 Oak Avenue', 17, 4700),
('Sam Brown', 3, '789 Pine Road', 15, 4300),
('Emily Davis', 4, '101 Maple Street', 16, 4900),
('Michael Johnson', 5, '202 Birch Lane', 17, 4600);
```

- **INT** – A normal-sized integer that can be signed or unsigned. If unsigned, the allowable range is from 0 to 4294967295. You can specify a width of up to 11 digits.
- **TINYINT** – A very small integer that can be signed or unsigned. If signed, the allowable range is from -128 to 127. If unsigned, the allowable range is from 0 to 255. You can specify a width of up to 4 digits.
- **SMALLINT** – A small integer that can be signed or unsigned. If unsigned, the allowable range is from 0 to 65535. You can specify a width of up to 5 digits.
- **MEDIUMINT** – A medium-sized integer that can be signed or unsigned. If unsigned, the allowable range is from 0 to 16777215. You can specify a width of up to 9 digits.
- **BIGINT** – A large integer that can be signed or unsigned. If unsigned, the allowable range is from 0 to 18446744073709551615. You can specify a width of up to 20 digits.

- **FLOAT(M,D)** – You can define the display length (M) and the number of decimals (D). This is not required and will default to 10,2, where 2 is the number of decimals and 10 is the total number of digits (including decimals).
-
- **DOUBLE(M,D)** – You can define the display length (M) and the number of decimals (D). This is not required and will default to 16,4, where 4 is the number of decimals. Decimal precision can go to 53 places for a DOUBLE. REAL is a synonym for DOUBLE.

- **DATE** – A date in YYYY-MM-DD format, For example, December 30th, 1973 would be stored as 1973-12-30.

- **DATETIME** – A date and time combination in YYYY-MM-DD HH:MM:SS format, For example, 3:30 in the afternoon on December 30th, 1973 would be stored as 1973-12-30 15:30:00.

- **TIMESTAMP** – A timestamp between midnight, January 1st, 1970 and sometime in 2037. This looks like the previous DATETIME format, only without the hyphens between numbers; 3:30 in the afternoon on December 30th, 1973 would be stored as 19731230153000 (YYYYMMDDHHMMSS).

- **TIME** – Stores the time in a HH:MM:SS format.

- **YEAR(M)** – Stores a year in a 2-digit or a 4-digit format. If the length is specified as 2 (for example YEAR(2)), YEAR can be between 1970 to 2069 (70 to 69). If the length is specified as 4, then YEAR can be 1901 to 2155. The default length is 4.

- **CHAR(M)** – A fixed-length string between 1 and 255 characters in length (for example CHAR(5)), right-padded with spaces to the specified length when stored. Defining a length is not required, but the default is 1.
- **VARCHAR(M)** – A variable-length string between 1 and 255 characters in length. For example, VARCHAR(25). You must define a length when creating a VARCHAR field.

- How to declare variable in Mysql :

Declare [variable_name] datatype(size) [default value];

Set variableName= value;

Example :

How to list procedure

```
SHOW PROCEDURE STATUS WHERE db = 'mystudentdb';
```

IF-ELSE STATEMENT

```
IF expression THEN
    statements;
ELSE
   else-statements;
 END IF;
```

The statements must end with a semicolon.

```
+------+---------+------------+------------+ |
 id   | Name    | Address    | Subject    |
+------+---------+------------+------------+
| 101  | YashPal | Amritsar   | History    |
| 105  | Gaurav  | Jaipur     | Literature |
| 125  | Raman   | Shimla     | Computers  |
 +------+---------+------------+------------+
```

```
CREATE PROCEDURE CheckAge(IN Age INT, OUT AgeCategory VARCHAR(50))
BEGIN
    DECLARE AgeStatus VARCHAR(50);

    -- Check if the age is 18 or more
    IF Age >= 18 THEN
        SET AgeStatus = 'You are an adult.';
    ELSE
        SET AgeStatus = 'You are a minor.';
    END IF;

    -- Set the output parameter
    SET AgeCategory = AgeStatus;
END //
```

```
CALL CheckAge(20, @Result);
SELECT @Result;
```

```
create procedure coursedetails(IN S_subject Varchar(30) , OUT S_course varchar(50))
    begin
        declare sub varchar(20);
         select subject INTO sub from student_info where S_subject = subject;
         IF sub = 'Computer' THEN
         SET S_course = 'Btech';
         ELSE
         SET S_course = 'Subject is not in CS';
         END IF;
       END //


 call coursedetails('Computers', @S_course) //


 Select @S_Course //
```

```
CREATE PROCEDURE coursedetails(IN S_subject VARCHAR(30), OUT S_course VARCHAR(50))
BEGIN
 IF S_subject = 'Computer' THEN
 SET S_course = 'Btech'; ELSE
SET S_course = 'Subject is not in CS’;
 END IF;
END //
```

Practice Questions

1. Check User Role and Permissions

•Write a stored procedure that takes a user ID as input and checks the user's role in the system.
•Depending on the role (admin, editor, or viewer), the procedure should return different sets of permissions.
•For example, admin has all permissions, editor can edit content, and viewer can only view content.

2. Calculate Discount Based on Customer Type

•Create a stored procedure that accepts a customer ID and the total purchase amount as input.
•The procedure should apply a discount based on the customer type (Regular, Premium, or VIP).
•For example, Regular customers get a 5 percent discount, Premium get 10 percent, and VIP get 15 percent.

3. Determine Pass/Fail Status for Students

•Write a stored procedure that takes a student ID and their total marks as input.
•The procedure should determine if the student has passed or failed based on a threshold (e.g., 40 percent).
•It should also return a custom message with the student's pass/fail status.

4. Dynamic Tax Calculation

•Create a stored procedure that calculates tax based on the type of item being purchased.
•The procedure should accept the item ID and purchase amount as input,
•then apply a different tax rate depending on whether the item is classified as Essential, Luxury, or Service.

5. Determine Order Priority Level

•Write a stored procedure that takes an order ID and checks the order's total value.

• The procedure should classify the order as Low, Medium, or High priority based on the total value.

•For example, orders below 100 dollars are Low, between 100 and 500 dollars are Medium, and above 500 dollars are High priority.

Solutions

1.

```
DELIMITER  //
CREATE PROCEDURE CheckUserRole(IN p_user_id INT, OUT p_permissions VARCHAR(255))
BEGIN
    DECLARE v_role VARCHAR(50);

    -- Assuming there is a table 'users' with columns 'id' and 'role'
    SELECT role INTO v_role FROM users WHERE id = p_user_id;

    IF v_role = 'admin' THEN
        SET p_permissions = 'ALL PERMISSIONS';
    ELSEIF v_role = 'editor' THEN
        SET p_permissions = 'EDIT, VIEW';
    ELSEIF v_role = 'viewer' THEN
        SET p_permissions = 'VIEW ONLY';
    ELSE
        SET p_permissions = 'NO PERMISSIONS';
    END IF;
END //

DELIMITER ;
```

2.

```
DELIMITER //

CREATE PROCEDURE CalculateDiscount(IN p_customer_id INT, IN p_total_amount DECIMAL(10,2), OUT p_final_amount DECIMAL(10,2))
BEGIN
    DECLARE v_customer_type VARCHAR(50);
    DECLARE v_discount_rate DECIMAL(5,2);

    -- Assuming there is a table 'customers' with columns 'id' and 'customer_type'
    SELECT customer_type INTO v_customer_type FROM customers WHERE id = p_customer_id;

    IF v_customer_type = 'Regular' THEN
        SET v_discount_rate = 0.05;
    ELSEIF v_customer_type = 'Premium' THEN
        SET v_discount_rate = 0.10;
    ELSEIF v_customer_type = 'VIP' THEN
        SET v_discount_rate = 0.15;
    ELSE
        SET v_discount_rate = 0;
    END IF;

    SET p_final_amount = p_total_amount - (p_total_amount * v_discount_rate);
END //
```

3.

```
DELIMITER $$

CREATE PROCEDURE CheckPassFail(IN p_student_id INT, IN p_total_marks DECIMAL(5,2), OUT p_status
VARCHAR(50))
BEGIN
    DECLARE v_threshold DECIMAL(5,2);

    -- Set pass threshold
    SET v_threshold = 40.00;

    IF p_total_marks >= v_threshold THEN
        SET p_status = 'Passed';
    ELSE
        SET p_status = 'Failed';
    END IF;
END$$

DELIMITER ;
```

4.

```
DELIMITER $$

CREATE PROCEDURE CalculateTax(IN p_item_id INT, IN p_purchase_amount DECIMAL(10,2), OUT p_tax_amount DECIMAL(10,2))
BEGIN
    DECLARE v_item_type VARCHAR(50);
    DECLARE v_tax_rate DECIMAL(5,2);

    -- Assuming there is a table 'items' with columns 'id' and 'item_type'
    SELECT item_type INTO v_item_type FROM items WHERE id = p_item_id;

    IF v_item_type = 'Essential' THEN
        SET v_tax_rate = 0.05;
    ELSEIF v_item_type = 'Luxury' THEN
        SET v_tax_rate = 0.20;
    ELSEIF v_item_type = 'Service' THEN
        SET v_tax_rate = 0.10;
    ELSE
        SET v_tax_rate = 0;
    END IF;

    SET p_tax_amount = p_purchase_amount * v_tax_rate;
END$$

DELIMITER ;
```

5.

```
DELIMITER $$

CREATE PROCEDURE CheckOrderPriority(IN p_order_id INT, OUT p_priority VARCHAR(50))
BEGIN
    DECLARE v_total_value DECIMAL(10,2);

    -- Assuming there is a table 'orders' with columns 'id' and 'total_value'
    SELECT total_value INTO v_total_value FROM orders WHERE id = p_order_id;

    IF v_total_value < 100 THEN
        SET p_priority = 'Low';
    ELSEIF v_total_value BETWEEN 100 AND 500 THEN
        SET p_priority = 'Medium';
    ELSE
        SET p_priority = 'High';
    END IF;
END$$

DELIMITER ;
```

LOOP

- **Types of Loops in MySQL Stored Procedures**

- MySQL supports the following loop constructs inside stored procedures:

- **LOOP**: A simple loop that executes until explicitly exited.
- **WHILE**: Repeats the loop as long as the given condition is true.
- **REPEAT UNTIL**: Repeats the loop until the given condition becomes true.

Type	Condition Checked	Execution Condition	Use Case
LOOP	Manual exit with LEAVE	Runs indefinitely unless exited	Useful when the exit condition is complex or requires more logic.
WHILE	Before each iteration	Continues while condition is true	Common when the number of iterations is not known beforehand.
REPEAT	After each iteration	Runs until the condition becomes true	Use when you need the block to execute at least once.

Q- Even number

```
CREATE PROCEDURE generate_even_numbers()
BEGIN
  DECLARE counter INT DEFAULT 2;

  WHILE counter <= 10 DO
    SELECT counter;
    SET counter = counter + 2;
  END WHILE;
END;
```

Calculating the Sum of Numbers from 1 to N

```
DELIMITER //

CREATE PROCEDURE sum_numbers_while(IN N INT, OUT total_sum INT)
BEGIN
    DECLARE counter INT DEFAULT 1;
    DECLARE sum INT DEFAULT 0;

    WHILE counter <= N DO
        SET sum = sum + counter;
        SET counter = counter + 1;
    END WHILE;

    SET total_sum = sum;
END //

DELIMITER ;
```

This procedure generates a multiplication table up to N x M

```
DELIMITER
CREATE PROCEDURE multiplication_table(IN N INT, IN M INT)
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE j INT DEFAULT 1;
    outer_loop: LOOP
        IF i > N THEN
            LEAVE outer_loop;
        END IF;

        SET j = 1;  -- Reset inner loop counter

        inner_loop: LOOP
            IF j > M THEN
                LEAVE inner_loop;
            END IF;
            -- Calculate and display the product
            SELECT CONCAT(i, ' x ', j, ' = ', i * j) AS Multiplication;

            SET j = j + 1;
        END LOOP inner_loop;

        SET i = i + 1;
    END LOOP outer_loop;
END //
DELIMITER ;
```