Module 13: MySQL Stored Routines – Procedure & Functions

Stored Routines:

- Stored Routines are group of SQL and PL/SQL statements which perform a specific task.
- MySQL engine parses and compiles the stored programs before they are actually called.
- With stored routines, we can write procedural code that controls the flow of execution which includes: if/else, loops and error handling code.

Benefits of Stored Routines:

- **More Flexible SQL Syntax:**
  - They provide extension to SQL syntax such as compound statements, selection statements and loop statements which make it easier to express complex logic.
- **Error Handling Capabilities:**
  - We can create error handlers which can be used when exceptional condition arises.
- **Less "Reinvention of Wheel":**
  - Collection of stored routines by skilled SQL developer can act as a library of solutions to problems faced by developers with less expertise.
- **Ease of Maintenance:**
  - Single copy of routine is easier to maintain than a copy embedded with each application.
- **Network Traffic:**
  - Stored routines are invoked by the calling program which pass only the name and parameters else it would had needed multiple sql statements.
- **Security:**
  - Permission can be granted to applications to access the stored programs without giving permission on underlying table.

Types of Stored Routines:

- MySQL supports four types of Stored Routines:
- **Stored Procedure:**
  - It can be called from an application that has access to the database.
- **Stored Function:**
  - It can be called from a SQL statement.
- **Trigger:**
  - It is executed in response to an INSERT, UPDATE or DELETE statement on a specified table.
- **Event:**
  - It is executed at a scheduled time.

Stored Procedure:

- Stored Procedure (also called sproc or procedure) is

- MySQL compiles the code for the procedure and stores the compiled code in the database.
- If there is any coding error, the system responds with an appropriate message and the procedure isn't created.
- The syntax of the CREATE PROCEDURE is:
    - **CREATE PROCEDURE procedure_name (**
    - **[par_nam1 data_type]**
    - **[, par_nam1 data_type]**
    - **...**
    - **)**
    - **sql_block**

Creating Procedure:

- CREATE PROCEDURE statement is used to create a stored procedure.
- After the name of the procedure we write a set of parenthesis.
- Within the parenthesis we can write 0 or more parameters for the procedure.
- We code block of statements between the BEGIN and END keywords.
- Within block of statements we write SQL statements.
- To display a message from stored procedure we use SELECT statement to return a result set.

DELIMITER statement:

- The DELIMITER statement changes the delimiter from the default ; to two $ ($$).
- This is necessary because the semicolon is used within CREATE PROCEDURE statement and hence $$ is to identify the end of CREATE PROCEDURE statement.

DROP Procedure:

- To drop a stored procedure from the database, we used DROP PROCEDURE statement.
    - The syntax is: **DROP PROCEDURE [IF EXISTS] procedure_name**
- DROP PROCEDURE keywords is followed by the name of the procedure.
- MySQL returns an error if procedure doesn't exists. In this case we can use IF EXISTS.

Structure of Stored Procedure:

- Each block commences with BEGIN statement and is terminated by END statement.
- Block consist of various types of declarations (eg: variables, cursors handlers) and program code.

Calling Procedure:

- We can execute or call a stored procedure by using the CALL statement.
- When we use CALL statement, we pass parameters by position.
- The order is same as the order as they were coded in the CREATE PROCEDURE statement.

Order:

- The order is which these appear are as follows:
  - 1) **V**ariables and **C**ondition declarations
  - 2) **C**ursor declarations
  - 3) **H**andler declarations
  - 4) **P**rogram code.

Listing and Displaying Stored Procedures:

- To display the stored procedures code:
  - **SHOW CREATE PROCEDURE stored_procedure_name**
- To list all stored procedures of the databases that you have access :
  - **SHOW PROCEDURE STATUS.**
- To list stored procedures in a particular database:
  - **SHOW PROCEDURE STATUS where db = 'db_name'**
- To show stored procedures that a particular pattern:
  - **SHOW PROCEDURE STATUS where name like 'pattern'**

Parameter Types: Parameter type can be:

- Input parameter, Output parameter and Input/Output parameter.
- The syntax is:
  - [**IN** | OUT | INOUT] parameter_name data_type

- **Input Parameters**
  - It accept values that are passed from the calling program.
  - These values cannot be changed by the body of the stored procedure.
  - By default parameters are defined as input parameters.
  - To identify an input parameter, we write IN keyword.
- **Output Parameters:**
  - These parameters store values that are passed back to the calling program.
  - These values must be set by the body of the stored procedure.
  - To identify an output parameter, we write OUT keyword.
- **Input/Output Parameters:**
  - They can store value that's passed from the calling program.
  - The body of the stored procedure can change this parameter.
  - To identify an input/output parameter, we write INOUT keyword.

Local Variable:

Declare Variables:

- ❑ A variable stores a value that can change as the procedure executes.
- ❑ The syntax is:
  - ⟶ ❑ **DECLARE**
- ❑ To declare a variable, we use DECLARE keyword followed by the variable name and data type.
- ❑ Each variable within a block must have a different name.
- ❑ We can use DEFAULT keyword to assign a default value to a variable when we declare it.
- ❑ The initial value is NULL for variables declared with no DEFAULT clause.

SELECT … INTO:

- ❑ The SELECT … INTO statement assigns the result of a SELECT statement into variables.
- ❑ The syntax is:
  - ❑ **SELECT col_name1, col_name2 INTO var_name1, var_name2;**
- ❑ To avoid name clashes it is best not to give a local variable the same name as any table columns that we refer within a routine.
- ❑ SELECT statement must select at most a single row. If it selects more than one row an error occurs.

SET variables:

- ❑ To assign a value to variable we use SET statement along with =.
- ❑ To assign a value to variable that's returned by a SELECT statement to a variable, we use INTO clause.
- ❑ The syntax is:
  - ❑ **SET variable_name = literal_value _or_expression;**

Stored Routines – MCQ-1

Q1) Which of the following are stored routines in MySQL? **Choose all that apply**

Options:

A. Stored Package B. Stored View
C. Triggers D. Stored Functions

Solution:

Q2) The body of stored routine is written between:

Options:

A. { } B. begin & end
C. start & end D. begin and stop

Solution:

Q3) How do u invoke stored procedure called sp1?

Options:

A. select sp1()	B. call procedure sp1()
C. sp1()	D. call sp1()

Solution:

Q4) What are the type of parameters in stored procedure? Choose all that apply.

Options:

A. IN	B. WRITE
C. INOUT	D. READ

Solution:

Q5) How do we declare INOUT parameters in the stored procedure?

Options:

A. info varchar(30)	B. info varchar(30) INOUT
C. info INOUT varchar(30)	D. INOUT info varchar(30)

Solution:

Q6) How do we assign default value to local variables?

Options:

A. declare res = 0;	B. declare res int = 0;
C. declare res int default 0;	D. default res int declare 0;

Solution:

Q7) How do we change the value of local variable res?

Options:

A. set res++;	B. set res = res + 1;
C. res = res + 1;	D. res++;

Solution:

Q8) What is the result of following code?

```
create procedure tp16()
BEGIN
declare i int default 2;
declare j int i + 2;
select i;
select j;
END $$
```

Options:

A. output: 4   B. it will not compile

C. ouput: 2null   D. ouput: 22

Solution:

Q9) consider the following code snippet:

```
declare i int;
declare j int;
set i = 10;
set j = 20;
declare k int;
set k = i + j;
select k;
```

What is the result?

Options:

A. code will not compile   B. 30

C. null   D. 1020

Solution:

Q10) consider the following code assuming procedure tp7 does not exists in db.

```
create procedure tp7()
begin end $$

create procedure tp7(a int)
begin end $$
```

What is the result?

Options:

A. Db contains two procedures called tp7() and tp7(a int).

B. Db contains one procedure called tp7().

C. DB contains one procedure called tp7(a int).

D. code will give compile time error

Solution:

Conditional Statement: The IF and CASE statements enable you to perform conditional testing.

IF Statement:

- IF Statement is used to execute one or more statements depending on one or more Boolean expressions. Boolean expression is an expression that evaluates to true or false.
- The syntax is:
  - **IF boolean_expression THEN**
    - **statement_1;**
    - **[statement_2;]…**
  - **[ELSEIF boolean_expression THEN**
    - **statement_1;**
    - **[statement_2;]…]**
  - **[ELSEIF boolean_expression THEN**
    - **statement_1;**
    - **[statement_2;]…]…**
  - **[ELSE**
    - **statement_1;**
    - **[statement_2;]…]**
  - **END IF;**

CASE Statement:

Simple case or Searched case statement can be used to execute one or more statements depending on a value that's returned by an expression.

- **Simple CASE: (used for = cases)**
  - In simple case the expression is evaluated and compared with expression_value_1.
    - if they are equal the statement list following THEN is executed.
    - It they are not equal and there are any following WHEN clauses, they are handled similarly in turn.
  - If no WHEN clause has a expression_value equal to expression and there is an ELSE clause, the ELSE clauses statement list is executed.
  - **Note:** if ELSE clause is missed then we get : ERROR 1339 (20000): Case not found for CASE statement
- The syntax is :
  - **CASE expression**
    - **WHEN expression_value_1 THEN**
      - **statement_1;**
      - **[statement_2;]…**
    - **[WHEN expression_value_2 THEN**
      - **statement_1;**
      - **[statement_2;]…]…**
    - **[ELSE**
      - **statement_1;**
      - **statement_2;]…]**
  - **END CASE;**

- ❑ **Searched CASE : (used for comparison values)**
  - ❑ The boolean_exrpression in each WHEN clause is executed until one is found to be true and then its statement list is executed.
  - ❑ If none are true and there is an ELSE clause, the ELSE clause's statement list is executed.
  - ❑ The syntax is:
    - ❑ **CASE**
      - ❑ **WHEN boolean_expression THEN**
        - ❑ **statement_1;**
        - ❑ **[statement_2;]…**
      - ❑ **[WHEN boolean_expression THEN**
        - ❑ **statement_1;**
        - ❑ **[statement_2;]…]…**
      - ❑ **[ELSE**
        - ❑ **statement_1;**
        - ❑ **statement_2;]…]**
    - ❑ **END CASE;**

Stored Routines – MCQs-2

Q1) Which of the following code snippet is valid assuming res is declared as local variable?
Options:

A.
if res % 2 = 0 then
    select 'even';
else
    select 'odd';
end if;
	B.
	if res % 2 = 0
	    select 'even';
	else
	    select 'odd';
	end if;

C.
if res % 2 == 0 then
    select 'even';
else
    select 'odd';
end if;
	D
	if res % 2 == 0
	    select 'even';
	else
	    select 'odd';
	end if;

Solution:

Q2) what is the output of following code:

```
declare i int;
if i then
        select 'kk';
else
        select 'jj';
end if;
```

Options:

A. jj   B. kk

C. code will not compile.   D. no output

Solution:

Q3) what is the output of following code:

```
declare res int default 2;
case res
        when 2 then select 'wow';
        when 2 then select 'now';
end case;
```

Options:

A. wow B. wow now

C. code will not compile. D. now

Solution:

Q4) what is the output of following code:

```
declare res int default 2;
case res
        when 3 then select 'wow';
        when 4 then select 'now';
end case;
```

Options:

A. now B. code will compile but will get error at runtime.

C. code will not compile. D. no output

Solution:

Q5) what is the output of following code:

```
declare res int default 2;
case
        when res < 2 then select 'wow';
        when res >=1 then select 'now';
end case;
```

Options:

A. now B. wow

C. code will not compile. D. no output

Solution: