My SQL
Stored Routines

Objectives

1. Stored routines definition
2. A simple procedure
3. A simple function
4. Procedure parameters
5. Random numbers
6. Finding routines

1. Stored routines definition

In MySQL there are two kinds of stored routines: **Stored procedures** and **Stored functions**.

- Stored procedures are called with the CALL statement. They do not return values. Stored functions return values. And are used with the SELECT statement.
- A stored routine is a set of SQL statements that can be stored in the server. Stored routines are not generally accepted. They have some advantages but also several disadvantages. Stored routines are typically used in data validation or access control.

2. A simple procedure

The procedure is created with a CREATE PROCEDURE statement.

```
CREATE PROCEDURE AllCars() SELECT * FROM Cars;
```

In this statement, we have created a new simple procedure called AllCars(). The select statement following the name of the procedure is the body of the procedure, which is executed when we call the procedure. The procedure selects all data from the Cars table.

```
mysql> CALL AllCars();
+----+------------+--------+
| Id | Name       | Cost   |
+----+------------+--------+
|  1 | Audi       |  52642 |
|  2 | Mercedes   |  57127 |
|  3 | Skoda      |   9000 |
|  4 | Volvo      |  29000 |
|  5 | Bentley    | 350000 |
|  6 | Citroen    |  21000 |
|  7 | Hummer     |  41400 |
|  8 | Volkswagen |  21600 |
+----+------------+--------+
```

We call the AllCars() procedure and its body is executed.

3. A simple function

A function is created with a CREATE FUNCTION statement. A function returns a value. It is called with a SELECT statement.

```
 -- this function computes the area
-- of a circle; it takes a radius as
-- a parameter
DELIMITER $$
DROP FUNCTION IF EXISTS CircleArea;
CREATE FUNCTION CircleArea(r DOUBLE) RETURNS DOUBLE
BEGIN
   DECLARE area DOUBLE;
   SET area = r * r * pi();
   RETURN area;
END
$$
DELIMITER ;
```

3. A simple function

- In this code, we create a CircleArea() function which computes the area of a circle. It takes a radius as a parameter. The best way to create a procedure or a function that has more than one line is to create an SQL file and read the file with the source command.

```
-- this function computes the area
-- of a circle; it takes a radius as
-- a parameter
```

Comments begin with double dashes.

```
DELIMITER $$
```

SQL statements are finished with a semicolon. To create a procedure or a function we need multiple statements. Because of this, we need to temporarily use a different delimiter. Here we have used $$ as a delimiter. We could use different characters. At the end of the function definition, we use this delimiter.

3. A simple function

We create a function called CircleArea. It takes a parameter r of type DOUBLE. The function returns a value of type DOUBLE.

```
CREATE FUNCTION CircleArea(r DOUBLE) RETURNS DOUBLE
```

The function body is placed between the BEGIN and END keywords.

```
BEGIN
...
END
```

We declare a new variable in the routine. Its name is area and data type is DOUBLE.

```
DECLARE area DOUBLE;
```

We compute the area of the circle with the given radius: `SET area = r * r * pi();`

We return the variable: `RETURN area;`

Here is the end of the routine.

```
$$
DELIMITER ;
```

3. A simple function

We use again the default delimiter.

```
mysql> source circlearea.sql
mysql> SELECT CircleArea(5.5);
+-------------------+
| CircleArea(5.5)   |
+-------------------+
| 95.03317777109125 |
+-------------------+
```

We create the CircleArea() function and call it with the SELECT statement.

4. Procedure parameters

We use again the default delimiter.

```
mysql> source circlearea.sql
mysql> SELECT CircleArea(5.5);
+-------------------+
| CircleArea(5.5)   |
+-------------------+
| 95.03317777109125 |
+-------------------+
```

We create the CircleArea() function and call it with the SELECT statement.

4. Procedure parameters

A procedure cannot return a value. However, it can work with three types of variables:

- IN
- OUT
- INOUT

The IN is the default type of parameter. It is used when no type is specified explicitly. The IN parameter is passed to the procedure. It can be modified inside the procedure, but it remains unchanged outside. In the case of the OUT parameter no value is passed to the procedure. It can be modified inside the procedure. And the variable is available outside the procedure. The INOUT variable is the blending of the both IN and OUT parameters. It can be passed to the procedure, changed there and can be retrieved outside the procedure.

4. Procedure parameters

```
-- this procedure computes the power
-- of a given value
DELIMITER $$
DROP PROCEDURE IF EXISTS Pow;
CREATE PROCEDURE Pow(IN val DOUBLE, OUT p DOUBLE)
BEGIN
    SET p = val * val;
END
$$
DELIMITER ;
```

In this procedure, we compute the power of a given value.

4. Procedure parameters

The procedure takes two parameters. The first is the value to compute the power. It is declared to be IN. It is passed to the routine and used there. The second variable is an OUT variable. It is the parameter where we store the result of this procedure. It can be used after the routine has finished.

```
CREATE PROCEDURE Pow(IN val DOUBLE, OUT p DOUBLE)
```

We create the procedure Pow(). We call it using the CALL statement. The result is stored in the @p variable. Finally, we select the @p variable to see its content.

```
mysql> source power.sql
mysql> CALL Pow(3, @p);
mysql> SELECT @p;
+------+
| @p   |
+------+
|    9 |
+------+
```

5. Random numbers

In the following example, we will create a procedure which produces five random numbers.

From 0 to 9.

```
-- this procedure generates
-- five random numbers from 0 to 9
DELIMITER $$
DROP PROCEDURE IF EXISTS FiveRandomNumbers;
CREATE PROCEDURE FiveRandomNumbers()
BEGIN
    SET @i = 0;
    REPEAT
        SELECT FLOOR(RAND() * 10) AS 'Random Number';
        SET @i = @i + 1;
    UNTIL @i >=5 END REPEAT;
END
$$
DELIMITER ;
```

5. Random numbers

In this procedure, we will use RAND() and FLOOR() built-in functions.

```
SET @i = 0;
```

This variable is a counter.

```
REPEAT
    SELECT FLOOR(RAND() * 10) AS 'Random Number';
    SET @i = @i + 1;
UNTIL @i >=5 END REPEAT;
```

The REPEAT, UNTIL keywords create a loop. The counter is used to control the number of iterations. In our case, we have five. The RAND() function returns a decimal number and the FLOOR() function is used to round it.

5. Random numbers

```
mysql> source fiverandomnumbers.sql;
mysql> CALL FiveRandomNumbers;
+---------------+
| Random Number |
+---------------+
|             9 |
+---------------+
1 row in set (0.00 sec)
+---------------+
| Random Number |
+---------------+
|             1 |
+---------------+
...
```

We create the procedure using the source command. And then call it.

6. Finding rountines

In MySQL, we can use SHOW PROCEDURE STATUS and SHOW FUNCTION STATUS to see routines and their characteristics in our database.
There is also a ROUTINES table in the information_schema database. We can query the table for information about stored routines.

```
mysql> SELECT SPECIFIC_NAME from information_schema.ROUTINES
    -> WHERE ROUTINE_TYPE='PROCEDURE';
+--------------------+
| SPECIFIC_NAME      |
+--------------------+
| AllCars            |
| FiveRandomNumbers  |
| Pow                |
+--------------------+
```

This statement shows all procedures in the database.

6. Finding rountines

This statement shows all functions in the database.

```
SELECT SPECIFIC_NAME from information_schema.ROUTINES  WHERE ROUTINE_TYPE='FUNCTION';
+---------------+
| SPECIFIC_NAME |
+---------------+
| CircleArea    |
+---------------+
```