Procedures and Functions

MySQL

Stored Procedure in MySQL

- Creating procedure
- Calling procedure
- Dropping procedure
- Variables in procedure
- Parameters in procedure
- Check the list and definition of stored procedures
- Conditional statements
- Loops
- Function

Stored Procedure

- If we want to save the query on the database server for execution later, then stored procedure is one way to do that
- By definition, a stored procedure is a segment of declarative SQL statements stored inside the MySQL Server

```
create procedure procedure_name()
begin
        //SQL statements
end
```

Reference: http://www.mysqltutorial.org/mysql-stored-procedure-tutorial.aspx

EXAMPLE

```
mysql> delimiter $$

mysql> create procedure sp_list()
       begin
              select sno, pno from sp;
       end$$

mysql> delimiter ;
```

Once the stored procedure is stored, you can invoke it by using the CALL statement

```
mysql> call sp_list();
```

To drop a procedure-

```
mysql> drop procedure [if exists] sp_list;
```

Variables in the stored procedure

- Typically variables are used in stored procedures to hold immediate results.
- These variables are local to the stored procedure.
- The variables must be declared before using it
- To declare a variable inside a stored procedure, use the DECLARE statement as follows:

```
DECLARE variable_name datatype(size) [DEFAULT default_value];

DECLARE x, y INT DEFAULT 0;
```

Once a variable is declared, it is ready to use. To assign a variable a value, we can use the SET statement:

```
SET variable_name = value;
```

EXAMPLE

```
create procedure supp_count()
begin
        declare suppCount int default 0;
        select count(*)
        into suppCount
        from suppliers;
        select suppCount;
end$$
```

```
mysql> call supp_count();
```

Stored Procedure Parameter

- Parameters are used to make the procedure more flexible and useful
- Three modes of parameters are – IN, OUT, INOUT
- IN parameter
  - IN is the default mode. When we define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure.
  - The value of the IN parameter may be changed inside the stored procedure, but its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

Stored Procedure Parameter

- OUT parameter
  - The value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program
- INOUT parameter
  - An INOUT parameter is a combination of IN and OUT parameters. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter, and pass the new value back to the calling program.

[IN | OUT | INOUT] parameter_name datatype[(length)]

Example of In Parameter

```
create procedure supp_city(in scity varchar(20))
begin
        select *
        from suppliers
        where city=scity;
end$$
```

Example of OUT parameter

```
 mysql> delimiter $$
create procedure supp_city_cnt(in scity varchar(20), out cnt int)
begin
        select count(sno) into cnt
        from suppliers
        where city=scity;
end$$
```

Pass a session variable, to get the returned value

```
   mysql> delimiter ;

  mysql> call supp_city_count('patna',@op);
  mysql> select @op;
```

Example of INOUT parameter

```
 mysql> delimiter $$
create procedure setCounter(inout cnt int, in inc int)
begin
     set cnt = cnt + inc;
end$$

 mysql> delimiter ;
 mysql> set @n=4
 mysql> call setCounter(@n,2);
 mysql> select @n;
```

Check the list and definition of stored procedures

- We can use the following syntax

```
SHOW PROCEDURE STATUS [LIKE 'pattern' | WHERE
search_condition]
```

```
SHOW PROCEDURE STATUS WHERE db = ‘SP';
```

```
SHOW PROCEDURE STATUS like = ‘%supp%';
```

- To check the definition of a stored procedure

```
SHOW CREATE PROCEDURE procedure_name;
```

CONDITIONAL STATEMENT- IF STATEMENT

- Allows to specify a set of statements based on a condition

```
IF condition THEN
   statements;
END IF;
```

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

```
IF condition THEN
   statements;
ELSEIF elseif-condition
THEN
   elseif-statements;
…
ELSE
   else-statements;
END IF;
```

Example

```
create procedure supp_city(in scity varchar(20), out op varchar(20))
begin
        declare cnt int;
        select count(sno) into cnt
        from suppliers
        where city=scity;
if (cnt>10) then
   set op='popular';
else
   set op='non-popular';
end if;
end$$
```

CASE STATEMENT

```
CASE case_value
WHEN when_value1 THEN statements
WHEN when_value1 THEN statements
…
ELSE else-statements
END CASE;
```

In this syntax, the simple CASE statement sequentially compares the case_value is with the when_value1, when_value2, … until it finds one is equal. When the CASE finds a case_value equal to a when_value, it executes statements in the corresponding THEN clause.

If CASE cannot find any when_value equal to the case_value, it executes the else-statements in the ELSE clause if the ELSE clause is available.

When the ELSE clause does not exist and the CASE cannot find any when_value equal to the case_value, it issues an error:

Case Example

```
CREATE PROCEDURE GetCustomerShipping(IN  pCustomerNUmber INT,
   OUT pShipping  VARCHAR(50))
BEGIN
   DECLARE customerCountry VARCHAR(100);

SELECT
   country INTO customerCountry FROM customers
WHERE customerNumber = pCustomerNUmber;

   CASE customerCountry
      WHEN  'USA' THEN
        SET pShipping = '2-day Shipping';
      WHEN 'Canada' THEN
        SET pShipping = '3-day Shipping';
      ELSE
        SET pShipping = '5-day Shipping';
   END CASE;
END$$
```

LOOP

Allows to execute one or more statements repeatedly. General syntax is-

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

LEAVE statement is used to terminate a loop immediately after the condition is met

```
[label]: LOOP
    ...
    -- terminate the loop
    IF condition THEN
        LEAVE [label];
    END IF;
    ...
END LOOP;
```

LOOP

ITERATE statement is used to skip the loop immediately and start a new iteration

```
CREATE PROCEDURE LoopDemo()
BEGIN
   DECLARE x  INT;
   DECLARE str  VARCHAR(255);
      SET x = 1;
      SET str =  '';
      loop_label:  LOOP
      IF  x > 10 THEN
         LEAVE  loop_label;
      END  IF;

      SET  x = x + 1;
      IF  (x mod 2) THEN
         ITERATE  loop_label;
      ELSE
         SET  str = CONCAT(str,x,',');
      END  IF;
   END LOOP;
   SELECT str;
END$$
```

WHILE LOOP

The WHILE loop is a loop statement that executes a block of code repeatedly as long as a condition is true.

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Let’s create a table as follows-

```
CREATE TABLE calendars(
    id INT AUTO_INCREMENT,
    fulldate DATE UNIQUE,
    day TINYINT NOT NULL,
    month TINYINT NOT NULL,
    quarter TINYINT NOT NULL,
    year INT NOT NULL,
    PRIMARY KEY(id)
);
```

WHILE LOOP EXAMPLE

```
CREATE PROCEDURE InsertCalendar(dt DATE)
BEGIN
   INSERT INTO calendars(
      fulldate,
      day,
      month,
      quarter,
      year
   )
   VALUES(
      dt,
      EXTRACT(DAY FROM dt),
      EXTRACT(MONTH FROM dt),
      EXTRACT(QUARTER FROM dt),
      EXTRACT(YEAR FROM dt)
   );
END$$
```

While loop example continues

```
CREATE PROCEDURE LoadCalendars(
    startDate DATE,
    day INT
)
BEGIN

    DECLARE counter INT DEFAULT 1;
    DECLARE dt DATE DEFAULT startDate;

    WHILE counter <= day DO
        CALL InsertCalendar(dt);
        SET counter = counter + 1;
        SET dt = DATE_ADD(dt,INTERVAL 1 day);
    END WHILE;

END$$
```

FUNCTION

- A stored function is a special kind of stored procedure that returns a single value

```
CREATE FUNCTION function_name(
    param1,
    param2,…
)
RETURNS datatype
[NOT] DETERMINISTIC
BEGIN
-- statements
END $$
```

Function example

```
CREATE FUNCTION CustomerLevel( credit DECIMAL(10,2))
RETURNS VARCHAR(20)
DETERMINISTIC
BEGIN
   DECLARE customerLevel VARCHAR(20);

   IF credit > 50000 THEN
      SET customerLevel = 'PLATINUM';
   ELSEIF (credit >= 50000 AND
         credit <= 10000) THEN
      SET customerLevel = 'GOLD';
   ELSEIF credit < 10000 THEN
      SET customerLevel = 'SILVER';
   END IF;
   -- return the customer level
   RETURN (customerLevel);
END$$
```