DBMS - Stored procedure (03.03.21)

- A **stored procedure** is a collection of pre-compiled SQL statements **stored** inside the database. It is a subroutine or a subprogram in the regular computing language.
- A **stored procedure** always contains a name, parameter lists, and SQL statements.
- To create a new stored procedure, you use the CREATE PROCEDURE statement.

Creating / Executing / Removing a stored procedure

From Single Table:

```
Drop procedure if exists sps;  ** Removing a stored procedure

Delimiter //
create procedure sps ()
begin
select * from Movie;
end //

delimiter ;

call sps()        **Executing a stored procedure
```

From Multiple Table:

```
delimiter $$
create procedure spm ()
begin
select m.mID,m.title,r.rID,r.stars from Movie m , Rating r where m.mID=r.mID;
end  $$

delimiter ;
```

Listing Stored Procedures

```
show procedure status;

show procedure status where db=’view’;

show procedure status like  ‘%p%’;
```

Stored Procedure Variables

EXAMPLE 1 :

```
DELIMITER $$

CREATE PROCEDURE  gettotalmovie()
BEGIN
        DECLARE totalmovie INT DEFAULT 0;
        SET totalmovie = 5;

    SELECT COUNT(*)  INTO totalmovie   FROM  Movie;

    SELECT totalmovie;

END $$

DELIMITER ;


CALL gettotalmovie();
```

EXAMPLE 2 :

```
DELIMITER $$
CREATE PROCEDURE  gettotalstars()
BEGIN
        DECLARE totalstars INT DEFAULT 0;

    SELECT sum(stars)  INTO totalstars   FROM  Rating;

    SELECT totalstars;

END $$

DELIMITER ;

CALL gettotalstars();
```

Stored Procedure Parameters

IN parameters :

```
DELIMITER //

CREATE PROCEDURE Getmoviedetail(IN movieID INT)

BEGIN
        SELECT *
        FROM Movie
        WHERE mID = movieID;
END //

DELIMITER ;

CALL Getmoviedetail('101');
```

OUT parameters :

```
DELIMITER $$


CREATE PROCEDURE GetMoviename (IN  movieID INT ,  OUT  name varchar(255))

BEGIN
        SELECT title
        INTO name
        FROM Movie
        WHERE mID = movieID;
END$$

DELIMITER ;

 call GetMoviename(101,@name);

select @name;
```

INOUT parameters :

```
DELIMITER $$

CREATE PROCEDURE SetCounter(INOUT counter INT,IN inc INT)

BEGIN
        SET counter = counter + inc;
END$$

DELIMITER ;

SET @counter = 1;

CALL SetCounter(@counter,1); -- 2
CALL SetCounter(@counter,1); -- 3
CALL SetCounter(@counter,5); -- 8
SELECT @counter;
```

MySQL IF-THEN-ELSEIF-ELSE statement

```
DELIMITER //

CREATE PROCEDURE GetMoviestatus(IN  movieID INT, OUT  movieLevel varchar(20))
BEGIN
    DECLARE movieyear INT DEFAULT 0;

    SELECT year INTO movieyear FROM Movie WHERE mID = movieID;
    IF movieyear>2000  THEN
        SET movieLevel = ‘1’;
    ELSEIF movieyear<=2000 AND movieyear>1950 THEN
        SET movieLevel = ‘2’;
    ELSE
        SET movieLevel= ‘3’;

    END IF;
END //
DELIMITER ;
```

MySQL CASE Statement (10.03.21)

Simple CASE statement

```
DELIMITER $$

CREATE PROCEDURE GetMoviename( IN  movieID INT,  OUT name  VARCHAR(50))
BEGIN
    DECLARE myear VARCHAR(100);

SELECT  year INTO myear FROM  Movie  WHERE   mID = movieID;

    CASE myear
            WHEN  '2009' THEN
              SET name = 'Recent';
            WHEN '1981' THEN
              SET name = 'OLD';
            ELSE
              SET  name = 'VERYOLD';
    END CASE;
END$$

DELIMITER ;

mysql> call GetMoviename(108,@result);

Query OK, 1 row affected (0.00 sec)



mysql> select  @result;

+---------+

| @result |

+---------+

| OLD     |

+---------+

1 row in set (0.00 sec)
```

MySQL LOOP / LEAVE

```
DROP PROCEDURE LoopDemo;

DELIMITER $$
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$$LEAVE  loop_label;

DELIMITER ;

CALL LoopDemo();
```

```
mysql> call LoopDemo();

+-------------+

| str         |

+-------------+

| 2,4,6,8,10, |

+-------------+

1 row in set (0.00 sec)



Query OK, 0 rows affected (0.00 sec)
```

MySQL WHILE Loop

```
DELIMITER $$
```

```
CREATE PROCEDURE WhileLoopDemo()
BEGIN
        DECLARE x  INT;
        DECLARE str  VARCHAR(255);

        SET x = 1;
        SET str =  '';

        loop_label:  WHILE  x < 10  DO


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

DELIMITER ;

mysql> call WhileloopDemo();

+-------------+

| str         |

+-------------+

| 2,4,6,8,10, |

+-------------+

1 row in set (0.00 sec)



Query OK, 0 rows affected (0.00 sec)
```

MySQL REPEAT Loop

```
DELIMITER $$

CREATE PROCEDURE RepeatDemo()
BEGIN
    DECLARE counter INT DEFAULT 0;
    DECLARE result VARCHAR(100) DEFAULT '';

    REPEAT
        SET result = CONCAT(result,counter,',');
        SET counter = counter + 1;
    UNTIL counter >= 10
    END REPEAT;

    SELECT result;
END$$

DELIMITER ;
```

```
mysql> call RepeatDemo();
+--------------------+
| result             |
+--------------------+
| 1,2,3,4,5,6,7,8,9, |
+--------------------+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

MySQL Error Handling in Stored Procedures

```
DELIMITER $$

CREATE PROCEDURE Ex( IN inSId INT,   IN inPId INT)

BEGIN

   DECLARE EXIT HANDLER FOR 1062
   BEGIN
      SELECT  'Duplicate key occurred' AS message from dual;
   END;


   INSERT INTO exception(sid,pid)
   VALUES(inSId,inPId);


   SELECT COUNT(*) FROM exception;

END$$

DELIMITER ;

call Ex(1,3);

+----------+

| COUNT(*) |

+----------+

|        1 |

+----------+

1 row in set (0.03 sec)



Query OK, 0 rows affected (0.03 sec)
```

```
mysql> call Ex(1,5);

+------------------------+

| message                |

+------------------------+

| Duplicate key occurred |

+------------------------+

1 row in set (0.00 sec)


+----------+

| COUNT(*) |

+----------+

|        1 |

+----------+

1 row in set (0.00 sec)


Query OK, 0 rows affected (0.00 sec)


mysql> call Ex(2,7);

+----------+

| COUNT(*) |

+----------+

|        2 |

+----------+

1 row in set (0.38 sec)
```

DELIMITER $$ **EXIT**

```
CREATE PROCEDURE Exex( IN inSId INT,   IN inPId INT)

BEGIN
   -- exit if the duplicate key occurs
   DECLARE EXIT HANDLER FOR 1062
   BEGIN
      SELECT  'Duplicate key occurred' AS message;
   END;

   -- insert a new row into the SupplierProducts
   INSERT INTO exception(sid,pid)
   VALUES(inSId,inPId);

   -- return the products supplied by the supplier id
   SELECT COUNT(*)
   FROM exception;

END$$

DELIMITER ;

mysql> call Exex(1,2);

+----------+

| COUNT(*) |

+----------+

|        1 |

+----------+

1 row in set (0.03 sec)

Query OK, 0 rows affected (0.03 sec)

mysql> call Exex(2,2);

+----------+

| COUNT(*) |

+----------+

|        2 |

+----------+

1 row in set (0.27 sec)
```

```
Query OK, 0 rows affected (0.27 sec)

mysql> call Exex(2,5);

+------------------------+

|     message                |

+------------------------+

| Duplicate key occurred |

+------------------------+

1 row in set (0.00 sec)

DELIMITER $$

CREATE PROCEDURE Exex( )

BEGIN
    -- exit if the duplicate key occurs
    DECLARE EXIT HANDLER FOR 1054
    BEGIN
        SELECT  'Wrong Column name' AS message;
    END;

    -- insert a new row into the SupplierProducts
   select Sed from  exception;


    -- return the products supplied by the supplier id
    SELECT COUNT(*)
    FROM exception;

END$$

DELIMITER ;

mysql> call Exex();

+-------------------+

| message           |

+-------------------+

| Wrong Column name |

+-------------------+

1 row in set (0.00 sec)
```

Raising Error Conditions with MySQL SIGNAL / RESIGNAL Statements

SIGNAL statement used to return an error or warning condition to the caller from a stored program . The SIGNAL statement provides you with control over which information for returning such as value and messageSQLSTATE.

RESIGNAL statement within an error or warning handler for to passes the error information.

```
DELIMITER $$

CREATE PROCEDURE Divide(IN numerator INT, IN denominator INT, OUT result double)
BEGIN
	DECLARE division_by_zero CONDITION FOR SQLSTATE '22012';

	DECLARE CONTINUE HANDLER FOR division_by_zero
	RESIGNAL SET MESSAGE_TEXT = 'Division by zero / Denominator cannot be zero';

	IF denominator = 0 THEN
		SIGNAL division_by_zero;
	ELSE
		SET result := numerator / denominator;
	END IF;
END $$

DELIMITER ;
```

```
mysql> call Divide(10,5,@r);

Query OK, 0 rows affected (0.00 sec)



mysql> select @r;

+------+

| @r   |

+------+

|    2 |

+------+

1 row in set (0.00 sec)



mysql> call Divide(10,0,@r);
```

```
ERROR 1644 (22012): Division by zero / Denominator cannot be zero
```

MySQL Stored Procedures That Return Multiple Values

MySQL stored function returns only one value. To develop stored programs that return multiple values, you need to use stored procedures with INOUT or OUT parameters.

```
DELIMITER $$


CREATE PROCEDURE get_order_by_cust(
	IN cust_no INT,
	OUT shipped INT,
	OUT canceled INT,
	OUT resolved INT,
	OUT disputed INT)
BEGIN
		-- shipped
		SELECT
            count(*) INTO shipped
        FROM
            orders
        WHERE
            customerNumber = cust_no
                AND status = 'Shipped';

		-- canceled
		SELECT
            count(*) INTO canceled
        FROM
            orders
        WHERE
            customerNumber = cust_no
                AND status = 'Canceled';

		-- resolved
		SELECT
            count(*) INTO resolved
        FROM
            orders
        WHERE
            customerNumber = cust_no
                AND status = 'Resolved';

		-- disputed
		SELECT
            count(*) INTO disputed
        FROM
        orders
      WHERE
        customerNumber = cust_no
          AND status = 'Disputed';

END $$

delimiter ;

mysql> CALL get_order_by_cust(105,@shipped,@canceled,@resolved,@disputed);

Query OK, 1 row affected (0.00 sec)

mysql> SELECT @shipped,@canceled,@resolved,@disputed;

+----------+-----------+-----------+-----------+

| @shipped | @canceled | @resolved | @disputed |

+----------+-----------+-----------+-----------+

|        1 |         1 |         1 |         2 |

+----------+-----------+-----------+-----------+

1 row in set (0.00 sec)
```

MySQL Stored Function

A stored function is a special kind stored program that returns a single value. Typically, you use stored functions to encapsulate common formulas or business rules that are reusable among SQL statements or stored programs.

The following CREATE FUNCTION statement creates a function that returns the customer level based on credit:

Example 1:

```
DELIMITER $$

CREATE FUNCTION CustomerLevel(credit int ) RETURNS VARCHAR(20)

BEGIN
   DECLARE customerLevel VARCHAR(20);

   IF credit > 5000 THEN
            SET customerLevel = 'PLATINUM';
   ELSEIF (credit <= 5000 AND  credit >= 1000) THEN
            SET customerLevel = 'GOLD';
   ELSEIF credit < 1000 THEN
          SET customerLevel = 'SILVER';
```

```
    END IF;

        RETURN (customerLevel);

END$$
DELIMITER ;



DELIMITER $$

CREATE PROCEDURE GetCustomerLevel(IN  customerNo INT,  OUT customerLevel
                                                                    VARCHAR(20) )
BEGIN
      DECLARE credit  int DEFAULT 0;

      SELECT cuscredits  INTO  credit  FROM cusdetails  WHERE cusid = customerNo;

      SET customerLevel = CustomerLevel(credit);

     UPDATE cusdetails SET cusstatus=customerLevel WHERE cusid=customerNo;

END$$

DELIMITER ;

mysql> call GetCustomerLevel(101,@res);

Query OK, 1 row affected (0.03 sec)



mysql> select @res;

+------+

| @res |

+------+

| GOLD |

+------+

1 row in set (0.00 sec)



mysql> call GetCustomerLevel(102,@res);

Query OK, 1 row affected (0.28 sec)
```

```
mysql> select @res;

+----------+

| @res     |

+----------+

| PLATINUM |

+----------+

1 row in set (0.00 sec)
```

EXCUTION OF FUNCTION

```
 set @res=CustomerLevel(105);

Query OK, 0 rows affected (0.00 sec)

mysql> select @res;

+--------+

| @res   |

+--------+

| SILVER |

+--------+

1 row in set (0.00 sec)
```

MySQL DROP FUNCTION

```
DROP FUNCTION [IF EXISTS] function_name;

DROP FUNCTION IF EXISTS NonExistingFunction;
```

Listing Stored Functions

```
SHOW FUNCTION STATUS    [LIKE 'pattern' | WHERE search_condition];

SHOW FUNCTION STATUS LIKE '%u%';

SHOW FUNCTION STATUS WHERE db = 'view';
```

```
DELIMITER $$

CREATE FUNCTION Fun() returns varchar(100)
BEGIN
   DECLARE counter INT DEFAULT 1;
   DECLARE result VARCHAR(100) DEFAULT '';

   REPEAT
      SET result = CONCAT(result,counter,'  ');
      SET counter = counter + 1;
   UNTIL counter >= 10
   END REPEAT;

   return result;
END$$

DELIMITER ;



DROP PROCEDURE WhileLoopDemo;

DELIMITER $$
CREATE PROCEDURE WhileLoopDemo()
BEGIN
      DECLARE x  INT;
      DECLARE str  VARCHAR(255);

      SET x = 0;
      SET str =  '';

       WHILE  x < 10  DO


            SET  x = x + 1;
            SET  str = CONCAT(str,x,',');

      END WHILE;
      SELECT str;
END$$

DELIMITER ;
```