

## Stored Procedures

A stored procedure is a collection of SQL statements that are stored in Database. One of the major advantages of stored procedures is that they can be used to encapsulate and represent business transactions.

To create a stored procedure, you use the following syntax:

```
CREATE PROCEDURE procedure_name [([IN/OUT] argument data-type, ...)]  
  
BEGIN  
  
SQL statements;  
  
...  
  
END;
```

Note the following important points about stored procedures and their syntax:

- Argument specifies the parameters that are passed to the stored procedure. A stored procedure could have zero or more arguments or parameters.
- IN/OUT indicates whether the parameter is for input, output, or both.
- data-type is one of the procedural SQL data types used in the RDBMS. The data types normally match those used in the RDBMS table creation statement.

### Example 1:

Consider the following table (*script available on Google Classroom in stored procedures lab folder*):

```
mysql> describe product;
```

Field	Type	Null	Key	Default	Extra
P_CODE	varchar(10)	NO	PRI	NULL	
P_DESCRIPT	varchar(35)	NO		NULL	
P_INDATE	date	NO		NULL	
P_ONHAND	float	NO		NULL	
P_MIN	float	NO		NULL	
P_PRICE	float	NO		NULL	
P_DISCOUNT	float	NO		NULL	
U_CODE	int(11)	YES	MUL	NULL	

```
8 rows in set (0.00 sec)  
  
mysql>
```

```
mysql> select * from product;
```

P_CODE	P_DESCRIPT	P_INDATE	P_ONHAND	P_MIN	P_PRICE	P_DISCOUNT	U_CODE
11QER/31	Power painter, 15 psi., 3-nozzle	2003-11-03	8	5	109.99	0	25595
13-Q2/P2	7.25-in. pwr. saw blade	2003-12-13	32	15	14.99	0.05	21344
14-Q1/L3	9.00-in. pwr. saw blade	2003-11-13	18	12	17.49	0	21344
1546-QQ2	Hrd. cloth, 1/4-in., 2x50	2004-01-04	15	8	39.95	0	23119
1558-QW1	Hrd. cloth, 1/2-in., 3x50	2004-01-15	23	5	43.99	0	23119
2232/QTY	B&D jigsaw, 12-in. blade	2003-12-10	8	5	109.92	0.05	24288
2232/QWE	B&D jigsaw, 8-in. blade	2003-12-24	6	5	99.87	0.05	24288
2238/QPD	B&D cordless drill, 1/2-in.	2004-01-20	12	5	38.95	0.05	25595
23109-HB	Claw hammer	2004-01-20	23	10	9.95	0.1	21225
23114-AA	Sledge hammer, 12 lb.	2004-01-20	8	5	14.4	0.05	NULL
54778-2T	Rat-tail file, 1/8-in. fine	2003-12-15	43	20	4.99	0	21344
89-WRE-Q	Hicut chain saw, 16 in.	2004-02-17	11	5	256.99	0.05	24288
PUC23DRT	PUC pipe, 3.5-in., 8-ft	2004-02-20	188	75	5.87	0	NULL
SM-18277	1.25-in. metal screw, 25	2004-03-01	172	75	6.99	0	21225
SW-23116	2.5-in. wd. screw, 50	2004-02-24	237	100	8.45	0	21231
WR3/TT3	Steel matting, 4'x8'x1/6", .5" mesh	2004-01-07	18	5	119.95	0.1	25595

```
16 rows in set (0.00 sec)
```

To illustrate stored procedures, assume that you want to create a procedure (PRC\_PROD\_DISCOUNT) to assign an additional 5 percent discount for all products when the quantity on hand is more than or equal to twice the minimum quantity.

```
CREATE PROCEDURE PRG_PROD()
BEGIN
UPDATE product
SET P_DISCOUNT = (P_DISCOUNT*0.05)+ P_DISCOUNT
WHERE P_ONHAND >= P_MIN*2;
END;
```

## Pick a Delimiter

The delimiter is the character or string of characters which is used to complete an SQL statement. By default, we use semicolon (;) as a delimiter. But this causes problem in stored procedure because a procedure can have many statements, and everyone must end with a semicolon. So, for your delimiter, pick a string which is rarely occur within statement or within procedure. Here we have used double dollar sign i.e. \$\$\$. You can use whatever you want. To resume using ";" as a delimiter later, say "DELIMITER ;

See here how to change the delimiter:

```
DELIMITER $$
```

```
CREATE PROCEDURE PRG_PROD()  
  
BEGIN  
  
UPDATE product  
  
SET P_DISCOUNT = (P_DISCOUNT*0.05)+ P_DISCOUNT  
  
WHERE P_ONHAND >= P_MIN*2;  
  
END $$  
  
DELIMITER ; /*to change the delimiter back*/
```

In this case we will write `call prg_prod();` to execute the procedure.

SHOW CREATE PROCEDURE statements used to describe the stored procedure.

In order to list all procedure, the following queries can be used:

```
SHOW PROCEDURE STATUS;
```

```
SELECT name, type FROM mysql.proc;
```

### **Declare a Variable:**

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

To provide a default value for a variable, include a DEFAULT clause. The value can be specified as an expression; it need not be constant. If the DEFAULT clause is missing, the initial value is NULL.

### **Local variables Example**

Local variables are declared within stored procedures and are only valid within the BEGIN...END block where they are declared. Local variables can have any SQL data type. The following example shows the use of local variables in a stored procedure.

```

DELIMITER $$
CREATE PROCEDURE my_procedure_Local_Variables()
BEGIN  /* declare local variables */
DECLARE a INT DEFAULT 10;
DECLARE b, c INT;      /* using the local variables */
SET a = a + 100;
SET b = 2;
SET c = a + b;
BEGIN      /* local variable in nested block */
DECLARE c INT;
SET c = 5;
/* local variable c takes precedence over the one of the
same name declared in the enclosing block. */
SELECT a, b, c;
END;
SELECT a, b, c;
END$$

DELIMITER ;

CALL my_procedure_Local_Variables();

```

### Parameter IN example

In the following procedure, we have used a IN parameter 'var1' (type integer) which accept a number from the user. Within the body of the procedure, there is a SELECT statement which fetches rows from 'P' table and display the data whose p\_ONHAND is equal to var1. Here is the procedure:

```

DELIMITER $$

CREATE PROCEDURE my_proc_IN (IN var1 INT)
BEGIN
    SELECT * FROM Product where P_ONHAND = var1;
END$$

DELIMITER ;

call my_proc_IN(8);

```

This procedure takes var 1 as a parameter and display that many numbers of rows from the table product;

```
DELIMITER $$

CREATE PROCEDURE LIMIT_ROW (IN var1 INT)
BEGIN
    SELECT * FROM Product LIMIT var1;
END$$

DELIMITER ;

call LIMIT_ROW (8);
```

### Parameter OUT example

The following example shows a simple stored procedure that uses an OUT parameter.

Using the product table again:

```
DELIMITER $$;

CREATE PROCEDURE PRG_AVG_PRICE(out avg_price decimal)
BEGIN
    SELECT AVG(P_PRICE) INTO avg_price FROM Product;
END$$
```

In order to execute the procedure, write:

```
call prg_avg_price(@out);
```

and then:

```
SELECT @out;
```

To print the output.

## DROP PROCEDURE

This statement is used to drop a stored procedure or function. That is, the specified routine is removed from the server.

```
DROP PROCEDURE procedure_name ;
```

Reference: <https://www.w3resource.com/mysql/mysql-procedure.php#SP>

## Lab Task

Table 1: Users [user\_id (p.k)]

user_id	username	password	email
1	abc123	def321	<a href="mailto:abc123@yahoo.com">abc123@yahoo.com</a>
2	xyz789	asd123	<a href="mailto:xyz789@gmail.com">xyz789@gmail.com</a>

Table 2: Summary [id(p.k)]

id	total_users	Yahoo	Hotmail	Gmail
1	2	1	0	1

Create a database name person and create the tables above.

1. Write a procedure that take id, total\_user, Yahoo, Hotmail, Gmail values as an input and insert the data into the table summary.
2. Write a procedure that take user\_id, username, password, email values as an input and insert the data into the table Users.
3. Write a procedure that output the average value of yahoo
4. Write a procedure that output the min value of Gmail.
5. Write a procedure that output the max value of Hotmail.
6. Write a procedure that increment the total user by 1 when yahoo value is less than equal to Hotmail.