------Creating Stored Procedures in MySQL------

--Make sure you have version 5 of MySQL:

SELECT VERSION();

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

| VERSION() |

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

| 5.0.15-nt |

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

1 row in set (0.00 sec)

--First pick a database to use (a procedure, like a table, is associated with

--a single database.) For these examples, I will use a database that is populated

--with the tables from HW 2:

USE ozaidan\_hw2;

--Next, change the delimiter, because we will use the semicolon WITHIN the

--procedure declarations, and therefore it cannot be the delimiter anymore:

DELIMITER //

--OK, let's get started. Creating procedures is straightforward:

CREATE PROCEDURE myFirstProc()

SELECT 'Hello World!' AS Output;

//

Query OK, 0 rows affected (0.00 sec)

--Whenever you create a procedure (successfully) you should get a 'Query OK' message.

--Calling a procedure is also straightforward:

CALL myFirstProc() //

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

| Output |

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

| Hello World! |

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

1 row in set (0.00 sec)

--By the way, procedure names are NOT case sensitive:

CALL myfirstproc() //

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

| Output |

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

| Hello World! |

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

1 row in set (0.00 sec)

--Another example:

CREATE PROCEDURE ListStudents()

SELECT \*

FROM Student;

//

CALL ListStudents() //

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

| StuID | LName | Fname | Age | Sex | Major | Advisor | city\_code |

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

| 1001 | Smith | Linda | 18 | F | 600 | 1121 | BAL |

| 1002 | Kim | Tracy | 19 | F | 600 | 7712 | HKG |

.

.

.

| 1034 | Epp | Eric | 18 | M | 50 | 5718 | BOS |

| 1035 | Schmidt | Sarah | 26 | F | 50 | 5718 | WAS |

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

34 rows in set (0.00 sec)

--Say we only want student ID's and names. To update a procedure, we must

--first DROP it:

DROP PROCEDURE IF EXISTS ListStudents //

Query OK, 0 rows affected (0.00 sec)

--Again, whenever you drop a procedure, you should get a 'Query OK' message.

--From now on, we will always use "DROP PROCEDURE IF EXISTS procName" as

--a standard practice before declaring procedures:

DROP PROCEDURE IF EXISTS ListStudents //

CREATE PROCEDURE ListStudents()

SELECT StuID, LName, FName

FROM Student;

//

CALL ListStudents() //

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

| StuID | LName | FName |

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

| 1001 | Smith | Linda |

| 1002 | Kim | Tracy |

.

.

.

| 1034 | Epp | Eric |

| 1035 | Schmidt | Sarah |

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

34 rows in set (0.00 sec)

--OK, let's use some parameters:

DROP PROCEDURE IF EXISTS sayHello //

CREATE PROCEDURE sayHello(IN name VARCHAR(20))

SELECT CONCAT('Hello ', name, '!') AS Greeting;

//

--The 'IN' keyword tells MySQL that is should be expecting an input value for

--the parameter......hunh? Why would a parameter NOT have an input value? You will

--see in a little bit. First, let's see if sayHello works:

CALL sayHello('Omar') //

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

| Greeting |

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

| Hello Omar! |

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

1 row in set (0.00 sec)

--Another example:

DROP PROCEDURE IF EXISTS saySomething //

CREATE PROCEDURE saySomething(IN phrase VARCHAR(20), IN name VARCHAR(20))

SELECT CONCAT(phrase, ' ', name, '!') AS Output;

//

CALL saySomething('Go','Blue Jays') //

CALL saySomething('Do','my homework') //

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

| Output |

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

| Go Blue Jays! |

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

1 row in set (0.00 sec)

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

| Output |

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

| Do my homework! |

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

1 row in set (0.00 sec)

--and another one:

DROP PROCEDURE IF EXISTS FindStudent //

CREATE PROCEDURE FindStudent(IN id INT)

SELECT StuID, CONCAT(FName, ' ', LName) AS 'Student Name'

FROM Student

WHERE StuID = id;

//

CALL FindStudent(1001) //

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

| StuID | Student Name |

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

| 1001 | Linda Smith |

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

1 row in set (0.00 sec)

--and yet another:

DROP PROCEDURE IF EXISTS calculate //

CREATE PROCEDURE calculate(IN x INT, IN y INT, OUT sum INT, OUT product INT)

SET sum = x + y;

SET product = x \* y;

//

ERROR 1064 (42000): You have an error in your SQL syntax; check the manual ...

--Well, that wasn't good. The reason is, we must use BEGIN/END if we have

--a compound statement:

DROP PROCEDURE IF EXISTS calculate //

CREATE PROCEDURE calculate(IN x INT, IN y INT, OUT sum INT, OUT product INT)

BEGIN

SET sum = x + y;

SET product = x \* y;

END;

//

--Did you notice the 'OUT' keyword for sum and product? This tells MySQL that those

--two parameters are not 'input' parameters but are 'output' parameters instead.

--Now, when calling the procedure, we need to provide four parameters: two input

--values, and two MySQL \*variables\* where the results will be stored:

CALL calculate(4,5,@s,@p) //

Query OK, 0 rows affected (0.00 sec)

--Here, @s and @p are MySQL variables. Notice that they start with @, although

--procedure \*parameters\* do not start with @

SELECT @s //

SELECT @p //

+------+

| @s |

+------+

| 9 |

+------+

1 row in set (0.00 sec)

+------+

| @p |

+------+

| 20 |

+------+

1 row in set (0.00 sec)

--Note: you can also have INOUT parameters, which serve as both input and output

--parameters.

--OK, let's do some interesting stuff. First off, flow control:

DROP PROCEDURE IF EXISTS mySign //

CREATE PROCEDURE mySign(IN x INT)

BEGIN

IF x > 0 THEN

SELECT x AS Number, '+' AS Sign;

ELSEIF x < 0 THEN

SELECT x AS Number, '-' AS Sign;

ELSE

SELECT x AS Number, 'Zero' AS Sign;

END IF;

END;

//

CALL mySign(2) //

CALL mySign(-5) //

CALL mySign(0) //

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

| Number | Sign |

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

| 2 | + |

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

1 row in set (0.00 sec)

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

| Number | Sign |

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

| -5 | - |

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

1 row in set (0.00 sec)

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

| Number | Sign |

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

| 0 | Zero |

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

1 row in set (0.00 sec)

--Before we get any further, let's introduce variables:

DROP PROCEDURE IF EXISTS mySign //

CREATE PROCEDURE mySign(IN x INT)

BEGIN

DECLARE result VARCHAR(20);

IF x > 0 THEN

SET result = '+';

ELSEIF x < 0 THEN

SET result = '-';

ELSE

SET result = 'Zero';

END IF;

SELECT x AS Number, result AS Sign;

END;

//

CALL mySign(2) //

CALL mySign(-5) //

CALL mySign(0) //

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

| Number | Sign |

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

| 2 | + |

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

1 row in set (0.00 sec)

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

| Number | Sign |

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

| -5 | - |

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

1 row in set (0.00 sec)

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

| Number | Sign |

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

| 0 | Zero |

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

1 row in set (0.00 sec)

--Using CASE:

DROP PROCEDURE IF EXISTS digitName //

CREATE PROCEDURE digitName(IN x INT)

BEGIN

DECLARE result VARCHAR(20);

CASE x

WHEN 0 THEN SET result = 'Zero';

WHEN 1 THEN SET result = 'One';

WHEN 2 THEN SET result = 'Two';

WHEN 3 THEN SET result = 'Three';

WHEN 4 THEN SET result = 'Four';

WHEN 5 THEN SET result = 'Five';

WHEN 6 THEN SET result = 'Six';

WHEN 7 THEN SET result = 'Seven';

WHEN 8 THEN SET result = 'Eight';

WHEN 9 THEN SET result = 'Nine';

ELSE SET result = 'Not a digit';

END CASE;

SELECT x AS Digit, result AS Name;

END;

//

CALL digitName(0) //

CALL digitName(4) //

CALL digitName(100) //

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

| Digit | Name |

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

| 0 | Zero |

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

1 row in set (0.00 sec)

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

| Digit | Name |

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

| 4 | Four |

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

1 row in set (0.00 sec)

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

| Digit | Name |

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

| 100 | Not a digit |

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

1 row in set (0.00 sec)

--As you'd expect, we have loops. For example, WHILE loops:

DROP PROCEDURE IF EXISTS fact //

CREATE PROCEDURE fact(IN x INT)

BEGIN

DECLARE result INT;

DECLARE i INT;

SET result = 1;

SET i = 1;

WHILE i <= x DO

SET result = result \* i;

SET i = i + 1;

END WHILE;

SELECT x AS Number, result as Factorial;

END;

//

CALL fact(1) //

CALL fact(2) //

CALL fact(4) //

CALL fact(0) //

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

| Number | Factorial |

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

| 1 | 1 |

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

1 row in set (0.00 sec)

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

| Number | Factorial |

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

| 2 | 2 |

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

1 row in set (0.00 sec)

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

| Number | Factorial |

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

| 4 | 24 |

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

1 row in set (0.01 sec)

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

| Number | Factorial |

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

| 0 | 1 |

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

1 row in set (0.00 sec)

--There is also REPEAT/UNTIL loops:

DROP PROCEDURE IF EXISTS fact //

CREATE PROCEDURE fact(IN x INT)

BEGIN

DECLARE result INT DEFAULT 1; /\* notice you can declare a variable\*/

DECLARE i INT DEFAULT 1; /\* and give it a value in one line \*/

REPEAT

SET result = result \* i;

SET i = i + 1;

UNTIL i > x

END REPEAT;

SELECT x AS Number, result as Factorial;

END;

//

CALL fact(1) //

CALL fact(2) //

CALL fact(4) //

CALL fact(0) //

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

| Number | Factorial |

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

| 1 | 1 |

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

1 row in set (0.00 sec)

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

| Number | Factorial |

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

| 2 | 2 |

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

1 row in set (0.00 sec)

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

| Number | Factorial |

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

| 4 | 24 |

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

1 row in set (0.00 sec)

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

| Number | Factorial |

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

| 0 | 1 |

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

1 row in set (0.00 sec)

--OK, do you remember this?

/\*

CREATE PROCEDURE FindStudent(IN id INT)

SELECT StuID, CONCAT(FName, ' ', LName) AS 'Student Name'

FROM Student

WHERE StuID = id;

//

\*/

--What if we only want to extract the name without printing it out?

--Obviously, we need some OUT parameters. Still, how do you extract

--information into those OUT parameters?

--

--Answer: something called a CURSOR:

DROP PROCEDURE IF EXISTS FindName //

CREATE PROCEDURE FindName(IN id INT, OUT fn VARCHAR(20), OUT ln VARCHAR(20))

BEGIN

DECLARE cur CURSOR FOR

SELECT FName, LName

FROM Student

WHERE StuID = id;

OPEN cur;

FETCH cur INTO fn, ln;

CLOSE cur;

END;

//

CALL FindName(1001,@f,@l) //

Query OK, 0 rows affected (0.00 sec)

--Remember that @f and @l are MySQL variables:

SELECT @f //

SELECT @l //

+-------+

| @f |

+-------+

| Linda |

+-------+

1 row in set (0.00 sec)

+-------+

| @l |

+-------+

| Smith |

+-------+

1 row in set (0.00 sec)

--What if we give an invalid student ID?

CALL FindName(0000,@f,@l) //

ERROR 1329 (02000): No data to FETCH

--MySQL complains, as expected. It would be nice to handle this more elegantly, however.

--We need an error HANDLER. Let's modify FindName:

DROP PROCEDURE IF EXISTS FindName //

CREATE PROCEDURE FindName(IN id INT, OUT fn VARCHAR(20), OUT ln VARCHAR(20))

BEGIN

DECLARE cur CURSOR FOR

SELECT FName, LName

FROM Student

WHERE StuID = id;

DECLARE EXIT HANDLER FOR NOT FOUND

SELECT 'Sorry; this ID was not found' AS 'Error Message';

OPEN cur;

FETCH cur INTO fn, ln;

CLOSE cur;

END;

//

CALL FindName(0000,@f,@l) //

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

| Error Message |

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

| Sorry; this ID was not found |

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

1 row in set (0.00 sec)

--Another use for handlers: multiple FETCH calls using a CONTINUE handler.

--

--In this case, we use a CONTINUE handler that, instead of exiting the procedure

--upon encountering a NOT FOUND error, simply sets a variable done = 1.

--

--Why would we do that? And how does that help us carry out multiple FETCH calls?

--

--Take a look at this procedure, which traverses all the entries of a table to

--find the maximum and minimum age:

DROP PROCEDURE IF EXISTS MaxMinAge //

CREATE PROCEDURE MaxMinAge(OUT maxAge INT, OUT minAge INT)

BEGIN

DECLARE currAge,maxSoFar,minSoFar,done INT;

DECLARE cur CURSOR FOR

SELECT Age

FROM Student;

DECLARE CONTINUE HANDLER FOR NOT FOUND

SET done = 1;

SET maxSoFar = 0;

SET minSoFar = 1000;

SET done = 0;

OPEN cur;

WHILE done = 0 DO

FETCH cur INTO currAge;

IF currAge > maxSoFar THEN

SET maxSoFar = currAge;

END IF;

IF currAge < minSoFar THEN

SET minSoFar = currAge;

END IF;

END WHILE;

CLOSE cur;

SET maxAge = maxSoFar;

SET minAge = minSoFar;

END;

//

CALL MaxMinAge(@max,@min) //

Query OK, 0 rows affected (0.00 sec)

SELECT @max //

SELECT @min //

+------+

| @max |

+------+

| 27 |

+------+

1 row in set (0.00 sec)

+------+

| @min |

+------+

| 16 |

+------+

1 row in set (0.00 sec)

--In summary, stored procedures in MySQL look like this:

DROP PROCEDURE IF EXISTS procName //

CREATE PROCEDURE procName(parameter list)

BEGIN

/\* variable declarations \*/

/\* CURSOR definitions \*/

/\* declaring handlers \*/

/\* procedure body...whatever you want it to do \*/

END;

//

--In more detail:

DROP PROCEDURE IF EXISTS procName //

CREATE PROCEDURE procName(IN/OUT/INOUT parName parType, ...)

BEGIN

/\* variable declarations \*/

DECLARE varName,... varType;

/\* e.g. DECLARE myName VARCHAR(20); DECLARE x,y,z INT; \*/

DECLARE varName varType DEFAULT value;

/\* e.g. DECLARE x INT DEFAULT 0; \*/

/\* CURSOR definitions \*/

DECLARE curName CURSOR FOR

SELECT ...

/\* e.g. DECLARE cur1 CURSOR FOR

SELECT FName, LName

FROM Student; \*/

/\* declaring handlers \*/

DECLARE EXIT/CONTINUE HANDLER FOR errorType/errorNumber

... action ...

/\* e.g. DECLARE EXIT HANDLER FOR NOT FOUND

SELECT 'Sorry; this ID was not found' AS 'Error Message'; \*/

/\* e.g. DECLARE CONTINUE HANDLER FOR NOT FOUND

SET done = 1; \*/

/\* procedure body...whatever you want it to do \*/

/\* IF statement \*/

IF cond1 THEN

action1

ELSEIF cond2 THEN

action2

ELSEIF cond3 THEN

action3

ELSE

elseaction

END IF;

/\* e.g. IF x > 0 THEN

SET result = '+';

ELSEIF x < 0 THEN

SET result = '-';

ELSE

SET result = 'Zero';

END IF; \*/

/\* CASE statement \*/

CASE varName

WHEN val1 THEN action1

WHEN val2 THEN action2

ELSE elseaction

END CASE;

/\* e.g. CASE position

WHEN 1 THEN SET result = 'Gold Medal';

WHEN 2 THEN SET result = 'Silver Medal';

WHEN 3 THEN SET result = 'Bronze Medal';

ELSE SET result = 'No Medal!';

END CASE; \*/

/\* WHILE loop \*/

WHILE cond DO

action1

action2

...

END WHILE

/\* e.g. WHILE i < 5 DO

SET result = result + i;

SET i = i + 1;

END WHILE; \*/

/\* REPEAT/UNTIL loop \*/

REPEAT

action1

action2

...

UNTIL cond

END REPEAT;

/\* e.g. REPEAT

SET result = result + i;

SET i = i + 1;

UNTIL i >= 5

END REPEAT; \*/

/\* using a CURSOR \*/

OPEN curName;

.

.

FETCH curName INTO var1, var2, ...;

.

.

CLOSE curName;

/\* e.g. Assume cur1 has id's, first names, and last names

let's find the name of the student whose StuID is x:

OPEN cur1;

SET found = 0;

WHILE found = 0 DO

FETCH cur1 INTO nextID, nextFName, nextLName;

IF nextID = x THEN

SET result = CONCAT(nextFName, ' ', nextLName);

SET found = 1;

END IF;

END WHILE;

CLOSE cur1; \*/

END;

//

Top of Form



Bottom of Form

* [Knowledge Base](https://www.javacodegeeks.com/)
* [Resources](https://www.javacodegeeks.com/resources)
* [Deals](https://deals.javacodegeeks.com/)
* [Join Us](https://www.javacodegeeks.com/join-us)
* [About](https://www.javacodegeeks.com/about)
* [Login](https://www.javacodegeeks.com/login?redirect_to=/2018/06/mysql-stored-procedure-tutorial.html)
* [Register](https://www.javacodegeeks.com/register?redirect_to=/2018/06/mysql-stored-procedure-tutorial.html)



* [Android](https://www.javacodegeeks.com/category/android)
* [Java](https://www.javacodegeeks.com/category/java)
* [JVM Languages](https://www.javacodegeeks.com/category/jvm-languages)
* [Software Development](https://www.javacodegeeks.com/category/software-development)
* [Agile](https://www.javacodegeeks.com/category/agile)
* [Career](https://www.javacodegeeks.com/category/career)
* [Communications](https://www.javacodegeeks.com/category/communications)
* [DevOps](https://www.javacodegeeks.com/category/devops)
* [Meta JCG](https://www.javacodegeeks.com/category/meta-jcg)



[Home](https://www.javacodegeeks.com/) » [Software Development](https://www.javacodegeeks.com/category/software-development) » MySQL Stored Procedure Tutorial

About Abhishek Kothari



Abhishek is a Web Developer with diverse skills across multiple Web development technologies. During his professional career, he has worked on numerous enterprise level applications and understood the technological architecture and complexities involved in making an exceptional project. His passion to share knowledge among the community through various mediums has led him towards being a Professional Online Trainer, Youtuber as well as Technical Content Writer.

MySQL Stored Procedure Tutorial

Posted by: [Abhishek Kothari](https://www.javacodegeeks.com/author/abhishek-kothari" \o ") in [Software Development](https://www.javacodegeeks.com/category/software-development) June 26th, 2018 [0](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#respond) 1059 Views

1. Introduction

In an enterprise level application, there is always a need to perform certain specific set of tasks on the database on a regular basis like database cleanup, processing payroll, creating a new entity with several default entries and lots more. Such a task could involve execution of multiple queries for each task. This process could be eased if there was a way to group these tasks into a single task and execute it. MySQL Stored Procedure is precisely for this purpose. MySQL stored procedure is a pieced of pre-compiled SQL code which can be executed to run multiple tasks along with performing certain logical operations. A MySQL procedure is analogous to functions in a general programming language. This article covers the creation of MySQL procedures step-by-step using a practical use-case scenario.

Table Of Contents

[1. Introduction](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section1)

[2. Getting started with MySQL procedures](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section2)

[3. Creating a procedure to insert records](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section3)

[4. Dropping a procedure](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section4)

[5. Variables in procedure](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section5)

[5.1 Adding variables inside procedure](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section51)

[5.2 Adding procedure parameters](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section52)

[6. Returning a specific value from a procedure](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section6)

[7. Conditional flows in a procedure](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section7)

[8. Looping through records](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section8)

[9. Conclusion](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#section9)

2. Getting started with MySQL procedures

To begin with, let us create a simple zero action procedure. A zero action procedure is a procedure that will not really perform any database task but provide us a feedback that the procedure executed successfully. Before we get started with the code, here are the pre-requisites to proceed with the tutorial.

* MySQL server up and running
* MySQL shell running

Once you have the MySQL shell started, select the database for the tutorial. If you do not have a database, use the below commands to create and select database.

|  |  |
| --- | --- |
| 1  2 | mysql> create schema procedure\_training;  mysql> use procedure\_training; |

These commands will create the schema and select it. Now, let us understand the syntax of a simple procedure creation without variables before we create our first procedure.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | mysql> DELIMITER //  mysql> CREATE PROCEDURE PROCEDURE\_NAME()      -> begin      -> PROCEDURE\_BODY      -> end      -> // |

A procedure contains new line characters as well as semicolons in its statement. Hence, the first step before creating a procedure is to change the delimiter that decides when a statement actually ends. Hence the code DELIMITER // has been written with the goal to indicate that the statement ends only when the MySQL prompt encounters the // symbol.  
To create a procedure, start with the statement CREATE PROCEDURE PROCEDURE\_NAME() where the PROCEDURE\_NAME can be replaced with the name of your choice. The next statement contains the keyword BEGIN. It indicates that the body of the procedure begins from that point. The next part can be a single statement or multiple statement replacing the PROCEDURE\_BODY in the above code. Finally, once the procedure body is completed, put the keyword END to indicate that the procedure body ends there.

Now that we have a basic understanding of the syntax, let us create our first procedure using the code below.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | mysql> DELIMITER //  mysql> CREATE PROCEDURE myfirstprocedure()      -> begin      -> select 'This is my first procedure';      -> end      -> //  Query OK, 0 rows affected (0.06 sec) |

In the above code, we create a procedure named **myfirstprocedure**. To keep the procedure simple, all we put in the body is a single statement to display a comment. As long as the entire code is written as expected, you will see the output as shown above.

Once the query is executed successfully, you can call the procedure using a simple command shown below.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | mysql> delimiter ;  mysql> call myfirstprocedure();  +----------------------------+  | This is my first procedure |  +----------------------------+  | This is my first procedure |  +----------------------------+  1 row in set (0.00 sec)    Query OK, 0 rows affected (0.00 sec) |

As it can be seen, I first changed the delimiter back to semicolon to avoid the trouble of writing // to end the statement. Once changed, execute the call statement to execute the procedure. Since, the body just contains a comment, it is displayed as shown above. Let us now create a slightly complex procedure.

3. Creating a procedure to insert records

An insert procedure is used to insert a bunch of predefined entries on creation of a specific record. A perfect use case of it is to assign few default access rights to a user when the user is created. In order to do such an activity, we need a table to work upon. Create a table using the create statement as provided below:

|  |  |
| --- | --- |
| 1  2  3  4  5 | CREATE TABLE `procedure\_training`.`user\_roles` (    `id` INT NOT NULL,    `rolename` VARCHAR(45) NULL,    `userid` INT NULL,    PRIMARY KEY (`id`)); |

The next step is to create a procedure that insert a bunch of roles for userid value 1. This can be done using below procedure.

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | DELIMITER //   CREATE PROCEDURE insertdefaultroles()     BEGIN     insert into user\_roles values (1, 'admin',1), (2, 'employee',1),(3,'learner',1);     END //   DELIMITER ; |

The above procedure inserts 3 records in the table when executed. The output of the same has been shown below.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | mysql> call insertdefaultroles();    Query OK, 3 rows affected (0.14 sec)    mysql> select \* from user\_roles;  +----+----------+--------+  | id | rolename | userid |  +----+----------+--------+  |  1 | admin    |      1 |  |  2 | employee |      1 |  |  3 | learner  |      1 |  +----+----------+--------+  3 rows in set (0.01 sec) |

4. Dropping a procedure

When you need to re-create the procedure, just like tables, you need to drop the procedure before creating it again. In an alternate database like Oracle DB, there is support to create or replace a procedure by using CREATE OR REPLACE instead of the plain CREATE. However, in MySQL, it is mandatory to delete the procedure before you could re-create it. As we proceed in the tutorial, we would require to drop the procedure frequently. The query to drop a procedure is similar to the DROPquery for MySQL tables. A query to drop the procedure **insertdefaultroles** has been shown below.

|  |  |
| --- | --- |
| 1 | mysql> DROP PROCEDURE insertdefaultroles; |

5. Variables in procedure

As you must have noticed in the above code, we supplied static user id and primary key value. This renders the procedure unusable after it has been used once. Hence, we need a way out to automate these values. Let us begin with the value of primary key first.

5.1 Adding variables inside procedure

The first step to use a variable is to declare it. Once it has been declared, we need to get the maximum value of primary key variable currently in the table. The code below does precisely the same.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | DROP PROCEDURE insertdefaultroles;  DELIMITER //   CREATE PROCEDURE insertdefaultroles()     BEGIN     DECLARE pkvalue int;     select max(id) into pkvalue from user\_roles;     insert into user\_roles values (pkvalue+1, 'admin',1), (pkvalue+2, 'employee',1),(pkvalue+3,'learner',1);     END //   DELIMITER; |

The code contains a drop procedure statement to drop the procedure before recreating it. If you do not do so, you would encounter an error mentioning that the procedure already exists. Once you are done with the execution of the code, call the procedure once again to view the results. The results would should that new entries have been added to the table with increasing value of primary keys.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | mysql> call insertdefaultroles();  Query OK, 3 rows affected (0.07 sec)  mysql> select \* from user\_roles;  +----+----------+--------+  | id | rolename | userid |  +----+----------+--------+  |  1 | admin    |      1 |  |  2 | employee |      1 |  |  3 | learner  |      1 |  |  4 | admin    |      1 |  |  5 | employee |      1 |  |  6 | learner  |      1 |  +----+----------+--------+  6 rows in set (0.00 sec) |

5.2 Adding procedure parameters

Although the trouble of dynamic primary key has been resolved, the user id is still static. The user id cannot be retrieved from the database like the primary key. Hence, the user id needs to be passed to the procedure in some way. Procedure allows function parameters for this.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | DROP PROCEDURE insertdefaultroles;  DELIMITER //   CREATE PROCEDURE insertdefaultroles(uid int)     BEGIN     DECLARE pkvalue int ;     select max(id) into pkvalue from user\_roles;     insert into user\_roles values (pkvalue+1, 'admin',uid), (pkvalue+2, 'employee',uid),(pkvalue+3,'learner',uid);     END //  DELIMITER ; |

The above code declares one such procedure. This procedure takes the user id in the variable uid. This variable can be used to replace the value of user id in the query. Execute the procedure and ensure that the execution is successful.

To run the procedure, pass the user id just like you pass it in a normal function. The code below shows the execution and results of the execution too.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18 | mysql> call insertdefaultroles(2);  Query OK, 3 rows affected (0.10 sec)    mysql> select \* from user\_roles;  +----+----------+--------+  | id | rolename | userid |  +----+----------+--------+  |  1 | admin    |      1 |  |  2 | employee |      1 |  |  3 | learner  |      1 |  |  4 | admin    |      1 |  |  5 | employee |      1 |  |  6 | learner  |      1 |  |  7 | admin    |      2 |  |  8 | employee |      2 |  |  9 | learner  |      2 |  +----+----------+--------+  9 rows in set (0.01 sec) |

As it can be seen, the new user id 2 was passed in the procedure parameter and the same has been replaced in the query. The procedure call now inserts the default roles for the user id 2.

6. Returning a specific value from a procedure

Until now, we either returned the results of procedure execution or returned a comment by executing a query. Now the need comes to perform a certain calculation and return the evaluated value. For instance, let us count the number of user having a specific role. In order to count the users, you would need to search the users by a specific role and get a count of records. In order to do so, execute the below query.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | $ mysql> select count(\*) from user\_roles where rolename = 'admin';  +----------+  | count(\*) |  +----------+  |        3 |  +----------+  1 row in set (0.00 sec) |

The query outputs the number of users as it can be seen above. This count of users can be returned as a value from a procedure. In order to do so, you need to select the value in a similar way as the comment. The below code shows such an implementation for you.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | DROP PROCEDURE countusers;  DELIMITER //   CREATE PROCEDURE countusers(role varchar(45))     BEGIN     DECLARE usercount int ;     select count(\*) into usercount from user\_roles where rolename = role;     select usercount;     END //   DELIMITER ; |

The above code creates a procedure that takes the value of count into a variable usercount and selects it to return the related value. As it can be noticed, the output of the first query is not displayed as it is overshadowed by the query that follows. In a procedure, the output is always dependent on the query that is executed at the end unless a deliberate select of a variable is executed in between.

7. Conditional flows in a procedure

So far, we have seen simple use cases of procedure. These use cases involve queries that are not really complicated. In this section, we would proceed with branching the flow of procedure conditionally. This is where the real application of procedure comes into picture. Conditional flows allow you to take decision on what operation needs to be done based on a query or a variable that is passed. Before we get into write a procedure containing a case, let us understand its syntax. The below code snippet shows the procedure syntax for a procedure containing cases.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE PROCEDURE procedure\_name  BEGIN  CASE case\_value      WHEN expression THEN queries\_to\_be\_executed      [WHEN expression THEN queries\_to\_be\_executed] ...      [WHEN expression THEN queries\_to\_be\_executed] ...      [ELSE queries\_to\_be\_executed]  END CASE  END |

In the above syntax, the items mentioned in square braces are optional. In the above procedure, WHEN indicates the starting of an if condition. The keyword CASE signifies the beginning of a conditional branching. The expression statement could be any variable or an expression being evaluated. THEN is used to indicate the beginning of body if the expression evaluates to true.

The rest of the statements are optional. They are used to add more if conditions and an else condition in case none of the provided expressions are evaluated to true. Now, proceeding towards writing an actual procedure. We would use simple body statements to understand the conditional branching well. In general, you could use any SQL query statements in the body.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | CREATE PROCEDURE firstcaseprocedure(variable varchar(10))  BEGIN  CASE variable      WHEN 'first' THEN select 'First condition executed';      WHEN 'second'  THEN select 'Second condition executed';      WHEN 'third'  THEN select 'Third condition executed';      ELSE select 'Else condition executed';  END CASE;  END; |

In the above code, carefully check the semicolons and single quotes placed. They are the major cause of errors if at all you encounter one. Once the procedure is created, test the procedure execution for each test case. The testing of the procedure has been shown below.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39 | mysql> call firstcaseprocedure('first');  +--------------------------+  | First condition executed |  +--------------------------+  | First condition executed |  +--------------------------+  1 row in set (0.00 sec)    Query OK, 0 rows affected (0.00 sec)    mysql> call firstcaseprocedure('second');  +---------------------------+  | Second condition executed |  +---------------------------+  | Second condition executed |  +---------------------------+  1 row in set (0.00 sec)    Query OK, 0 rows affected (0.00 sec)    mysql> call firstcaseprocedure('third');  +--------------------------+  | Third condition executed |  +--------------------------+  | Third condition executed |  +--------------------------+  1 row in set (0.00 sec)    Query OK, 0 rows affected (0.00 sec)    mysql> call firstcaseprocedure('xyz');  +-------------------------+  | Else condition executed |  +-------------------------+  | Else condition executed |  +-------------------------+  1 row in set (0.00 sec)    Query OK, 0 rows affected (0.00 sec) |

As it can be seen in the above execution results, the output obtained is conditional based on the supplied argument. Although we supplied the argument, it is also possible to define the argument within the procedure. For instance, consider a procedure that updates whether a student passed or failed. In such a procedure, the procedure dynamically retrieves the marks and check whether the student passed or fail and updates the record accordingly.

A sample procedure with a similar implementation is shown below. The table has been assumed to be existing as the example is just to clarify a use case.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11 | CREATE PROCEDURE passorfail(sid int)  BEGIN  DECLARE marks int;  select student\_marks into marks from student\_result where student\_id = sid;  CASE      WHEN marks<35 THEN select 'Fail';      WHEN marks<50 THEN select 'C Grade';      WHEN marks<70 THEN select 'B Grade';      ELSE select 'A Grade';  END CASE;  END; |

Here, notice the difference in the case carefully. The case variable is no longer common between the statements. Whenever we need to use comparison or logical operators, we need to use it the way it is shown above. The condition marks<35 can be replaced by any relevant condition involving a different variable too.

In the above code, the flow control takes place using CASE statement. However, often we are more comfortable using IF…. ELSE for its simplicity in using the expressions. Unlike CASE, where you are allowed to evaluate only one specific variable in the condition, IF…ELSE allows you to check for multiple variables with different combinations. Let us understand the syntax of an IF…ELSE statement in procedure.

|  |  |
| --- | --- |
| 1  2  3  4 | IF search\_condition THEN query\_statements      [ELSEIF search\_expression THEN query\_statements] ...      [ELSE statement\_list]  END IF |

The above syntax can be updated and used in any procedure body. The statements in the square braces are optional and can be avoided if not required. The **query\_statements** part can contains single or multiple SQL conditions to be executed if the search\_condition evaluates to true. The above CASE…WHEN statement is converted into IF…ELSE in the code below.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10 | CREATE PROCEDURE passorfail(sid int)  BEGIN  DECLARE marks int;  select student\_marks into marks from student\_result where student\_id = sid;  IF marks<35 THEN select 'Fail';  ELSEIF marks<50 THEN select 'C Grade';  ELSEIF marks<70 THEN select 'B Grade';  ELSE select 'A Grade';  END IF  END; |

Observe the amount of simplicity it introduces. It just gets easier to understand conditions for even a layman.

8. Looping through records

Notice that in the above use case of student marks, all we did is print the pass grade of a student based on the supplied id. A procedure as discussed before is used to do large pre-defined tasks. Hence, it would be more useful if one could update the grade of each student by just calling the procedure. In order to do so, let us create a table and add few details to it. Execute the below queries in order to do the same.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18 | mysql> create table student\_marks (student\_id int, marks int, grade varchar(5));  Query OK, 0 rows affected (0.12 sec)    mysql> insert into student\_marks values (1, 50,null),(2,60,null),(3,34,null),(4,70,null),(5,43,null);  Query OK, 5 rows affected (0.07 sec)  Records: 5  Duplicates: 0  Warnings: 0    mysql> select \* from student\_marks;  +------------+-------+-------+  | student\_id | marks | grade |  +------------+-------+-------+  |          1 |    50 | NULL  |  |          2 |    60 | NULL  |  |          3 |    34 | NULL  |  |          4 |    70 | NULL  |  |          5 |    43 | NULL  |  +------------+-------+-------+  5 rows in set (0.01 sec) |

Now, we would create a procedure that checks the marks of each student and assigns a grade to each student. Before proceeding, let us understand the syntax and terminologies. The code snippet shows the syntax of the procedure containing a loop.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | CREATE PROCEDURE curdemo()  BEGIN    //Declaration of variables    //Open cursors      read\_loop: LOOP      //FETCH Cursors      //Loop through cursors      IF done THEN        LEAVE read\_loop;      END IF;      //BODY      END IF;    END LOOP;    //Close cursors  END; |

In the above syntax, the term to be noted is **Cursor**. A cursor is similar to a pointer to a row in the database. You could assign a result set to a cursor. Once assigned, the cursor can help you iterate through the records one by one. The read\_loop label is used to continue an iteration from a specific label. Thus, when you write read\_loop:LOOP, the code keep rotating until it encounters the statement LEAVE read\_loop. Now that we have understood the syntax, let us write our first procedure with an iteration in it.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18 | CREATE PROCEDURE iterdemo()  BEGIN    DECLARE done INT DEFAULT FALSE;    DECLARE sid int;    DECLARE mks int;    DECLARE cur1 CURSOR FOR select student\_id,marks from student\_mks;    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;    OPEN cur1;      read\_loop: LOOP      FETCH cur1 INTO sid, mks;      IF  done THEN LEAVE read\_loop; END IF;      IF mks<35 THEN update student\_marks set  grade ='Fail' where student\_id = sid;      ELSE update student\_marks set  grade ='Pass' where student\_id = sid;      END IF;    END LOOP;    CLOSE cur1;  END; |

The above procedure assigns the result status of a student – whether the student has passed the exam or failed based on the scored marks. The procedure executes a loop around the records of table student\_marks. The cursor basically points to the records that are selected. As it can be seen in the above code, the cursor cur1 points to all the records of student\_marks table. On executing the loop read\_loop, the cur1 cursor is used to fetch data row by row. The statement FETCH cur1 into sid,mks puts the data of current row into the variables and moves the cursor to point to the next row. Thus, each row is being processed for the if condition provided in the body. The if condition checks for the marks and decides whether the student passed or failed the exam. Accordingly the rows of the table will be updated. On calling the procedure, you can check the results to be similar to the one shown below:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15 | mysql> delimiter ;  mysql> call iterdemo();  Query OK, 0 rows affected (0.01 sec)    mysql> select \* from student\_marks;  +------------+-------+-------+  | student\_id | marks | grade |  +------------+-------+-------+  |          1 |    50 | Pass  |  |          2 |    60 | Pass  |  |          3 |    34 | Fail  |  |          4 |    70 | Pass  |  |          5 |    43 | Pass  |  +------------+-------+-------+  5 rows in set (0.00 sec) |

As it can be seen, only the grade for student with marks below 35 is updated to fail. We can also use an IF ELSE ladder to update the exact grade of the student. The below code does the same.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | mysql> drop procedure iterdemo;  Query OK, 0 rows affected (0.12 sec)  mysql> DELIMITER //  mysql>  mysql> CREATE PROCEDURE iterdemo()      -> BEGIN      ->   DECLARE done INT DEFAULT FALSE;      ->   DECLARE sid int;      ->   DECLARE mks int;      ->   DECLARE cur1 CURSOR FOR select student\_id,marks from student\_marks;      ->   DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;      ->   OPEN cur1;      ->   read\_loop: LOOP      ->     FETCH cur1 INTO sid, mks;      ->     IF  done THEN LEAVE read\_loop; END IF;      ->     IF mks ELSEIF mks ELSEIF mks ELSE update student\_marks set  grade ='A' where student\_id = sid;      -> END IF;      ->   END LOOP;      ->   CLOSE cur1;      -> END;//  Query OK, 0 rows affected (0.02 sec)  mysql> delimiter ;  mysql> call iterdemo;  Query OK, 0 rows affected (0.05 sec)  mysql> select \* from student\_marks;  +------------+-------+-------+  | student\_id | marks | grade |  +------------+-------+-------+  |          1 |    50 | B     |  |          2 |    60 | B     |  |          3 |    34 | Fail  |  |          4 |    70 | A     |  |          5 |    43 | C     |  +------------+-------+-------+  5 rows in set (0.00 sec) |

The above code processes the marks value for multiple IF condition and updates the grade accordingly. This is precisely how a procedure can help in doing major tasks with simplicity.

9. Conclusion

To conclude, procedure is a set of rules and queries to be executed in order to perform database intensive tasks with simplicity. With the procedures, you can execute normal SQL queries, branch the query flow conditionally and iterate through a list of records and do mass update when necessary.

Tagged with: [MYSQL](https://www.javacodegeeks.com/tag/mysql) [MYSQL SERVER](https://www.javacodegeeks.com/tag/mysql-server)

 (***+1****rating,****1****votes*)  
*You need to be a registered member to rate this.* [Start the discussion](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html#respond) 1059 Views [Tweet it!](https://twitter.com/share?url=https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html&via=javacodegeeks&text=MySQL%20Stored%20Procedure%20Tutorial)

Do you want to know how to develop your skillset to become a Java Rockstar?

Subscribe to our newsletter to start Rocking right now!

To get you started we give you our best selling eBooks for FREE!

**1.** JPA Mini Book

**2.** JVM Troubleshooting Guide

**3.** JUnit Tutorial for Unit Testing

**4.** Java Annotations Tutorial

**5.** Java Interview Questions

**6.** Spring Interview Questions

**7.** Android UI Design

and many more ....



I agree to the [Terms](https://www.javacodegeeks.com/about/terms-of-use)and [Privacy Policy](https://www.javacodegeeks.com/about/privacy-policy)

[**Sign up**](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html)

LIKE THIS ARTICLE? READ MORE FROM JAVA CODE GEEKS

Leave a Reply

Top of Form

avatar



Bottom of Form

This site uses Akismet to reduce spam. [Learn how your comment data is processed](https://akismet.com/privacy/" \t "_blank).

  Subscribe

NEWSLETTER

Insiders are already enjoying weekly updates and complimentary whitepapers!

**Join them now** to gain exclusive access to the latest news in the Java world, as well as insights about Android, Scala, Groovy and other related technologies.



I agree to the [Terms](https://www.javacodegeeks.com/about/terms-of-use" \t "_blank)and [Privacy Policy](https://www.javacodegeeks.com/about/privacy-policy" \t "_blank)

**[Sign up](https://www.javacodegeeks.com/2018/06/mysql-stored-procedure-tutorial.html)**

JOIN US

With 1,240,600 monthly unique visitors and over 500 authors we are placed among the top Java related sites around. Constantly being on the lookout for partners; we encourage you to join us. So If you have a blog with unique and interesting content then you should check out our **[JCG](https://www.javacodegeeks.com/join-us/jcg)** partners program. You can also be a **[guest writer](https://www.javacodegeeks.com/join-us/w4g)** for Java Code Geeks and hone your writing skills!

KNOWLEDGE BASE

* [Courses](https://www.javacodegeeks.com/courses" \t "_blank)
* [Examples](https://examples.javacodegeeks.com/" \t "_blank)
* [Minibooks](https://www.javacodegeeks.com/minibook" \t "_blank)
* [Resources](https://www.javacodegeeks.com/resources" \t "_blank)
* [Tutorials](https://www.javacodegeeks.com/tutorials" \t "_blank)

PARTNERS

* [Mkyong](https://www.mkyong.com/" \t "_blank)

THE CODE GEEKS NETWORK

* [.NET Code Geeks](https://www.dotnetcodegeeks.com/" \t "_blank)
* [Java Code Geeks](https://www.javacodegeeks.com/" \t "_blank)
* [System Code Geeks](https://www.systemcodegeeks.com/" \t "_blank)
* [Web Code Geeks](https://www.webcodegeeks.com/" \t "_blank)

HALL OF FAME

* [“Android Full Application Tutorial” series](https://www.javacodegeeks.com/2010/10/android-full-application-tutorial.html" \t "_blank)
* [11 Online Learning websites that you should check out](https://www.javacodegeeks.com/2013/01/15-online-learning-websites-that-you-should-check-out.html" \t "_blank)
* [Advantages and Disadvantages of Cloud Computing – Cloud computing pros and cons](https://www.javacodegeeks.com/2013/04/advantages-and-disadvantages-of-cloud-computing-cloud-computing-pros-and-cons.html" \t "_blank)
* [Android Google Maps Tutorial](https://www.javacodegeeks.com/2011/02/android-google-maps-tutorial.html" \t "_blank)
* [Android JSON Parsing with Gson Tutorial](https://www.javacodegeeks.com/2011/01/android-json-parsing-gson-tutorial.html" \t "_blank)
* [Android Location Based Services Application – GPS location](https://www.javacodegeeks.com/2010/09/android-location-based-services.html" \t "_blank)
* [Android Quick Preferences Tutorial](https://www.javacodegeeks.com/2011/01/android-quick-preferences-tutorial.html" \t "_blank)
* [Difference between Comparator and Comparable in Java](https://www.javacodegeeks.com/2013/03/difference-between-comparator-and-comparable-in-java.html" \t "_blank)
* [GWT 2 Spring 3 JPA 2 Hibernate 3.5 Tutorial](https://www.javacodegeeks.com/2010/05/gwt-2-spring-3-jpa-2-hibernate-35-2.html" \t "_blank)
* [Java Best Practices – Vector vs ArrayList vs HashSet](https://www.javacodegeeks.com/2010/08/java-best-practices-vector-arraylist.html" \t "_blank)

ABOUT JAVA CODE GEEKS

JCGs (Java Code Geeks) is an independent online community focused on creating the ultimate Java to Java developers resource center; targeted at the technical architect, technical team lead (senior developer), project manager and junior developers alike. JCGs serve the Java, SOA, Agile and Telecom communities with daily news written by domain experts, articles, tutorials, reviews, announcements, code snippets and open source projects.

DISCLAIMER

All trademarks and registered trademarks appearing on Java Code Geeks are the property of their respective owners. Java is a trademark or registered trademark of Oracle Corporation in the United States and other countries. Examples Java Code Geeks is not connected to Oracle Corporation and is not sponsored by Oracle Corporation.

Java Code Geeks and all content copyright © 2010-2019, [Exelixis Media P.C.](https://www.exelixismedia.com/) | [Terms of Use](https://www.javacodegeeks.com/about/terms-of-use) | [Privacy Policy](https://www.javacodegeeks.com/about/privacy-policy) | [Contact](mailto:support@javacodegeeks.com)