

STORED PROCEDURE



STORED PROCEDURE

- It is a subroutine like a subprogram in a regular computing language, stored in database.
- It has a name, a parameter list, and SQL statement(s).
- Stored procedures are invoked using the CALL statement.

WHY STORED PROCEDURES?

- Stored procedures are fast.
 - MySQL server do cache the data. Repetitive task that requires checking, looping, multiple statements, do it with a single call to a procedure that's stored on the server.
- Stored procedures are portable.
 - It runs on every platform that MySQL runs on, without obliging you to install an additional runtime-environment package.
- Stored procedures are always available as 'source code' in the database itself.
And it makes sense to link the data with the processes that operate on the data.

CREATE PROCEDURE

- By default, a procedure is associated with the default database (currently used database).
- To associate the procedure with a given database, specify the name as `database_name.stored_procedure_name` when you create it.
- Before creating stored procedure
 - Check database version: `select version();`
 - Check the privileges assigned: `show privileges;`
 - `CREATE PROCEDURE`, `CREATE FUNCTION` require the `CREATE ROUTINE` privilege.
 - Pick a delimiter

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.
- This causes problem in stored procedure because a procedure can have many statements, and everyone must end with a semicolon.
- Pick a string which is rarely occur within statement or within procedure.
- You can use whatever you want.

DELIMITER CONT...

- Here, double dollar sign i.e. \$\$ is used as a delimiter.
 - DELIMITER \$\$
- Now, the default delimiter is \$\$
 - Select * from table_name \$\$
- Now execute the following command to resume ";" as a delimiter :
 - DELIMITER ;

```
225 delimiter $$
226 • SELECT * FROM user $$
227 DELIMITER ;
```



Result Grid



Filter Rows:

	userid	password	name
▶	scott123	123@sco	Scott
	ferp6734	dloeu@&3	Palash
	diana094	ku\$j@23	Diana

CREATE PROCEDURE

- CREATE PROCEDURE command creates the stored procedure.
- Next part is the procedure name (here, np).
- Parentheses, "()" holds the parameter(s) list as there are no parameters in this procedure
- Here, \$\$ is a real statement ender.

```
229 Delimiter $$
230 • CREATE PROCEDURE np()
231 BEGIN
232     select * from user;
233 END $$
234
235 Delimiter ;
236 • call np();
```

< **Result Grid** | Filter Rows:

	userid	password	name
▶	scott123	123@sco	Scott
	ferp6734	dloeu@&3	Palash
	diana094	ku\$j@23	Diana

CREATE PROCEDURE THROUGH WORKBENCH TOOL

File Edit View Query Database Server Tools Scripting Help



Navigator

SQL File 10* HR_Script.sql* mysqlsampledatabase lab 10* new_procedure - Routine x

SCHEMAS

Filter objects

- ecse2111_lab1
- er2relational
- example
- hr
 - Tables
 - Views
 - Stored Procedures
 - ABC
 - emp
 - loc_variable
 - new_Proc
 - new_procedure
 - XYZ_
 - Functions
- lab3
- lab4eb06
- lab7
- labeb05
- lahch06

Create Stored Procedure...

Refresh All

Administration Schemas

Information

Schema: hr

Routine

Name: new_procedure

The name of the routine is parsed automatically from the DDL statement. The DDL is parsed automatically while you type.

DDL:



```
1 • CREATE PROCEDURE `np1` ()  
2 BEGIN  
3  
4 END  
5
```

Click on Apply button

Apply

Revert

Apply SQL Script to Database

Navigator

SCHEMAS

Filter objects

- ecse211_lab1
- er2relational
- example
- hr
 - Tables
 - Views
 - Stored Procedures
 - ABC
 - emp
 - loc_variable
 - new_Proc
 - new_procedure
 - XYZ_
 - Functions
- lab3
- lab4eb06
- lab7
- lab
- lab
- Administr

Schema: hr

Stored Procedure (np1) is not created yet

Review SQL Script

Apply SQL Script

Review the SQL Script to be Applied on the Database

Online DDL

Algorithm:

Default

Lock Type:

Default

```
1  USE `hr`;  
2  DROP procedure IF EXISTS `np1`;  
3  
4  DELIMITER $$  
5  USE `hr`$$  
6  CREATE PROCEDURE `np1` ()  
7  BEGIN  
8    select * from user;  
9  END$$  
10  
11  DELIMITER ;  
12  
13
```

SQL Query

Click on

Back

Apply

Cancel

Apply

Revert

Navigator

SCHEMAS

Filter objects

- ecse211l_lab1
- er2relational
- example
- hr
 - Tables
 - Views
 - Stored Procedures
 - ABC
 - emp
 - loc_variable
 - new_Proc
 - new_procedure
 - np1
 - XYZ_
 - Functions
- lab3
- lab4eb06
- lab7
- laheh05

Administration Schemas

Information

Schema: hr

Apply SQL Script to Database

Review SQL Script

Apply SQL Script

Applying SQL script to the database

The following tasks will now be executed. Please monitor the execution. Press Show Logs to see the execution logs.

☒ Execute SQL Statements

SQL script was successfully applied to the database.

Show Logs

Back

Finish

Cancel

Now, np1 is created

Apply

Revert

Filter objects

- ecse211l_lab1
- er2relational
- example
- hr
 - Tables
 - Views
 - Stored Procedures
 - ABC
 - emp
 - loc_variable
 - new_Proc
 - new_procedure
 - np1
 - XYZ_
 - Functions
- lab3
- lab4eb06
- lab7
- laheh05

Administration Schemas

Information

Schema: hr

DDL:



```
1 CREATE DEFINER='root'@'localhost' PROCEDURE `np1`()
2 BEGIN
3   select * from user;
4 END
```

After clicking on Finish button. You can do changes here and click on apply changes

Apply

Revert

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 103	16:56:30	call np()	3 row(s) returned	0.000 sec / 0.000 sec
✓ 104	17:19:18	Apply changes to np1	Changes applied	

Object Info Session

Local instance MySQL Router x

File Edit View Query Database Server Tools Scripting Help

Navigator

SCHEMAS

Filter objects

- ecse211l_lab1
- er2relational
- example
- hr**
 - Tables
 - Views
 - Stored Procedures
 - ABC
 - emp
 - loc_variable
 - new_Proc
 - new_procedure
 - np1**
 - XYZ_
 - Functions
- lab3
- lab4eb06
- lab7
- laheh05

Administration Schemas

Information

SQL File 10* HR_Script.sql* mysqlsampledatabase lab 10* np1 - Routine np1 - Routine **np1**

Limit to 1000 rows

```
1 call hr.np1();
2
```

Stored procedure is called

Click

This window will open




Result Grid Filter Rows: Export: Wrap Cell Content:

	userid	password	name
▶	scott123	123@sco	Scott
	ferp6734	dloeu@&3	Palash
	diana094	ku\$j@23	Diana

Procedure: **np1**

SHOW PROCEDURE

239 • `show create procedure np1;`

Result Grid  Filter Rows: <input data-bbox="410 654 677 736" type="text"/> Export:  Wrap Cell Content: 					
Procedure	sql_mode	Create Procedure	character_set_client	collation_connection	Database Collation
np1	STRICT_TRANS_TABLES,NO_ENGINE_SUBSTIT...	CREATE DEFINER='root' @'localhost' PROCE...	utf8mb4	utf8mb4_0900_ai_ci	utf8mb4_0900_ai_ci

Declare Statement

- It is used to define various items local to a program
 - E.g., local variables, conditions, handler and cursors.
- It is used only inside a BEGIN ... END statement.
- Declarations follow the following order :
 - Cursor declarations must appear before handler declarations.
 - Variable and condition declarations must appear before cursor or handler declarations.

Variables in Stored Programs

- System variables and user-defined variables can be used in stored programs (SP).
- SP uses DECLARE to define local variables
- **Declare a Variable:**
 - DECLARE var_name [, var_name] ... type [DEFAULT value]
- To provide a default value for a variable, include a DEFAULT clause.
- If the DEFAULT clause is missing, the initial value is NULL.

LOCAL VARIABLES AND GLOBAL VARIABLES

```
240 DELIMITER $$
241 • CREATE PROCEDURE Local_Var()
242   BEGIN /* declare local variables */
243     DECLARE a INT DEFAULT 10;
244     DECLARE b, c INT; /* using the local variables */
245     SET a = a + 100;
246     SET b = 2;
247     SET c = a + b;
248     BEGIN /* local variable in nested block */
249       DECLARE c INT;
250       SET c = 5;
251       SELECT a, b, c;
252     END;
253   END$$
254   delimiter ;
255 • call Local_Var();
```

```
240 DELIMITER $$
241 • CREATE PROCEDURE Local_Var1()
242   BEGIN /* declare local variables */
243     DECLARE a INT DEFAULT 10;
244     DECLARE b, c INT; /* using the local variables */
245     SET a = a + 100;
246     SET b = 2;
247     SET c = a + b;
248     BEGIN /* local variable in nested block */
249       DECLARE c INT;
250       SET c = 5;
251     END;
252     SELECT a, b, c;
253   END$$
254   delimiter ;
255 • call Local_Var1();
```

LOCAL VARIABLES AND GLOBAL VARIABLES

```
240 DELIMITER $$
241 • CREATE PROCEDURE Local_Var()
242 BEGIN /* declare local variables */
243 DECLARE a INT DEFAULT 10;
244 DECLARE b, c INT; /* using the local variables */
245 SET a = a + 100;
246 SET b = 2;
247 SET c = a + b;
248 BEGIN /* local variable in nested block */
249 DECLARE c INT;
250 SET c = 5;
251 SELECT a, b, c;
252 END;
253 END$$
254 delimiter ;
255 • call Local_Var();
```

<  Filter Rows: | Export:  | Wrap Cell Content: 

	a	b	c
▶	110	2	5

```
240 DELIMITER $$
241 • CREATE PROCEDURE Local_Var1()
242 BEGIN /* declare local variables */
243 DECLARE a INT DEFAULT 10;
244 DECLARE b, c INT; /* using the local variables */
245 SET a = a + 100;
246 SET b = 2;
247 SET c = a + b;
248 BEGIN /* local variable in nested block */
249 DECLARE c INT;
250 SET c = 5;
251 END;
252 SELECT a, b, c;
253 END$$
254 Delimiter ;
255 • call Local_Var1();
```

<  Filter Rows: | Export:  | Wrap Cell Content: 

	a	b	c
▶	110	2	112

USER VARIABLES

```
257 DELIMITER $$
258 • CREATE PROCEDURE User_Variables()
259   BEGIN
260     SET @x = 15;
261     SET @y = 5;
262     SELECT @x, @y, @x-@y;
263   END$$
264 Delimiter ;
265 • call User_Variables();
```

<

Result Grid | Filter Rows: | Export:  | Wrap Cell Content: 

	@x	@y	@x-@y
▶	15	5	10

PROCEDURE PARAMETERS

- CREATE PROCEDURE statement in the following ways :
 - **CREATE PROCEDURE sp_name ()** → the parameter list is empty.
 - **CREATE PROCEDURE sp_name ([IN] param_name type)** → IN parameter passes a value into a procedure but the modification is not visible to the caller when the procedure returns.
 - **CREATE PROCEDURE sp_name ([OUT] param_name type)** → OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns.
 - **CREATE PROCEDURE sp_name ([INOUT] param_name type)** → INOUT parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.
- In a procedure, each parameter is an IN parameter by default.

PARAMETER 'IN' EXAMPLE

- IN parameter name → 'var1'
- Type →
- The SELECT statement fetches rows from 'employees' table and the number of rows is limited by the user.

```
269 delimiter $$
270 • CREATE PROCEDURE my_proc_IN (IN var1 INT)
271 BEGIN
272     SELECT * FROM locations LIMIT var1;
273 END$$
274 delimiter ;
275 • call my_proc_IN(5);
```

Result Grid



Filter Rows:

Export:



Wrap Cell Content:

	LOCATION_ID	STREET_ADDRESS	POSTAL_CODE	CITY	STATE_PROVINCE	COUNTRY_ID
▶	1000	1297 Via Cola di Rie	989	Roma		IT
	1100	93091 Calle della Testa	10934	Venice		IT
	1200	2017 Shinjuku-ku	1689	Tokyo	Tokyo Prefecture	JP
	1300	9450 Kamiya-cho	6823	Hiroshima		JP
	1400	2014 Jabberwocky Rd	26192	Southlake	Texas	US

PARAMETER 'OUT' EXAMPLE

```
277 delimiter $$
278 • CREATE PROCEDURE my_proc_OUT (OUT average_salary INT)
279 • BEGIN
280 •   SELECT avg(SALARY) INTO average_salary FROM employees;
281 • END$$
282 delimiter ;
283 • call my_proc_OUT(@avg_sal);
284 • select @avg_sal;
```

User Variable

Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

@avg_sal
6462


```
277 delimiter $$
278 • CREATE PROCEDURE proc_OUT (OUT average_salary INT, OUT highest_salary INT)
279 • BEGIN
280 •   SELECT avg(SALARY) INTO average_salary FROM employees;
281 •   SELECT MAX(MAX_SALARY) INTO highest_salary FROM JOBS;
282 • END$$
283 delimiter ;
284 • call proc_OUT(@avg_sal, @max_sal);
285 • select @avg_sal, @max_sal;
```




Result Grid | Filter Rows: | Export: | Wrap Cell Content: |

@avg_sal	@max_sal
6462	40000

PARAMETER 'INOUT' EXAMPLE

```
296 delimiter $$
297 • CREATE PROCEDURE INOUT_eg (INOUT EMP_ID int, IN given_sal decimal(8,2))
298 BEGIN
299     SELECT count(EMPLOYEE_ID) INTO EMP_ID FROM employees WHERE salary <= given_sal;
300 END$$
301 • call INOUT_eg(@sal, '24000.00');
302 select @sal;
303
```

< 

Result Grid |  Filter Rows: | Export:  | Wrap Cell Content: 

	@sal
▶	107

FLOW CONTROL STATEMENTS: IF STATEMENT

```
308 delimiter $$
309 • create PROCEDURE GetUserName(INOUT user_name varchar(16), IN user_id varchar(16)
310   BEGIN
311     DECLARE uname varchar(16);
312     SELECT name INTO uname FROM user WHERE userid = user_id;
313     IF user_id = "scott123" THEN SET user_name = "Scott";
314     ELSEIF user_id = "ferp6734" THEN SET user_name = "Palash";
315     ELSEIF user_id = "diana094" THEN SET user_name = "Diana";
316   END IF;
317 END $$
318 delimiter ;
319 • CALL GetUserName(@A, 'ferp6734');
320 • select @A;
```

< 

Result Grid   Filter Rows: Export:  Wrap Cell Content: 

	@A
▶	Palash

FLOW CONTROL STATEMENTS: CASE STATEMENT

```
323 DELIMITER $$
324 • CREATE PROCEDURE proc_CASE_eg (INOUT no_employees INT, IN salary INT)
325 BEGIN
326 CASE
327 WHEN (salary > 5000) THEN (SELECT COUNT(job_id) INTO no_employees FROM jobs WHERE min_salary > 5000);
328 WHEN (salary < 5000) THEN (SELECT COUNT(job_id) INTO no_employees FROM jobs WHERE min_salary < 5000);
329 ELSE (SELECT COUNT(job_id) INTO no_employees FROM jobs WHERE min_salary = 5000);
330 END CASE;
331 END$$
332 delimiter ;
333 • call proc_CASE_eg (@no_employees, 6000);
334 • select @no_employees;
335
```

Result Grid			Filter Rows: <input type="text"/>	Export:	Wrap Cell Content:
@no_employees					
9					

FLOW CONTROL STATEMENTS: LOOP STATEMENT

```
338 DELIMITER $$
339 • CREATE PROCEDURE proc_LOOP_ex (IN num INT)
340 • BEGIN
341 • DECLARE x INT;
342 • SET x = 0;
343 • loop_label: LOOP
344 • INSERT INTO number1 VALUES (rand());
345 • SET x = x + 1;
346 • IF x >= num THEN
347 • LEAVE loop_label;
348 • END IF;
349 • END LOOP;
350 • END$$
351 delimiter ;
352 • call proc_LOOP_ex(5);
353 • select * from number1;
```

Result Grid | | Filter Rows: | Export: | Wrap Cell Content:

rand_number
0.2918747
0.4500370
0.3745611
0.5226943
0.4897882

FLOW CONTROL STATEMENT: WHILE STATEMENT

```
357 DELIMITER $$
358 • CREATE PROCEDURE proc_WHILE(IN n INT)
359   BEGIN
360     SET @sum = 0;
361     SET @x = 1;
362     WHILE @x < n
363     DO
364       IF mod(@x, 2) <> 0 THEN
365         SET @sum = @sum + @x;
366       END IF;
367       SET @x = @x + 1;
368     END WHILE;
369   END$$
370 delimiter ;
371 • call proc_WHILE(10);
372 • select @sum, @x;
```

Result Grid

Filter Rows:

Export:

Wrap Cell Content:

	@sum	@x
▶	25	10

DROP PROCEDURE

- You can drop a procedure by writing this command:
 - Drop procedure procedure_name;
- You can check the existence of a procedure
 - SHOW CREATE PROCEDURE procedure_name