STORED PROCEDURE

STORED PROCEDURE

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

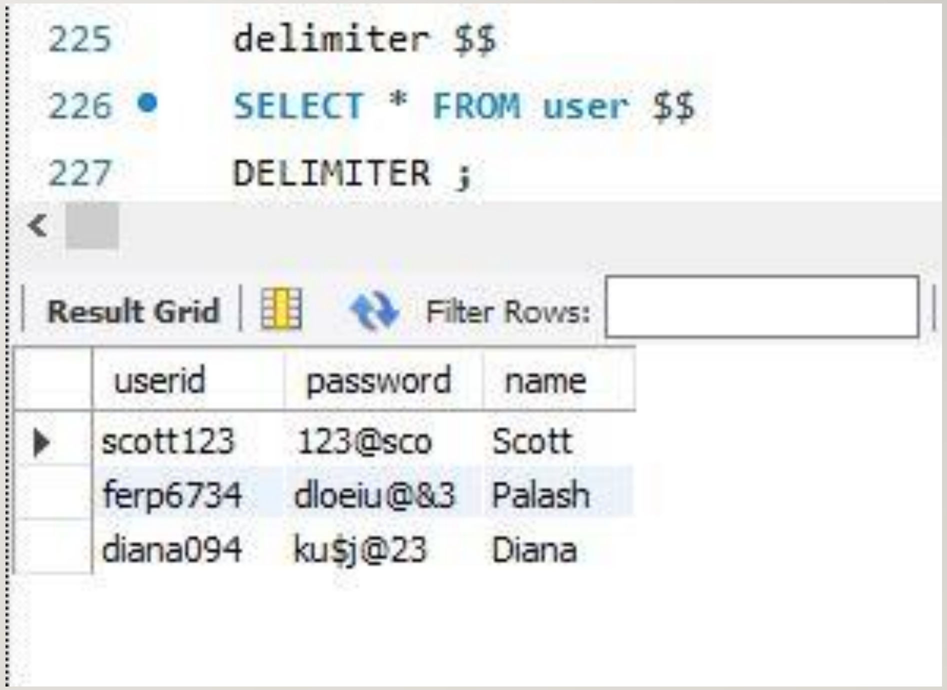
- By default, a procedure is associated with the default database (currently used database).
- To associate the procedure with a given database, specify the name as database_name**.**stored_procedure_name when you create it.
- Before creating stored procedure
  - Check database version: select version();
  - 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 ;

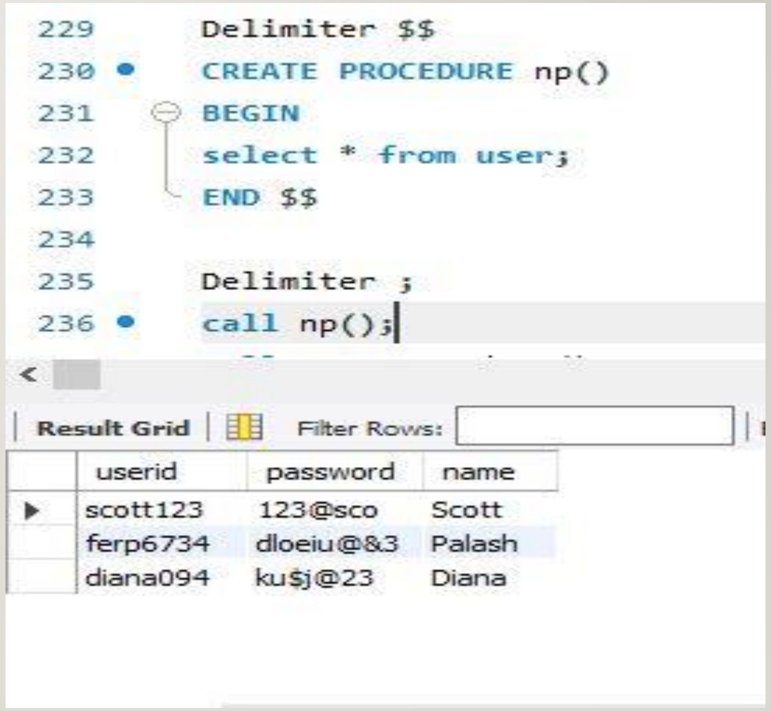


```
delimiter $$
SELECT * FROM user $$
DELIMITER ;
```

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

CREATE PROCEDURE

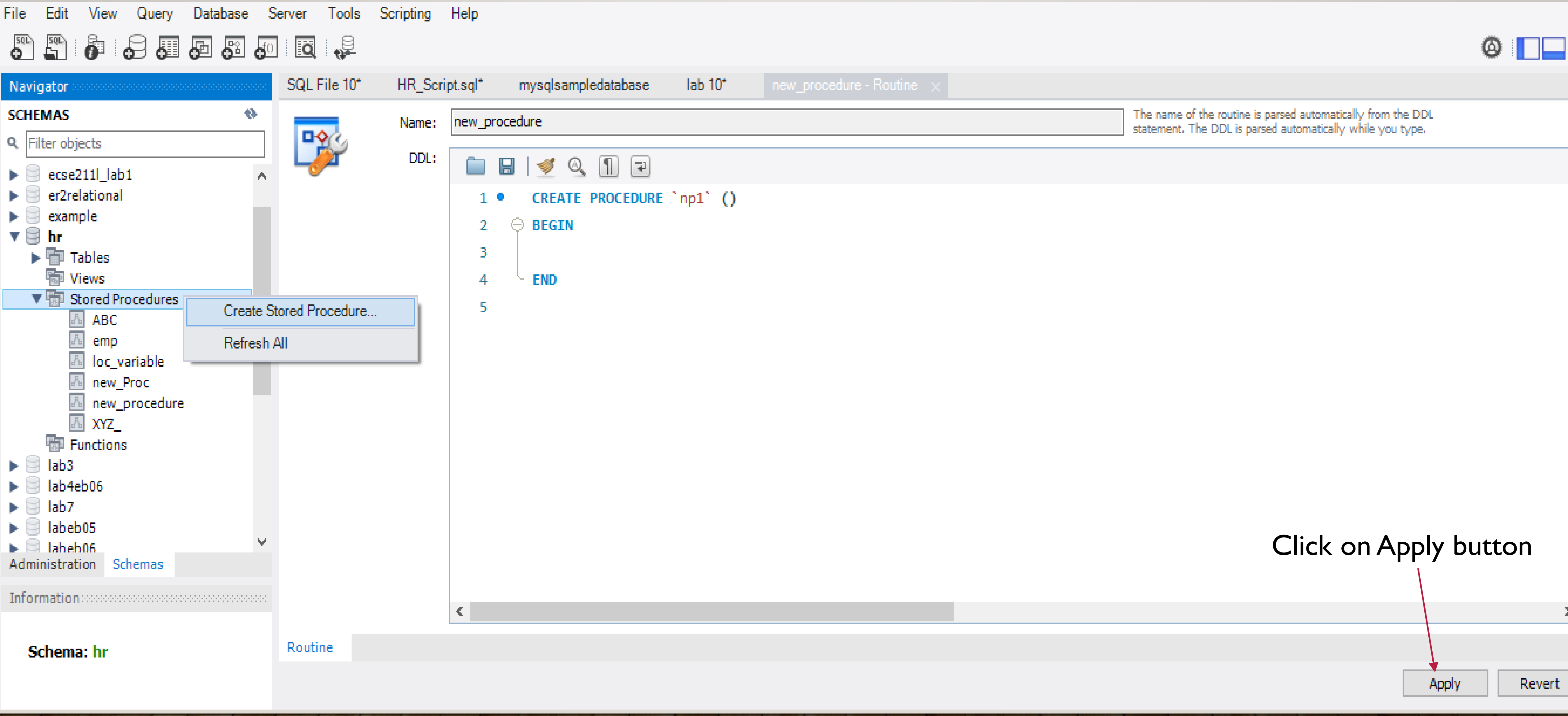
- CREATE PROCEDURE command creates the stored procedure.
- Next part is the procedure name (here, np).
- Parentheses, "()" holds the parameter(s) list as there are no parameters in this procedure
- Here, $$ is a real statement ender.



```
Delimiter $$
CREATE PROCEDURE np()
BEGIN
select * from user;
END $$

Delimiter ;
call np();
```

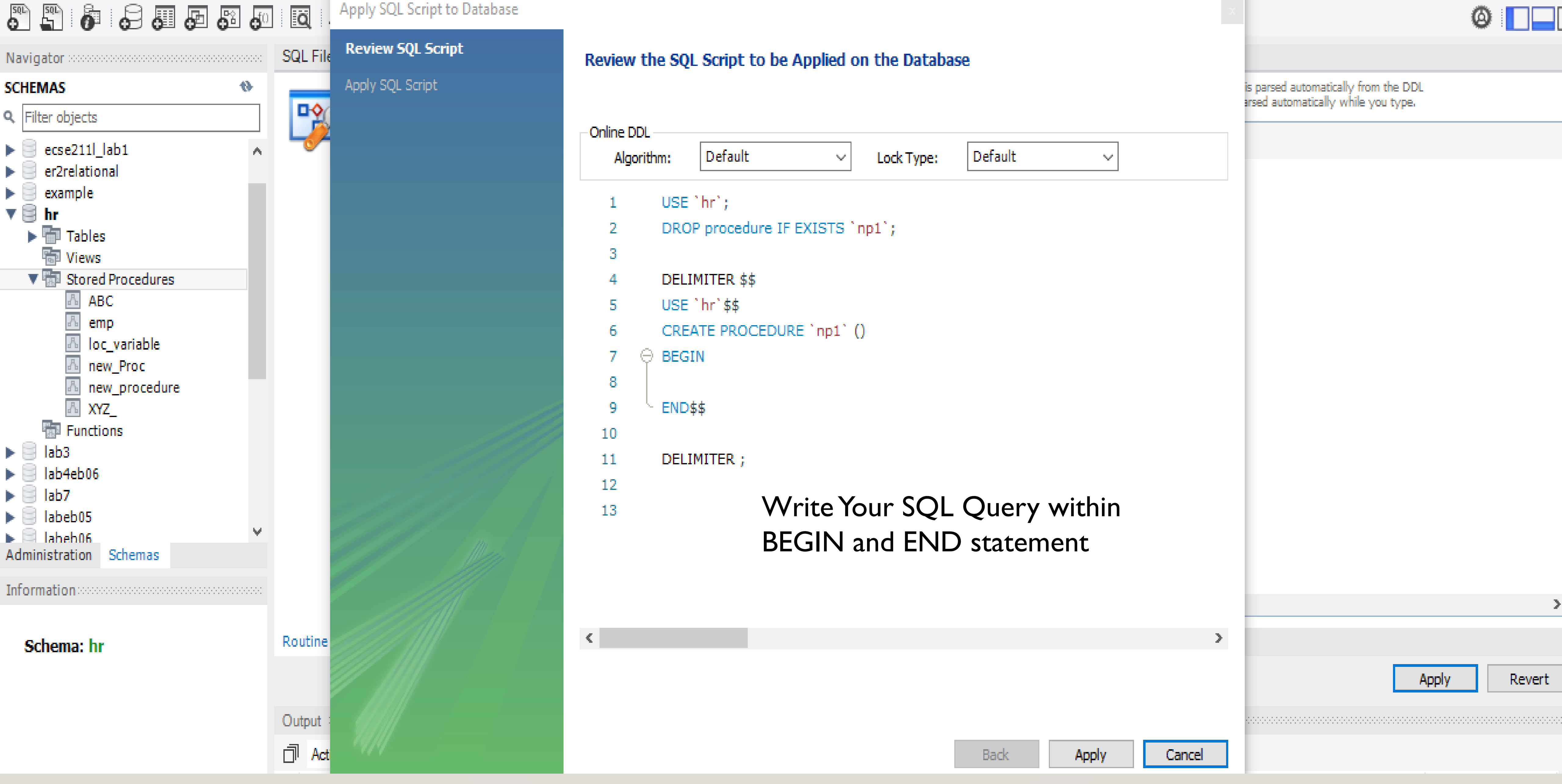
userid	password	name
scott123	123@sco	Scott
ferp6734	dloeiu@&3	Palash
diana094	ku$j@23	Diana

CREATE PROCEDURE THROUGH WORKBENCH TOOL



```
CREATE PROCEDURE `np1` ()
BEGIN

END
```

Click on Apply button



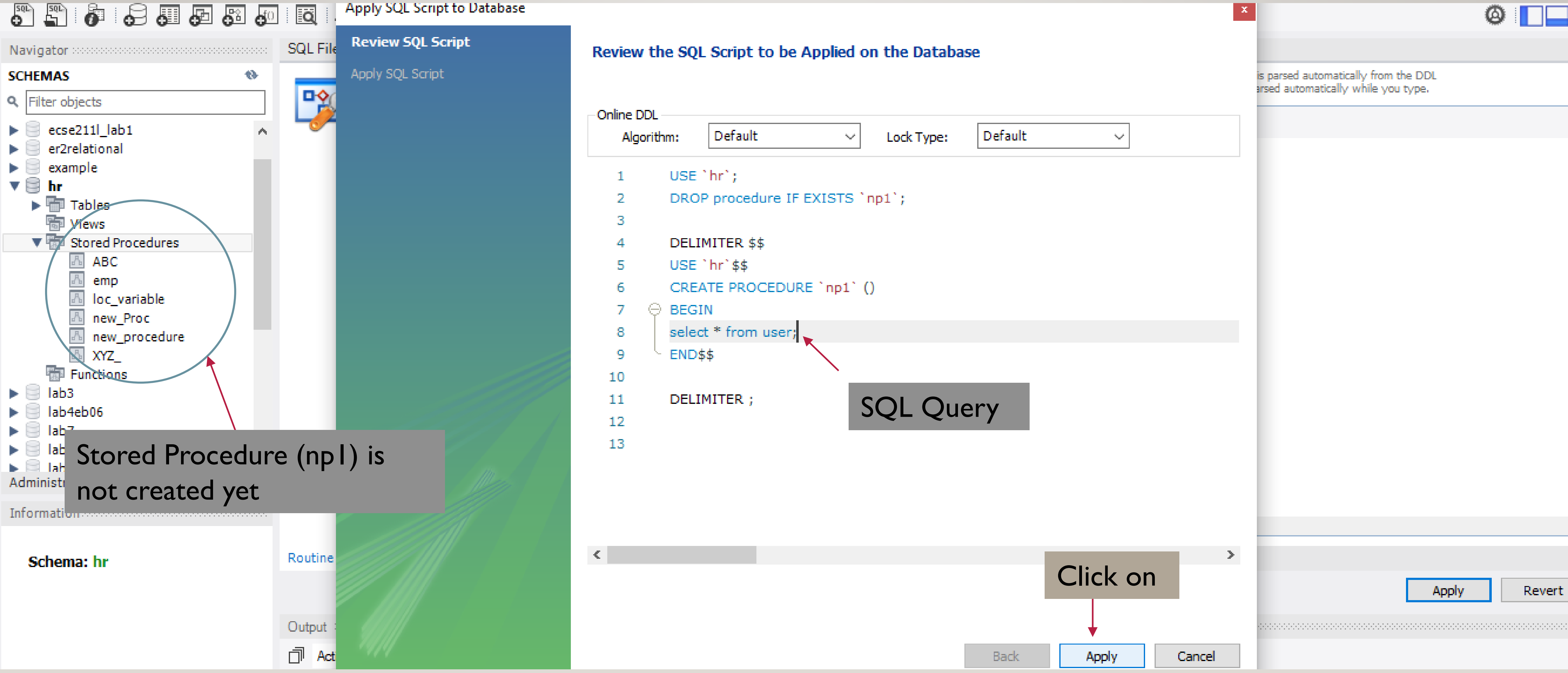
Review the SQL Script to be Applied on the Database

Online DDL

Algorithm: Default   Lock Type: Default

```
USE `hr`;
DROP procedure IF EXISTS `np1`;

DELIMITER $$
USE `hr`$$
CREATE PROCEDURE `np1` ()
BEGIN

END$$

DELIMITER ;

```

Write Your SQL Query within BEGIN and END statement



Review the SQL Script to be Applied on the Database

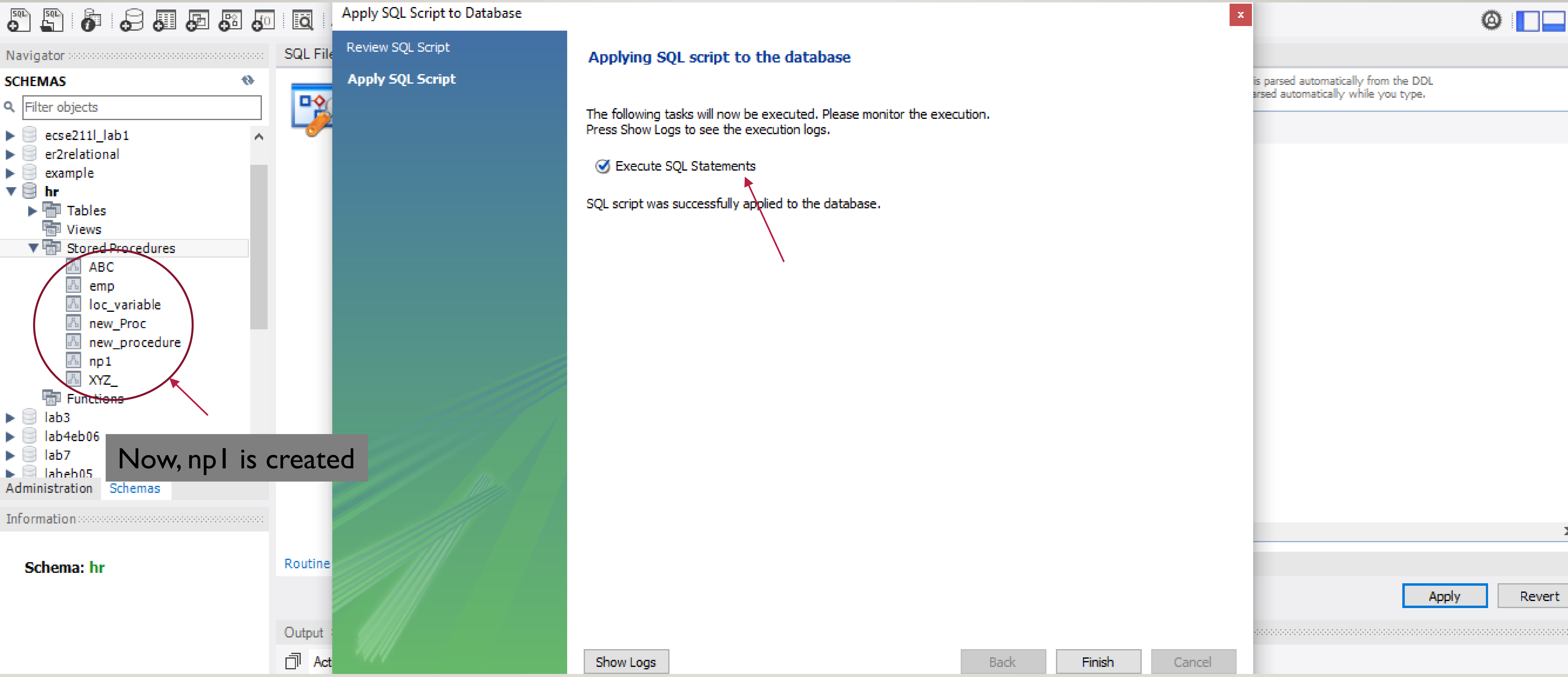
Online DDL
Algorithm: Default
Lock Type: Default

```
USE `hr`;
DROP procedure IF EXISTS `np1`;

DELIMITER $$
USE `hr`$$
CREATE PROCEDURE `np1` ()
BEGIN
select * from user;
END$$

DELIMITER ;
```

SQL Query

Stored Procedure (np1) is not created yet

Click on



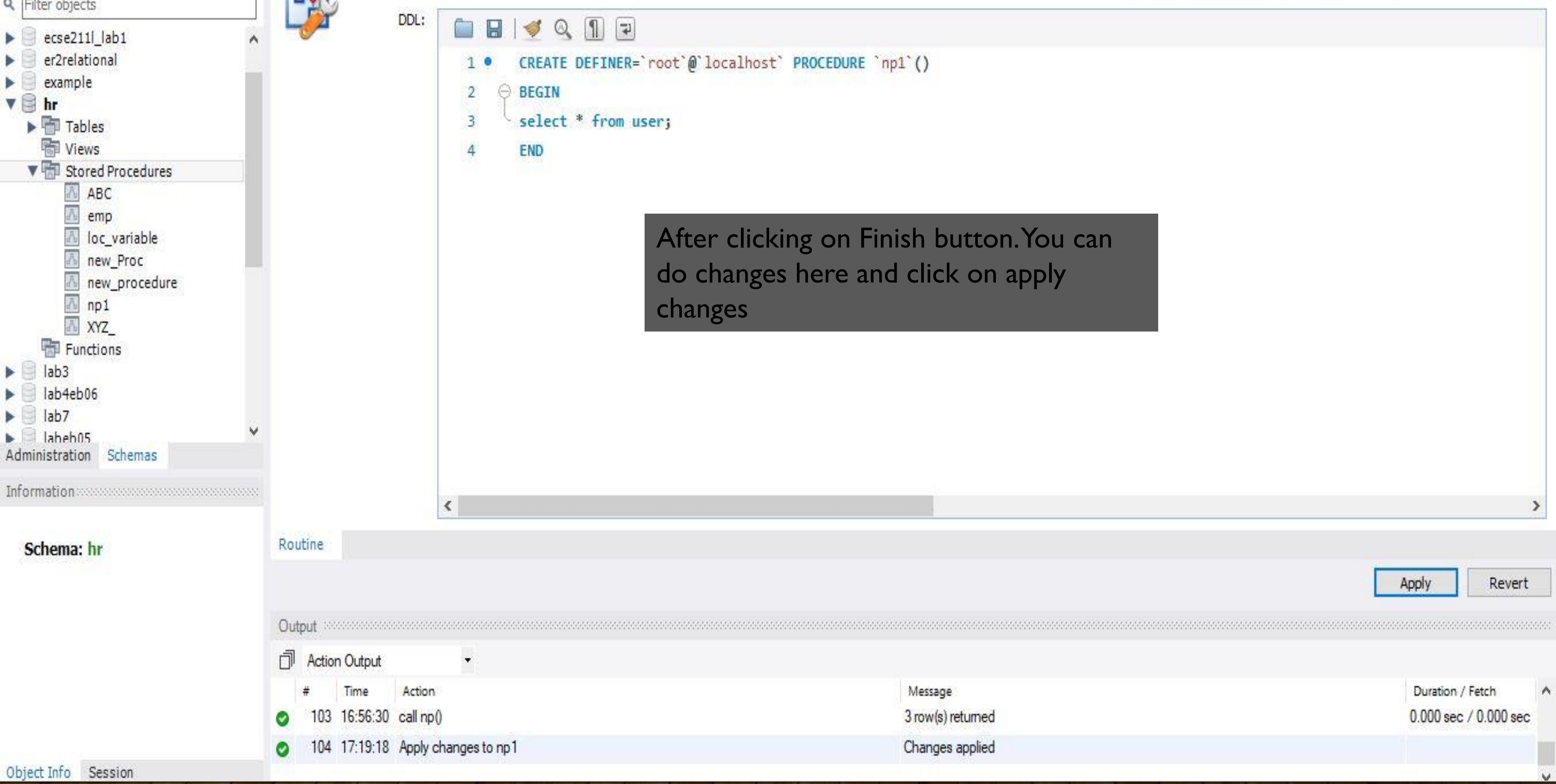
Applying SQL script to the database

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.

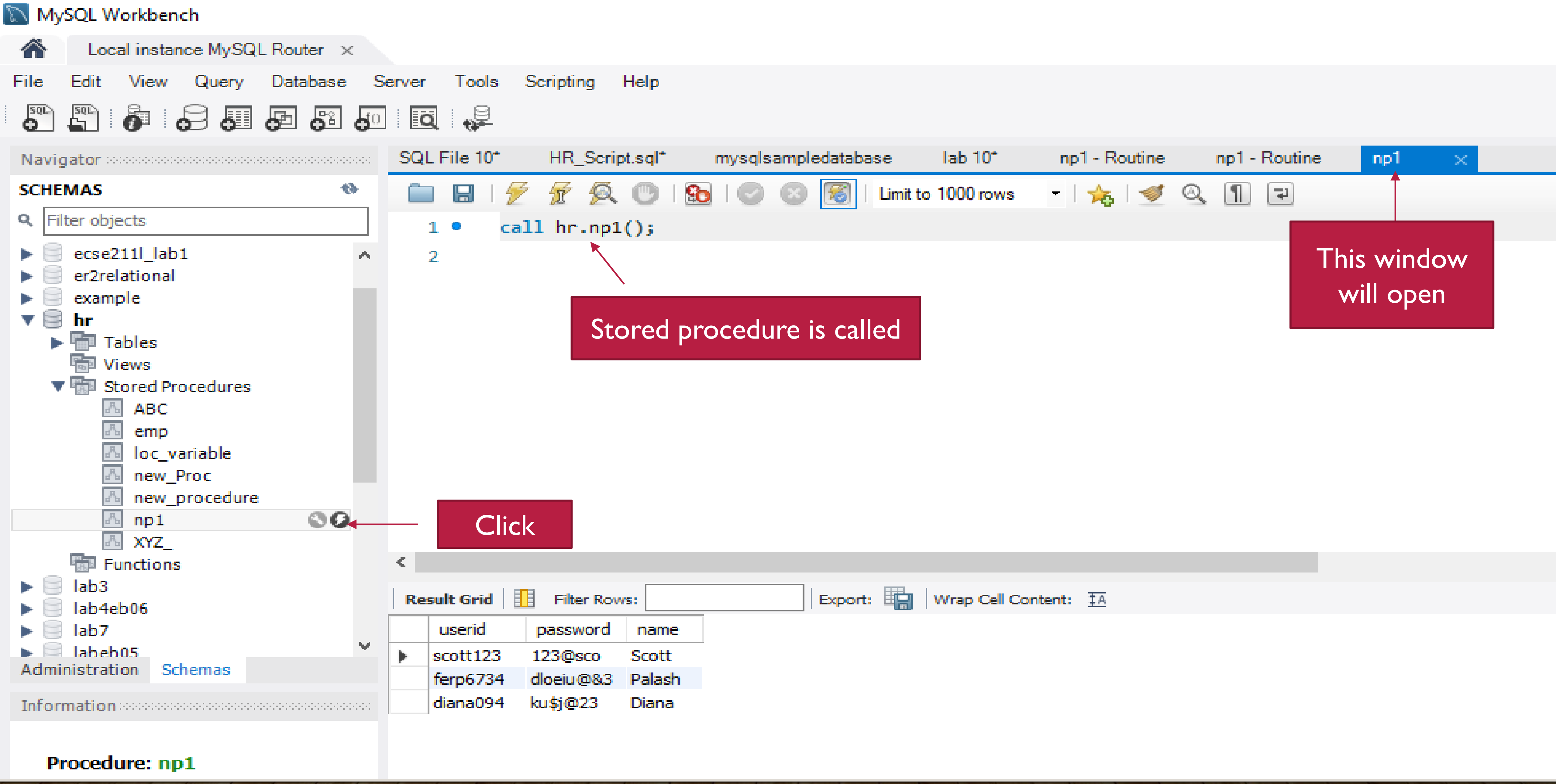
Now, np1 is created



```
CREATE DEFINER=`root`@`localhost` PROCEDURE `np1`()
BEGIN
select * from user;
END
```

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

```
   #    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
```



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

Stored procedure is called

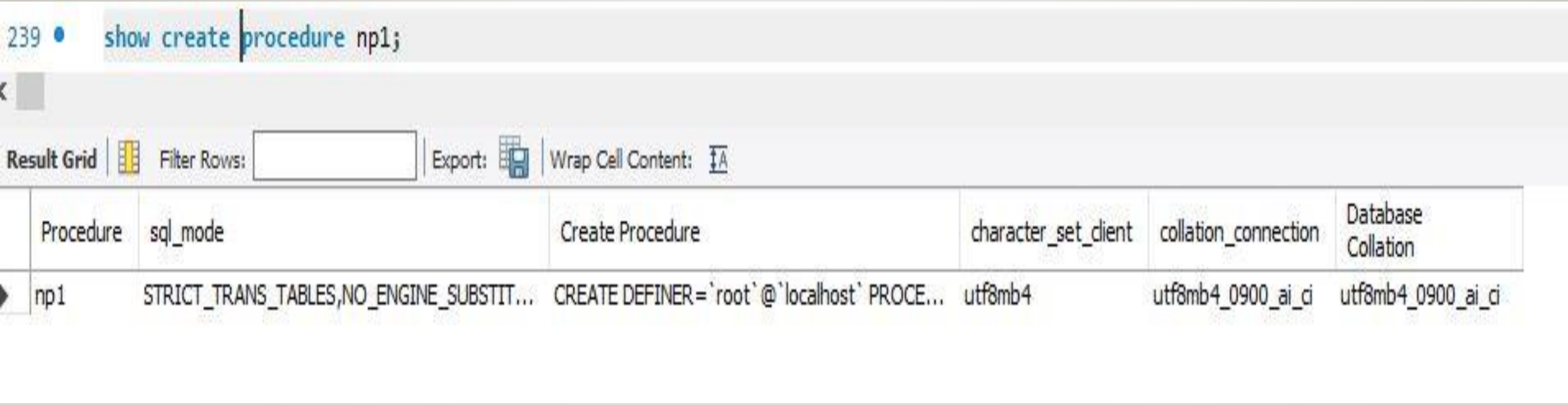
This window will open

Click

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

Procedure: np1

SHOW PROCEDURE



```
show create procedure np1;
```

Result Grid | Filter Rows: | 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

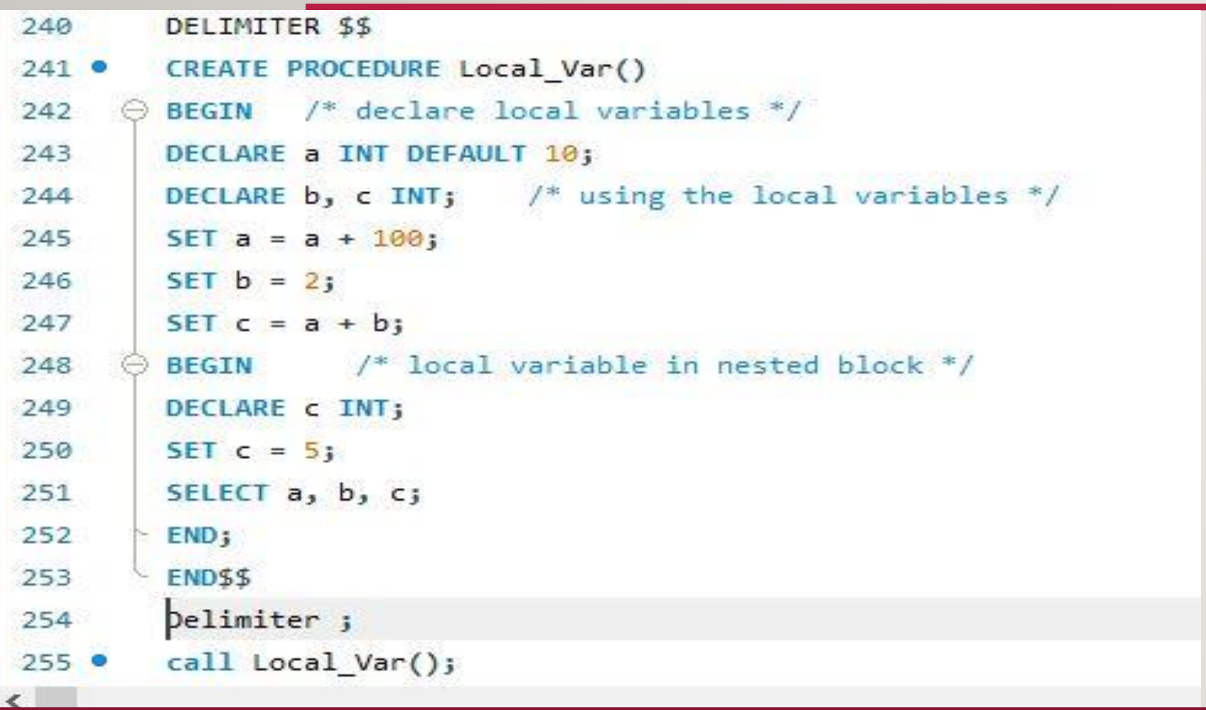
Declare Statement

- 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.

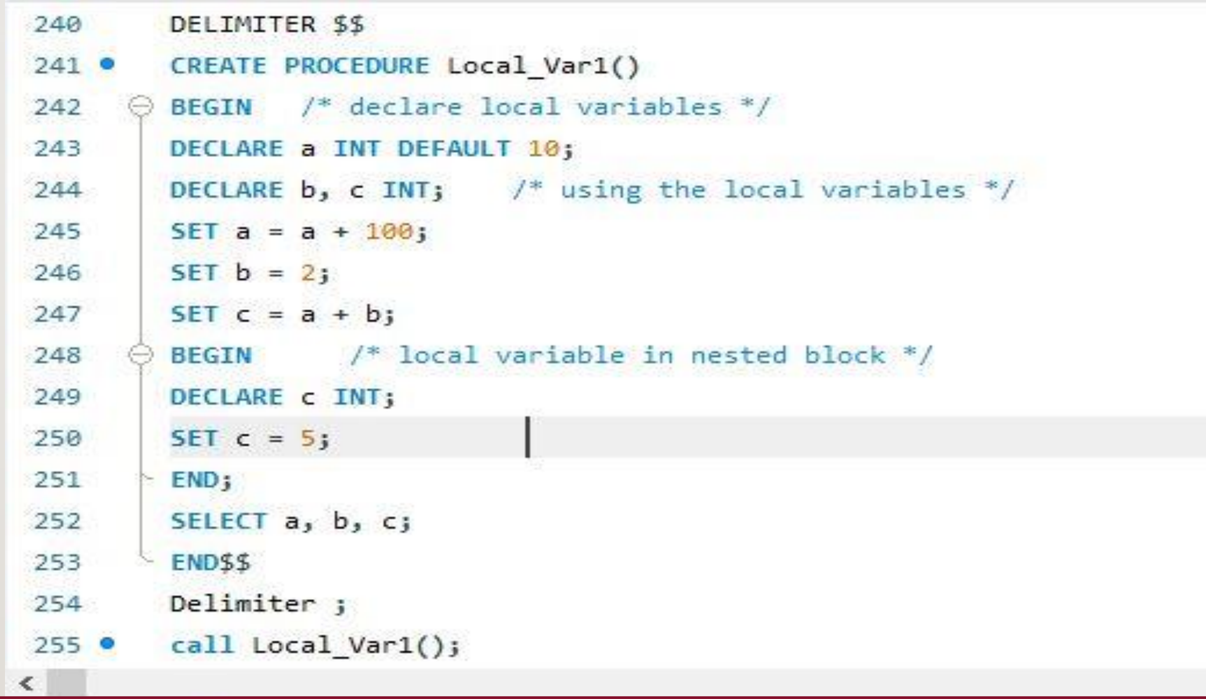
Variables in Stored Programs

- 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

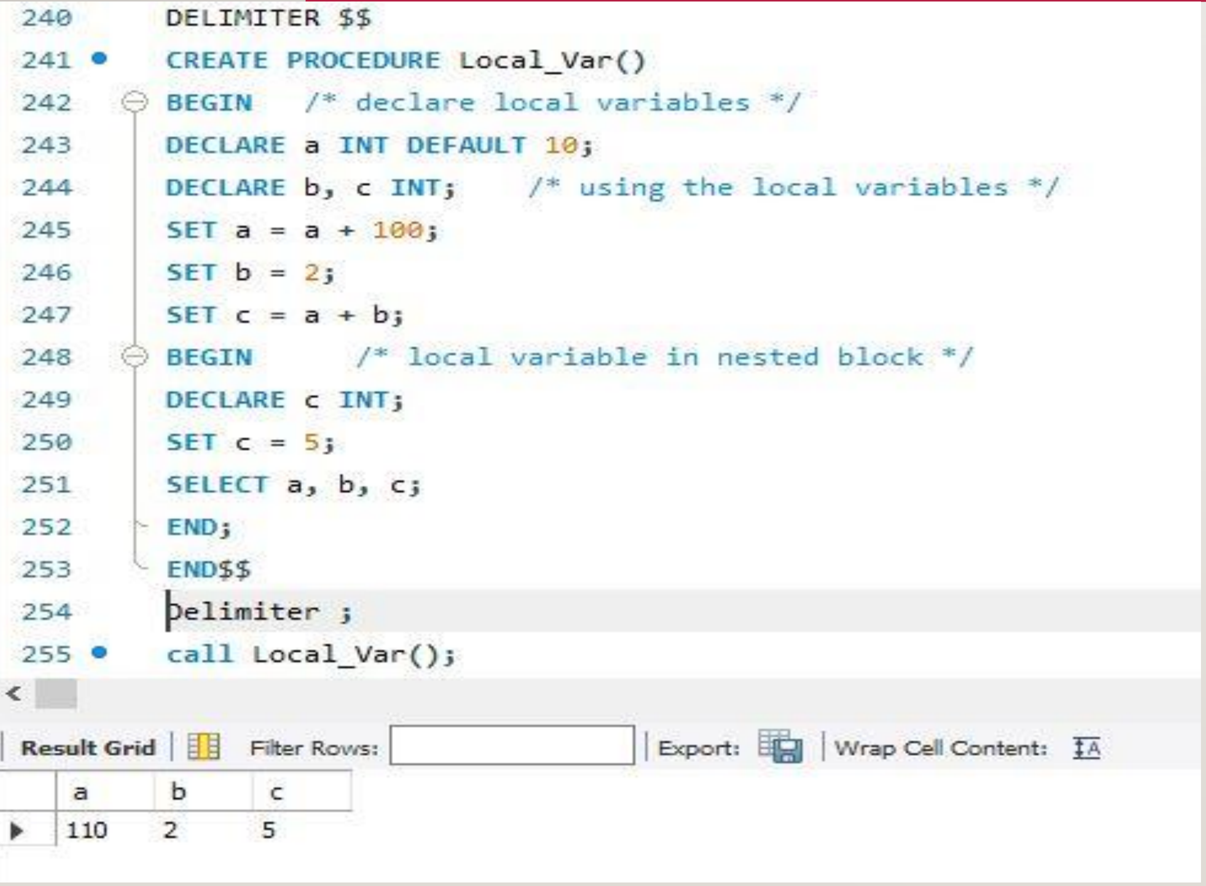


```
DELIMITER $$
CREATE PROCEDURE Local_Var()
BEGIN   /* declare local variables */
DECLARE a INT DEFAULT 10;
DECLARE b, c INT;    /* using the local variables */
SET a = a + 100;
SET b = 2;
SET c = a + b;
BEGIN      /* local variable in nested block */
DECLARE c INT;
SET c = 5;
SELECT a, b, c;
END;
END$$
Delimiter ;
call Local_Var();
```



```
DELIMITER $$
CREATE PROCEDURE Local_Var1()
BEGIN   /* declare local variables */
DECLARE a INT DEFAULT 10;
DECLARE b, c INT;    /* using the local variables */
SET a = a + 100;
SET b = 2;
SET c = a + b;
BEGIN      /* local variable in nested block */
DECLARE c INT;
SET c = 5;
END;
SELECT a, b, c;
END$$
Delimiter ;
call Local_Var1();
```

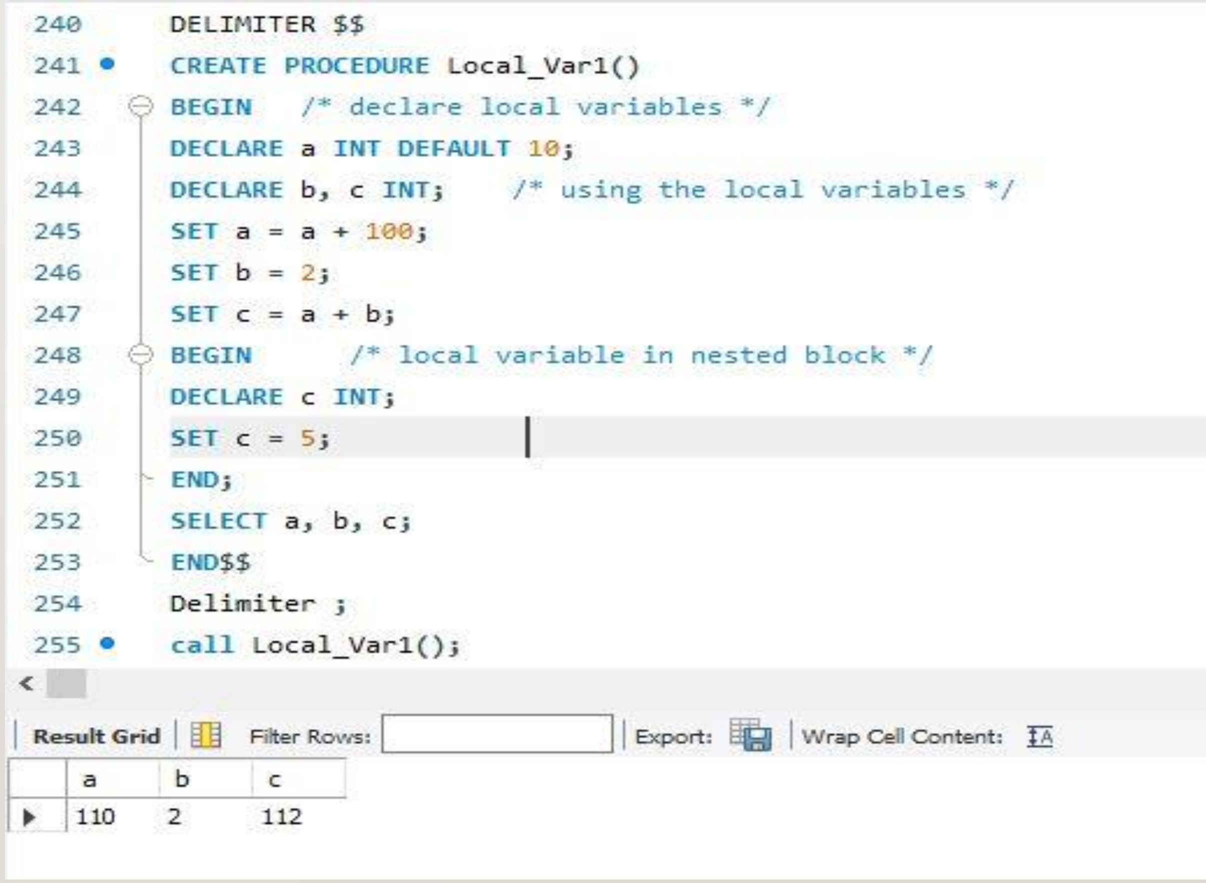
LOCAL VARIABLES AND GLOBAL VARIABLES



```
DELIMITER $$
CREATE PROCEDURE Local_Var()
BEGIN   /* declare local variables */
DECLARE a INT DEFAULT 10;
DECLARE b, c INT;    /* using the local variables */
SET a = a + 100;
SET b = 2;
SET c = a + b;
BEGIN      /* local variable in nested block */
DECLARE c INT;
SET c = 5;
SELECT a, b, c;
END;
END$$
Delimiter ;
call Local_Var();
```

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

a	b	c
110	2	5

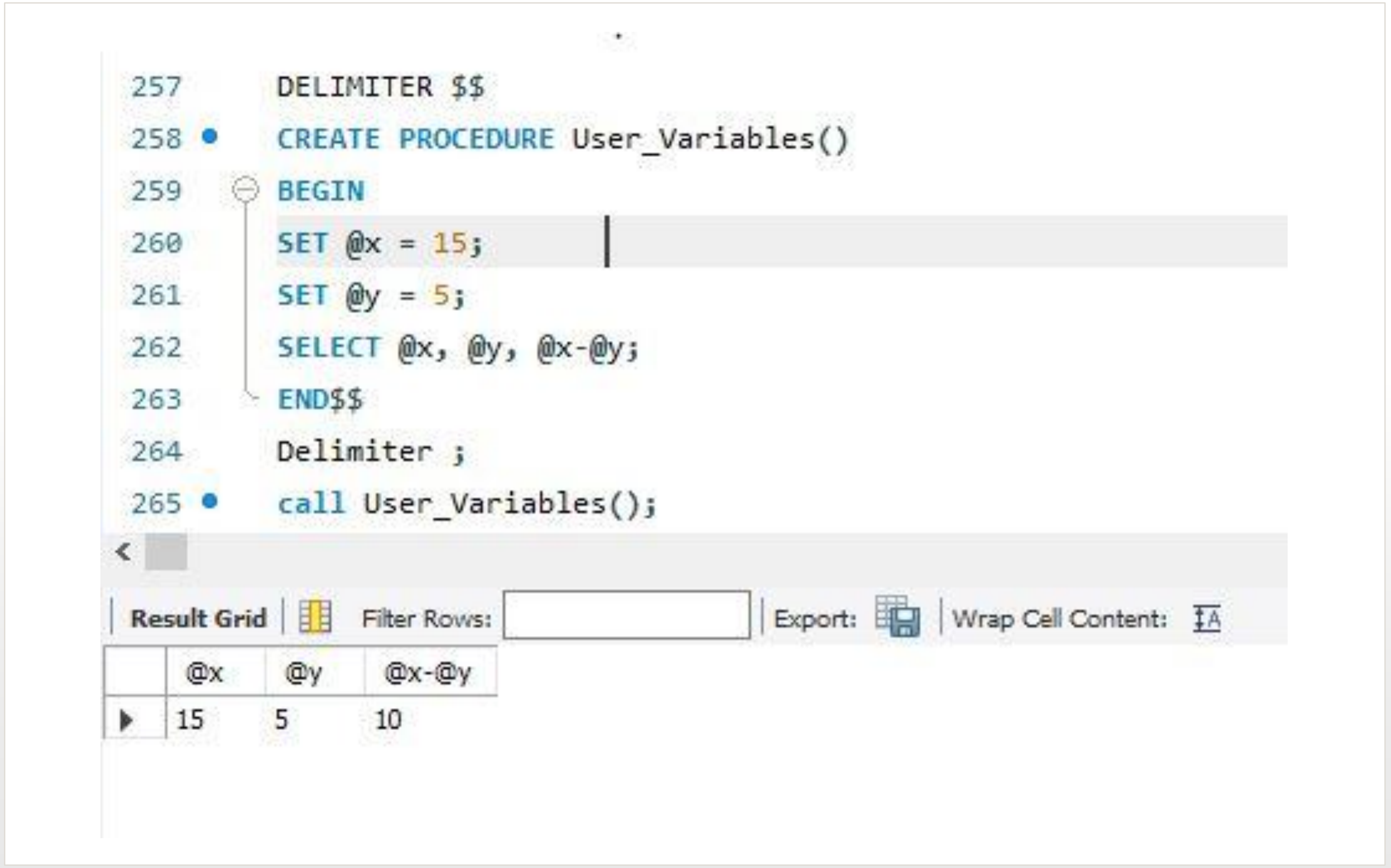


```
DELIMITER $$
CREATE PROCEDURE Local_Var1()
BEGIN   /* declare local variables */
DECLARE a INT DEFAULT 10;
DECLARE b, c INT;    /* using the local variables */
SET a = a + 100;
SET b = 2;
SET c = a + b;
BEGIN      /* local variable in nested block */
DECLARE c INT;
SET c = 5;
END;
SELECT a, b, c;
END$$
Delimiter ;
call Local_Var1();
```

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

a	b	c
110	2	112

USER VARIABLES



```
DELIMITER $$
CREATE PROCEDURE User_Variables()
BEGIN
SET @x = 15;
SET @y = 5;
SELECT @x, @y, @x-@y;
END$$
Delimiter ;
call User_Variables();
```

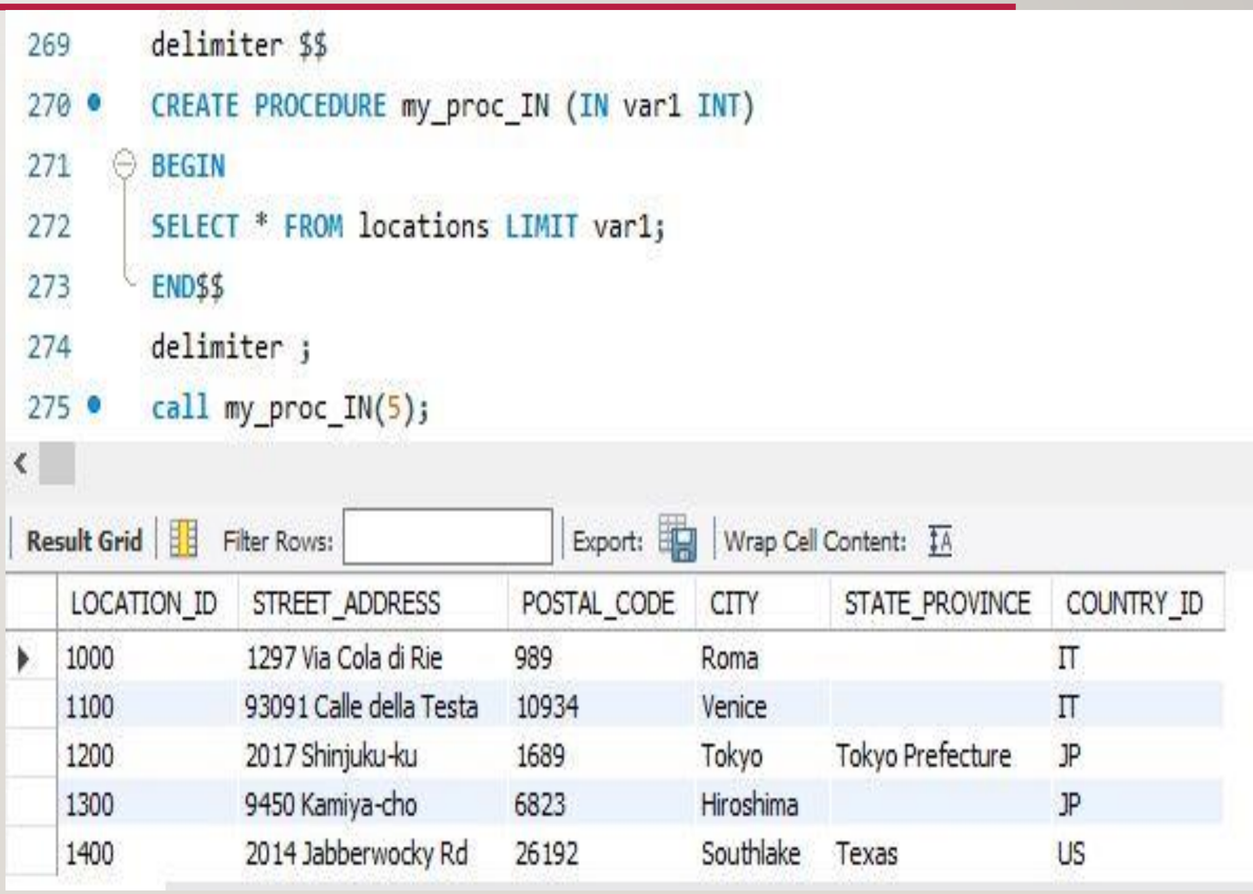
```
@x   @y   @x-@y
15   5    10
```

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.

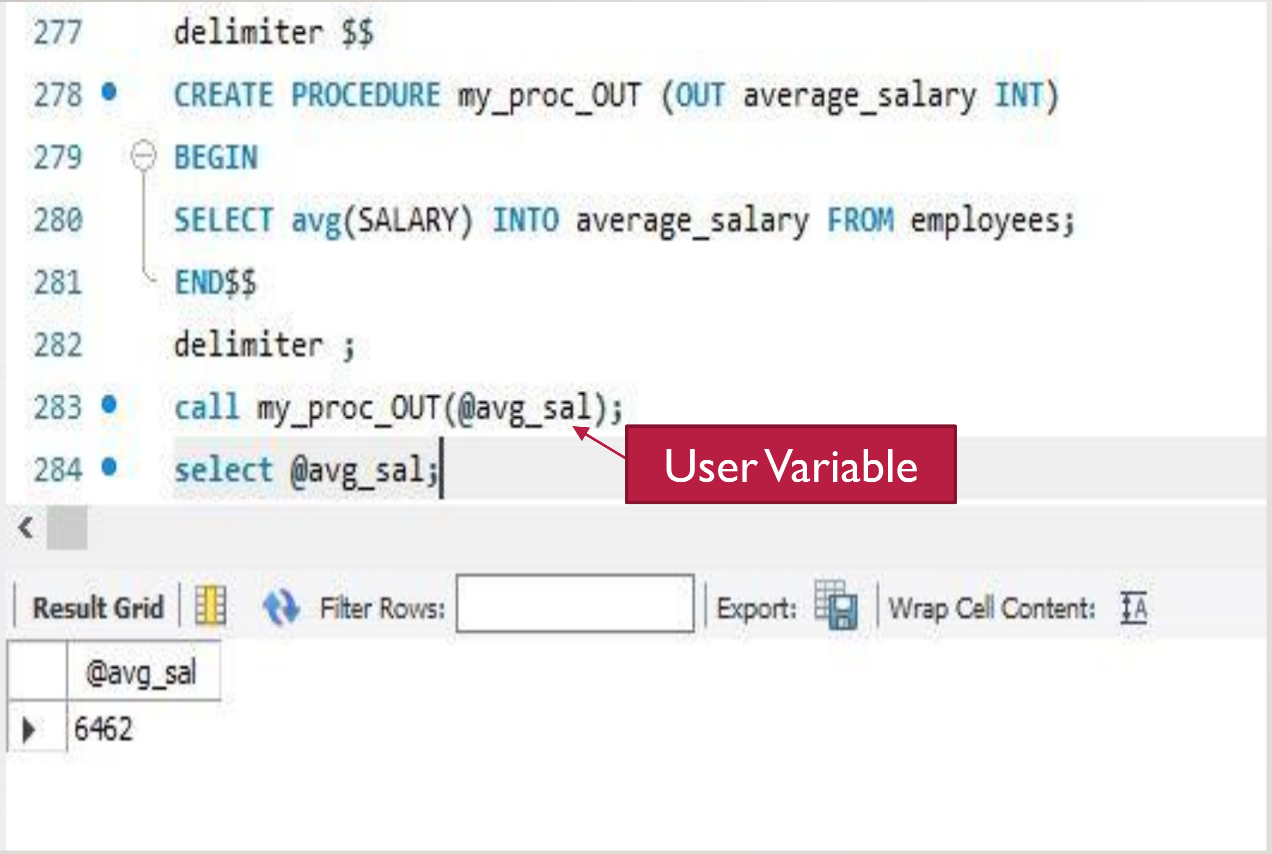


```
delimiter $$
CREATE PROCEDURE my_proc_IN (IN var1 INT)
BEGIN
SELECT * FROM locations LIMIT var1;
END$$
delimiter ;
call my_proc_IN(5);
```

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

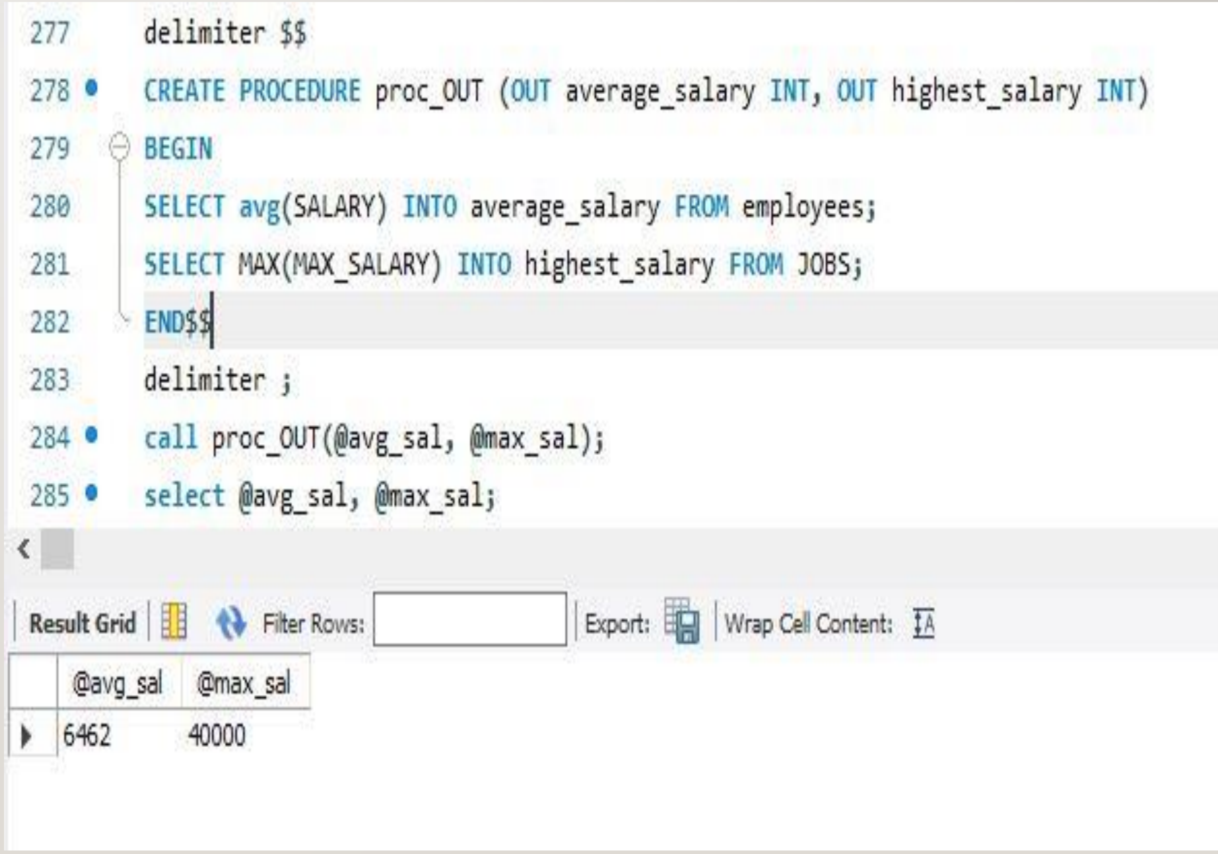
PARAMETER 'OUT' EXAMPLE



```
delimiter $$
CREATE PROCEDURE my_proc_OUT (OUT average_salary INT)
BEGIN
SELECT avg(SALARY) INTO average_salary FROM employees;
END$$
delimiter ;
call my_proc_OUT(@avg_sal);
select @avg_sal;
```

User Variable

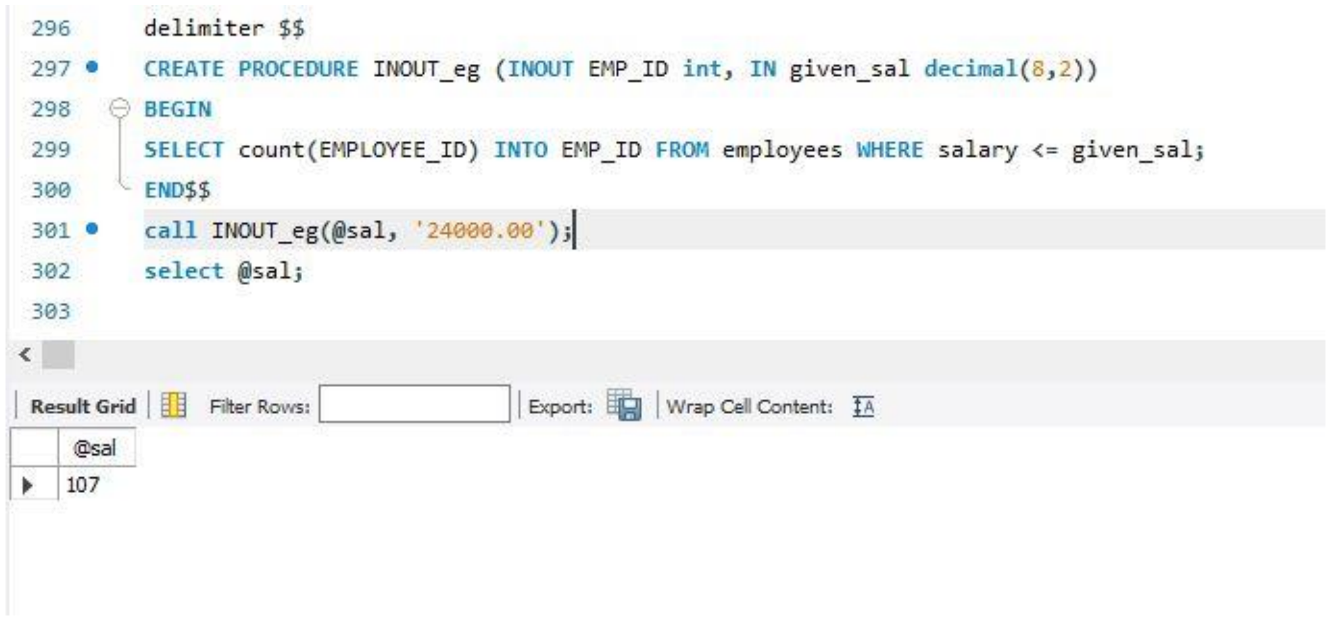
```
@avg_sal
6462
```



```
delimiter $$
CREATE PROCEDURE proc_OUT (OUT average_salary INT, OUT highest_salary INT)
BEGIN
SELECT avg(SALARY) INTO average_salary FROM employees;
SELECT MAX(MAX_SALARY) INTO highest_salary FROM JOBS;
END$$
delimiter ;
call proc_OUT(@avg_sal, @max_sal);
select @avg_sal, @max_sal;
```

```
@avg_sal  @max_sal
6462      40000
```

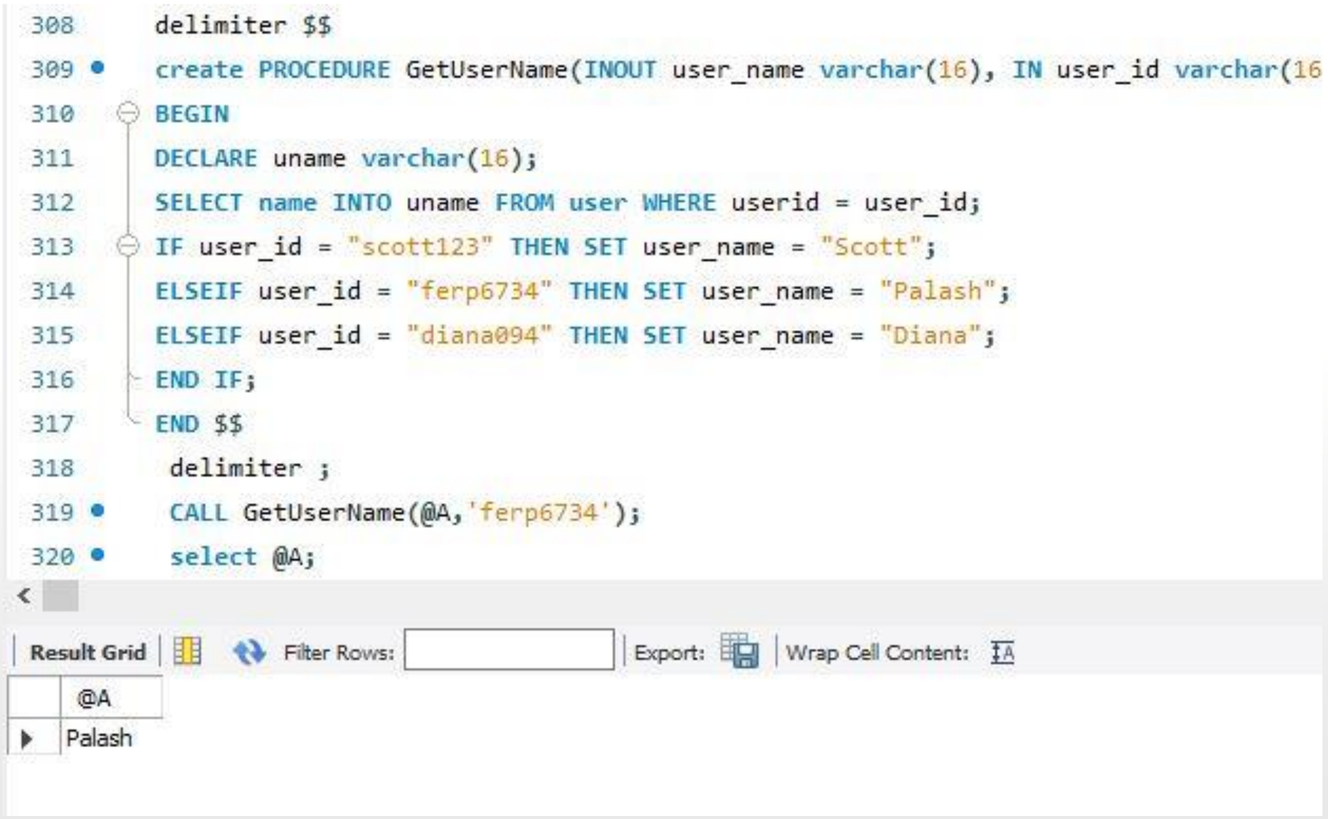
PARAMETER 'INOUT' EXAMPLE



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

```
@sal
107
```

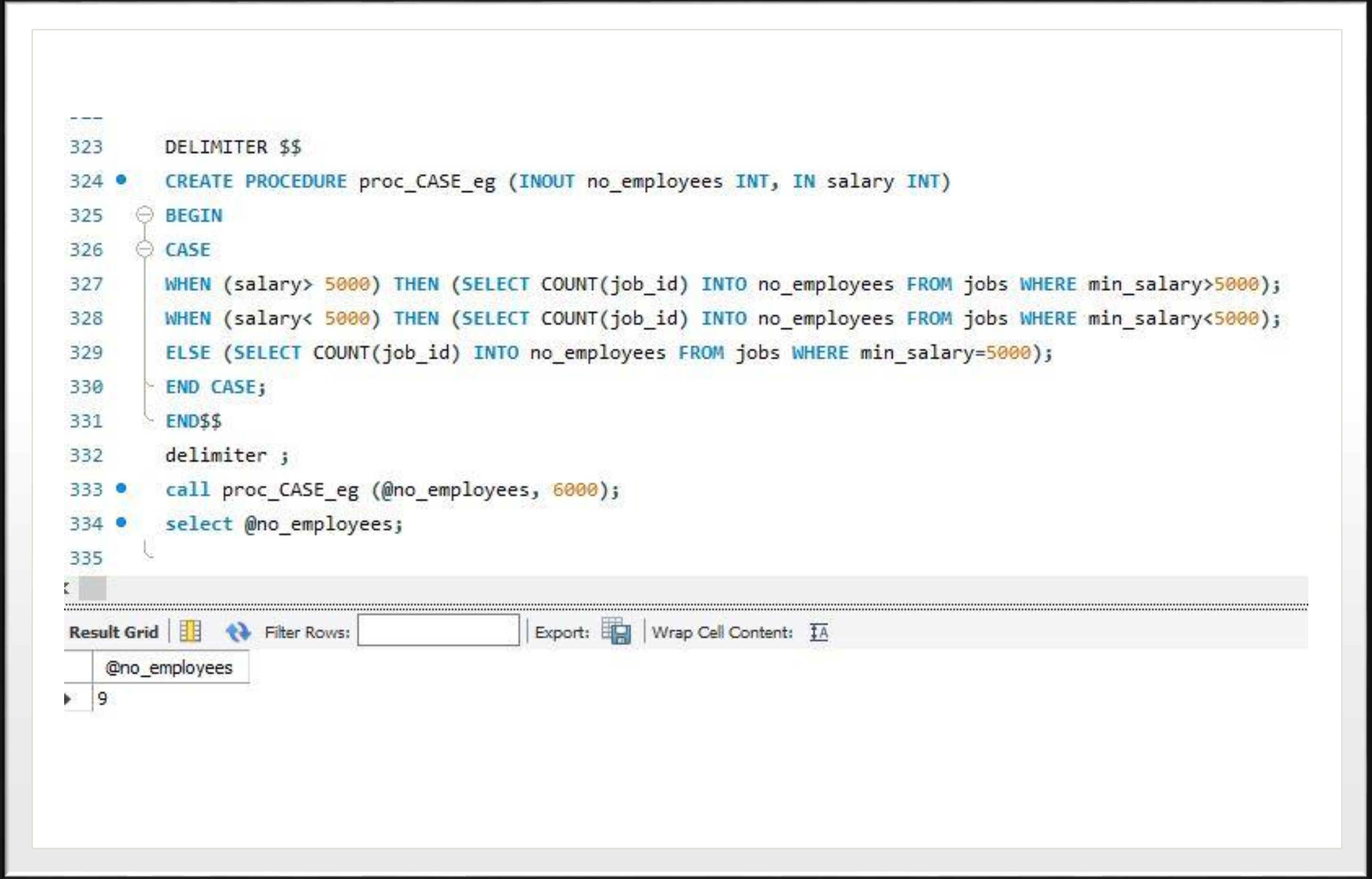
FLOW CONTROL STATEMENTS: IF STATEMENT



```
delimiter $$
create PROCEDURE GetUserName(INOUT user_name varchar(16), IN user_id varchar(16
BEGIN
DECLARE uname varchar(16);
SELECT name INTO uname FROM user WHERE userid = user_id;
IF user_id = "scott123" THEN SET user_name = "Scott";
ELSEIF user_id = "ferp6734" THEN SET user_name = "Palash";
ELSEIF user_id = "diana094" THEN SET user_name = "Diana";
END IF;
END $$
 delimiter ;
 CALL GetUserName(@A,'ferp6734');
 select @A;
```

```
@A
Palash
```

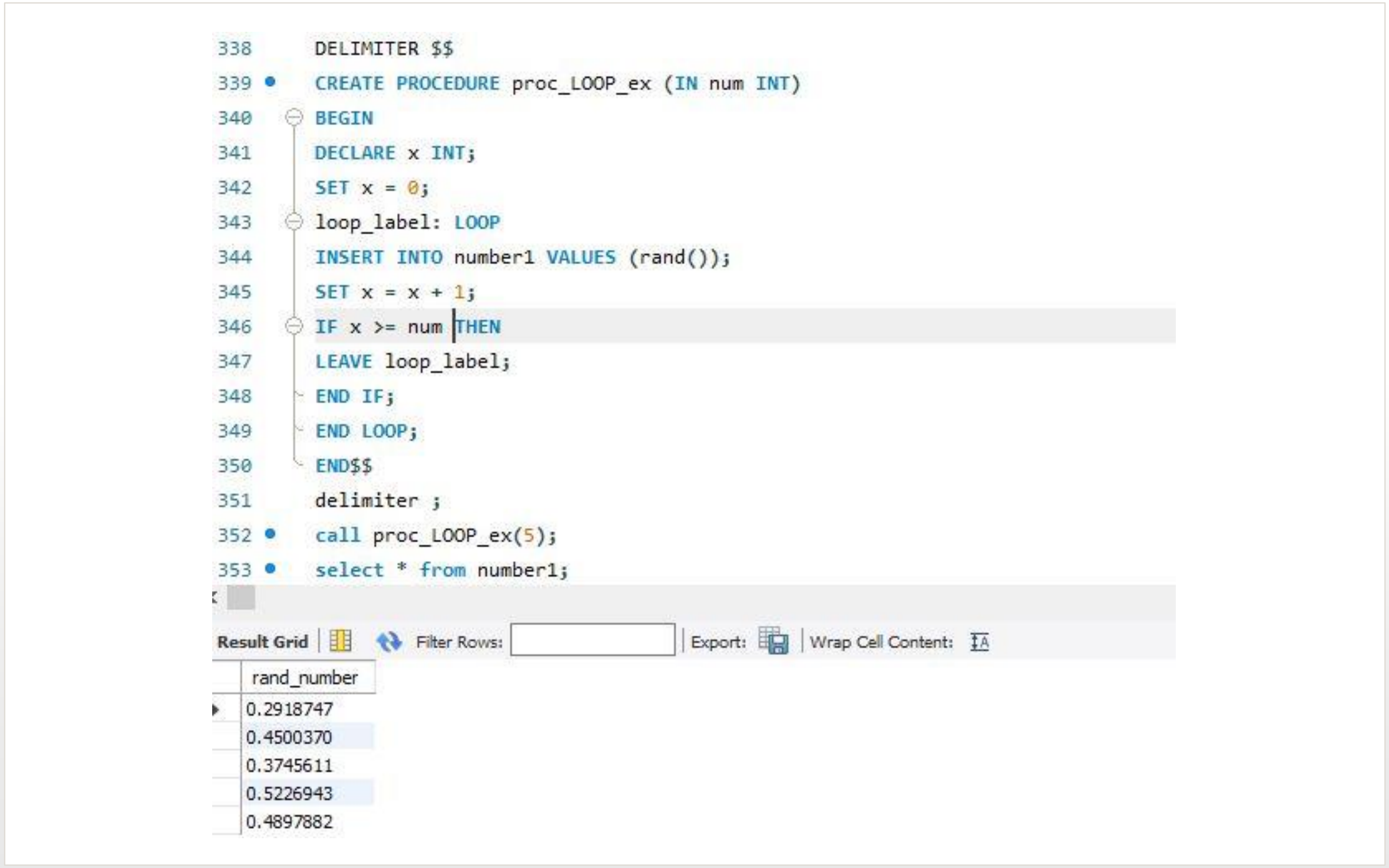
FLOW CONTROL STATEMENTS: CASE STATEMENT



```
DELIMITER $$
CREATE PROCEDURE proc_CASE_eg (INOUT no_employees INT, IN salary INT)
BEGIN
CASE
WHEN (salary> 5000) THEN (SELECT COUNT(job_id) INTO no_employees FROM jobs WHERE min_salary>5000);
WHEN (salary< 5000) THEN (SELECT COUNT(job_id) INTO no_employees FROM jobs WHERE min_salary<5000);
ELSE (SELECT COUNT(job_id) INTO no_employees FROM jobs WHERE min_salary=5000);
END CASE;
END$$
delimiter ;
call proc_CASE_eg (@no_employees, 6000);
select @no_employees;
```

```
@no_employees
9
```

FLOW CONTROL STATEMENTS:LOOP STATEMENT

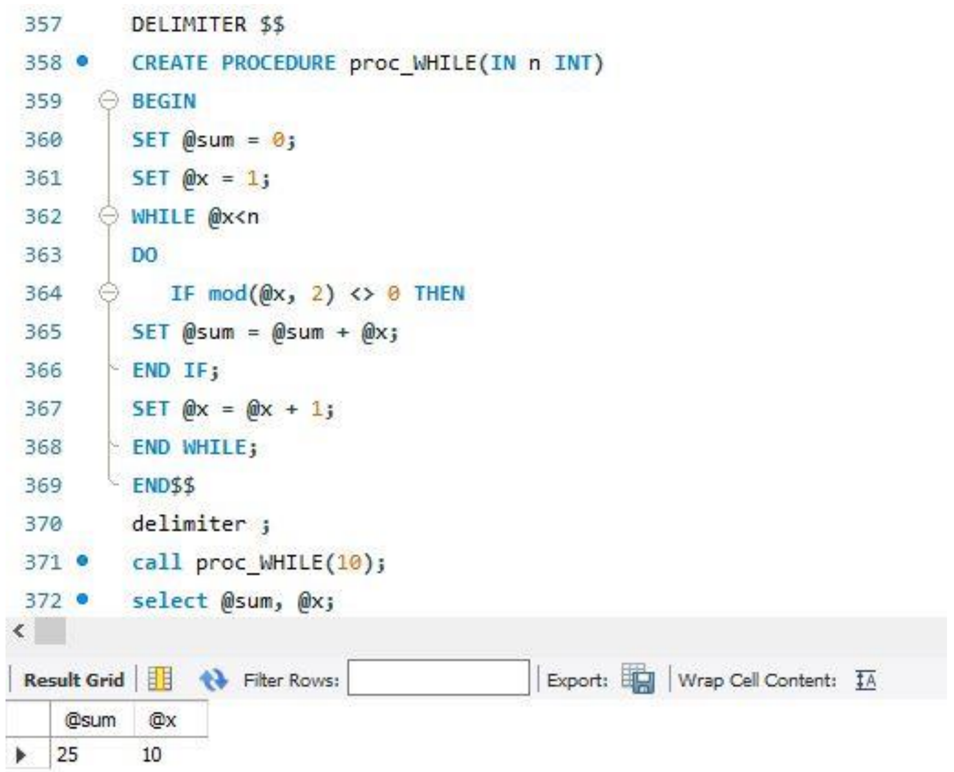


```
DELIMITER $$
CREATE PROCEDURE proc_LOOP_ex (IN num INT)
BEGIN
DECLARE x INT;
SET x = 0;
loop_label: LOOP
INSERT INTO number1 VALUES (rand());
SET x = x + 1;
IF x >= num THEN
LEAVE loop_label;
END IF;
END LOOP;
END$$
delimiter ;
call proc_LOOP_ex(5);
select * from number1;
```

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



```
DELIMITER $$
CREATE PROCEDURE proc_WHILE(IN n INT)
BEGIN
SET @sum = 0;
SET @x = 1;
WHILE @x<n
DO
   IF mod(@x, 2) <> 0 THEN
SET @sum = @sum + @x;
END IF;
SET @x = @x + 1;
END WHILE;
END$$
delimiter ;
call proc_WHILE(10);
select @sum, @x;
```

	@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