

# MySQL Programming Lab

(taken from several tutorials on the [mysqltutorial.org](http://mysqltutorial.org) website)

## Note:

This lab will require the MySQL workbench. I thought that you could run it from NetBeans, but it would seem that there are special characters (like the grave mark: `) that MySQL can interpret fine, but which NetBeans cannot. Rather than figuring out where all of those are and making a NetBeans version of this lab, I decided that I would ask you to install the MySQL workbench and use that to work this lab. The MySQL installer can be found at URL: <https://dev.mysql.com/downloads/workbench/>.

Note that you do not **have** to install MySQL itself. If you install the MySQL database management system on your computer, then you will be able to use localhost as the “server” where MySQL runs when you create the connection to the database. Otherwise, you will need to connect to the CSULB campus MySQL instance that I wrote about on BeachBoard | Content | Projects | Miming’s Cantonese Restaurant Term Project | Using MySQL from CSULB servers.














## Loading the Test Database

Step 1. Download the [mysqlprogramming.zip](#) from my code page. It contains all the files needed for this lab.

<a href="#">mysqlsampledatabase.sql</a>	<a href="#">Classic Models Database used for the stored procedures, functions and triggers sections</a>
<a href="#">storedprocedure.txt</a>	<a href="#">Code for GetAllProducts stored procedure</a>
<a href="#">function.txt</a>	<a href="#">Code for CustomerLevel() function</a>
<a href="#">testfunction.txt</a>	<a href="#">Code to test the CustomerLevel() function</a>
<a href="#">getcustomerlevelbefore.txt</a>	<a href="#">Code for GetCustomerLevel stored procedure before the CustomerLevel() function</a>
<a href="#">getcustomerlevelafter.txt</a>	<a href="#">Code for GetCustomerLevel stored procedure after the function</a>
<a href="#">employeesaudittable.sql</a>	<a href="#">Code to create the employees_audit table</a>
<a href="#">employeesbeforeupdate.txt</a>	<a href="#">Code to create the employees_before_update trigger</a>
<a href="#">testtrigger.sql</a>	<a href="#">Code to test the trigger</a>

Step 2. Unzip the downloaded file into a temporary folder. You can use any folder you want. To make it simple we will unzip it to the C:\temp folder as follows.

(C:) ▸ Temp ▸ mysqlprogramming

<input type="checkbox"/> Name	Date modified	Type	Size
 CallSPDemo.java	11/25/2016 10:5...	JAVA File	2 KB
 employeesauditable.sql	11/25/2016 10:3...	SQL File	1 KB
 employeesbeforeupdate.txt	11/25/2016 10:4...	TXT File	1 KB
 function.txt	11/25/2016 10:0...	TXT File	1 KB
 getcustomerlevel.txt	11/25/2016 10:2...	TXT File	1 KB
 getcustomerlevelafter.txt	11/25/2016 10:3...	TXT File	1 KB
 getcustomerlevelbefore.txt	11/25/2016 10:3...	TXT File	1 KB
 mysqljdbc.sql	5/14/2015 8:17 ...	SQL File	13 KB
 mysqlsampledatabase.sql	11/19/2016 7:42...	SQL File	206 KB
 SQLExceptionDemo.java	11/25/2016 10:5...	JAVA File	2 KB
 storedprocedure.txt	11/25/2016 10:1...	TXT File	1 KB
 testfunction.txt	11/25/2016 10:1...	TXT File	1 KB
 testtrigger.sql	11/25/2016 10:4...	SQL File	1 KB

Step 3. Launch MySQL Workbench application.

Step 4. Either connect to your local database using an existing connection or create a new one.

To add a new database connection for querying, click New Connection as follows:

Step 5. Setup a new connection: you must enter all connection parameters on this Setup New Connection window. The following information is required:

- Connection name: the name of the connection. If you connect to the localhost , just type local . In case you connect to a specific host, use the host name for the name of the connection to make it clear.
- Host name: in this case it is 127.0.0.1 i.e., localhost . You can enter either IP address or the name of the database server.
- Username: the user that you use connect to the MySQL database. In this case it is the root user.  
**Note:** if you are connecting to MySQL on the CSULB campus, you must use the account and password that you received earlier by email.

You can also provide the following information optionally:

- Password: the password of the user that you use to connect to the database.
- Default Schema: is the database that you want to connect. You can leave it blank and select later using the use database command.

**Setup New Connection**

Connection Name:  Type a name for the connection

Connection Method:  Method to use to connect to the RDBMS

Parameters **Advanced**

Hostname:  Port:  Name or IP address of the server host. - TCP/IP port.

Username:  Name of the user to connect with.

Password:   The user's password. Will be requested later if it's not set.

Default Schema:  The schema to use as default schema. Leave blank to select it later.

You should click on Test Connection to make sure that the parameters you provided are correct, and then click OK button to create a new connection. Once you complete, you will see connections window as follows:

Step 6. Click the local database connection to connect to MySQL database server. Because we didn't provide the password in the previous step, MySQL asks us to enter the password for the root account. We enter our password and click OK button.

**Connect to MySQL Server**

**Please enter password for the following service:**

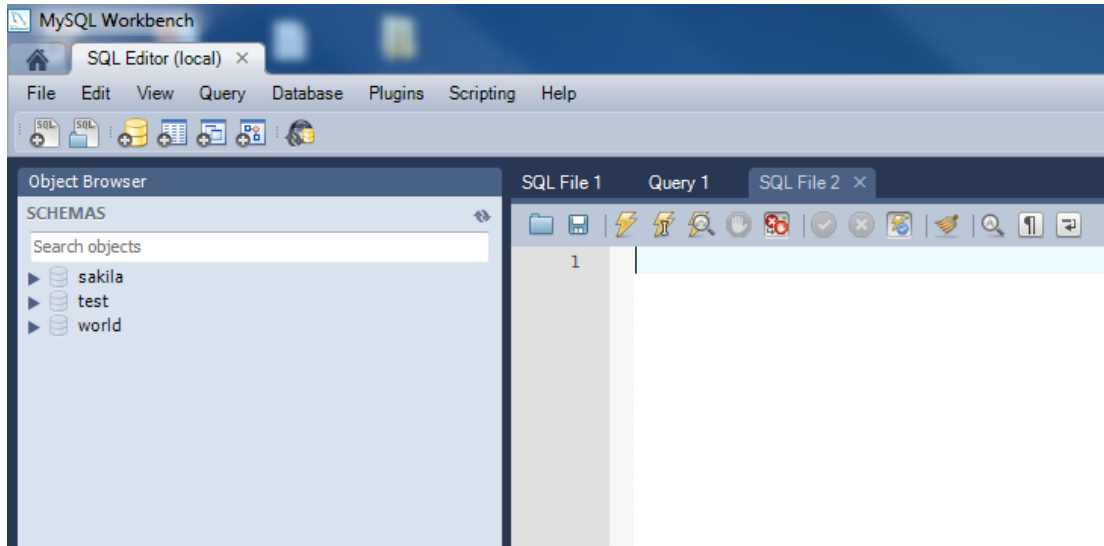
Service: Mysql@127.0.0.1:3306

User: root

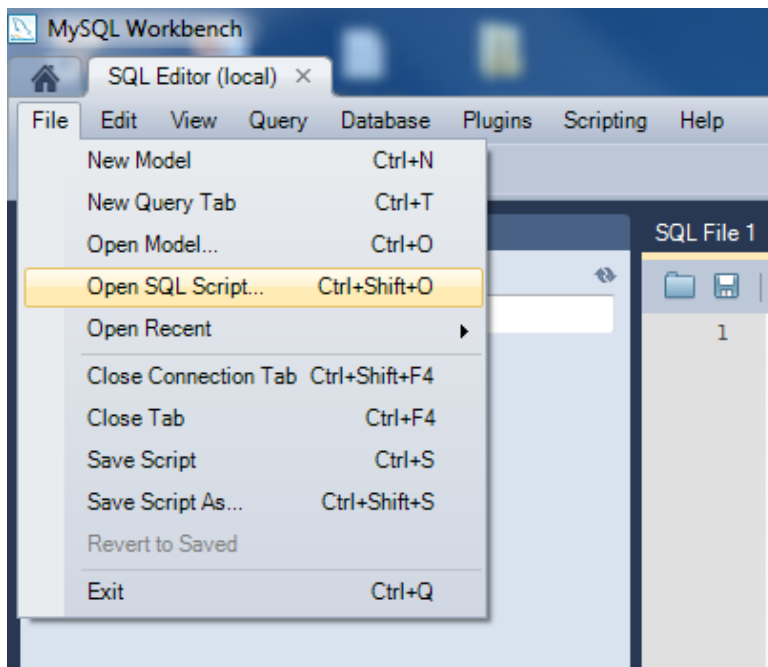
Password:

☒ Save password in vault

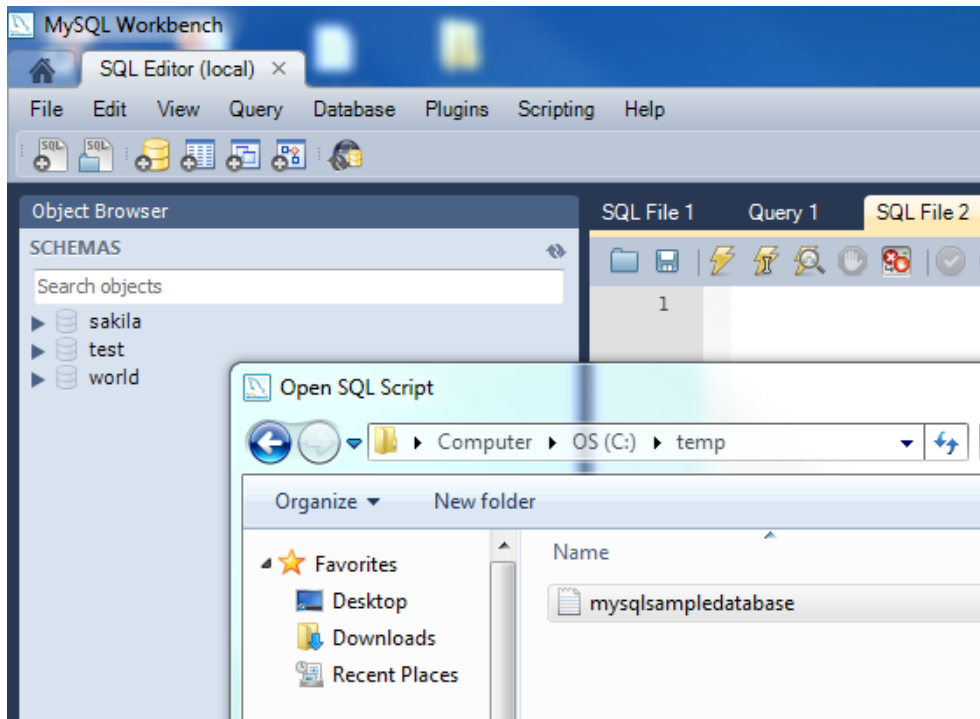
If you enter both user and password correctly, you will see the following window:



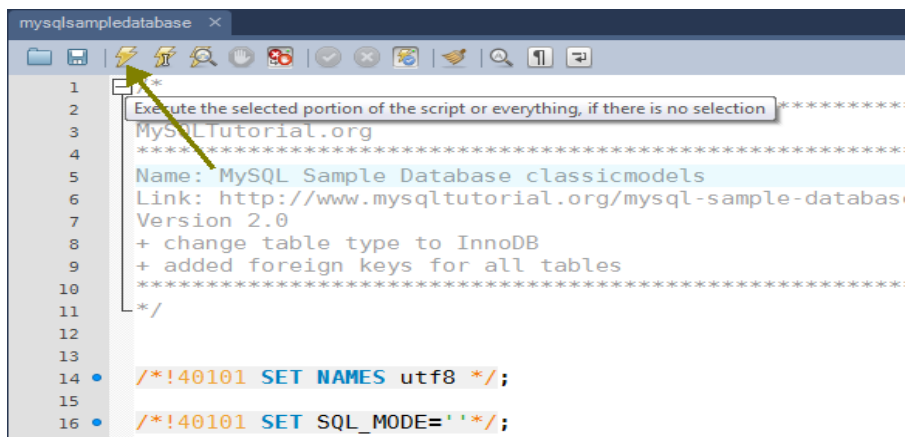
Step 7. Open SQL Script by choosing File > Open SQL Script or press the Ctrl+Shift+O keyboard shortcut.



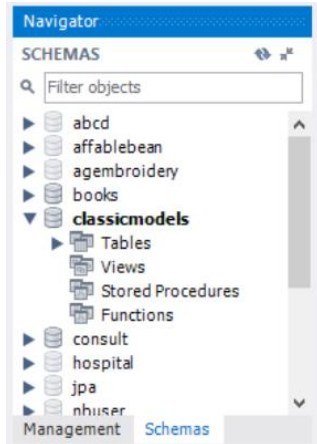
Step 8. Choose SQL Script File by selecting the file  
**C:\temp\mysqlprogramming\mysqlsampledatabase.sql**



Step 10. To execute SQL Script, you click execute button from the toolbar as following:



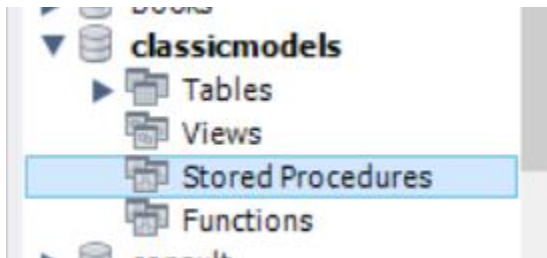
Step 11. Right click inside the Schemas panel and click Refresh All button to update the panel. The **classicmodels** database is loaded successfully into MySQL database server.



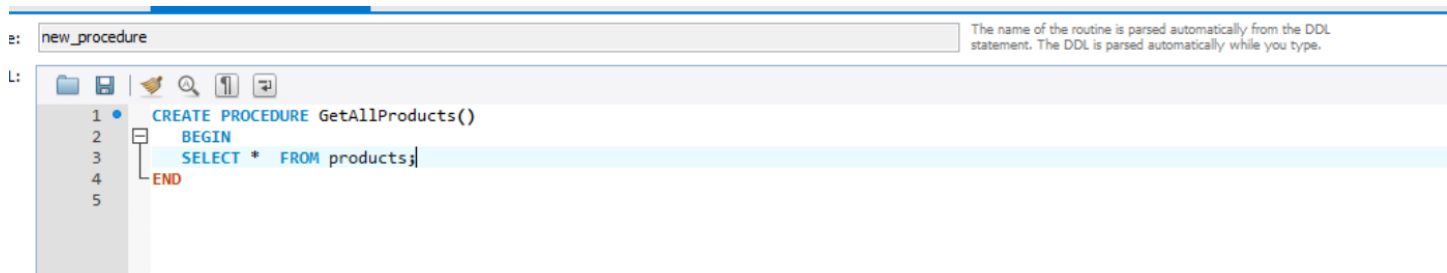
## MySQL Stored Procedure Example

For example, in MySQL Workbench, you can create a new stored procedure as follows:

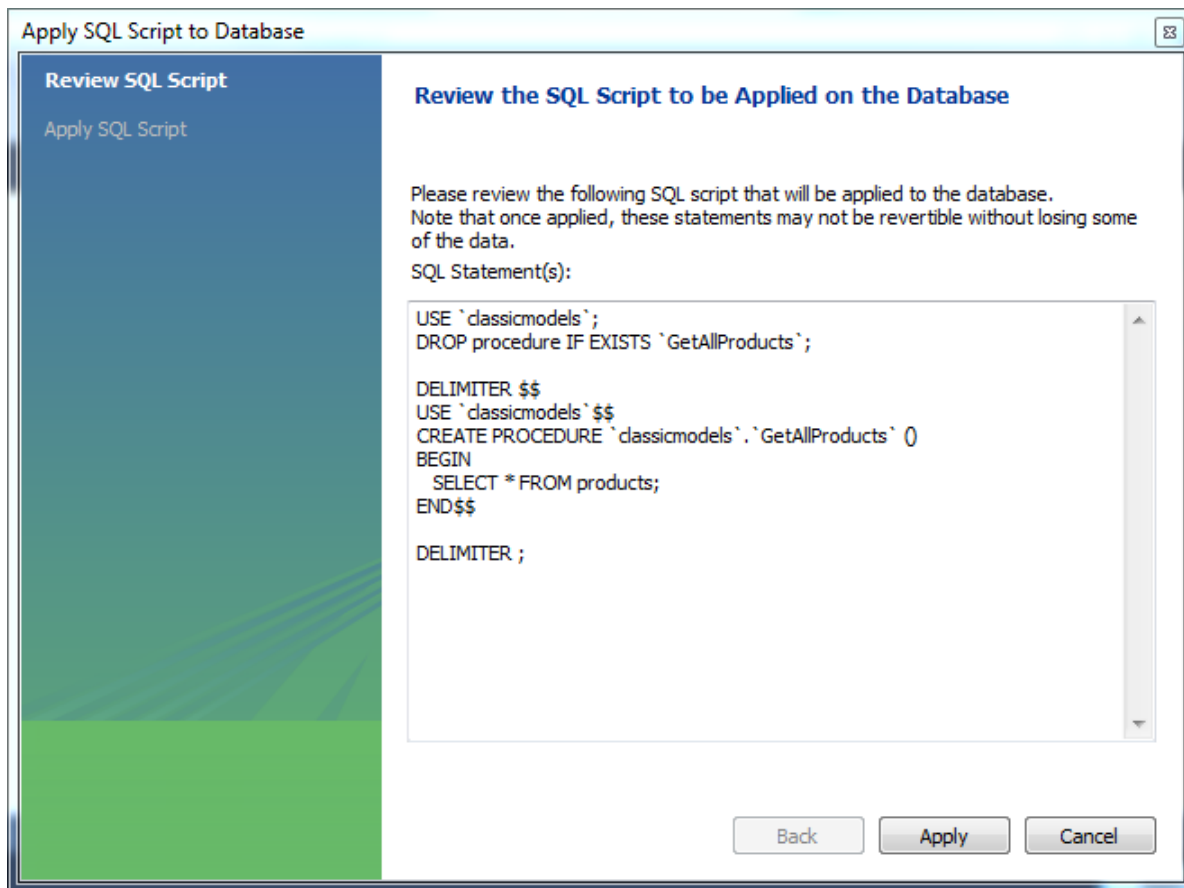
First, right mouse click on the Stored Procedures and choose “Create Procedure...” menu item.



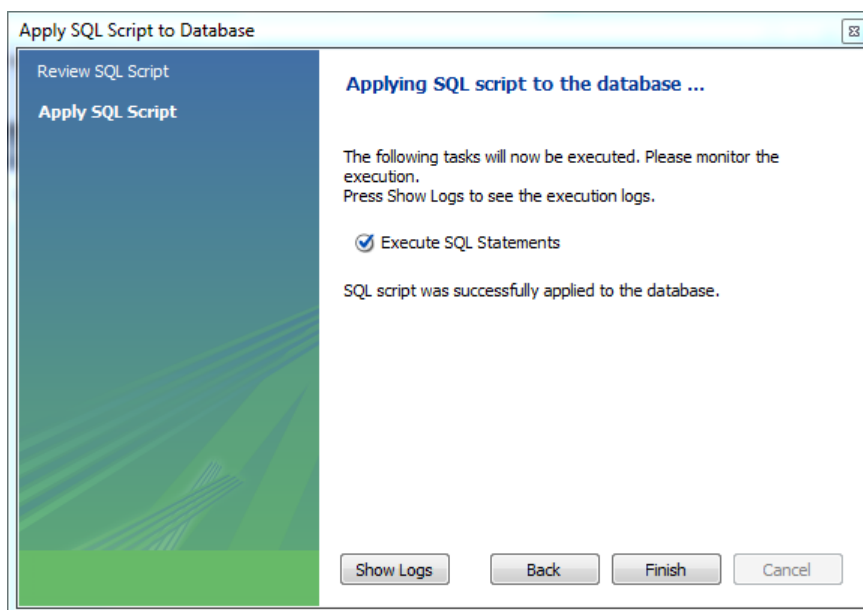
Next, enter the stored procedure code and click the Apply button ([storedprocedure.txt](#))



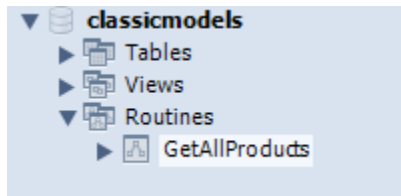
Then, you can review the code before MySQL stores it in the database. Click Apply button if everything is good.



After that, MySQL compiles and puts the stored procedure in the database catalog; click the Finish button.



Finally, you can see a new stored procedure created under *Stored Procedures* of the **classicmodels** database.



We have created a new stored procedure. Now, it's time to learn how to use it.

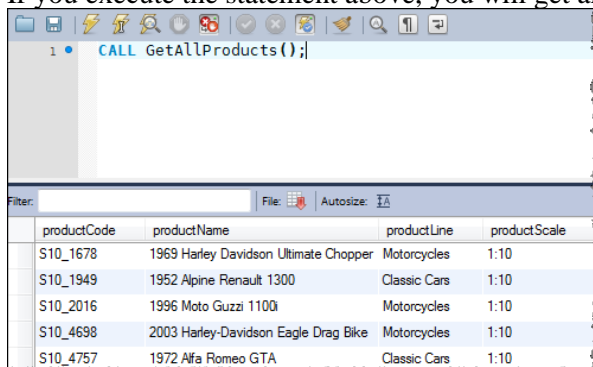
In order to call a stored procedure, you use the following SQL command:

```
CALL STORED_PROCEDURE_NAME();
```

You use the CALL statement to call a stored procedure e.g., to call the GetAllProducts() stored procedure, you use the following statement:

```
CALL GetAllProducts();
```

If you execute the statement above, you will get all products in the products table.



## MySQL Stored Function Example

Let's take a look at an example of using stored function. We will use the customers table in the [classics database](#) for the demonstration.

For example, in MySQL Workbench, you can create a new function as follows:

First, right mouse click on the Functions and choose "Create Functions..." menu item.

The following example is a function that returns the level of a customer based on credit limit. We use the IF statement to decide the credit limit. (**function.txt**)

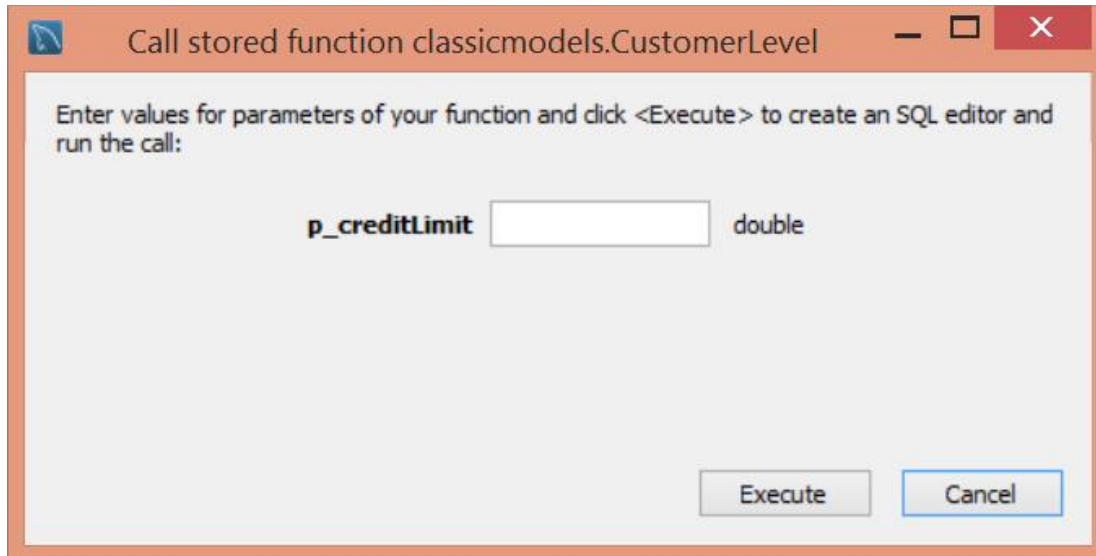
```
CREATE FUNCTION CustomerLevel(p_creditLimit double) RETURNS VARCHAR(10)
DETERMINISTIC
```

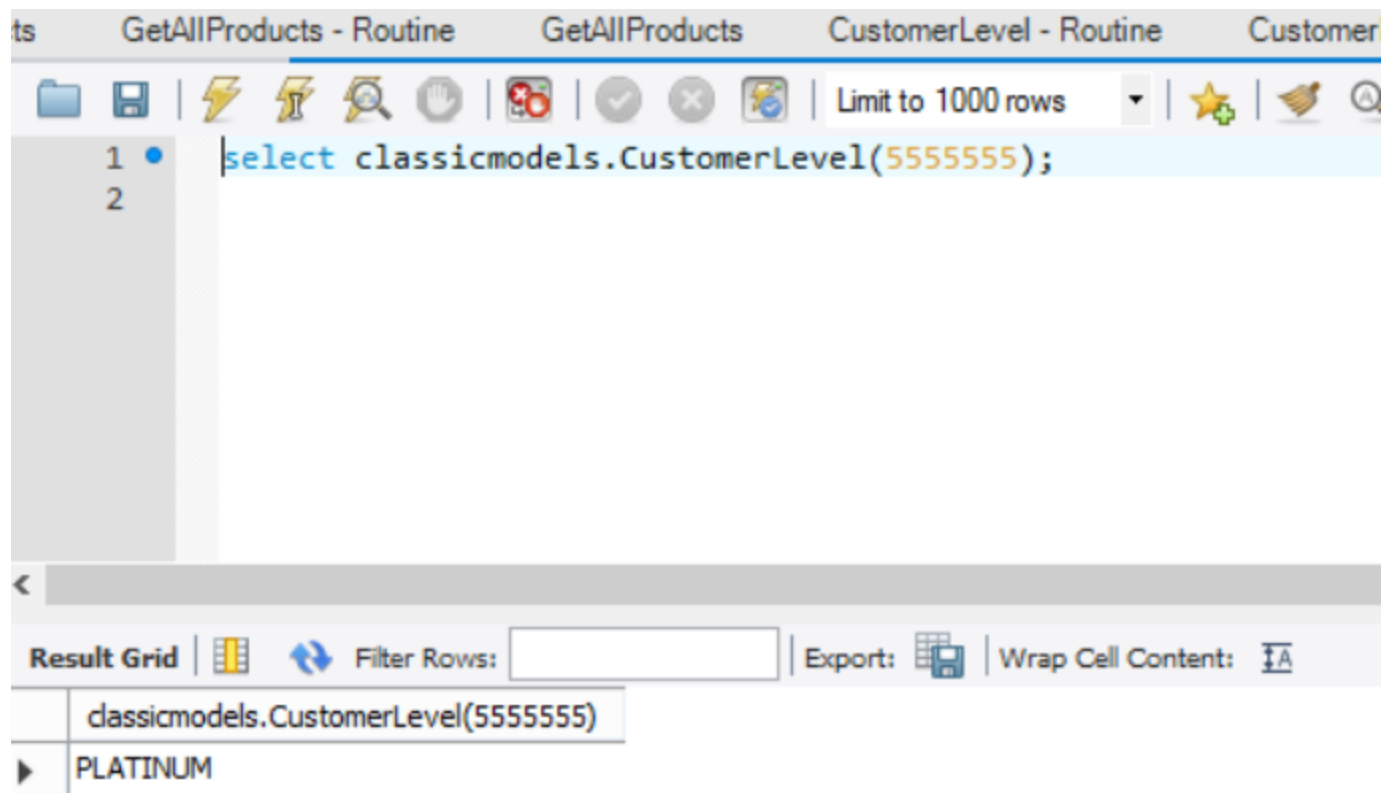


```
BEGIN
  DECLARE lvl varchar(10);
  IF p_creditLimit > 50000 THEN
SET lvl = 'PLATINUM';
  ELSEIF (p_creditLimit <= 50000 AND p_creditLimit >= 10000) THEN
    SET lvl = 'GOLD';
  ELSEIF p_creditLimit < 10000 THEN
    SET lvl = 'SILVER';
  END IF;
RETURN (lvl);
END
```

We can test the function as follows:

Click on function then hit the run icon. A dialog box will pop up requesting you to input a credit limit. Then hit Execute.





Now, we can call the CustomerLevel() in an SQL SELECT statement as follows: (**testfunction.txt**)

```
SELECT
  customerName, CustomerLevel(creditLimit)
FROM
  customers
ORDER BY customerName;
```

	customerName	CustomerLevel(creditLimit)
►	Alpha Cognac	PLATINUM
	American Souvenirs Inc	SILVER
	Amica Models & Co.	PLATINUM
	ANG Resellers	SILVER
	Anna's Decorations, Ltd	PLATINUM
	Anton Designs, Ltd.	SILVER
	Asian Shopping Network, Co	SILVER
	Asian Treasures, Inc.	SILVER
	Atelier graphique	GOLD
	Australian Collectables, Ltd	PLATINUM
	Australian Collectors, Co.	PLATINUM

We also rewrite the GetCustomerLevel() stored procedure that was originally developed using a MySQL IF statement as follows:

## BEFORE

```
CREATE PROCEDURE GetCustomerLevel(
    in p_customerNumber int(11),
    out p_customerLevel varchar(10))
BEGIN
    DECLARE creditlim double;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber =
p_customerNumber;

    IF creditlim > 50000 THEN
SET p_customerLevel = 'PLATINUM';

    ELSEIF (creditlim <= 50000 AND creditlim >= 10000) THEN
```

## AFTER

```
CREATE PROCEDURE GetCustomerLevel(
    IN p_customerNumber INT(11),
    OUT p_customerLevel varchar(10)
)
BEGIN
    DECLARE creditlim DOUBLE;

    SELECT creditlimit INTO creditlim
    FROM customers
    WHERE customerNumber = p_customerNumber;

    SELECT CUSTOMERLEVEL(creditlim)
    INTO p_customerLevel;

END
```

```
    SET p_customerLevel = 'GOLD';  
  
ELSEIF creditlim < 10000 THEN  
  
    SET p_customerLevel = 'SILVER';  
  
END IF;
```

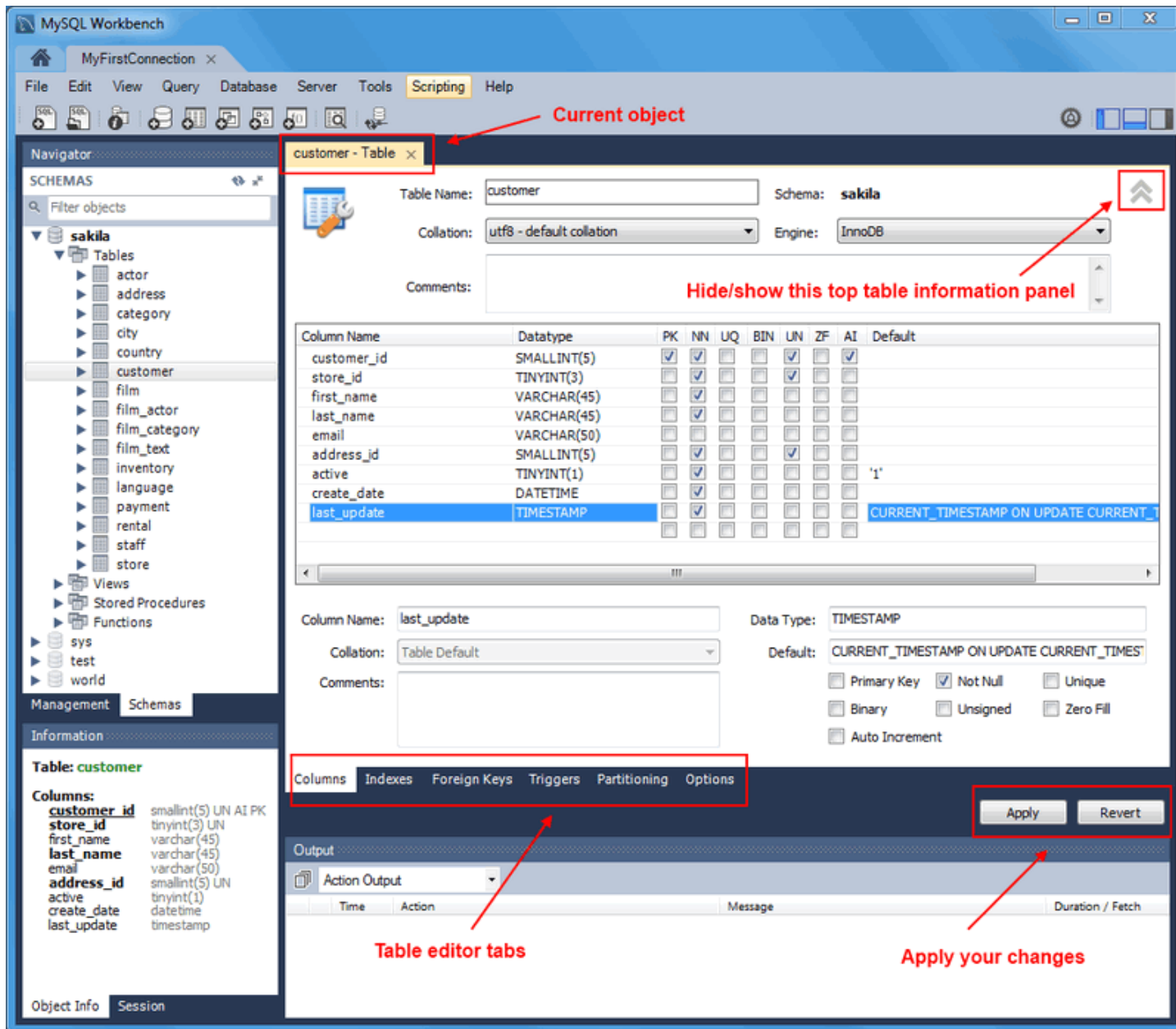
As you can see, the `GetCustomerLevel()` stored procedure is much more readable when using the `CustomerLevel()` stored function.

Notice that a stored function returns a single value only. If you include a `SELECT` statement without the `INTO` clause, you will get an error.

In addition, if a stored function contains `SQL` statements, you should not use it inside other `SQL` statements; otherwise, the stored function will slow down the speed of the query.

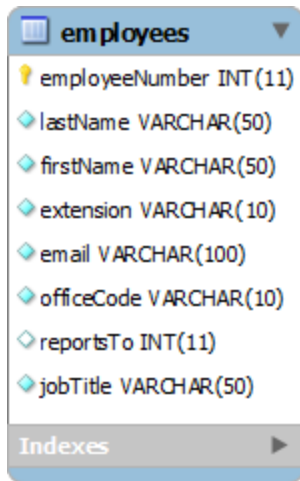
## MySQL trigger example

To create a trigger for an existing table, access the MySQL Table Editor, right-click on a table name in the Object Viewer (the left navigator panel that lists all schemas) and choose ALTER TABLE. This opens a new tab within the main SQL Editor window.



The Triggers tab opens a textbox to create or edit existing triggers. Click the Add Trigger Button.

Let's start creating a trigger in MySQL to log the changes of the employees table.



First, create a new table named employees\_audit to keep the changes of the employee table. The following statement creates the employee\_audit table. ([employeesaudittable.sql](#))

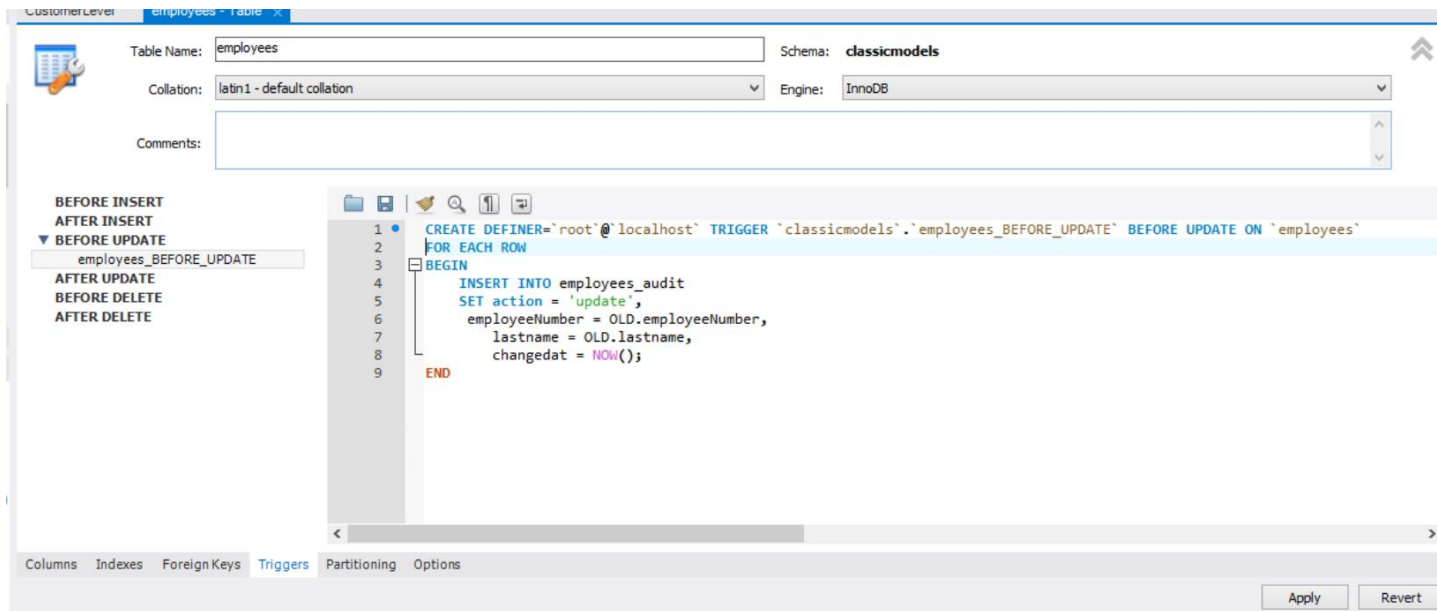
```
CREATE TABLE employees_audit (  
    id INT AUTO_INCREMENT PRIMARY KEY,  
    employeeNumber INT NOT NULL,  
    lastname VARCHAR(50) NOT NULL,  
    changedat DATETIME DEFAULT NULL,  
    action VARCHAR(50) DEFAULT NULL  
);
```

Next, create a BEFORE UPDATE trigger that is invoked before a change is made to the employees table.

Click the [+] to the right of the BEFORE UPDATE event. An editor window will open with the initial part of the trigger already created. Enter the following block of code to complete the trigger then hit Apply. ([employeesbeforeupdate.txt](#))

```
BEGIN  
    INSERT INTO employees_audit  
    SET action = 'update',  
        employeeNumber = OLD.employeeNumber,  
        lastname = OLD.lastname,  
        changedat = NOW();
```

END



Inside the body of the trigger, we used the OLD keyword to access employeeNumber and lastname column of the row affected by the trigger.

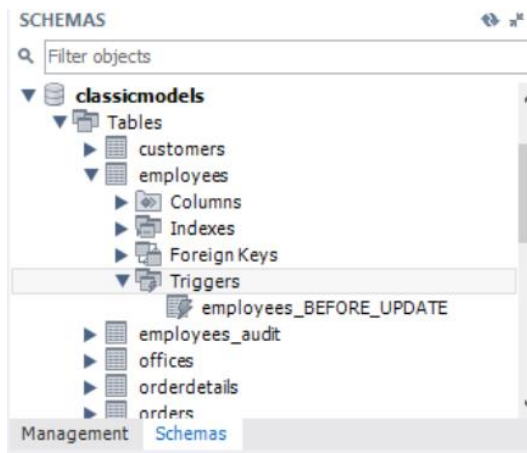
Notice that in a trigger defined for INSERT, you can use NEW keyword only. You cannot use the OLD keyword. However, in the trigger defined for DELETE, there is no new row so you can use the OLD keyword only. In the UPDATE trigger, OLD refers to the row before it is updated and NEW refers to the row after it is updated.

Then, to view all triggers in the current database, you use SHOW TRIGGERS statement as follows:

SHOW TRIGGERS;

Trigger	Event	Table	Statement	Timing	Created	sql_mode	Definer	character_set_client	collation_connection
employees_BEFORE_UPDATE	UPDATE	employees	BEGIN INSERT INTO employees_audit SE...	BEFORE		STRICT_TRANS_TABLES,NO_AUTO_CREATE_U...	root@localhost	utf8	utf8_general_ci

In addition, if you look at the schema using MySQL Workbench under the employees > triggers, you will see the employees\_BEFORE\_UPDATE trigger as shown in the screenshot below:



After that, update the employees table to check whether the trigger is invoked. ([testtrigger.sql](#))

```
UPDATE employees
SET
    lastName = 'Phan'
WHERE
    employeeNumber = 1056;
```

Finally, to check if the trigger was invoked by the UPDATE statement, you can query the employees\_audit table using the following query:

```
SELECT * FROM employees_audit;
```

The following is the output of the query:

Result Grid					
Filter Rows:					
	id	employeeNumber	lastname	changedat	action
▶	1	1056	Patterson	2016-11-22 08:13:01	update
★	NULL	NULL	NULL	NULL	NULL

As you see, the trigger was really invoked and it inserted a new row into the employees\_audit table.

#### What to put into the Dropbox:

- Screenshot of your select \* from employees\_audit execution to prove to me that the trigger worked properly.
- Your team's Collaboration document. You can find the template for that at BeachBoard | Contents | Student Helps | Lab Collaboration Document.