Assignment No.7

Title of Assignment: PL/SQL Stored Procedure and Stored Function.
Write a Stored Procedure namely proc_Grade for the categorization of student.
If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class. Write a PL/SQL block for using procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class)
Frame the separate problem statement for writing PL/SQL Stored Procedure And function, inline with above statement.

Objective:

- To learn about **MySQL stored procedures**, their advantages and disadvantages.
- To learn about variables in stored procedure, its scope, how to declare and use variables.
- To learn how to write MySQL stored procedures with parameters.
- To learn how to use **MySQL IF statement** to execute a block of SQL code based on conditions.
- To learn how to use **MySQL CASE** statements to construct complex conditionals.
- TO learn how to use various loop statements in MySQL including WHILE, REPEAT and LOOP to run a block of code repeatedly based on a condition.
- To learn how to create stored functions using CREATE FUNCTION statement

Relevant Theory

Definition of stored procedures

A stored procedure is a segment of declarative SQL statements stored inside the database catalog. A stored procedure can be invoked by triggers, other stored procedures or applications such as Java, C#, PHP, etc.
A stored procedure that calls itself is known as a recursive stored procedure. Most database management system supports recursive stored procedures.
However MySQL does not support it very well. You should check your version of MySQL database before implementing recursive stored procedures in MySQL.

Stored Procedures in MySQL

MySQL is known as the most popular open source RDBMS which is widely used by both community and enterprise. However, during the first decade of its existence, it did not support stored procedures, stored functions, triggers and events. Since MySQL version 5.0, those features were added to MySQL database engine to make it more flexible and powerful.

MySQL stored procedures advantages

- Typically stored procedures help increase the performance of the applications. Once created, stored procedures are compiled and stored in the database. However MySQL implements the stored procedures slightly different. MySQL stored procedures are compiled on demand. After compiling a stored procedure, MySQL puts it to a cache. And MySQL maintains its own stored procedure cache for every single connection. If an application uses a stored procedure multiple times in a single connection, the compiled version is used, otherwise the stored procedure works like a query.

- A stored procedure helps reduce the traffic between application and database server because instead of sending multiple lengthy SQL statements, the application has to send only name and parameters of the stored procedure.
- Stored procedures are reusable and transparent to any applications. Stored procedures expose the database interface to all applications so that developers don't have to develop functions that are already supported in stored procedures.
- Stored procedures are secure. Database administrator can grant appropriate permissions to applications that access stored procedures in the database without giving any permission on the underlying database tables.

Besides those advantages, stored procedures have their own disadvantages, which you should be aware of before using the store procedures.

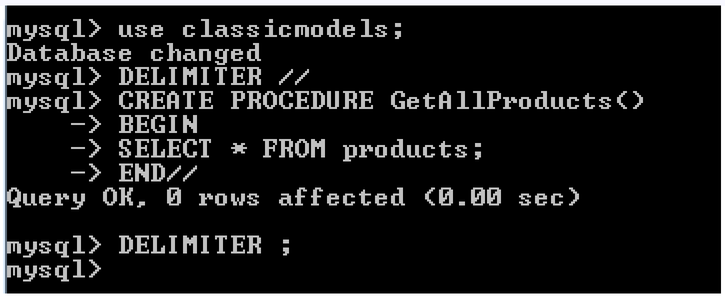
MySQL stored procedures disadvantages

- If you use a lot of stored procedures, the memory usage of every connection that is using those stored procedures will increase substantially. In addition, if you overuse a large number of logical operations inside store procedures, the CPU usage will also increase because database server is not well-designed for logical operations.
- A constructs of stored procedures make it more difficult to develop stored procedures that have complicated business logic.
- It is difficult to debug stored procedures. Only few database management systems allow you to debug stored procedures. Unfortunately, MySQL does not provide facilities for debugging stored procedures.
- It is not easy to develop and maintain stored procedures. Developing and maintaining stored procedures are often required specialized skill set that not all application developers possess. This may lead to problems in both application development and maintenance phases.

Writing the first MySQL stored procedure

We are going to develop a simple stored procedure named GetAllProducts() to help you get familiar with the syntax. The GetAllProducts() stored procedure selects all products from the products table.
Launch the mysql client tool and type the following commands:

```
DELIMITER //
 CREATE PROCEDURE GetAllProducts()
   BEGIN
   SELECT *  FROM products;
   END //
 DELIMITER ;
```



```
mysql> use classicmodels;
Database changed
mysql> DELIMITER //
mysql> CREATE PROCEDURE GetAllProducts()
    -> BEGIN
    -> SELECT * FROM products;
    -> END//
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
```

Let's examine the stored procedure in greater detail:

- The first command is DELIMITER //, which is not related to the stored procedure syntax. The DELIMITER statement changes the standard delimiter which is semicolon (;) to another. In this case, the delimiter is changed from the semicolon(;) to double-slashes //.
- Why do we have to change the delimiter? Because we want to pass the stored procedure to the server as a whole rather than letting mysql tool to interpret eachstatement at a time. Following the END keyword, we use the delimiter // to indicate the end of the stored procedure. The last command (DELIMITER;*)* changes the delimiter back to the standard one.
- We use the CREATE PROCEDURE statement to create a new stored procedure. We specify the name of stored procedure after the CREATE PROCEDURE statement.
- In this case, the name of the stored procedure is GetAllProducts. We put the parentheses after the name of the stored procedure.
- The section between BEGIN and END is called the body of the stored procedure. You put the declarative SQL statements in the body to handle business logic.
- In this stored procedure, we use a simple SELECT statement to query data from the products table.

Calling stored procedures
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 GetAllProductsstored 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 Procedure Variables

A variable is a named data object whose value can change during the stored procedure execution. We typically use the variables in stored procedures to hold the immediate results. These variables are local to the stored procedure.
You must declare a variable before you can use it.

Declaring variables
To declare a variable inside a stored procedure, you use the DECLARE statement as follows:

```
DECLARE variable_name datatype(size) DEFAULT default_value;
```

Let's examine the statement above in more detail:

- First, you specify the variable name after the DECLARE keyword. The variable name must follow the naming rules of MySQL table column names.
- Second, you specify the data type of the variable and its size. A variable can have anyMySQL data types such as INT, VARCHAR, DATETIME, etc.
- Third, when you declare a variable, its initial value is NULL. You can assign the variable a default value by using DEFAULT keyword.

For example, we can declare a variable named total_sale with the data type INT and default value 0 as follows:

DECLARE total_sale INT DEFAULT 0

MySQL allows you to declare two or more variables that share the same data type using a single DECLARE statement as following:

DECLARE x, y INT DEFAULT 0

We declared two INT variables x and y , and set their default values to zero.

Assigning variables
Once you declared a variable, you can start using it. To assign a variable another value, you use the SET statement, for example:

```
DECLARE total_count INT DEFAULT 0
SET total_count = 10;
```

The value of the total_count variable is 10 after the assignment.
Besides the SET statement, you can use SELECT INTO statement to assign the result of a query to a variable. Notice that the query must return a scalar value.

```
DECLARE total_products INT DEFAULT 0

SELECT COUNT(*) INTO total_products
FROM products
```

In the example above:

- First, we declare a variable named total_products and initialize its value to 0.
- Then, we used the SELECT INTO statement to assign the total_products variable the number of products that we selected from the products from the products table.

Variables scope

- A variable has its own scope, which defines its life time. If you declare a variable inside a stored procedure, it will be out of scope when the END statement of stored procedure reached.
- If you declare a variable inside BEGIN END block, it will be out of scope if the END is reached. You can declare two or more variables with the same name in different scopes because a variable is only effective in its own scope. However, declaring variables with the same name in different scopes is not good programming practice.
- A variable that begins with the @ sign at the beginning is session variable. It is available and accessible until the session ends.

Introduction to MySQL stored procedure parameters
Almost stored procedures that you develop require parameters. The parameters make the stored procedure more flexible and useful. In MySQL, a parameter has one of three modes IN, OUT or INOUT.

- **IN –** is the default mode. When you define an IN parameter in a stored procedure, the calling program has to pass an argument to the stored procedure. In addition, the value of an IN parameter is protected. It means that even the value of the IN parameter is changed inside the stored procedure, its original value is retained after the stored procedure ends. In other words, the stored procedure only works on the copy of the IN parameter.

- **OUT –** the value of an OUT parameter can be changed inside the stored procedure and its new value is passed back to the calling program. Notice that the stored procedure cannot access the initial value of the OUT parameter when it starts.
- **INOUT –** an INOUT parameter is the combination of IN parameter and OUT parameter. It means that the calling program may pass the argument, and the stored procedure can modify the INOUT parameter and pass the new value back to the calling program.

The syntax of defining a parameter in the stored procedures is as follows:

```
MODE param_name param_type(param_size)
```

The MODE could be IN, OUT or INOUT , depending on the purpose of parameter in the stored procedure.
The param_name is the name of the parameter. The name of parameter must follow the naming rules of the column name in MySQL.
Followed the parameter name is its data type and size. Like a variable, the data type of the parameter can by any MySQL data type.
Each parameter is separated by a comma (,) if the stored procedure has more than one parameter.
Let’s practice with some examples to get a better understanding.

MySQL stored procedure parameter examples
1. IN parameter example
The following example illustrates how to use the IN parameter in the GetOfficeByCountrystored procedure that selects offices located in a specified country.

```
DELIMITER //
CREATE PROCEDURE GetOfficeByCountry(IN countryName VARCHAR(255))
   BEGIN
      SELECT *
       FROM offices
       WHERE country = countryName;
   END //
DELIMITER ;
```

The countryName is the IN parameter of the stored procedure. Inside the stored procedure, we select all offices that locate in the country specified by the countryName parameter.
Suppose, you want to get all offices in the USA, you just need to pass a value (USA) to the stored procedure as follows:

```
CALL GetOfficeByCountry('USA')
```

To get all offices in France, you pass the France literal string to the GetOfficeByCountrystored procedure as follows:

```
CALL GetOfficeByCountry('France')
```

2. OUT parameter example

The following stored procedure returns the number of orders by order status. It has two parameters:

- orderStatus: IN parameter that is the order status which you want to count the orders.
- total: OUT parameter that stores the number of orders for a specific order status.

The following is the source code of the CountOrderByStatus stored procedure.

```
DELIMITER $$
CREATE PROCEDURE CountOrderByStatus(
        IN orderStatus VARCHAR(25),
        OUT total INT)
BEGIN
    SELECT count(orderNumber)
    INTO total
    FROM orders
    WHERE status = orderStatus;
END$$
DELIMITER ;
```

To get the number of shipped orders, we call the CountOrderByStatus stored procedure and pass the order status as Shipped, and also pass an argument (@total) to get the return value.

```
CALL CountOrderByStatus('Shipped',@total);

SELECT @total;
```

To get the number of orders that are in process, we call the CountOrderByStatus stored procedure as follows:

```
CALL CountOrderByStatus('in process',@total);

SELECT @total AS  total_in_process;
```

3. INOUT parameter example

The following example demonstrates how to use INOUT parameter in the stored procedure.

```
DELIMITER $$
CREATE PROCEDURE set_counter(INOUT count INT(4),IN inc INT(4))
BEGIN
    SET count = count + inc;
END$$
DELIMITER ;
```

How it works

The set_counter stored procedure accepts one INOUT parameter (count) and one IN parameter (inc).

Inside the stored procedure, we increase the counter (count) by the value of the incparameter.

See how we call the set_counter stored procedure:

```
SET @counter = 1;
CALL set_counter(@counter,1); -- 2
CALL set_counter(@counter,1); -- 3
CALL set_counter(@counter,5); -- 8
SELECT @counter; -- 8
```

MySQL IF Statement

The MySQL IF statement allows you to execute a set of SQL statements based on a certain condition or value of an expression. To form an expression in MySQL, you can combine literals,variables, operators, and even functions. An expression can return three value TRUE, FALSE or NULL.

MySQL IF statement syntax

The following illustrates the syntax of the IF statement:

```
IF if_expression THEN commands

   [ELSEIF elseif_expression THEN commands]
   [ELSE commands]
   END IF;
```

If the *if_expression* evaluates to TRUE the commands in the IF branch will execute. If it evaluates to FALSE, MySQL will check the *elseif_expression* and execute the commands in ELSEIF branch if the *elseif_expression* evaluates to TRUE.
The IF statement may have multiple ELSEIF branches to check multiple expressions. If no expression evaluates to TRUE, the commands in the ELSE branch will execute.

MySQL IF statement examples

Let’s take a look at an example of how to use MySQL IF statements.

```
DELIMITER $$

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

END$$
```

We pass customer number to the stored procedure to get customer level based on credit limit. We use IF ELSEIF and ELSE statement to check customer credit limit against multiple values.

MySQL CASE Statement
Besides the IF statement, MySQL also provides an alternative conditional statement called MySQL CASE.
The MySQL CASE statement makes the code more readable and efficient.
There are two forms of the CASE statements: simple and searched CASE statements.

Simple CASE statement
Let's take a look at the syntax of the simple CASE statement:

```
ASE  case_expression
   WHEN when_expression_1 THEN commands
   WHEN when_expression_2 THEN commands
   ...
   ELSE commands
END CASE;
```

You use the simple CASE statement to check the value of an expression against a set of unique values.

The case_expression can be any valid expression. We compare the value of thecase_expression with when_expression in each WHEN clause

e.g.,when_expression_1, when_expression_2, etc. If the value of the case_expressionand when_expression_n are equal, the commands in the corresponding WHEN branch executes.
In case none of the when_expression in the WHEN clause matches the value of thecase_expression, the commands in the ELSE clause will execute. The ELSE clause is optional. If you omit the ELSE clause and no match found, MySQL will raise an error.
The following example illustrates how to use the simple CASE statement:

```
DELIMITER $$

CREATE PROCEDURE GetCustomerShipping(
        in  p_customerNumber int(11),
        out p_shiping        varchar(50))
BEGIN
    DECLARE customerCountry varchar(50);

    SELECT country INTO customerCountry
    FROM customers
    WHERE customerNumber = p_customerNumber;

    CASE customerCountry
        WHEN  'USA' THEN
           SET p_shiping = '2-day Shipping';
        WHEN 'Canada' THEN
           SET p_shiping = '3-day Shipping';
        ELSE
           SET p_shiping = '5-day Shipping';
    END CASE;
```

```
END$$
```

How the stored procedure works.

- The GetCustomerShipping stored procedure accepts customer number as an INparameter and returns shipping period based on the country of the customer.
- Inside the stored procedure, first we get the country of the customer based on the input customer number. Then we use the simple CASE statement to compare the country of the customer to determine the shipping period. If the customer locates in USA, the shipping period is 2-day shipping. If the customer is in Canada, the shipping period is 3-day shipping. The customers from other countries have 5-day shipping.

The following is the test script for the stored procedure above:

```
SET @customerNo = 112;

SELECT country into @country
FROM customers
WHERE customernumber = @customerNo;

CALL GetCustomerShipping(@customerNo,@shipping);

SELECT @customerNo AS Customer,
       @country    AS Country,
       @shipping   AS Shipping;
```

Searched CASE statement

The simple CASE statement only allows you match a value of an expression against a set of distinct values. In order to perform more complex matches such as ranges you use the *searched*CASE statement. The searched CASE statement is equivalent to the IF statement, however its construct is much more readable.

The following illustrates the syntax of the searched CASE statement:

```
CASE
    WHEN condition_1 THEN commands
    WHEN condition_2 THEN commands
    ...
    ELSE commands
END CASE;
```

MySQL evaluates each condition in the WHEN clause until it finds a condition whose value is TRUE, then corresponding commands in the THEN clause will execute.

If no condition is TRUE , the command in the ELSE clause will execute. If you don't specify the ELSE clause and no condition is TRUE, MySQL will issue an error message.

MySQL does not allow you to have empty commands in the THEN or ELSE clause. If you don't want to handle the logic in the ELSE clause while preventing MySQL raise an error, you can put an empty BEGIN END block in the ELSE clause.

The following example demonstrates using searched CASE statement to find customer level SILVER, GOLD or PLATINUM based on customer's credit limit.

```
DELIMITER $$
```

```
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;

    CASE
        WHEN creditlim > 50000 THEN
           SET p_customerLevel = 'PLATINUM';
        WHEN (creditlim <= 50000 AND creditlim >= 10000) THEN
           SET p_customerLevel = 'GOLD';
        WHEN creditlim < 10000 THEN
           SET p_customerLevel = 'SILVER';
    END CASE;

END$$
```

If the credit limit is

- greater than 50K, then the customer is PLATINUM customer
- less than 50K and greater than 10K, then the customer is GOLD customer
- less than 10K, then the customer is SILVER customer.

We can test our stored procedure by executing the following test script:

```
CALL GetCustomerLevel(112,@level);
SELECT @level AS 'Customer Level';
```

Loop in Stored Procedures
MySQL provides loop statements that allow you to execute a block of SQL code repeatedly based on a condition. There are three loop statements in MySQL: WHILE, REPEAT and LOOP.

WHILE loop
The syntax of the WHILE statement is as follows:

```
WHILE expression DO
   Statements
END WHILE
```

The WHILE loop checks the expression at the beginning of each iteration. If the expression valuates to TRUE, MySQL will executes statements between WHILE and END WHILE until the expression evaluates to FALSE. The WHILE loop is called pretest loop because it checks the expression before the statements execute.

Here is an example of using the WHILE loop statement in stored procedure:

```
DELIMITER $$
 DROP PROCEDURE IF EXISTS WhileLoopProc$$
 CREATE PROCEDURE WhileLoopProc()
```

```
      BEGIN
             DECLARE x  INT;
             DECLARE str  VARCHAR(255);
             SET x = 1;
             SET str =  '';
             WHILE x  <= 5 DO
                        SET  str = CONCAT(str,x,',');
                        SET  x = x + 1;
             END WHILE;
             SELECT str;
      END$$

   DELIMITER ;
```

In the stored procedure above:

- First, we build str string repeatedly until the value of the x variable is greater than 5.
- Then, we display the final string using the SELECT statement.

Notice that if we don't initialize x variable, its default value is NULL. Therefore the condition in the WHILE loop statement is always TRUE and you will have a indefinite loop, which is not expected.

REPEAT loop

The syntax of the REPEAT loop statement is as follows:

```
REPEAT
Statements;
UNTIL expression
END REPEAT
```

First MySQL executes the statements, and then it evaluates the expression. If the expression evaluates to TRUE, MySQL executes the statements repeatedly until the expression evaluates to FALSE.

Because the REPEAT loop statement checks the expression after the execution of statements therefore the REPEAT loop statement is also known as post-test loop.

We can rewrite the stored procedure that uses WHILE loop statement above using the REPEAT loop statement:

```
DELIMITER $$
 DROP PROCEDURE IF EXISTS RepeatLoopProc$$
 CREATE PROCEDURE RepeatLoopProc()
      BEGIN
             DECLARE x  INT;
             DECLARE str  VARCHAR(255);
             SET x = 1;
             SET str =  '';
             REPEAT
                        SET  str = CONCAT(str,x,',');
                        SET  x = x + 1;
             UNTIL x  > 5
             END REPEAT;
             SELECT str;
      END$$
 DELIMITER ;
```

It is noticed that there is no delimiter semicolon (;) in the UNTIL expression.

LOOP, LEAVE and ITERATE Statements

The LEAVE statement allows you to exit the loop immediately without waiting for checking the condition.

The LEAVE statement works like the break statement in other languages such as PHP, C/C++, Java, etc.

The ITERATE statement allows you to skip the entire code under it and start a new iteration. The ITERATE statement is similar to the continue statement in PHP, C/C++, Java, etc.

MySQL also gives you a LOOP statement that allows you to execute a block of code repeatedly with an additional flexibility of using a loop label.

The following is an example of using the LOOP loop statement.

```
DELIMITER $$
 DROP PROCEDURE IF EXISTS LOOPLoopProc$$
 CREATE PROCEDURE LOOPLoopProc()
      BEGIN
              DECLARE x  INT;
              DECLARE str  VARCHAR(255);
              SET x = 1;
              SET str =  '';
              loop_label:  LOOP
                          IF  x > 10 THEN
                              LEAVE  loop_label;
                          END  IF;
                          SET  x = x + 1;
                          IF  (x mod 2) THEN
                              ITERATE  loop_label;
                          ELSE
                              SET  str = CONCAT(str,x,',');
                          END  IF;

              END LOOP;
              SELECT str;
      END$$
 DELIMITER ;
```

The stored procedure only constructs string with even numbers e.g., 2, 4, 6, etc.

- We put a loop_label loop label before the LOOP statement.
- If the value of x is greater than 10, the loop is terminated because of the LEAVE statement.
- If the value of the x is an odd number, the ITERATE statement ignores everything below it and starts a new iteration.
- If the value of the x is an even number, the block in the ELSE statement will build the string with even numbers.

MySQL Stored Function

A stored function is a special kind stored program that returns a single value. You use stored functions to encapsulate common formulas or business rules that may be reusable among SQL statements or stored programs.

Different from a stored procedure, you can use a stored function in SQL statements wherever an expression is used. This helps improve the readability and maintainability of the procedural code.

MySQL stored function syntax

The following illustrates the simplest syntax for creating a new stored function:

```
CREATE FUNCTION function_name(param1,param2,…)
    RETURNS datatype
   [NOT] DETERMINISTIC
 statements
```

First, you specify the name of the stored function after CREATE FUNCTION keywords.
Second, you list all parameters of the stored function. By default, all parameters are implicitly IN parameters. You cannot specify IN, OUT or INOUT modifiers to the parameters.
Third, you must specify the data type of the return value in the RETURNS statement. It can be any valid MySQL data types.
Fourth, for the same input parameters, if the stored function returns the same result, it is considered deterministic and not deterministic otherwise.
You have to decide whether a stored function is deterministic or not. If you declare it incorrectly, the stored function may produced an unexpected result, or the available optimization is not used which degrade the performance.
Fifth, you write the code in the statements section. It can be a single statement or a compound statement. Inside the statements section, you have to specify at least one RETURN statement.
The RETURN statement returns a value to the caller. Whenever the RETURN statement is reached, the stored function's execution is terminated immediately.

MySQL stored function example

The following example is a function that returns level of customer based on credit limit.

```
DELIMITER $$

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
```

Now we can call the CustomerLevel() in an SQL SELECT statement as follows:

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

We also rewrite the GetCustomerLevel() stored procedure that we developed in the MySQL IF statement as follows:

```
DELIMITER $$

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
```

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 statementwithout 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 cause the performance of the SQL statements to degrade.

Exercises:

Conclusion:

- In this Assignment, we have learned how to write a simple stored procedure by using the CREATE PROCEDURE statement and call it by using the CALL statement.
- We have shown you how to declare a variable inside stored procedures and discussed about the variable scopes.

- We have shown you how to define parameters in stored procedures, and introduced you to different parameter modes including IN, OUT and INOUT.
- We have learned how to use MySQL IF statement to execute a block of SQL code based on conditions.
- We've shown you how to use two forms of the MySQL CASE statements including simple CASE statement and searched CASE statement.
- WE have learned various MySQL loop statements to execute a block of code repeatedly based on a condition.