DATABASE MANAGEMENT SYSTEMS

Subject Teacher: Zartasha Baloch

STORED FUNCTIONS & PROCEDURES

Lecture # 36, 37 & 38

Disclaimer: The material used in this presentation to deliver the lecture i.e., definitions/text and pictures/graphs etc. does not solely belong to the author/presenter. The presenter has gathered this lecture material from various sources on web/textbooks. Following sources are especially acknowledged:

1. Connolly, Thomas M., and Carolyn E. Begg. *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.
2. *https://www.tutorialspoint.com*
3. *https://www.oracle.com*
4. Greenberg, Nancy, and Instructor Guide PriyaNathan. "Introduction to Oracle9i: SQL." ORACLE, USA (2001).

OBJECTIVES

After completing this lesson, you should be able to do the following:

- Describe stored functions and procedures
- Creating and using procedures
- Creating and using functions

PL/SQL BLOCK TYPES

- Anonymous Procedure Function

```
[DECLARE]


BEGIN
  --statements

[EXCEPTION]

END;
```

```
PROCEDURE name
IS

BEGIN
  --statements

[EXCEPTION]

END;
```

```
FUNCTION name
RETURN datatype
IS
BEGIN
  --statements
  RETURN value;
[EXCEPTION]

END;
```

STORED PROCEDURES AND FUNCTIONS

- Stored procedures and functions (subprograms) can be compiled and stored in an Oracle Database, ready to be executed.
- Once compiled, it is a schema object known as a stored procedure or stored function, which can be referenced or called any number of times by multiple applications connected to Oracle Database.
- Both stored procedures and functions can accept parameters when they are executed (called).
- To execute a stored procedure or function, you only need to include its object name.

WHAT IS A PROCEDURE?

- A procedure:
  - Is a type of subprogram that performs an action
  - Can be stored in the database as a schema object
  - Promotes reusability and maintainability

SYNTAX FOR CREATING PROCEDURES

CREATE [OR REPLACE] PROCEDURE ***procedure_name***

[(*parameter1* [*mode*] *datatype1*, *parameter2* [*mode*] *datatype2*, ...)]

IS|AS

[*local_variable_declarations*; …]

BEGIN

-- *actions;*

END [*procedure_name*];

WHAT ARE PARAMETERS?

Parameters:

- **Are declared after the subprogram name in the PL/SQL header**
- **Pass or communicate data between the caller and the subprogram**
- **Are used like local variables but are dependent on their parameter-passing mode:**
  - **An IN parameter (the default) provides values for a subprogram to process.**
  - **An OUT parameter returns a value to the caller.**
  - **An IN OUT parameter supplies an input value, which may be returned (output) as a modified value.**

SUMMARY OF PARAMETER MODES

IN	OUT	IN OUT
Default mode	Must be specified	Must be specified
Value is passed into subprogram	Returned to calling environment	Passed into subprogram; returned to calling environment
Formal parameter acts as a constant	Uninitialized variable	Initialized variable
Actual parameter can be a literal, expression, constant, or initialized variable	Must be a variable	Must be a variable
Can be assigned a default value	Cannot be assigned a default value	Cannot be assigned a default value

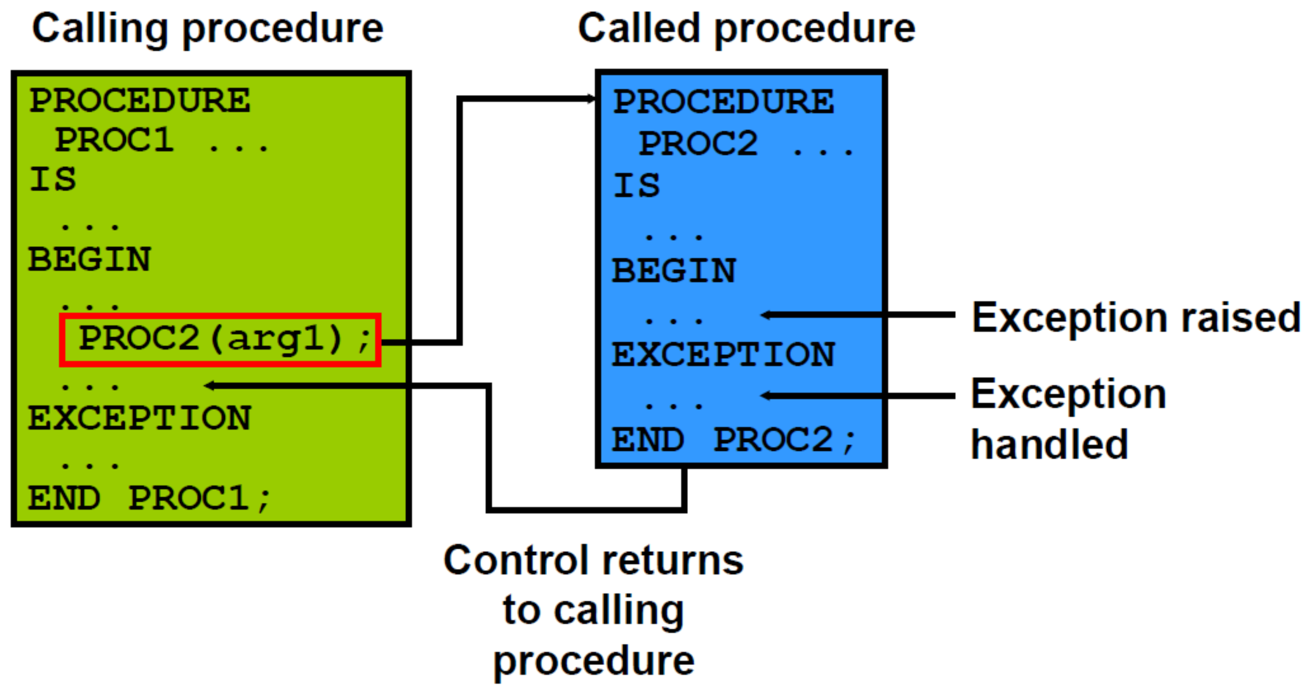
STORED PROCEDURE EXAMPLE

An example of a simple stored procedure that displays current date.

```
CREATE OR REPLACE PROCEDURE today_is AS
BEGIN
-- display the current system date in long format
  DBMS_OUTPUT.PUT_LINE( 'Today is ' || TO_CHAR(SYSDATE, 'DL') );
END today_is;
/
-- to call the procedure today_is, you can use the following block
BEGIN
  today_is(); -- the parentheses are optional here
END;
/
```

EXAMPLE-2

HANDLED EXCEPTION



```
PROCEDURE
 PROC1 ...
IS
 ...
BEGIN
 ...
  PROC2(arg1);
 ...
EXCEPTION
 ...
END PROC1;
```

```
PROCEDURE
 PROC2 ...
IS
 ...
BEGIN
 ...
EXCEPTION
 ...
END PROC2;
```

HANDLED EXCEPTION: EXAMPLE

```
CREATE PROCEDURE add_department(
name VARCHAR2, mgr NUMBER, loc NUMBER) IS
BEGIN
INSERT INTO DEPARTMENTS (department_id,
department_name, manager_id, location_id)
VALUES (DEPARTMENTS_SEQ.NEXTVAL, name, mgr, loc);
DBMS_OUTPUT.PUT_LINE('Added Dept: '||name);
EXCEPTION
WHEN OTHERS THEN
DBMS_OUTPUT.PUT_LINE('Err: adding dept: '||name);
END;
```

Calling Program →

```
BEGIN
add_department('Media', 100, 1800);
add_department('Editing', 99, 1800);
add_department('Advertising', 101, 1800);
END;
```

REMOVING PROCEDURES

```
Syntax:
     DROP PROCEDURE procedure_name

Example:
     DROP PROCEDURE raise_salary;
```

CREATING STORED FUNCTIONS

OVERVIEW OF STORED FUNCTIONS

A function:

- **Is a named PL/SQL block that returns a value**
- **Can be stored in the database as a schema object for repeated execution**
- **Is called as part of an expression or is used to provide a parameter value**

SYNTAX FOR CREATING FUNCTIONS

The PL/SQL block must have at least one RETURN statement.

```
CREATE [OR REPLACE] FUNCTION function_name
[(parameter1 [mode1] datatype1, ...)]
RETURN datatype IS|AS
[local_variable_declarations; …]
BEGIN
-- actions;
RETURN expression;
END [function_name];
```

STORED FUNCTION: EXAMPLE

Create the function:

```
CREATE OR REPLACE FUNCTION get_sal
(id employees.employee_id%TYPE) RETURN
NUMBER IS
sal employees.salary%TYPE := 0;
BEGIN
SELECT salary
INTO sal
FROM employees
WHERE employee_id = id;
RETURN sal;
END get_sal;
/
```

- Invoke the function as an expression or as a parameter value:

```
EXECUTE dbms_output.put_line(get_sal(100))
```

WAYS TO EXECUTE FUNCTIONS

Invoke as part of a PL/SQL expression

– Using a host variable to obtain the result

```
VARIABLE salary NUMBER
EXECUTE :salary := get_sal(100)
```

– Using a local variable to obtain the result

```
DECLARE sal employees.salary%type;
BEGIN
sal := get_sal(100); ...
END;
```

- Use as a parameter to another subprogram

```
EXECUTE dbms_output.put_line(get_sal(100))
```

- Use in a SQL statement (subject to restrictions)

```
SELECT job_id, get_sal(employee_id) FROM employees;
```

EXAMPLES

ADVANTAGES OF USER-DEFINED FUNCTIONS IN SQL STATEMENTS

- **Can extend SQL where activities are too complex, too awkward, or unavailable with SQL**
- **Can increase efficiency when used in the `WHERE` clause to filter data, as opposed to filtering the data in the application**
- **Can manipulate data values**

LOCATIONS TO CALL USER-DEFINED FUNCTIONS

User-defined functions act like built-in single-row functions and can be used in:

- **The `SELECT` list or clause of a query**
- **Conditional expressions of the `WHERE` and `HAVING` clauses**
- **The `CONNECT BY`, `START WITH`, `ORDER BY`, and `GROUP BY` clauses of a query**
- **The `VALUES` clause of the `INSERT` statement**
- **The `SET` clause of the `UPDATE` statement**

RESTRICTIONS ON CALLING FUNCTIONS FROM SQL EXPRESSIONS

- User-defined functions that are callable from SQL expressions must:
  - Be stored in the database
  - Accept only `IN` parameters with valid SQL data types, not PL/SQL-specific types
  - Return valid SQL data types, not PL/SQL-specific types
- When calling functions in SQL statements:
  - Parameters must be specified with positional notation
  - You must own the function or have the `EXECUTE` privilege

CONTROLLING SIDE EFFECTS WHEN CALLING FUNCTIONS FROM SQL EXPRESSIONS

Functions called from:

- **A `SELECT` statement cannot contain DML statements**
- **An `UPDATE` or `DELETE` statement on a table `T` cannot query or contain DML on the same table `T`**
- **SQL statements cannot end transactions (that is, cannot execute `COMMIT` or `ROLLBACK` operations)**

Note: Calls to subprograms that break these restrictions are also not allowed in the function.

RESTRICTIONS ON CALLING FUNCTIONS FROM SQL: EXAMPLE

```
CREATE OR REPLACE FUNCTION dml_call_sql(sal NUMBER)
RETURN NUMBER IS
BEGIN
      INSERT INTO employees(employee_id, last_name, email, hire_date, job_id, salary)
                   VALUES(1, 'Frost', 'jfrost@company.com', SYSDATE, 'SA_MAN', sal);
      RETURN (sal + 100);
END;
```

```
UPDATE employees
SET salary = dml_call_sql(2000)
WHERE employee_id = 170;
```

```
UPDATE employees SET salary = dml_call_sql(2000)
              *
ERROR at line 1:
ORA-04091: table PLSQL.EMPLOYEES is mutating,
trigger/function may not see it
ORA-06512: at "PLSQL.DML_CALL_SQL", line 4
```

REMOVING FUNCTIONS

Removing a stored function:

- **You can drop a stored function by using the following syntax:**

```
DROP FUNCTION function_name
```

Example:

```
DROP FUNCTION get_sal;
```

- **All the privileges that are granted on a function are revoked when the function is dropped.**
- **The `CREATE OR REPLACE` syntax is equivalent to dropping a function and re-creating it. Privileges granted on the function remain the same when this syntax is used.**

VIEWING FUNCTIONS IN THE DATA DICTIONARY

Information for PL/SQL functions is stored in the following Oracle data dictionary views:

- **You can view source code in the `USER_SOURCE` table for subprograms that you own, or the `ALL_SOURCE` table for functions owned by others who have granted you the `EXECUTE` privilege.**

```
SELECT text
FROM user_source
WHERE type = 'FUNCTION'
ORDER BY line;
```

You can view the names of functions by using `USER_OBJECTS`.

```
SELECT object_name
FROM user_objects
WHERE object_type = 'FUNCTION';
```

PROCEDURES VERSUS FUNCTIONS

Procedures	Functions
Execute as a PL/SQL statement	Invoke as part of an expression
Do not contain `RETURN` clause in the header	Must contain a RETURN clause in the header
Can return values (if any) in output parameters	Must return a single value
Can contain a `RETURN` statement without a value	Must contain at least one RETURN statement

CREATING STORED FUNCTIONS & PROCEDURES IN MYSQL

CREATE PROCEDURE

```
CREATE [DEFINER = user] PROCEDURE sp_name
              ([proc_parameter[,...]])
routine_body


proc_parameter:    [ IN | OUT | INOUT ] param_name type
type:               Any valid MySQL data type
```

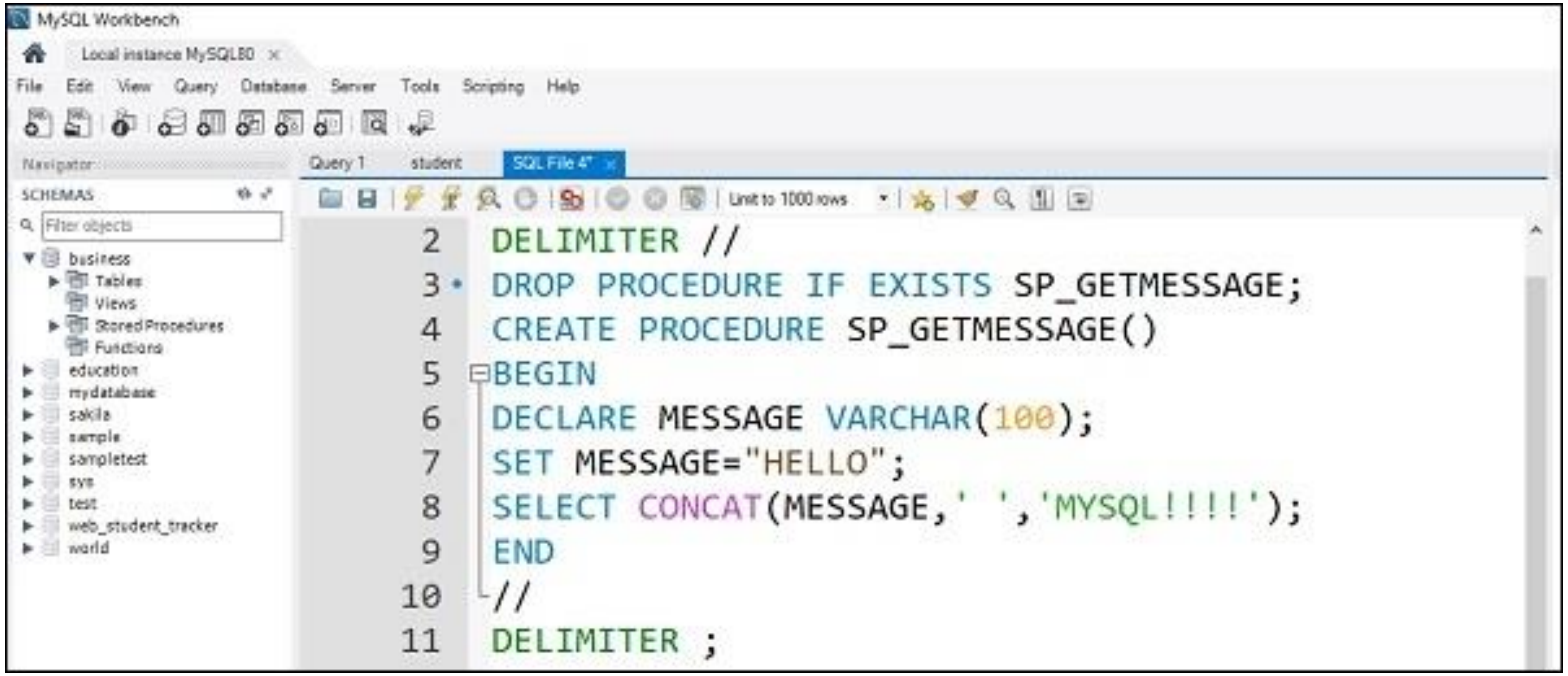
STORED PROCEDURES IN MYSQL

```
mysql> delimiter //
mysql> CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)
          BEGIN
            SELECT COUNT(*) INTO cities
            FROM world.city
            WHERE CountryCode = country;
          END//
Query OK, 0 rows affected (0.01 sec)
mysql> delimiter ;
mysql> CALL citycount('JPN', @cities); -- cities in Japan
 Query OK, 1 row affected (0.00 sec)
mysql> SELECT @cities;
+---------+
| @cities |
 +---------+
| 248 |
 +---------+
1 row in set (0.00 sec)
```

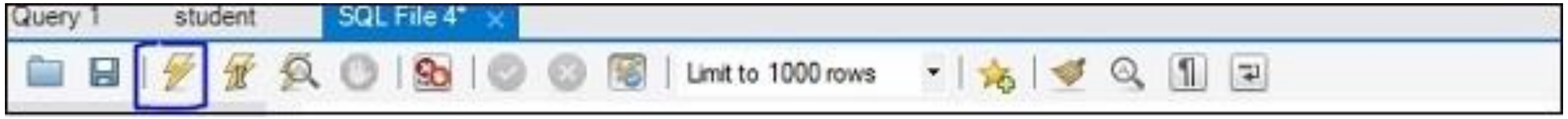
STORED PROCEDURE IN MYSQL WORKBENCH

```
use business;
DELIMITER //
DROP PROCEDURE IF EXISTS SP_GETMESSAGE;
CREATE PROCEDURE SP_GETMESSAGE()
BEGIN
DECLARE MESSAGE VARCHAR(100);
SET MESSAGE="HELLO";
SELECT CONCAT(MESSAGE,' ','MYSQL!!!!’);
END // DELIMITER ;
```

STORED PROCEDURE IN MYSQL WORKBENCH

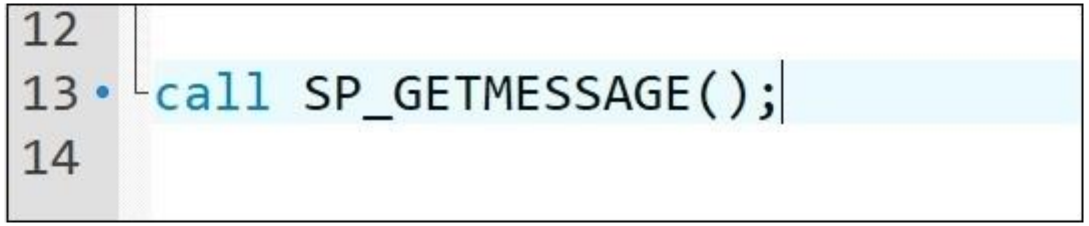


```
DELIMITER //
DROP PROCEDURE IF EXISTS SP_GETMESSAGE;
CREATE PROCEDURE SP_GETMESSAGE()
BEGIN
DECLARE MESSAGE VARCHAR(100);
SET MESSAGE="HELLO";
SELECT CONCAT(MESSAGE,' ','MYSQL!!!!');
END
//
DELIMITER ;
```



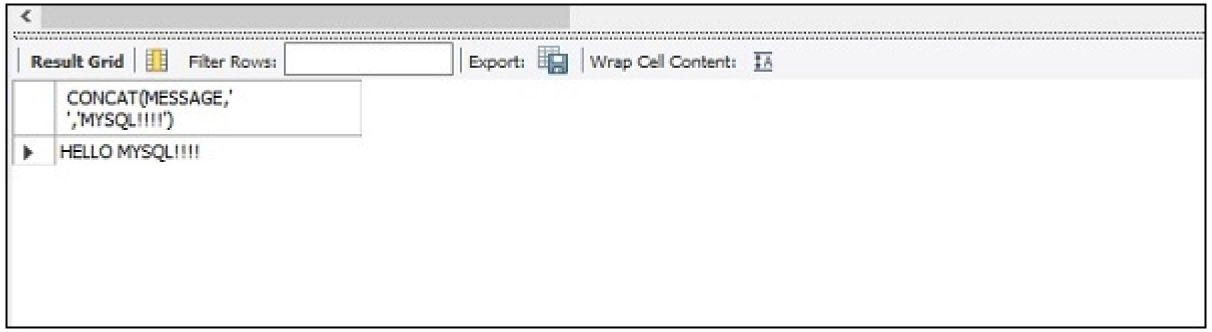
STORED PROCEDURE IN MYSQL WORKBENCH

Calling Procedure:



```
call SP_GETMESSAGE();
```

Output:



```
CONCAT(MESSAGE,' ','MYSQL!!!!')
HELLO MYSQL!!!!
```

STORED FUNCTIONS IN MYSQL

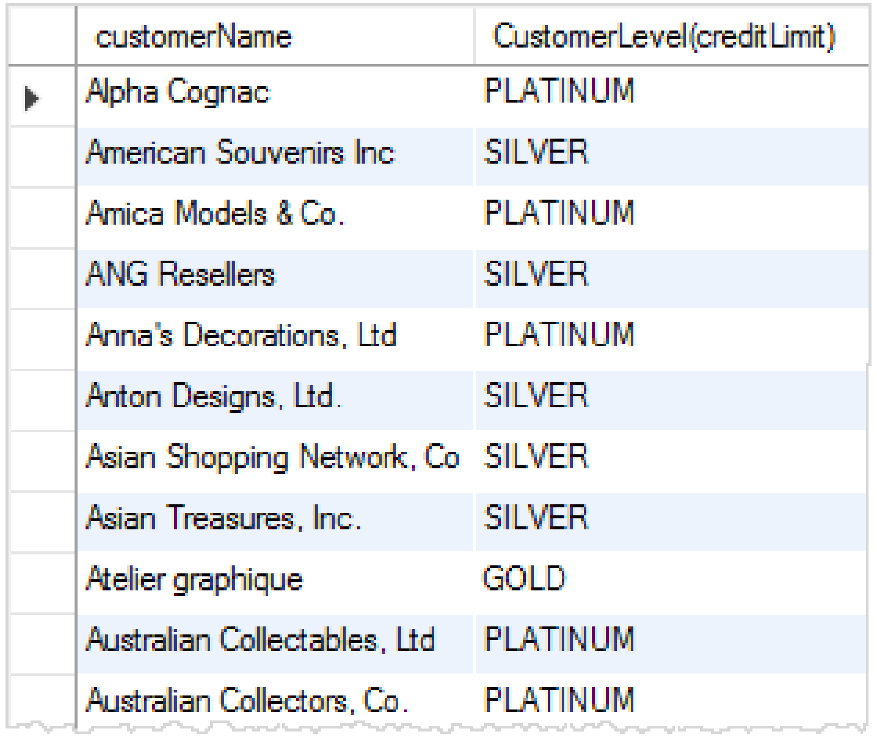
```
CREATE FUNCTION function_name(func_parameter1, func_parameter2, ..)
RETURN datatype [characteristics]
func_body
```

STORED FUNCTION IN MYSQL

```
DELIMITER $$
CREATE FUNCTION CustomerLevel( credit DECIMAL(10,2) )
RETURNS VARCHAR(20) DETERMINISTIC
BEGIN
        DECLARE customerLevel VARCHAR(20);
        IF credit > 50000 THEN
                SET customerLevel = 'PLATINUM';
        ELSEIF (credit >= 50000 AND credit <= 10000) THEN
                SET customerLevel = 'GOLD';
        ELSEIF credit < 10000 THEN
                SET customerLevel = 'SILVER';
        END IF;
        -- return the customer level
        RETURN (customerLevel);
END$$
DELIMITER ;
```

CALLING A STORED FUNCTION IN AN SQL STATEMENT

```
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

EXAMPLE

Find the number of years the employee has been in the company.

```
DELIMITER //
CREATE FUNCTION no_of_years(date1 date) RETURNS int DETERMINISTIC
BEGIN
DECLARE date2 DATE;
Select current_date()into date2;
RETURN year(date2)-year(date1);
END //
DELIMITER ;
```

EXAMPLE

```
Select emp_id, fname, lname, no_of_years(start_date) as 'years'
from employee;
```

EMP_ID	FNAME	LNAME	YEARS
1	Michael	Smith	18
2	Susan	Barker	17
3	Robert	Tvler	19
4	Susan	Hawthorne	17