SE Comp - B Div	Roll number : **8942**
Experiment no. : 9	Date of Implementation : **13-05-2021**

Aim : To implement Functions and Triggers

Tool Used : MySQL/PostgreSQL

Related Course outcome : At the end of the course, Students will be able to Use SQL : Standard language of relational database

Rubrics for assessment of Experiment:

Indicator	Poor	Average	Good
Timeliness • Maintains assignment deadline (3)	Assignment not done (0)	One or More than One week late (1-2)	Maintains deadline (3)
Completeness and neatness • Complete all parts of assignment(3)	N/A	< 80% complete (1-2)	100% complete (3)
Originality • Extent of plagiarism(2)	Copied it from someone else(0)	At least few questions have been done without copying(1)	Assignment has been solved completely without copying (2)
Knowledge • In depth knowledge of the assignment(2)	Unable to answer 2 questions(0)	Unable to answer 1 question (1)	Able to answer 2 questions (2)

Assessment Marks :

Timeliness	
Completeness and neatness	
Originality	
Knowledge	
Total	

Total : **(Out of 10)**

Teacher's Sign :

EXPERIMENT 09	***Functions and Triggers***
Aim	To implement PL/SQL function and trigger
Tools	MySQL http://www.postgresqltutorial.com/postgresql-create-function/ http://www.postgresqltutorial.com/plpgsql-function-overloading/
Theory	CREATE FUNCTION defines a new function. CREATE OR REPLACE FUNCTION will either create a new function, or replace an existing definition. To be able to define a function, the user must have the USAGE privilege on the language. If a schema name is included, then the function is created in the specified schema. Otherwise it is created in the current schema. The name of the new function must not match any existing function with the same input argument types in the same schema. However, functions of different argument types can share a name (this is called *overloading*). **Syntax for Function** CREATE [OR REPLACE] FUNCTION name ([[argmode] [argname] argtype [{ DEFAULT | = } default_expr] [, ...]]) [RETURNS rettype | RETURNS TABLE (column_name column_type [, ...])] { LANGUAGE lang_name | WINDOW | IMMUTABLE | STABLE | VOLATILE | CALLED ON NULL INPUT | RETURNS NULL ON NULL INPUT | STRICT | [EXTERNAL] SECURITY INVOKER | [EXTERNAL] SECURITY DEFINER | COST execution_cost | ROWS result_rows | SET configuration_parameter { TO value | = value | FROM CURRENT } | AS 'definition' | AS 'obj_file', 'link_symbol' } ... [WITH (attribute [, ...])] If you drop and then recreate a function, the new function is not the same entity as the old; you will have to drop existing rules, views, triggers, etc. that refer to the old function. Use CREATE OR REPLACE FUNCTION to change a function definition without breaking objects that refer to the function. The trigger can be specified to fire before the operation is attempted on a row (before constraints are checked and the INSERT, UPDATE, or DELETE is attempted); or after the operation has completed (after constraints are checked and the INSERT, UPDATE, or DELETE has completed); or instead of the operation (in the case of inserts, updates or deletes on a view). If the trigger fires before or instead of the event, the trigger can skip the operation for the current row, or change the row being inserted (for INSERT and UPDATE operations only). If the trigger fires after the event, all changes, including the effects of other triggers, are "visible" to the trigger.

	Syntax of Trigger CREATE [CONSTRAINT] TRIGGER name { BEFORE | AFTER | INSTEAD OF } { event [OR ...] } ON table [FROM referenced_table_name] [NOT DEFERRABLE | [DEFERRABLE] { INITIALLY IMMEDIATE | INITIALLY DEFERRED }] [FOR [EACH] { ROW | STATEMENT }] [WHEN (condition)] EXECUTE PROCEDURE function_name (arguments) where event can be one of: INSERT UPDATE [OF column_name [, ...]] DELETE TRUNCATE To create a trigger on a table, the user must have the TRIGGER privilege on the table. The user must also have EXECUTE privilege on the trigger function. Use DROP TRIGGER to remove a trigger.
Procedure	1. Write a function to find factorial of a number 2. Create table emp (id,name,salary) and insert 3 records in it. 3. Write a function to find average salary from emp table 4. Write a row-level trigger that would fire before insert and checks id salary < 0 then it sets the salary = 0. 5. Write a row –level trigger that would fire when user tries to update name. Triggers should not allow user to update name field and displays appropriate message. 6. Write a row level trigger that would fire AFTER delete operation is performed on emp table displaying date on which data is deleted.
Post Lab Questions:	1. Difference between procedures and Function on SQL.

1. Write a function to find factorial of a number

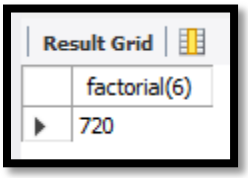
CODE:

```
DELIMITER $$
CREATE FUNCTION factorial (n int) RETURNS int
DETERMINISTIC
BEGIN
DECLARE i , f int;
set i = 1;
set i =1;
set f = 1;
while i <= n do
set f = f * i;

set i = i+1;
end while;
RETURN f;
END$$
```

OUTPUT:

Select factorial(6);



Result Grid

	factorial(6)
▶	720

2. Create table emp (id,name,salary) and insert 3 records in it.

CODE:

```
CREATE TABLE emp (
   id VARCHAR(4),
   name VARCHAR(20),
   salary DECIMAL(6 , 2 )
);
insert into emp values('E001','Gini Chacko', 5000.00);
insert into emp values('E002','Praisy Praveen', 6000.00);
insert into emp values('E003','Elwin Dainu', 4000.00);
```

OUTPUT:

Result Grid | Filter Rows:

id	name	salary
E001	Gini Chacko	5000.00
E002	Praisy Praveen	6000.00
E003	Elwin Dainu	4000.00

3. Write a function to find average salary from emp table

CODE:

```
DELIMITER //
CREATE FUNCTION AVERAGE_SALARY() returns double deterministic
begin
declare sum int;
declare avg1 double;
select avg(salary) INTO avg1
from emp;
Return avg1;
end;//
```

OUTPUT:

select average_salary();

average_salary()
5000

4. Write a row-level trigger that would fire before insert and checks id salary < 0 then it sets the salary = 0.

CODE:

Step 1: Create a following trigger

This trigger is activated before each insert statement into emp table and checks the ifsalary < 0 then sets the salary =0.

```
DELIMITER //
CREATE TRIGGER before_insert_emp_salary BEFORE INSERT ON emp FOR
EACH ROW
begin
    if new.salary< 0 then set new.salary = 0;
    end if;
end;//
```

Step 2: Now to check this trigger insert following row

```
insert into emp values('E004','Elaine Gomes', -4000.00);
```

Step 3: Now display records of the table

```
Select * from emp;
```

OUTPUT:

Result Grid | Filter Rows:

id	name	salary
E001	Gini Chacko	5000.00
E002	Praisy Praveen	6000.00
E003	Elwin Dainu	4000.00
E004	Elaine Gomes	0.00

Note: that employee Elaine's salary is set to 0 since we are trying to insert negative salary.

5. Write a row –level trigger that would fire when user tries to update name. Triggers should not allow user to update name field and displays appropriate message.

CODE:

```
DELIMITER //
CREATE TRIGGER before_update_emp_name1 BEFORE UPDATE ON emp
FOR EACH ROW
begin
DECLARE error_msg VARCHAR(255);
   SET error_msg = ('The Name attribute cannot be updated');
    set new.name = old.name;
```

```
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = error_msg;
end;//
```

OUTPUT:

```
update emp
set name ='Jerry';
where id = 'E001';
```

It will show you error message.

```
25  13:07:50  Update emp set name = 'Jerry' where id = 'E001'    Error Code: 1644. The Name attribute cannot be updated    0.125 sec
```

6. Write a row level trigger that would fire AFTER delete operation is performed on emp table displaying date on which data is deleted.

CODE:

```
DELIMITER //
CREATE TRIGGER after_delete_emp_Record AFTER DELETE ON emp FOR
EACH ROW
begin
DECLARE del_msg VARCHAR(255);
   SET del_msg = concat('The record is deleted on',now());
    SIGNAL SQLSTATE '45000'
    SET MESSAGE_TEXT = del_msg;
end;//
```

OUTPUT:

```
delete from emp where id = 'E001';
```

It will show you error message.

```
27  13:10:06  delete from emp where id = 'E001'    Error Code: 1644. The record is deleted on2021-05-14 13:10:06    0.250 sec
```

POSTLAB QUESTIONS:

1. Difference between procedures and Function on SQL.

Ans:

SR.NO	KEY	FUNCTION	PROCEDURE
1.	**Definition**	A function is used to calculate result using given inputs.	A procedure is used to perform certain task in order.
2.	**Call**	A function can be called by a procedure.	A procedure cannot be called by a function
3.	**DML**	DML statements cannot be executed within a function.	DML statements can be executed within a procedure.
4.	**SQL,Query**	A function can be called within a query.	A procedure cannot be called within a query.
5.	**SQL, Call**	Whenever a function is called, it is first compiled before being called.	A procedure is compiled once and can be called multiple times without being compiled.
6.	**SQL, Return**	A function returns a value and control to calling function or code.	A procedure returns the control but not any value to calling function or code.
7.	**try-catch**	A function has no support for try-catch.	A procedure has support for try-catch blocks.
8.	**SELECT**	A select statement can have a function call.	A select statement can't have a procedure call.
9.	**Explicit Transaction Handling**	A function cannot have explicit transaction handling.	A procedure can use explicit transaction handling.