Lab - 3	SQL_Assignment_3
IT214 Database Management System, Autumn'2020; Instructor: minal_bhise@daiict, TA: mayank@daiict	

Objectives: I) Understand & Run functions.

II) Use triggers to execute functions.

III) Understand Store Procedures (For PgSQL 11+).

Submission: Each student team needs to upload a **single .pdf** file, which will contain the following things for all the functions and triggers listed in your specific section's lab file.

1) Write English query, SQL function, &/or Trigger SQL statement in the given sequence.
2) Screenshot of results.
3) Count of tuples in the results.

I. Understand & Run functions.

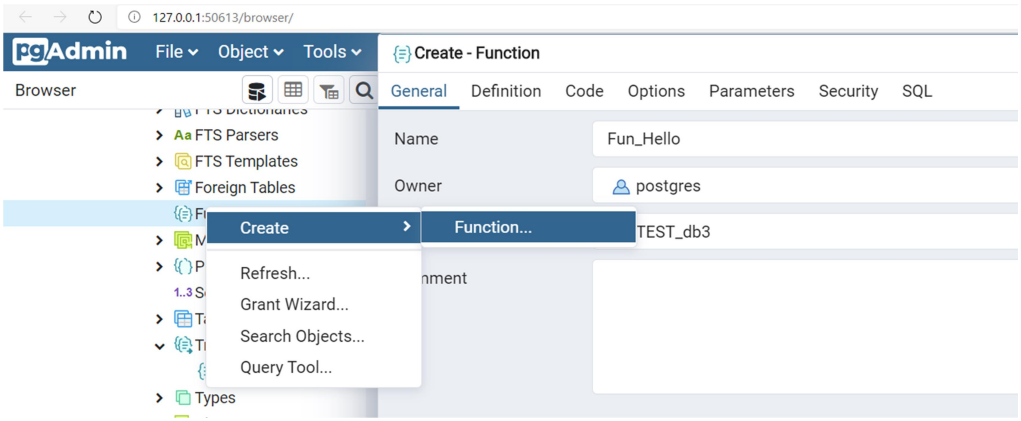
a. Create a simple function to print "Hello World" using GUI.

SQL Functions work similarly to normal coding functions.

Step1.

Using GUI – Right click on Functions => Create => Functions

Write Name – Fun_Hello

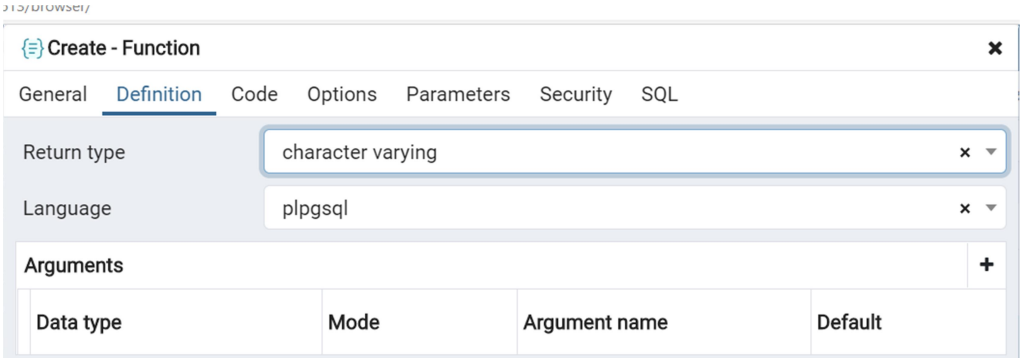


Step2.

Using GUI – Go to next tab => Definition

Choose Return Type => character varying

& Language => **plpgsql (It will throw errors if plpgsql not selected)**



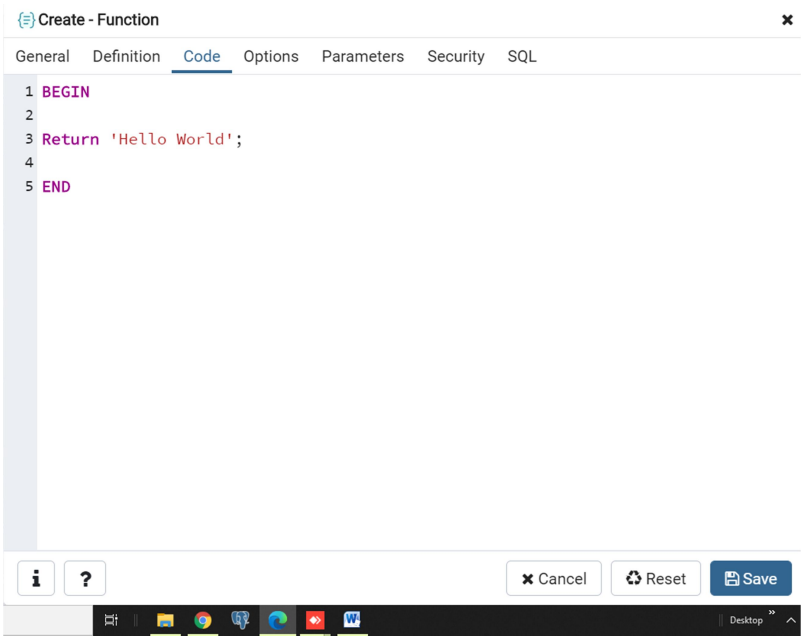
Step3.

Using GUI – Go to next tab => Code

Write code

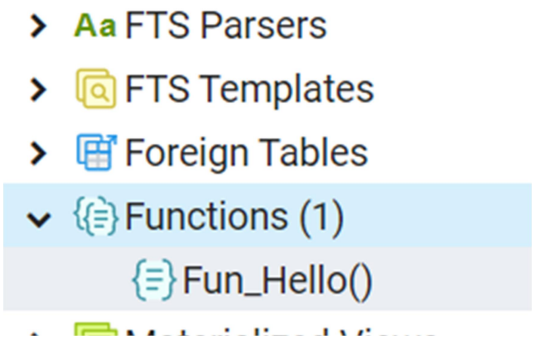
```
BEGIN
Return 'Hello World';
END
```

Using GUI => Save.



Step4.

Using GUI – Refresh to Check if a function is created under "Functions".



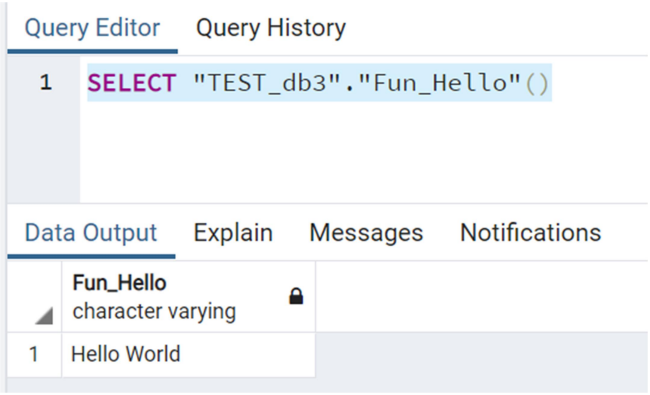
b. Create simple function to print "Hello World" using SQL.

Step1.

```
set search_path to "TEST_db3";

CREATE OR REPLACE FUNCTION "TEST_db3"."Fun_Hello2"()
    RETURNS character varying
    LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
return 'Hello World2';
END;
$BODY$;
```

c. Using the function. "Function calls".

```
SELECT "TEST_db3"."Fun_Hello"();
```



d. Create functions with parameters that return the SUM of two values.

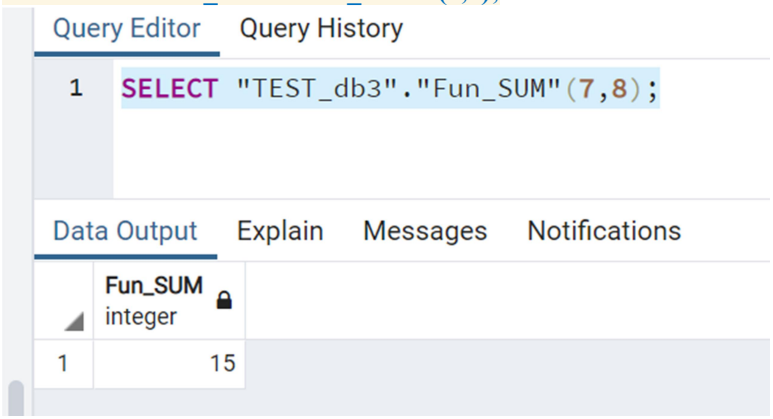
Step1. CREATE function.

```
CREATE OR REPLACE FUNCTION "TEST_db3"."Fun_SUM"(a integer, b integer)
  RETURNS integer
  LANGUAGE 'plpgsql'
AS $BODY$
BEGIN
```

```
Return (a+b);
END
$BODY$;
```

Step2. CALL :

```
SELECT "TEST_db3"."Fun_SUM"(7,8);
```



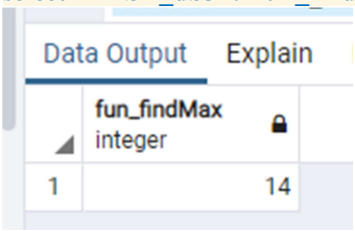
e. **Create function with If condition to find the largest number of two.**

Step1.

```
CREATE OR REPLACE function "TEST_db3"."fun_findMax"(a int, b int)
RETURNS integer
LANGUAGE 'plpgsql'

AS $BODY$

DECLARE
c integer;
BEGIN

if(a>b) then
        c=a;
else
        c=b;
end if;

RETURN c;

END;
$BODY$;
```

Step2. CALL :

```
select "TEST_db3"."fun_findMax"(14,7);
```



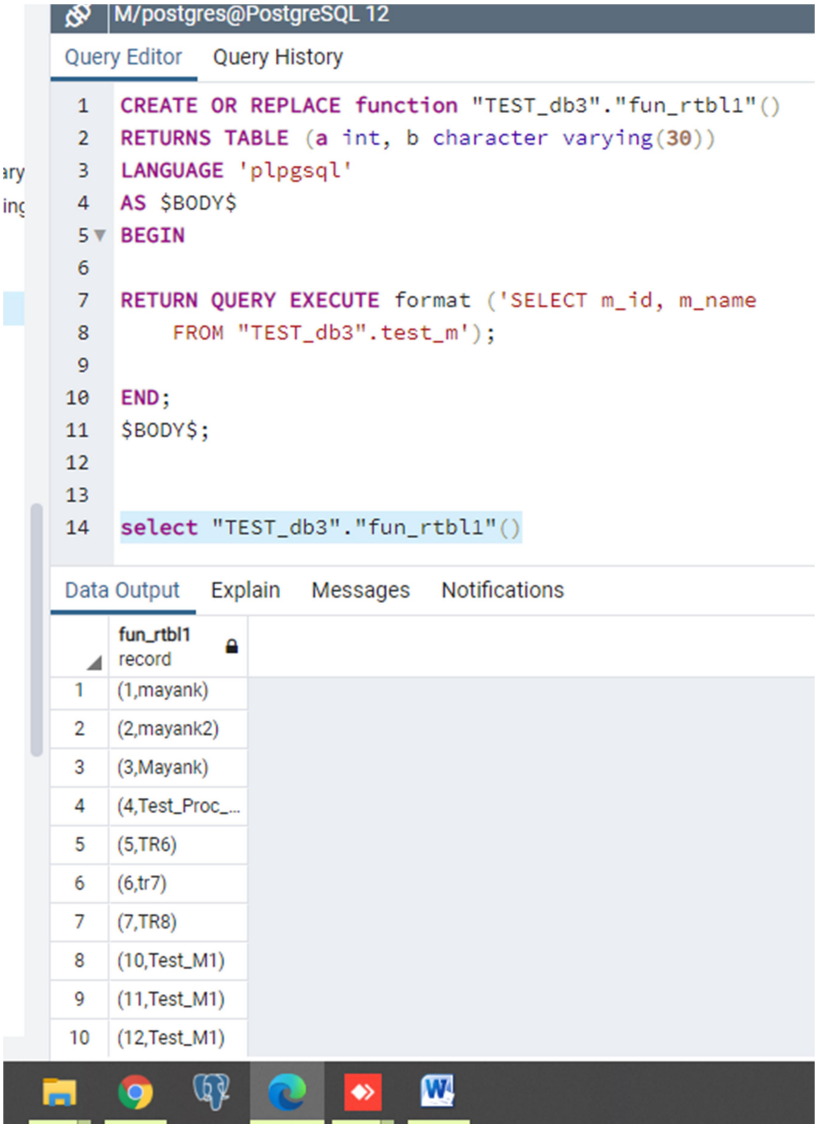
f. Create functions with a table as a return value.

Step1.

```
CREATE OR REPLACE function "TEST_db3"."fun_rtbl1"()
RETURNS TABLE (a int, b character varying(30))
LANGUAGE 'plpgsql'
AS $BODY$

BEGIN
RETURN QUERY EXECUTE format ('SELECT m_id, m_name
        FROM "TEST_db3".test_m');
 END;
$BODY$;
```

Step2. CALL :

```
"TEST_db3"."fun_rtbl1"()
```



g. Create functions with a Temporary table and use of FOR Loop.

Step1.

```
CREATE OR REPLACE FUNCTION "TEST_db3".fun_loop2(      )
    RETURNS TABLE(a integer, b character varying)
    LANGUAGE 'plpgsql'

AS $BODY$
DECLARE
R_LIST2 record;

BEGIN

CREATE TEMP TABLE test1 (a1 int, b1 character varying(30)) ON COMMIT
DROP;

FOR R_LIST2 in (select m_id, m_name from "TEST_db3".test_m)
loop
Insert into test1 (a1, b1) values (R_LIST2.m_id,R_LIST2.m_name);
end loop;

RETURN QUERY TABLE test1;

END;
$BODY$;
```

```
1   SELECT "TEST_db3".fun_loop2()
```

Data Output | Explain | Messages | Notificat

	fun_loop2 record
1	(1,mayank)
2	(2,mayank2)
3	(3,Mayank)
4	(4,Test_Proc_M)
5	(5,TR6)
6	(6,tr7)
7	(7,TR8)
8	(10,Test_M1)
9	(11,Test_M1)
10	(12,Test_M1)

II. Use triggers to execute functions.

The main difference between a normal function & a trigger function is its return type. The trigger function gets automatically called based on some events set earlier, like Insert, Update, &/or Delete on some table.

a. Create a trigger function.

Step1. Create.

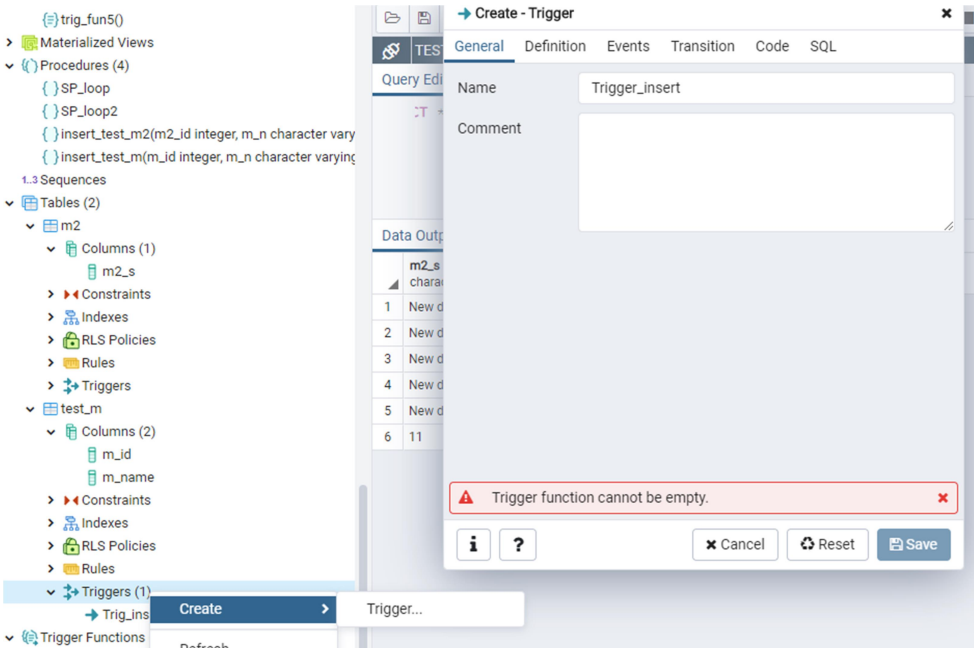
```
CREATE OR REPLACE FUNCTION "TEST_db3".Trig_fun2()
    RETURNS trigger
    LANGUAGE 'plpgsql'

AS $BODY$
BEGIN

Insert into "TEST_db3".m2 (m2_s) values (concat('New id
added=',NEW.m_id));    --NEW.m_id is the new data values
RETURN NEW;
END
$BODY$;
```

Step2. Create Trigger CALL

Create a trigger to call for the above-created function.

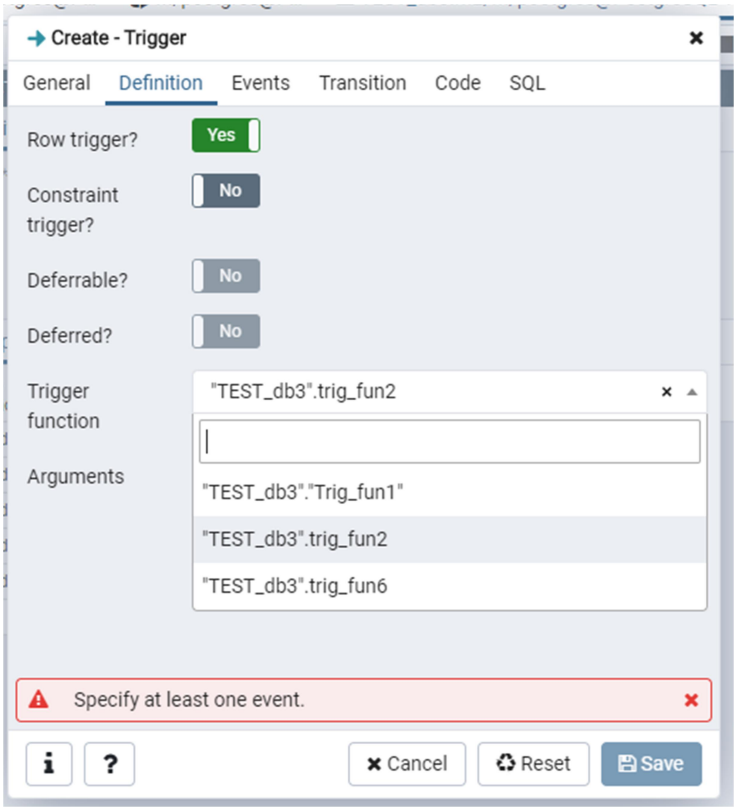
GUI =>Right Click on Triggers => Create => Trigger…

Add Name => Trigger_Insert



Step3. Select Trigger Function

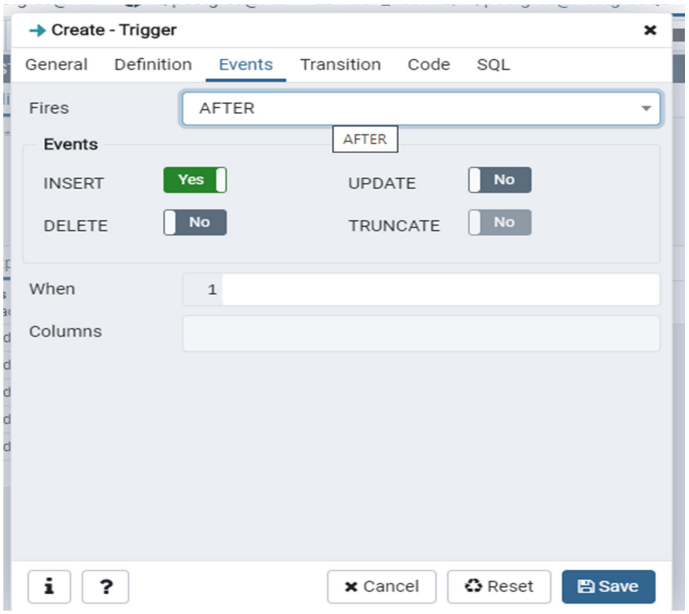
Go to Definition tab => Select the Trigger Function



Step4. Select the events when the function needs to be called.

Fires(Executes) when ? **BEFORE** or **AFTER => AFTER**

& **Switch on (Yes)** the events when you want to run the trigger function.



Step2. OR Add the trigger function call using below given code.

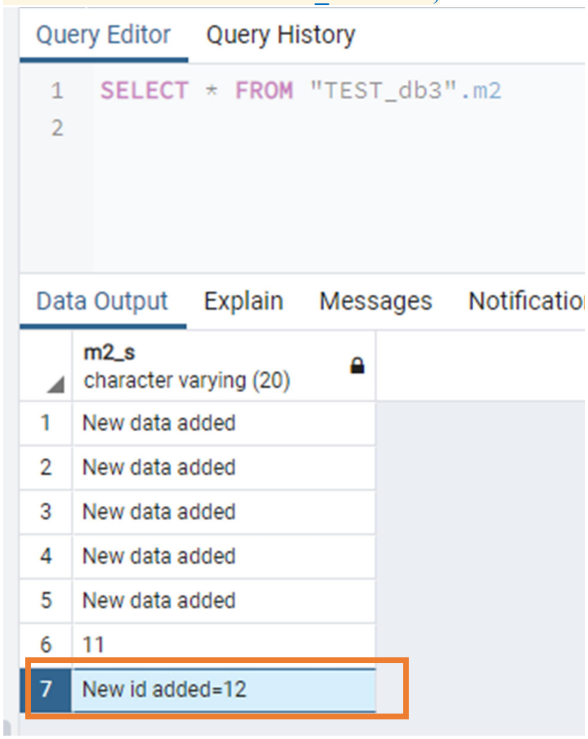
```
CREATE TRIGGER "Trigger_insert"
    AFTER INSERT
    ON "TEST_db3".test_m
    FOR EACH ROW
    EXECUTE PROCEDURE "TEST_db3".trig_fun2();
```

Step5. Auto call to trigger a function when inserts happen on the table where we had created the trigger call.

```
INSERT INTO "TEST_db3".test_m(
    m_id, m_name)
    VALUES (12, 'Test_12');
```

Step6. Check that the trigger function added a new record automatically in the m2 table.

```
SELECT * FROM “TEST_db3”.m2;
```



III. Understand Store Procedures. (For PgSQL 11+).

Store procedures are similar to normal functions. Except minor difference in the create statement and calling.

a. Create a simple stored procedure to insert records in a table.

Step1. CREATE stored procedure.

```
CREATE OR REPLACE PROCEDURE "TEST_db3".insert_test_m(
        m_id integer,
        m_n character varying)
LANGUAGE 'plpgsql'
AS $BODY$
BEGIN

INSERT INTO "TEST_db3".test_m (m_id, m_name) values (M_id, m_n);
COMMIT;

END
$BODY$;
```

Step2. **CALL the stored procedure.**

```
CALL "TEST_db3".insert_test_m(15, 'Test15');
```

It will insert the new records in the test_m table with ID 15 and the name 'Test15'.

b. One more sample stored procedure to list all records as a message.

Step1. CREATE stored procedure.

```
CREATE OR REPLACE PROCEDURE "TEST_db3"."SP_loop2"()
LANGUAGE 'plpgsql'
AS $BODY$
DECLARE
R_LIST2 record;

BEGIN
FOR R_LIST2 in (select m_id, m_name from sv_db3.test_m)
loop
RAISE NOTICE 'X = %, Y = %', R_LIST2.m_id, R_LIST2.m_name;
end loop;
```

```
END;
$BODY$;
```

Step2. CALL the stored procedure.

```
CALL "TEST_db3"."SP_loop"();
```