Lab Submission: 9

Student Name: Michael Daly

Course and Section: CST8215 – 362

========================================================================

Lab 9 – Creating a database trigger

PostgreSQL **Triggers** are database callback functions, which are automatically performed/invoked when a specified database event occurs.

The following are important points about PostgreSQL triggers −

* PostgreSQL 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)
  + After the operation has completed (after constraints are checked and the INSERT, UPDATE, or DELETE has completed)
  + Instead of the operation (in the case of inserts, updates or deletes on a view)

<https://www.tutorialspoint.com/postgresql/postgresql_triggers.htm>

**Grading: Lab 9 is worth 5 marks**

========================================================================

**Creating and using a Postgres Trigger**

This first part is the database preparation --- creating the database tables for test, creating the function and the trigger.

**Step 1 -- Create two tables for this exercise – Employees and Employee\_Audits**

CREATE TABLE **EMPLOYEES**

( employee\_id int4 serial primary key,

first\_name varchar(40) NOT NULL,

last\_name varchar(40) NOT NULL );

CREATE TABLE **EMPLOYEE\_AUDITS**

( id int4 serial primary key,

employee\_id int4 NOT NULL,

last\_name varchar(40) NOT NULL,

changed\_on timestamp(6) NOT NULL);

**Step 2 – Populate Employee table and verify new rows**

INSERT INTO employees (first\_name, last\_name) VALUES ('John', 'Doe');

INSERT INTO employees (first\_name, last\_name) VALUES ('Lily', 'Bush');

Validate the insert statements by issuing the following SQL command…..

SELECT \* FROM employees;



**Step 3 – Create a new database function which will perform the ‘insert’ into the EMPLOYEE\_AUDITS table.**

We need to create a new database function called -- **log\_last\_name\_changes**:

This new function is responsible for checking to see if the last name of employee changes. If it detects a change, it will insert the old last name, employee\_id and time of change into the **EMPLOYEE\_AUDITS** table.

CREATE OR REPLACE FUNCTION **log\_last\_name\_changes()**

RETURNS trigger

AS $BODY$

BEGIN

IF NEW.last\_name <> OLD.last\_name THEN

INSERT INTO **EMPLOYEE\_AUDITS** (employee\_id, last\_name, changed\_on)

VALUES (OLD.employee\_id, OLD.last\_name, now());

END IF;

RETURN NEW;

END; $BODY$

**Step 4 - Create Database Trigger called “last\_name\_changed”**

Triggers are linked to a table so we say we are ‘binding’ the new database trigger to a table.

As the requirement here is to ‘audit’ changes to the employee last name field, we will bind this trigger to the Employee table and have it perform a ‘before insert’ check on incoming ‘last name’ values. The instant that the database gets a command to ‘UPDATE EMPLOYEES….’ This trigger will immediately execute. – this trigger is designed to call the ‘function’/’procedure’ called ‘log\_last\_name\_changed’.

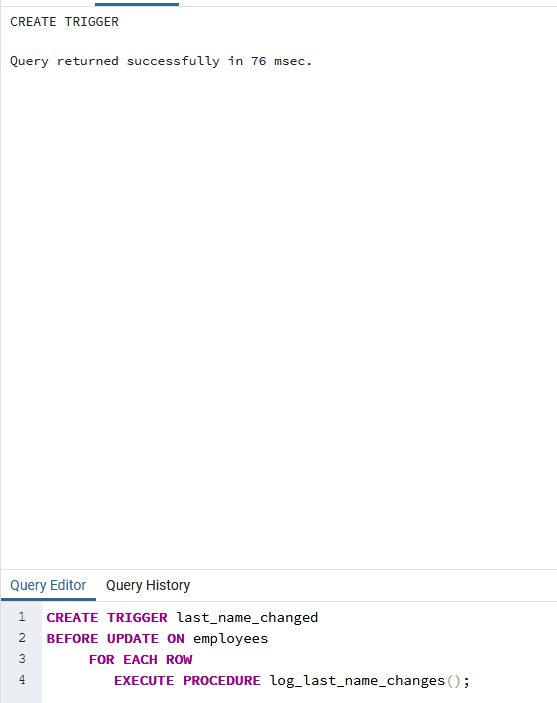
CREATE TRIGGER **last\_name\_changed**

BEFORE UPDATE ON employees

FOR EACH ROW

EXECUTE PROCEDURE **log\_last\_name\_changes();**

*Provide screenshot of your sql statement and results here*



You will note that when we execute a function in this manner we use the command ‘execute procedure’

Actual way these database objects work together in day to day operations

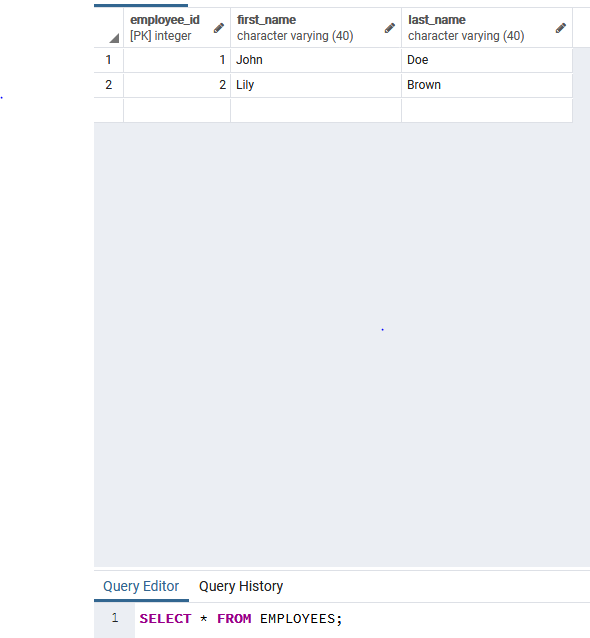
Suppose Lily Bush gets married and she needs to change her last name to Lily Brown. We can update it as the following query:

UPDATE **EMPLOYEES** SET last\_name = 'Brown' WHERE ID = 2;

Verify if the last name of Lily has been updated. As you can see, the Lily’s last name has been updated.

SELECT \* FROM **EMPLOYEES**;

*Provide screenshot of your results here*

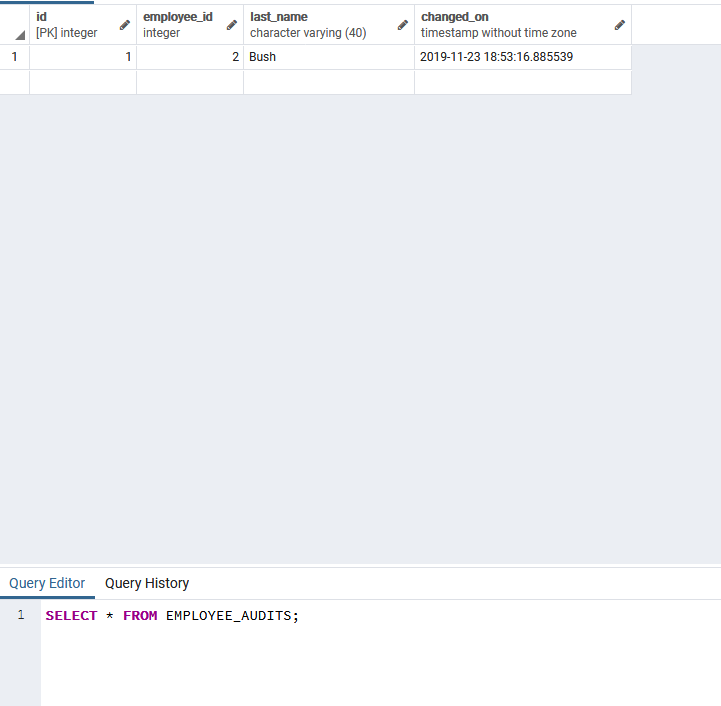


The business rule was to ensure that when the EMPLOYEES table was updated, the trigger would execute the function which will verify if the last name was changed in the update statement. If the last name was changed --- then it would insert the old/new last name data for the employee into the EMPLOYEE\_AUDITS table. Now that Lily Bush changed her last name in the EMPLOYEES table --- let’s verify that our trigger and function both did their job…..

Verify that data was entered into EMPLOYEE\_AUDITS, showing Bush changed to Brown….

SELECT \* FROM **EMPLOYEE\_AUDITS**;

*Provide screenshot of your results here*



We now need to test that our function only manages the changes to the ‘last name’ column and nothing else….. Test this scenario --Lily Brown (nee Brown), changed her last name to ‘Lillian’….

UPDATE **EMPLOYEES** SET first\_name = 'Liliian' WHERE ID = 2;

Verify if the last name of Lily has been updated….Did Lily’s first name get changed?

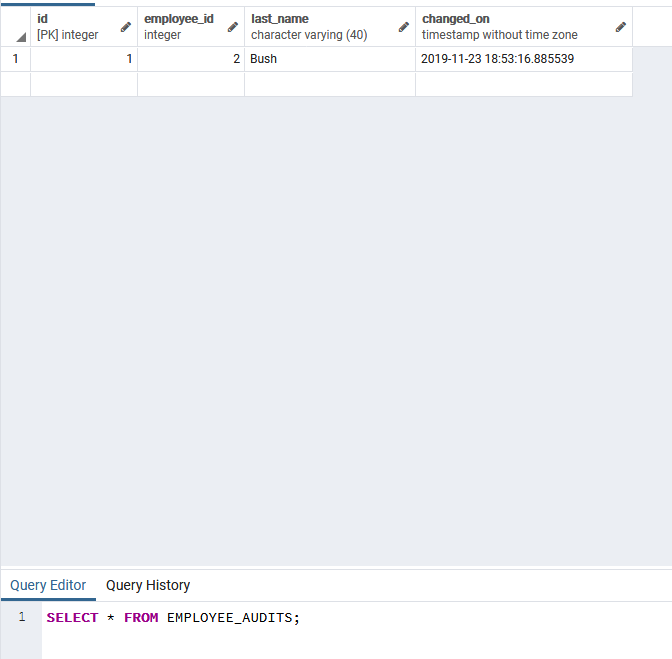
SELECT \* FROM **EMPLOYEES**;

*Provide screenshot of your results here*



Now, let’s make sure that this ‘first name’ change **did not** get logged.

SELECT \* FROM **EMPLOYEE\_AUDITS**;

*Provide screenshot of your results here*