INFO 610 Fall 2020

Week 10.5, Oct 22, 2020

Triggers

chrisfauerbach.github.io/info610_fall_2020/

Database Trigger

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database.

The trigger is mostly used for maintaining the integrity of the information on the database.

For example, when a new record (representing a new worker) is added to the employees table, new records should also be created in the tables of the taxes, vacations and salaries.

Triggers can also be used to log historical data, for example to keep track of employees' previous salaries.

Example

Let's work through through the example above.

- When a new record is inserted into the employees table
  - Insert a record into the taxes_paid table
  - Insert into the vacations tables
  - Insert into the salary table

Tables

```
DROP TABLE IF EXISTS vacation_balance;
DROP TABLE IF EXISTS salary;
DROP TABLE IF EXISTS employee;
CREATE TABLE employee (id serial primary key,
       fname varchar, lname varchar,
       start_date date,
       insert_ts timestamp default now(),
       last_update_ts timestamp default now());

CREATE TABLE vacation_balance(employee_id int primary key,
       year int, vacation_balance int,
       foreign key (employee_id) REFERENCES employee(id));

CREATE TABLE salary(employee_id int primary key,
       annual_salary int,
       effective_ts timestamp default now(),
       foreign key (employee_id) REFERENCES employee(id));
```

Rules

- When an employee is inserted into the employee table
  - Create a vacation balance of 0
  - Define an initial salary of 0 effective now
- For fun make sure the 'last_update_ts' of the employee is changed

  whenever the row is altered

Triggers

A database trigger is procedural code that is automatically executed in response to certain events on a particular table or view in a database.

```
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
```

- Create a vacation balance of 0
- When a new record is inserted into the employees table
- AFTER event
- Event == INSERT on employees table

```
CREATE TRIGGER trigger_employee_salary
    BEFORE UPDATE ON employee
    FOR EACH ROW
    EXECUTE PROCEDURE add_new_salary();

CREATE TRIGGER trigger_employee_vacation
    BEFORE UPDATE ON employee
    FOR EACH ROW
    EXECUTE PROCEDURE add_new_vacation();
```

```
ERROR: function add_new_salary() does not exist
ERROR: function add_new_vacation() does not exist
```

Functions

Code that can be executed in the database

Build example to handle our triggers

```
CREATE OR REPLACE FUNCTION add_new_salary() RETURNS TRIGGER AS $example_table$
   BEGIN
      INSERT INTO salary(employee_id, annual_salary)
      VALUES (new.id,  0);
      RETURN NEW;
   END;
$example_table$ LANGUAGE plpgsql;


CREATE OR REPLACE FUNCTION add_new_vacation() RETURNS TRIGGER AS $example_table$
   BEGIN
      INSERT INTO vacation_balance(employee_id, year, vacation_balance)
      VALUES (new.id, EXTRACT(YEAR FROM NOW()), 0);
      RETURN NEW;
   END;
$example_table$ LANGUAGE plpgsql;
```

Putting it together

Setup:

```
SELECT * FROM EMPLOYEE;
SELECT * FROM SALARY;
SELECT * FROM VACATION_BALANCE;
```

```
no rows selected
no rows selected
no rows selected
```

```
INSERT  INTO employee(fname, lname, start_date)
   VALUES ('Chris', 'Fauerbach', '01-01-2001');
SELECT * FROM SALARY;
SELECT * FROM VACATION_BALANCE;
```

```
1 row inserted.

employee_id annual_salary effective_ts
----------- ------------- --------------------------
3           0             2019-11-19 15:08:21.847554

employee_id year        vacation_balance
----------- ----------- ----------------
3           2019        0
```

Special Variables

When triggers are executed, they have access to several special variables that are automatically created.

These are 'global' variables for your procedure.

- Listed on next page.

```
NEW
Data type RECORD; variable holding the new database row for INSERT/UPDATE operations in row-level
triggers. This variable is NULL in statement-level triggers and for DELETE operations.

OLD
Data type RECORD; variable holding the old database row for UPDATE/DELETE operations in row-level
triggers. This variable is NULL in statement-level triggers and for INSERT operations.

TG_NAME
Data type name; variable that contains the name of the trigger actually fired.

TG_WHEN
Data type text; a string of BEFORE, AFTER, or INSTEAD OF, depending on the trigger's definition.

TG_LEVEL
Data type text; a string of either ROW or STATEMENT depending on the trigger's definition.

TG_OP
Data type text; a string of INSERT, UPDATE, DELETE, or TRUNCATE telling for which operation the trigger
was fired.

TG_RELID
Data type oid; the object ID of the table that caused the trigger invocation.

TG_TABLE_NAME
Data type name; the name of the table that caused the trigger invocation.

TG_TABLE_SCHEMA
Data type name; the name of the schema of the table that caused the trigger
invocation.

TG_NARGS
Data type integer; the number of arguments given to the trigger procedure in the CREATE TRIGGER
statement.

TG_ARGV[]
Data type array of text; the arguments from the CREATE TRIGGER statement.
The index counts from 0.  Invalid indexes (less than 0 or greater than or equal to
tg_nargs) result in a null value.
```

How do we use it?

```
CREATE TABLE emp (
    empname text,
    salary integer,
    last_date timestamp,
    last_user text
);

CREATE FUNCTION emp_stamp() RETURNS trigger AS $emp_stamp$
    BEGIN
        -- Check that empname and salary are given
        IF NEW.empname IS NULL THEN
            RAISE EXCEPTION 'empname cannot be null';
        END IF;
        IF NEW.salary IS NULL THEN
            RAISE EXCEPTION '% cannot have null salary', NEW.empname;
        END IF;

        -- Who works for us when she must pay for it?
        IF NEW.salary < 0 THEN
            RAISE EXCEPTION '% cannot have a negative salary', NEW.empname;
        END IF;

        -- Remember who changed the payroll when
        NEW.last_date := current_timestamp;
        NEW.last_user := current_user;
        RETURN NEW;
    END;
$emp_stamp$ LANGUAGE plpgsql;

CREATE TRIGGER emp_stamp BEFORE INSERT OR UPDATE ON emp
    FOR EACH ROW EXECUTE PROCEDURE emp_stamp();
```