Lecture 12

Trigger

A PostgreSQL **trigger** is a function invoked automatically whenever an event associated with a table occurs. An event could be any of the following: INSERT, UPDATE, DELETE or TRUNCATE.

A **trigger** is a special user-defined function associated with a table. To create a new trigger, you define a trigger function first, and then bind this trigger function to a table.

PostgreSQL trigger types

PostgreSQL provides two main types of triggers:

- Row-level triggers
- Statement-level triggers.

The differences between the two kinds are how many times the trigger is invoked and at what time.

When to use triggers

Triggers are useful in case the database is accessed by various applications, and you want to keep the cross-functionality within the database that runs automatically whenever the data of the table is modified. For example, if you want to keep the history of data without requiring the application to have logic to check for every event such as INSERT or UDPATE.

PostgreSQL triggers vs SQL standard triggers

Even though PostgreSQL implements SQL standard, triggers in PostgreSQL has some specific features:

- PostgreSQL fires trigger for the TRUNCATE event.
- PostgreSQL allows you to define the statement-level trigger on views.
- PostgreSQL requires you to define a user-defined function as the action of the trigger, while the SQL standard allows you to use any SQL commands.

PostgreSQL CREATE TRIGGER

The following illustrates the syntax of creating trigger function:

```
CREATE FUNCTION trigger_function()
   RETURNS TRIGGER
   LANGUAGE PLPGSQL
AS $$
BEGIN
   -- trigger logic
END;
$$
```

Introduction to PostgreSQL CREATE TRIGGER statement

The **CREATE TRIGGER** statement creates a new trigger. The following illustrates the basic syntax of the CREATE TRIGGER statement:

```
CREATE TRIGGER trigger_name
   {BEFORE | AFTER} { event }
   ON table_name
   [FOR [EACH] { ROW | STATEMENT }]
       EXECUTE PROCEDURE trigger_function
```

PostgreSQL CREATE TRIGGER example

```
DROP TABLE IF EXISTS employees;


CREATE TABLE employees(
   id INT GENERATED ALWAYS AS IDENTITY,
   first_name VARCHAR(40) NOT NULL,
   last_name VARCHAR(40) NOT NULL,
   PRIMARY KEY(id)
);
```

Example

```
CREATE TABLE employee_audits (
    id INT GENERATED ALWAYS AS IDENTITY,
    employee_id INT NOT NULL,
    last_name VARCHAR(40) NOT NULL,
    changed_on TIMESTAMP(6) NOT NULL
);
```

Example

```
CREATE OR REPLACE FUNCTION log_last_name_changes()  RETURNS TRIGGER
LANGUAGE PLPGSQL  AS
$$  BEGIN
IF NEW.last_name <> OLD.last_name THEN
INSERT INTO employee_audits(employee_id,last_name,changed_on)
VALUES(OLD.id,OLD.last_name,now());
END IF;
RETURN NEW;
END;
$$
```

Example

```
CREATE TRIGGER last_name_changes  BEFORE UPDATE
ON employees  FOR EACH ROW
EXECUTE PROCEDURE log_last_name_changes();
```

Example

```
INSERT INTO employees (first_name, last_name)  VALUES ('John', 'Doe');
INSERT INTO employees (first_name, last_name)  VALUES ('Lily', 'Bush');
SELECT * FROM employees;
```

id [PK] integer	first_name character varying (40)	last_name character varying (40)
1	John	Doe
2	Lily	Bush

Example

```
UPDATE employees  SET last_name = 'Brown'  WHERE ID = 2;
SELECT * FROM employees;
SELECT * FROM employee_audits;
```

id [PK] integer	first_name character varying (40)	last_name character varying (40)
1	John	Doe
2	Lily	Brown

id integer	employee_id integer	last_name character varying (40)	changed_on timestamp without time zone (6)
1	2	Bush	2025-12-01 09:40:05.931772

PostgreSQL DROP TRIGGER

To delete a trigger from a table, you use the DROP TRIGGER statement with the following syntax:

```
DROP TRIGGER [IF EXISTS] trigger_name
ON table_name [ CASCADE | RESTRICT ];
```

Example

```
DROP TRIGGER last_name_changes
ON employee;
```

PostgreSQL ALTER TRIGGER

The **ALTER TRIGGER** statement allows you to rename a trigger. The following shows the syntax of the **ALTER TRIGGER** statement:

```
ALTER TRIGGER trigger_name
ON table_name
RENAME TO new_trigger_name;
```

Example

```
ALTER TRIGGER before_update_salary
ON employees
RENAME TO salary_before_update;
```