**TRIGGERS**

Triggers are used when you want an action to automatically occur when an event occurs. Common events include the commands insert, update, delete and truncate. Triggers can also be associated with tables, foreign tables or views.

Triggers can execute before or after an event executes. Triggers also can execute instead of another event.

You can put multiple triggers on a table and they execute in alphabetical order. They can’t be triggered manually by a user. Triggers also can’t receive parameters.

If a Trigger is Row Level the Trigger is called for each row that is modified. If a Trigger is Statement level it will execute once regardless of the number of rows.

When can you perform certain actions with triggers SLIDE

This table shows what triggers can execute based on when they are to execute.

For example if a trigger is to execute Before if an event is insert, update, or delete it can perform actions on tables if row level and on tables or views if at statement level.

**Pros of Triggers SLIDE**

 Can be used for auditing, so if something is deleted a trigger could save it in case it is needed later

 They can be used to validate data

 Make certain events always happen to maintain integrity of data

 Insure integrity between different databases

 They can call functions or procedures

 Triggers are recursive so a trigger on a table can call another table with a trigger

**Cons of Triggers SLIDE**

 Triggers add execution overhead

 Nested / recursive trigger errors can be hard to debug

 Invisible to the client which can cause confusion when actions aren’t allowed

-- Create trigger function

CREATE FUNCTION trigger\_function()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$body$

BEGIN

END;

$body$

**-- Create trigger**

CREATE TRIGGER trigger\_name

{BEFORE | AFTER} {event} -- Event : insert, update, insert

ON table\_name

[FOR [EACH] {ROW | STATEMENT}]

EXECUTE PROCEDURE trigger\_function

Trigger Data Logging / Auditing

**-- Log changes to distributor table**

CREATE TABLE distributor(

id SERIAL PRIMARY KEY,

name VARCHAR(100));

**-- Insert distributors**

INSERT INTO distributor (name) VALUES

('Parawholesale'),

('J & B Sales'),

('Steel City Clothing');

SELECT \* FROM distributor;

**-- Table that stores changes to distributor**

CREATE TABLE distributor\_audit(

id SERIAL PRIMARY KEY,

dist\_id INT NOT NULL,

name VARCHAR(100) NOT NULL,

edit\_date TIMESTAMP NOT NULL);

**-- Create trigger function**

CREATE OR REPLACE FUNCTION fn\_log\_dist\_name\_change()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$body$

BEGIN

-- If name changes log the change

IF NEW.name <> OLD.name THEN

INSERT INTO distributor\_audit

(dist\_id, name, edit\_date)

VALUES

(OLD.id, OLD.name, NOW());

END IF;

-- Trigger information Variables

RAISE NOTICE 'Trigger Name : %', TG\_NAME;

RAISE NOTICE 'Table Name : %', TG\_TABLE\_NAME;

RAISE NOTICE 'Operation : %', TG\_OP;

RAISE NOTICE 'When Executed : %', TG\_WHEN;

RAISE NOTICE 'Row or Statement : %', TG\_LEVEL;

RAISE NOTICE 'Table Schema : %', TG\_TABLE\_SCHEMA;

-- Return the updated new data

RETURN NEW;

END;

$body$

**-- Bind function to trigger**

CREATE TRIGGER tr\_dist\_name\_changed

-- Call function before name is updated

BEFORE UPDATE

ON distributor

-- We want to run this on every row where an update occurs

FOR EACH ROW

EXECUTE PROCEDURE fn\_log\_dist\_name\_change();

**-- Update distributor name and log changes**

UPDATE distributor

SET name = 'Western Clothing'

WHERE id = 2;

**-- Check the log**

SELECT \* FROM distributor\_audit;

**Conditional Triggers**

You can revoke delete on tables for some users, or you can use triggers.

-- Block insert, update and delete on the weekend

CREATE OR REPLACE FUNCTION fn\_block\_weekend\_changes()

RETURNS TRIGGER

LANGUAGE PLPGSQL

AS

$body$

BEGIN

RAISE NOTICE 'No database changes allowed on the weekend';

RETURN NULL;

END;

$body$

**-- Bind function to trigger**

CREATE TRIGGER tr\_block\_weekend\_changes

-- Call function before name is updated

BEFORE UPDATE OR INSERT OR DELETE OR TRUNCATE

ON distributor

-- We want to run this on statement level

FOR EACH STATEMENT

-- Block if on weekend

WHEN(

EXTRACT('DOW' FROM CURRENT\_TIMESTAMP) BETWEEN 6 AND 7

)

EXECUTE PROCEDURE fn\_block\_weekend\_changes();

UPDATE distributor

SET name = 'Western Clothing'

WHERE id = 2;

**-- Drop triggers**

DROP EVENT TRIGGER tr\_block\_weekend\_changes;