**Assignment Day 9**

1. Create AFTER UPDATE trigger to track product price changes

· Create product\_price\_audit table with below columns:

audit\_id SERIAL PRIMARY KEY,

product\_id INT,

product\_name VARCHAR(40),

old\_price DECIMAL(10,2),

new\_price DECIMAL(10,2),

change\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

user\_name VARCHAR(50) DEFAULT CURRENT\_USER

create table if not exists product\_price\_audit(

audit\_id SERIAL PRIMARY KEY,

product\_id INT,

product\_name VARCHAR(40),

old\_price DECIMAL(10,2),

new\_price DECIMAL(10,2),

change\_date TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

user\_name VARCHAR(50) DEFAULT CURRENT\_USER

);

A white rectangular object with a blue border

AI-generated content may be incorrect.

Create a trigger function with the below logic:

INSERT INTO product\_price\_audit (

product\_id,

product\_name,

old\_price,

new\_price

)

VALUES (

OLD.product\_id,

OLD.product\_name,

OLD.unit\_price,

NEW.unit\_price

);

-----Defining the function

CREATE FUNCTION log\_price\_changes()

RETURNS TRIGGER AS $$

BEGIN

INSERT INTO product\_price\_audit (

product\_id,

product\_name,

old\_price,

new\_price

) VALUES (

OLD.product\_id,

OLD.product\_name,

OLD.unit\_price,

NEW.unit\_price

);

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

A white background with black and white clouds

AI-generated content may be incorrect.

· Create a row level trigger for below event:

AFTER UPDATE OF unit\_price ON products

CREATE OR REPLACE TRIGGER track\_price\_changes

AFTER UPDATE OF unit\_price ON products

FOR EACH ROW

WHEN(OLD.unit\_price <> NEW.unit\_price)

EXECUTE FUNCTION log\_price\_changes()

A screenshot of a computer

AI-generated content may be incorrect.

·

Test the trigger by updating the product price by 10% to any one product\_id.

update products

set unit\_price = unit\_price \*1.10

where product\_id = 3

A screenshot of a computer

AI-generated content may be incorrect.

SELECT \* FROM product\_price\_audit ORDER BY change\_date DESC;

A close-up of a computer screen

AI-generated content may be incorrect.

2. Create stored procedure using IN and INOUT parameters to assign tasks to employees

Parameters:

IN p\_employee\_id INT,

IN p\_task\_name VARCHAR(50),

INOUT p\_task\_count INT DEFAULT 0

Inside Logic: Create table employee\_tasks:

CREATE TABLE IF NOT EXISTS employee\_tasks (

task\_id SERIAL PRIMARY KEY,

employee\_id INT,

task\_name VARCHAR(50),

assigned\_date DATE DEFAULT CURRENT\_DATE

);

CREATE TABLE IF NOT EXISTS employee\_tasks(

task\_id SERIAL PRIMARY KEY,

employee\_id INT,

task\_name VARCHAR(50),

assigned\_date DATE DEFAULT CURRENT\_DATE

);

Insert employee\_id, task\_name into employee\_tasks

· Count total tasks for employee and put the total count into p\_task\_count .

· Raise NOTICE message:  
 RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',

p\_task\_name, p\_employee\_id, p\_task\_count;

-----Create Stored Procedure

CREATE OR REPLACE PROCEDURE assign\_tasks(

-----Input parameters

IN p\_employee\_id INT,

IN p\_task\_name VARCHAR(50),

----Output parameters

INOUT p\_task\_count INT Default 0)

LANGUAGE plpgsql

AS $$

BEGIN

insert into employee\_tasks(employee\_id,task\_name)

values(p\_employee\_id,p\_task\_name);

----calculate total tasks for each employee

SELECT

COUNT(\*)::INT

INTO p\_task\_count

FROM employee\_tasks

where employee\_id = p\_employee\_id

AND

task\_name = p\_task\_name;

---Raise Notice

RAISE NOTICE 'Task "%" assigned to employee %. Total tasks: %',

p\_task\_name, p\_employee\_id, p\_task\_count;

END

$$;

A white rectangular object with a black border

AI-generated content may be incorrect.

After creating stored procedure test by calling it:

CALL assign\_task(1, 'Review Reports');

CALL assign\_tasks(1, 'Review Reports')

A screenshot of a computer

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

You should see the entry in employee\_tasks table.

A screenshot of a computer

AI-generated content may be incorrect.