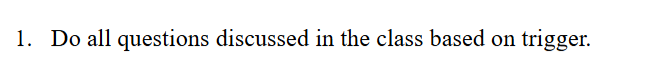
**Database management systems**

# **PLSQL Labsheet – 2**

****

1. **Update the emp\_no by incrementing it every time an employee joins.**

-- Step 1: Create Tables

CREATE TABLE Dept (

Dno INT PRIMARY KEY,

Dname VARCHAR(50),

Emp\_count INT DEFAULT 0

);

CREATE TABLE Emp (

Eno INT PRIMARY KEY,

Ename VARCHAR(50),

Sal NUMERIC(10, 2),

Dno INT REFERENCES Dept(Dno)

);

-- Step 2: Function to Increment Employee Count on Insert

CREATE FUNCTION update\_count() RETURNS TRIGGER AS

$$

BEGIN

UPDATE Dept

SET Emp\_count = Emp\_count + 1

WHERE Dno = NEW.Dno;

RETURN NEW;

END;

$$

LANGUAGE 'plpgsql';

-- Step 3: Trigger to Call update\_count Function on Insert

CREATE TRIGGER emp\_update

AFTER INSERT ON Emp

FOR EACH ROW

EXECUTE FUNCTION update\_count();

-- Step 4: Insert Departments

INSERT INTO Dept (Dno, Dname) VALUES

(1, 'HR'),

(2, 'IT'),

(3, 'Finance');

-- Step 5: Insert Employees

INSERT INTO Emp (Eno, Ename, Sal, Dno) VALUES

(101, 'Alice', 50000, 1),

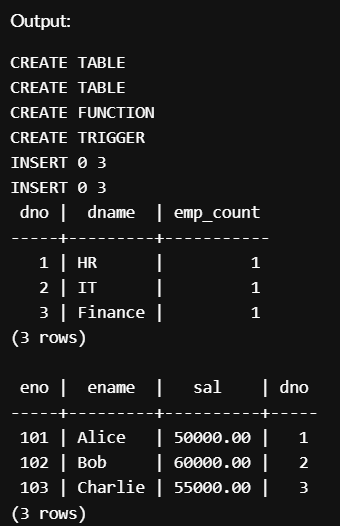
(102, 'Bob', 60000, 2),

(103, 'Charlie', 55000, 3);

-- Step 6: Verify Results

SELECT \* FROM Dept;

SELECT \* FROM Emp;

****

1. **Consider the following Emp table; whenever the salary column to any employee an audit to that change must be automatically need to be added to the sal\_audit table.**

CREATE TABLE Emp (

eno INT PRIMARY KEY,

name TEXT,

sal NUMERIC

);

CREATE TABLE sal\_audit (

eno INT,

old\_sal NUMERIC,

new\_sal NUMERIC,

modified\_by TEXT,

modified\_at TIMESTAMP

);

CREATE FUNCTION update\_sal\_audit()

RETURNS TRIGGER AS

$$

BEGIN

-- Insert the old and new salary information into the sal\_audit table

INSERT INTO sal\_audit (eno, old\_sal, new\_sal, modified\_by, modified\_at)

VALUES (OLD.eno, OLD.sal, NEW.sal, USER, now());

-- Return the NEW row to allow the update to proceed

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER sal\_audit\_trigger

AFTER UPDATE OF sal ON Emp

FOR EACH ROW

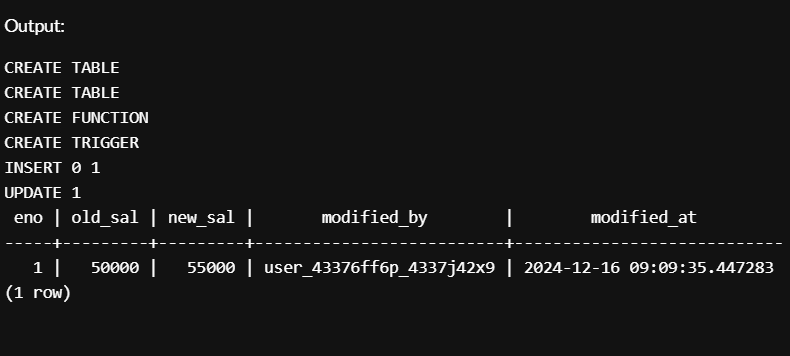
EXECUTE FUNCTION update\_sal\_audit();

--example

INSERT INTO Emp (eno, name, sal) VALUES (1, 'Alice', 50000);

UPDATE Emp SET sal = 55000 WHERE eno = 1;

SELECT \* FROM sal\_audit;





CREATE TABLE Emp (

eno INT,

name TEXT,

sal NUMERIC

);

CREATE FUNCTION data\_validate()

RETURNS TRIGGER AS

$$

BEGIN

-- Validate that 'eno' is not NULL

IF NEW.eno IS NULL THEN

RAISE EXCEPTION 'Employee number (eno) cannot be NULL';

END IF;

-- Validate that 'sal' is greater than 0

IF NEW.sal <= 0 THEN

RAISE EXCEPTION 'Salary must be greater than 0';

END IF;

-- Return the NEW row to proceed with the operation

RETURN NEW;

END;

$$

LANGUAGE plpgsql;

CREATE TRIGGER validate\_emp\_data

BEFORE INSERT OR UPDATE ON Emp

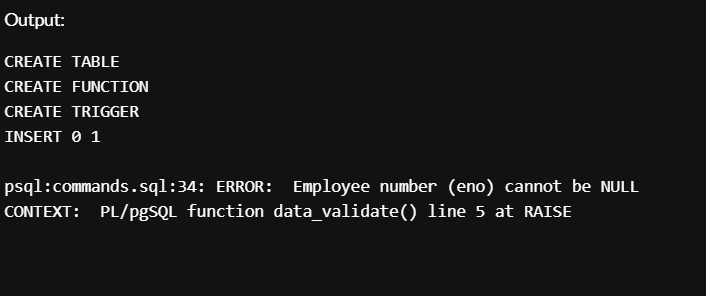
FOR EACH ROW

EXECUTE FUNCTION data\_validate();

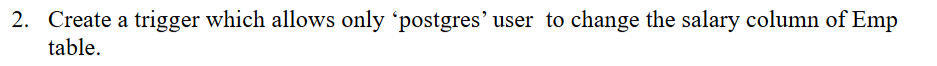
INSERT INTO Emp (eno, name, sal) VALUES (1, 'Alice', 50000); -- Succeeds

INSERT INTO Emp (eno, name, sal) VALUES (NULL, 'Bob', 30000); -- Fails with "Employee number (eno) cannot be NULL"

INSERT INTO Emp (eno, name, sal) VALUES (2, 'Charlie', -5000); -- Fails with "Salary must be greater than 0"





****

**-- Create Emp table**

**CREATE TABLE Emp (**

**emp\_id INT PRIMARY KEY,**

**emp\_name VARCHAR(100),**

**salary DECIMAL**

**);**

**-- Insert sample data**

**INSERT INTO Emp (emp\_id, emp\_name, salary)**

**VALUES (1, 'John Doe', 50000),**

**(2, 'Jane Smith', 60000),**

**(3, 'Alice Johnson', 70000);**

**CREATE OR REPLACE FUNCTION check\_postgres\_user()**

**RETURNS TRIGGER AS $$**

**BEGIN**

**-- Check if the user is 'postgres'**

**IF current\_user <> 'postgres' THEN**

**RAISE EXCEPTION 'Only the postgres user can modify the salary column';**

**END IF;**

**RETURN NEW;**

**END;**

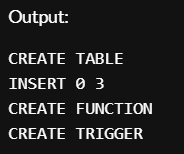
**$$ LANGUAGE plpgsql;**

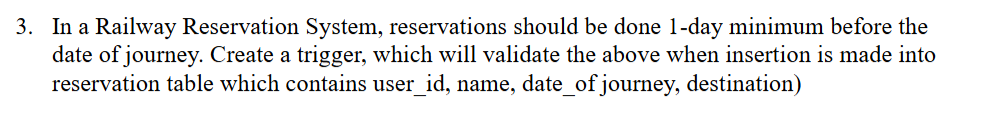
**CREATE TRIGGER salary\_update\_trigger**

**BEFORE UPDATE OF salary ON Emp**

**FOR EACH ROW**

**EXECUTE FUNCTION check\_postgres\_user();**

****

****

**-- Create reservation table**

**CREATE TABLE reservation (**

**user\_id INT,**

**name VARCHAR(100),**

**date\_of\_journey DATE,**

**destination VARCHAR(100),**

**PRIMARY KEY (user\_id, date\_of\_journey)**

**);**

**-- Insert sample data**

**INSERT INTO reservation (user\_id, name, date\_of\_journey, destination)**

**VALUES (1, 'John Doe', '2024-12-17', 'Paris'),**

**(2, 'Alice Smith', '2024-12-19', 'London'),**

**(3, 'Bob Brown', '2024-12-16', 'New York'); -- This should raise an exception in the trigger**

**-- Create trigger function to validate reservation date**

**CREATE OR REPLACE FUNCTION validate\_reservation\_date()**

**RETURNS TRIGGER AS $$**

**BEGIN**

**IF NEW.date\_of\_journey < CURRENT\_DATE + INTERVAL '1 day' THEN**

**RAISE EXCEPTION 'Reservations must be made at least one day in advance.';**

**END IF;**

**RETURN NEW;**

**END;**

**$$ LANGUAGE plpgsql;**

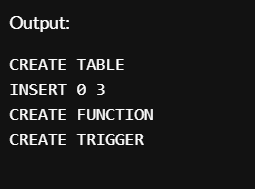
**-- Create trigger to apply the function on insert into reservation table**

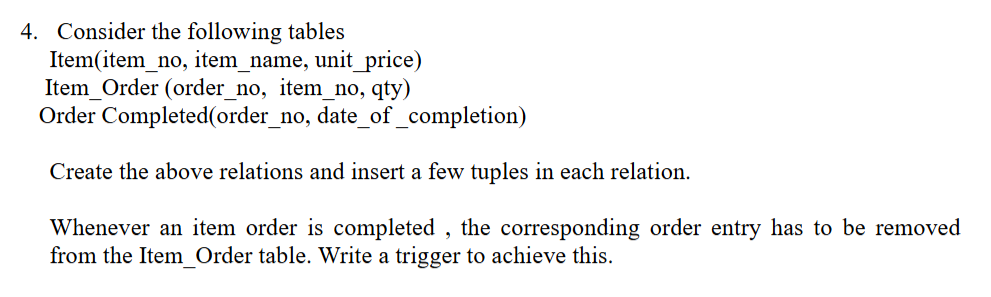
**CREATE TRIGGER reservation\_date\_trigger**

**BEFORE INSERT ON reservation**

**FOR EACH ROW**

**EXECUTE FUNCTION validate\_reservation\_date();**

****

****

**-- Create Item table**

**CREATE TABLE Item (**

**item\_no INT PRIMARY KEY,**

**item\_name VARCHAR(100),**

**unit\_price DECIMAL**

**);**

**-- Create Item\_Order table**

**CREATE TABLE Item\_Order (**

**order\_no INT PRIMARY KEY,**

**item\_no INT,**

**qty INT,**

**FOREIGN KEY (item\_no) REFERENCES Item(item\_no)**

**);**

**-- Create Order\_Completed table**

**CREATE TABLE Order\_Completed (**

**order\_no INT,**

**date\_of\_completion DATE,**

**FOREIGN KEY (order\_no) REFERENCES Item\_Order(order\_no)**

**);**

**-- Insert sample data into Item table**

**INSERT INTO Item (item\_no, item\_name, unit\_price)**

**VALUES (1, 'Laptop', 1000),**

**(2, 'Phone', 500),**

**(3, 'Headphone', 100);**

**-- Insert sample data into Item\_Order table**

**INSERT INTO Item\_Order (order\_no, item\_no, qty)**

**VALUES (101, 1, 2),**

**(102, 2, 3);**

**-- Insert sample data into Order\_Completed table**

**INSERT INTO Order\_Completed (order\_no, date\_of\_completion)**

**VALUES (101, '2024-12-15'),**

**(102, '2024-12-16');**

**-- Create trigger function to remove item order entry after completion**

**CREATE OR REPLACE FUNCTION remove\_item\_order()**

**RETURNS TRIGGER AS $$**

**BEGIN**

**DELETE FROM Item\_Order WHERE order\_no = NEW.order\_no;**

**RETURN NEW;**

**END;**

**$$ LANGUAGE plpgsql;**

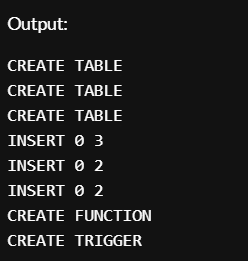
**-- Create trigger to apply the function on insert into Order\_Completed**

**CREATE TRIGGER remove\_item\_order\_trigger**

**AFTER INSERT ON Order\_Completed**

**FOR EACH ROW**

**EXECUTE FUNCTION remove\_item\_order();**

****