DATABASE MANAGEMENT SYSTEMS

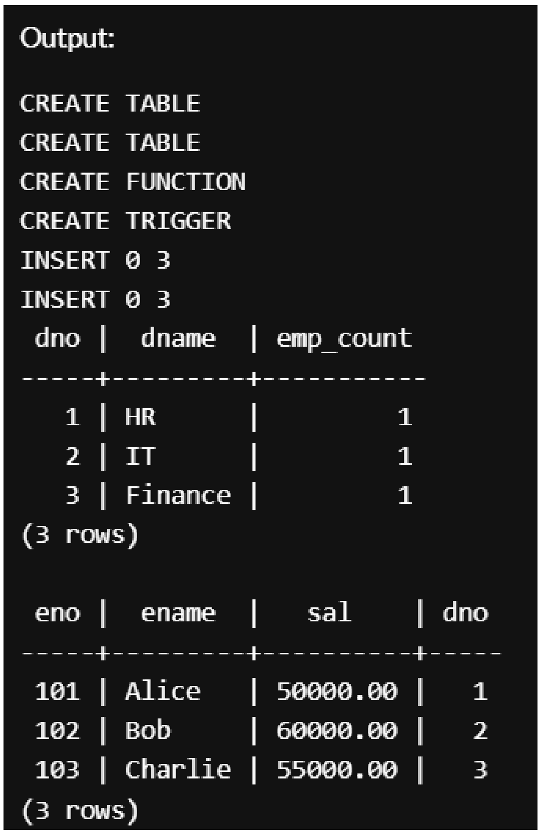
PLSQL Labsheet – 2

1. Do all questions discussed in the class based on trigger.

   a. **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;
```

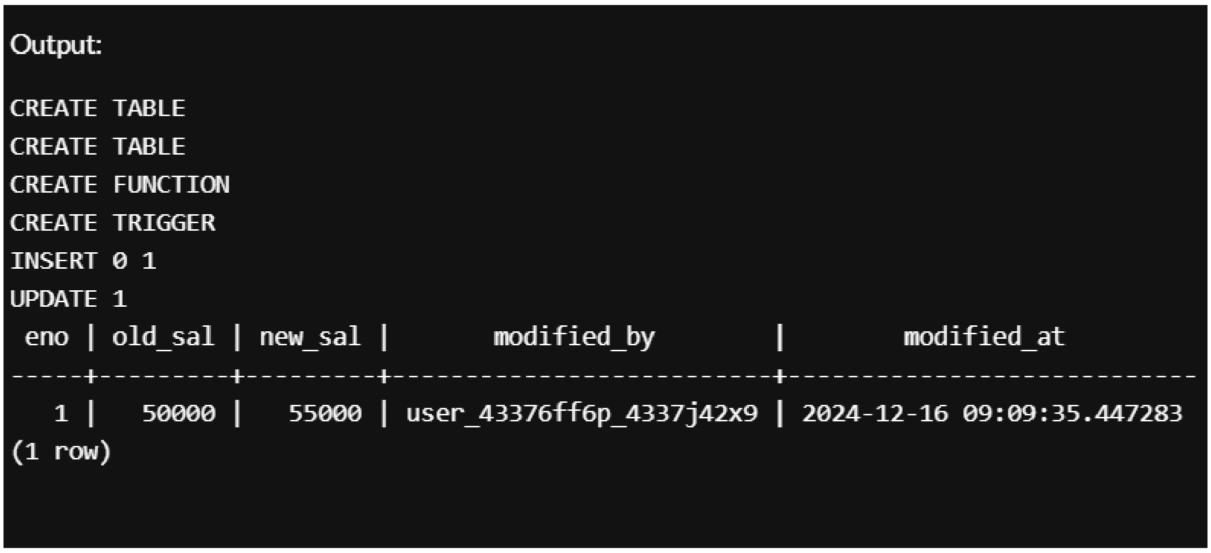
```
RETURN NEW;

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;
```



```
Output:

CREATE TABLE
CREATE TABLE
CREATE FUNCTION
CREATE TRIGGER
INSERT 0 3
INSERT 0 3
 dno |  dname   | emp_count
-----+---------+-----------
   1 | HR      |         1
   2 | IT      |         1
   3 | Finance |         1
(3 rows)

 eno |  ename   |   sal    | dno
-----+---------+----------+-----
 101 | Alice   | 50000.00 |   1
 102 | Bob     | 60000.00 |   2
 103 | Charlie | 55000.00 |   3
(3 rows)
```

b. 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;
```

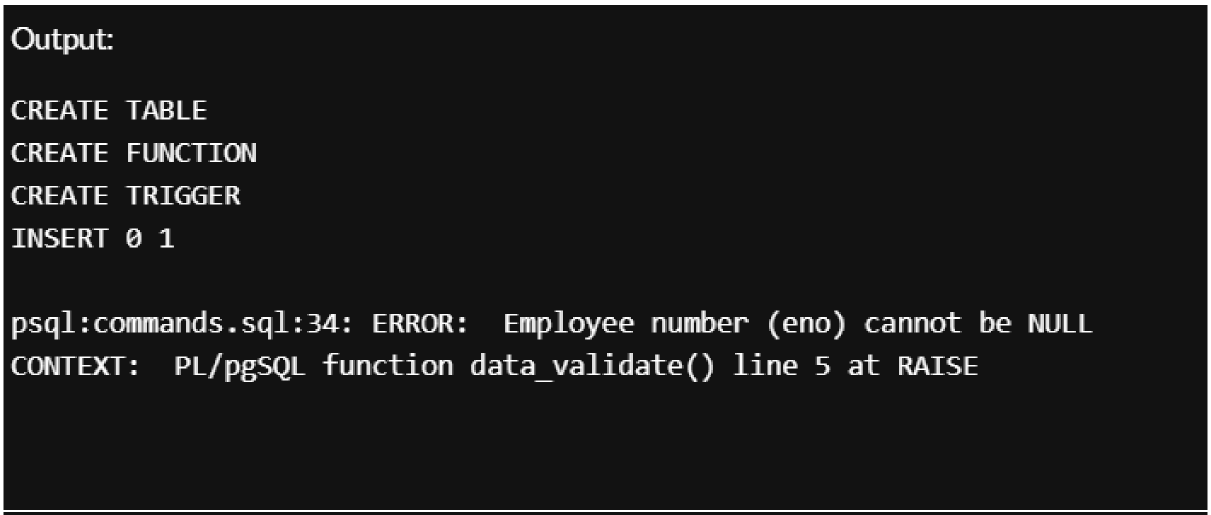

Output:

```
CREATE TABLE
CREATE TABLE
CREATE FUNCTION
CREATE TRIGGER
INSERT 0 1
UPDATE 1
 eno | old_sal | new_sal |       modified_by        |        modified_at
-----+---------+---------+--------------------------+----------------------------
   1 |   50000 |   55000 | user_43376ff6p_4337j42x9 | 2024-12-16 09:09:35.447283
(1 row)
```

c.

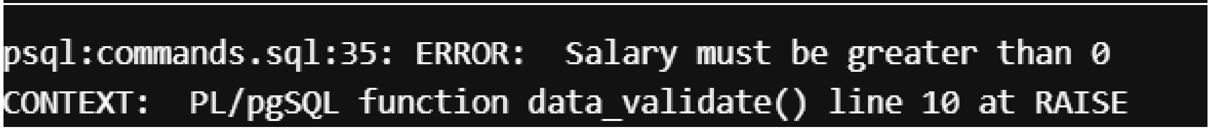
```
CREATE TABLE Emp (
    eno INT,
    name TEXT,
    sal NUMERIC
);

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"
```



Output:

```
CREATE TABLE
CREATE FUNCTION
CREATE TRIGGER
INSERT 0 1

psql:commands.sql:34: ERROR:  Employee number (eno) cannot be NULL
CONTEXT:  PL/pgSQL function data_validate() line 5 at RAISE
```

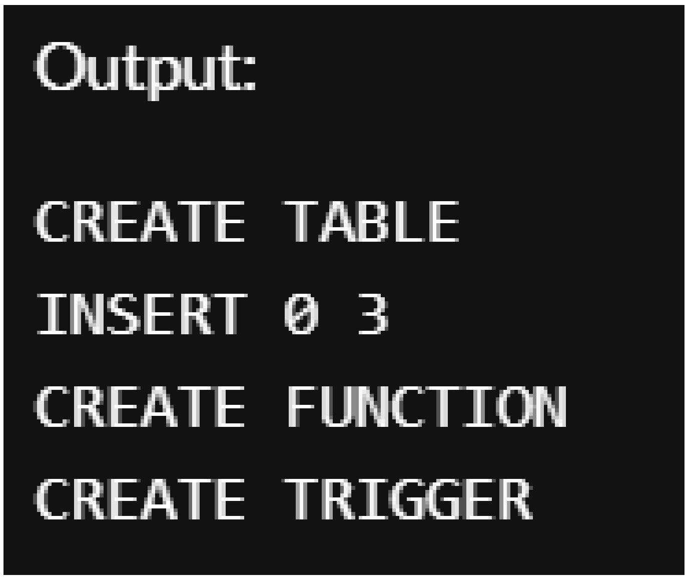


```
psql:commands.sql:35: ERROR:  Salary must be greater than 0
CONTEXT:  PL/pgSQL function data_validate() line 10 at RAISE
```

2. Create a trigger which allows only ‘postgres’ user to change the salary column of Emp table.

```
-- 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();
```



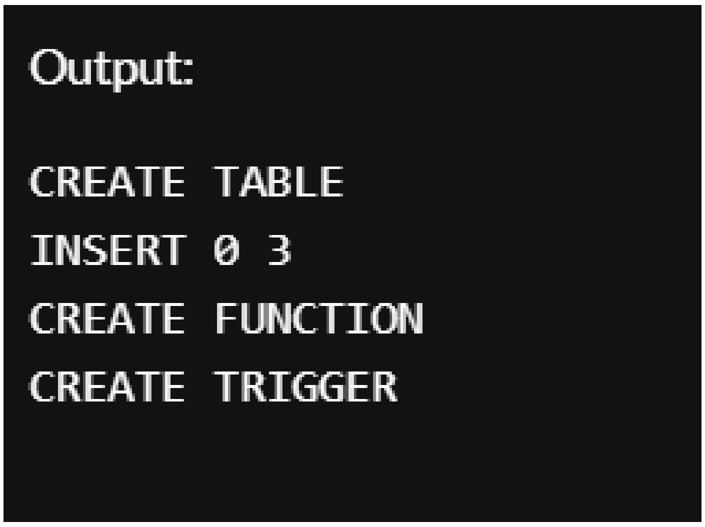
Output:

```
CREATE TABLE
INSERT 0 3
CREATE FUNCTION
CREATE TRIGGER
```

3. In a Railway Reservation System, reservations should be done 1-day minimum before the date of journey. Create a trigger, which will validate the above when insertion is made into reservation table which contains user_id, name, date_of journey, destination)

```
-- 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();
```



Output:

```
CREATE TABLE
INSERT 0 3
CREATE FUNCTION
CREATE TRIGGER
```

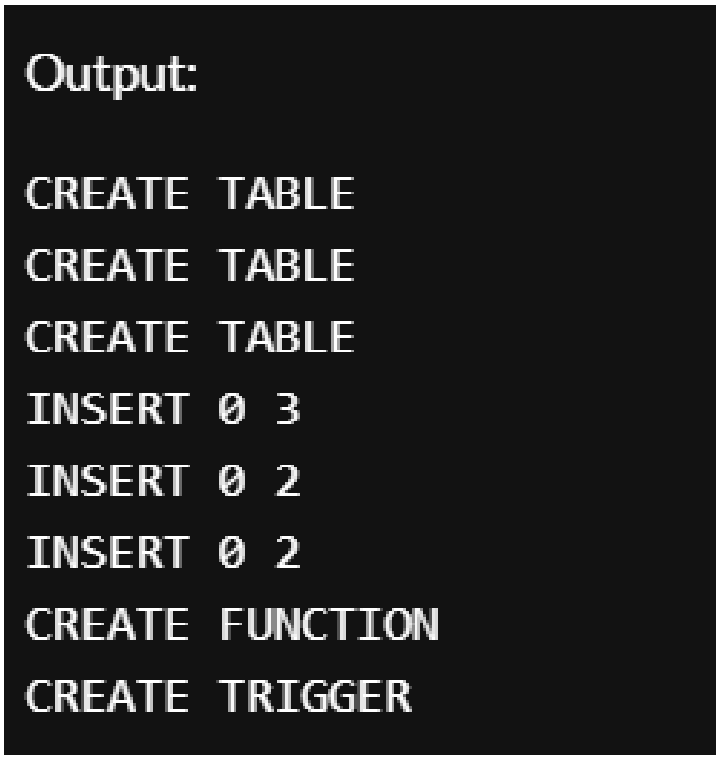
4. Consider the following tables
   Item(item_no, item_name, unit_price)
   Item_Order (order_no, item_no, qty)
   Order Completed(order_no, date_of _completion)

   Create the above relations and insert a few tuples in each relation.

   Whenever an item order is completed , the corresponding order entry has to be removed from the Item_Order table. Write a trigger to achieve this.

```
-- 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();
```



```
Output:

CREATE TABLE
CREATE TABLE
CREATE TABLE
INSERT 0 3
INSERT 0 2
INSERT 0 2
CREATE FUNCTION
CREATE TRIGGER
```