Lab 4

1. Create a multiply function that accepts two numbers and returns their Product.

```
postgres=# CREATE OR REPLACE FUNCTION multiply(num1 NUMERIC, num2 NUMERIC)
RETURNS NUMERIC AS $$
BEGIN
    RETURN num1 * num2;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT multiply(5, 3);  -- Returns 15
 multiply
----------
       15
(1 row)

postgres=#
```

2. Create a hello_world function that takes a name as input and returns a personalized welcome message for that name.

```
postgres=# CREATE OR REPLACE FUNCTION hello_world(name VARCHAR)
RETURNS VARCHAR AS $$
BEGIN
    RETURN 'Hello, ' || name || '! Welcome to our database system.';
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT hello_world('Ayooya');
                  hello_world
-----------------------------------------------
 Hello, Ayooya! Welcome to our database system.
(1 row)

postgres=#
```

3. Create a function that accepts a number and determines whether it is

odd or even.

```
postgres=# CREATE OR REPLACE FUNCTION check_odd_even(num INTEGER)
RETURNS VARCHAR AS $$
BEGIN
    IF num % 2 = 0 THEN
        RETURN num || ' is even';
    ELSE
        RETURN num || ' is odd';
    END IF;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
postgres=# SELECT check_odd_even(4);
 check_odd_even
----------------
 4 is even
(1 row)

postgres=# SELECT check_odd_even(7);
 check_odd_even
----------------
 7 is odd
(1 row)
```

4. Create a function that takes a Student ID as input and retrieves all

information related to that student.

```
        ^
student_tracking=# CREATE OR REPLACE FUNCTION get_student_info(student_id INTEG
R)
RETURNS TABLE (
    student_name VARCHAR,
    email VARCHAR,
    address TEXT,
    track_name VARCHAR,
    birth_date DATE,
    gender VARCHAR
) AS $$
BEGIN
    RETURN QUERY
    SELECT s.e_name, s.email, s.address, t.track_name, s.birth_date, s.gender
    FROM student s
    JOIN track t ON s.track_id = t.id
    WHERE s.id = student_id;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
```

```
 student_name |      email       |   address   | track_name | birth_date | genc
e
--------------+------------------+-------------+------------+------------+-----
--
 Mohammed Ali | mohammed@iti.com | 123 Main St | OS         | 1990-05-15 | Male
(1 row)
```

5. Implement a function that takes the name of a subject and calculates

the average grades for that subject.

```
student_tracking=# CREATE OR REPLACE FUNCTION calculate_subject_avg(subject_n
 VARCHAR)
RETURNS NUMERIC AS $$
DECLARE
    avg_grade NUMERIC;
BEGIN
    SELECT AVG(g.grade) INTO avg_grade
    FROM grades g
    JOIN subject sub ON g.sub_id = sub.id
    WHERE sub.sub_name = subject_name;

    RETURN ROUND(avg_grade, 2);
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
student_tracking=# SELECT calculate_subject_avg('Database Systems');
 calculate_subject_avg
-----------------------
                 85.00
(1 row)
```

6. Create a trigger to automatically save deleted student records from the Student table to the Deleted_Students table.

```
student_tracking=# CREATE TABLE deleted_students (
    id INTEGER,
    e_name VARCHAR(100),
    email VARCHAR(100),
    address TEXT,
    track_id INTEGER,
    birth_date DATE,
    gender VARCHAR(6),
    deletion_time TIMESTAMP DEFAULT NOW()
);

CREATE TABLE
student_tracking=# CREATE OR REPLACE FUNCTION archive_deleted_student()
RETURNS TRIGGER AS $$
BEGIN
    INSERT INTO deleted_students (id, e_name, email, address, track_id, birth
te, gender)
    VALUES (OLD.id, OLD.e_name, OLD.email, OLD.address, OLD.track_id, OLD.bir
date, OLD.gender);

    RETURN OLD;
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
```

```
student_tracking=# CREATE TRIGGER before_student_delete
BEFORE DELETE ON student
FOR EACH ROW EXECUTE FUNCTION archive_deleted_student();
CREATE TRIGGER
```

7. Create a trigger to monitor changes made to the student table, including additions, updates, and deletions. This trigger will record the time of each action and provide a description of the action in another table.

```
 audit_id | student_id | action |        action_time         |                                 old_data                                  |
                         new_data                                        |          description
----------+------------+--------+----------------------------+---------------------------------------------------------------------------+----
--------------------------------------------------------------------------+---------------------------------
        1 |          7 | INSERT | 2025-05-06 08:24:41.940858 |                                                                           | {"i
d": 7, "email": "test@example.com", "e_name": "Test Student", "gender": null, "address": null, "track_id": 1, "birth_date": null}    | New student added: Test Student
        2 |          7 | UPDATE | 2025-05-06 08:24:48.327639 | {"id": 7, "email": "test@example.com", "e_name": "Test Student", "gender": null, "address": null, "track_id": 1, "birth_date": null}    | {"i
d": 7, "email": "updated@example.com", "e_name": "Test Student", "gender": null, "address": null, "track_id": 1, "birth_date": null} | Student updated: Test Student
        3 |          7 | DELETE | 2025-05-06 08:24:54.876975 | {"id": 7, "email": "updated@example.com", "e_name": "Test Student", "gender": null, "address": null, "track_id": 1, "birth_date": null} |
                                                                                                                                      | Student deleted: Test Student
(3 rows)
```

```
student_tracking=# CREATE TABLE student_audit (
    audit_id SERIAL PRIMARY KEY,
    student_id INTEGER,
    action VARCHAR(10),   -- INSERT, UPDATE, DELETE
    action_time TIMESTAMP DEFAULT NOW(),
    old_data JSONB,
    new_data JSONB,
    description TEXT
);
CREATE TABLE
student_tracking=# CREATE OR REPLACE FUNCTION log_student_changes()
RETURNS TRIGGER AS $$
DECLARE
    action_desc TEXT;
BEGIN
    IF TG_OP = 'INSERT' THEN
        action_desc := 'New student added: ' || NEW.e_name;
        INSERT INTO student_audit (student_id, action, new_data, description)
        VALUES (NEW.id, 'INSERT', to_jsonb(NEW), action_desc);
    ELSIF TG_OP = 'UPDATE' THEN
        action_desc := 'Student updated: ' || NEW.e_name;
        INSERT INTO student_audit (student_id, action, old_data, new_data, description)
        VALUES (NEW.id, 'UPDATE', to_jsonb(OLD), to_jsonb(NEW), action_desc);
    ELSIF TG_OP = 'DELETE' THEN
        action_desc := 'Student deleted: ' || OLD.e_name;
        INSERT INTO student_audit (student_id, action, old_data, description)
        VALUES (OLD.id, 'DELETE', to_jsonb(OLD), action_desc);
    END IF;

    RETURN NULL; -- For AFTER trigger
END;
$$ LANGUAGE plpgsql;
CREATE FUNCTION
student_tracking=# CREATE TRIGGER after_student_changes
AFTER INSERT OR UPDATE OR DELETE ON student
FOR EACH ROW EXECUTE FUNCTION log_student_changes();
CREATE TRIGGER
student_tracking=# INSERT INTO student (e_name, email, track_id)
VALUES ('Test Student', 'test@example.com', 1);
INSERT 0 1
student_tracking=# UPDATE student SET email = 'updated@example.com' WHERE e_name = 'Test Student'
UPDATE 1
student_tracking=# DELETE FROM student WHERE e_name = 'Test Student';
DELETE 1
student_tracking=# SELECT * FROM student_audit;
```