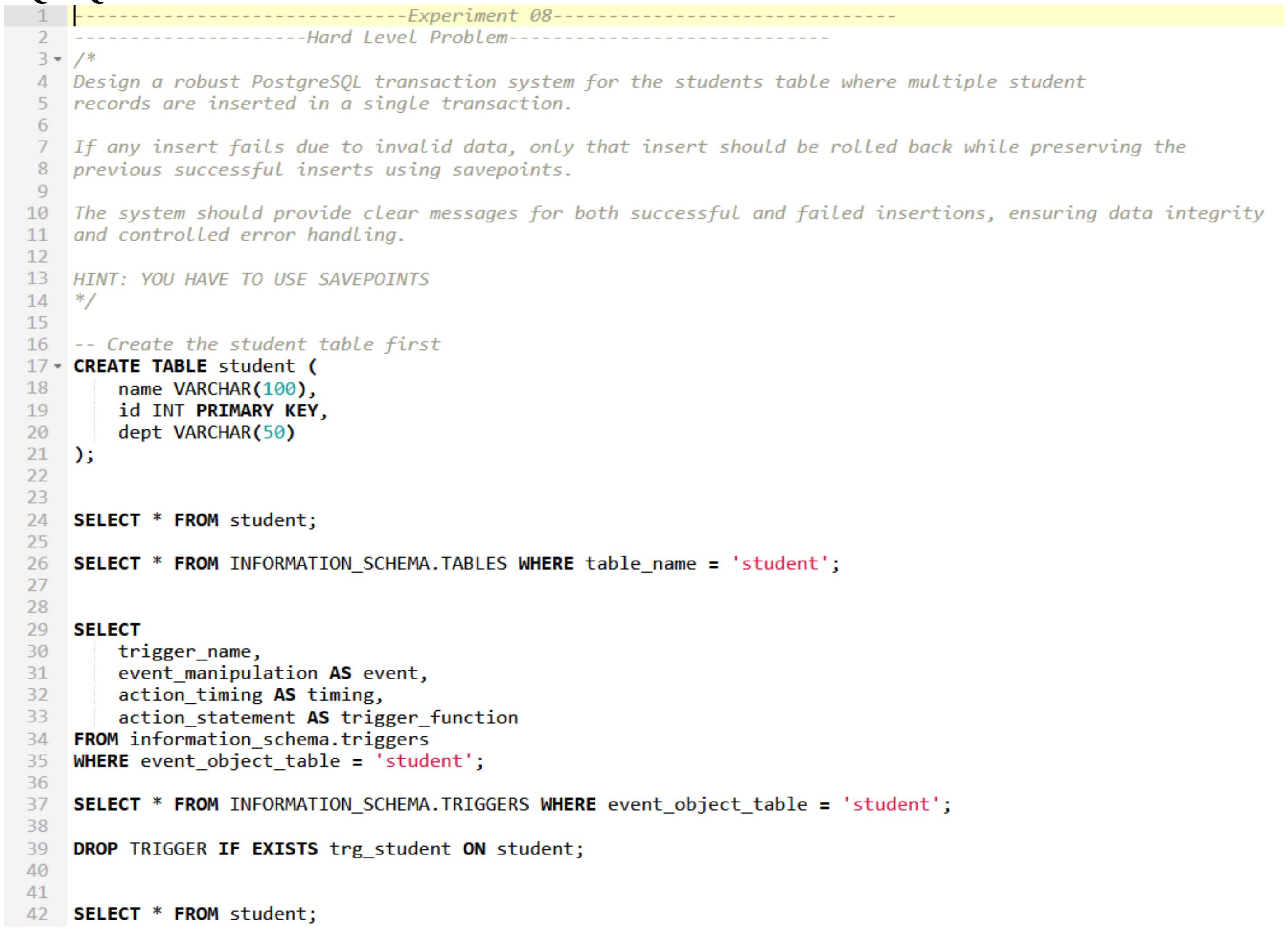
Experiment No 8

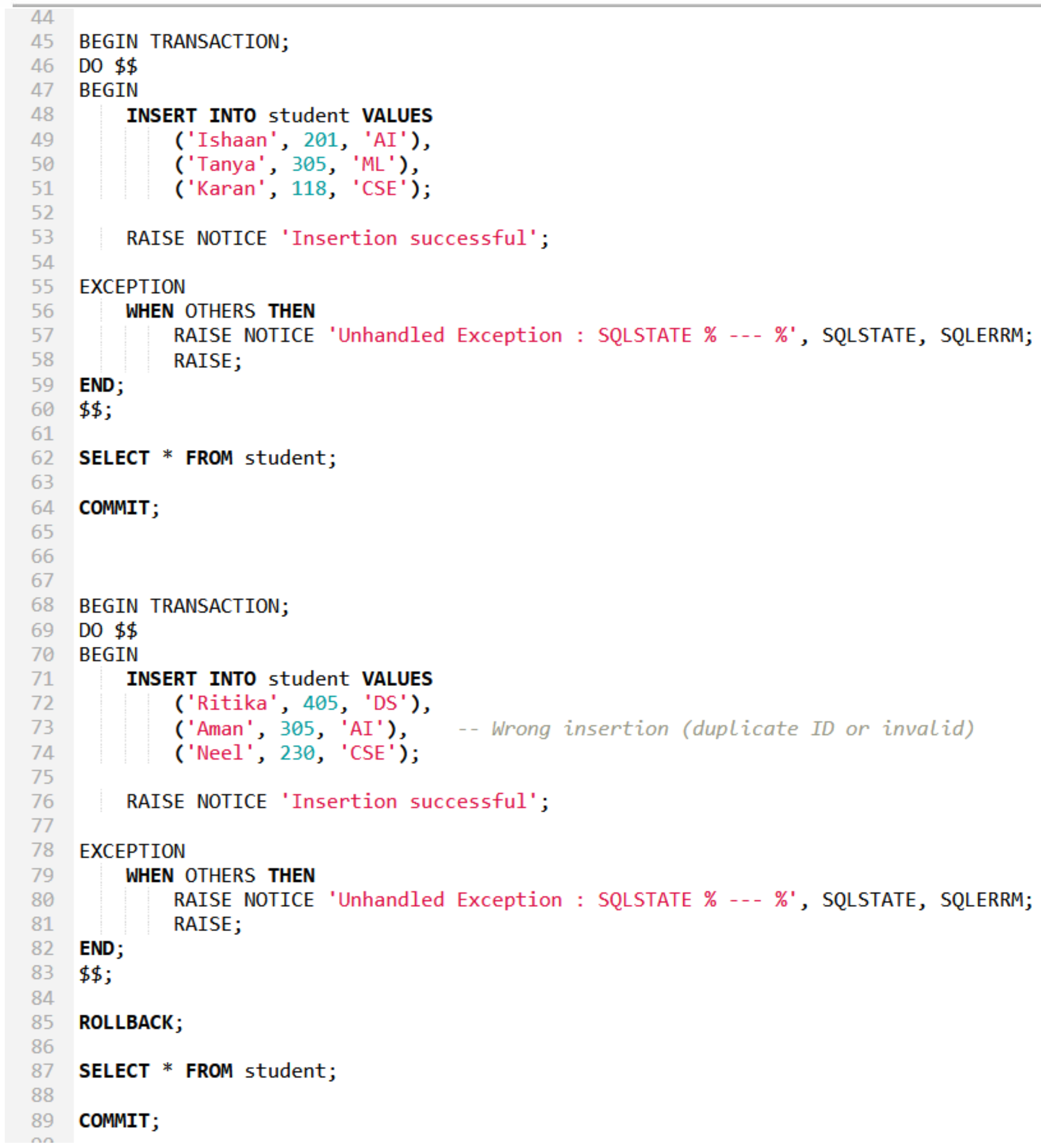
Student Name: Bhavya Gupta **UID: 23BIS70147**
Branch: BE CSE (IS) **Section/Group: 23AIT-KRG(2A)**
Semester: 5th **Date of Performance:17-Oct-2025**
Subject:ADBMS **Subject Code: 23-CSP-333**

Requirements: Design a robust PostgreSQL transaction system for the students table where multiple student records are inserted in a single transaction.
If any insert fails due to invalid data, only that insert should be rolled back while preserving the previous successful inserts using savepoints.
The system should provide clear messages for both successful and failed insertions, ensuring data integrity
and controlled error handling.

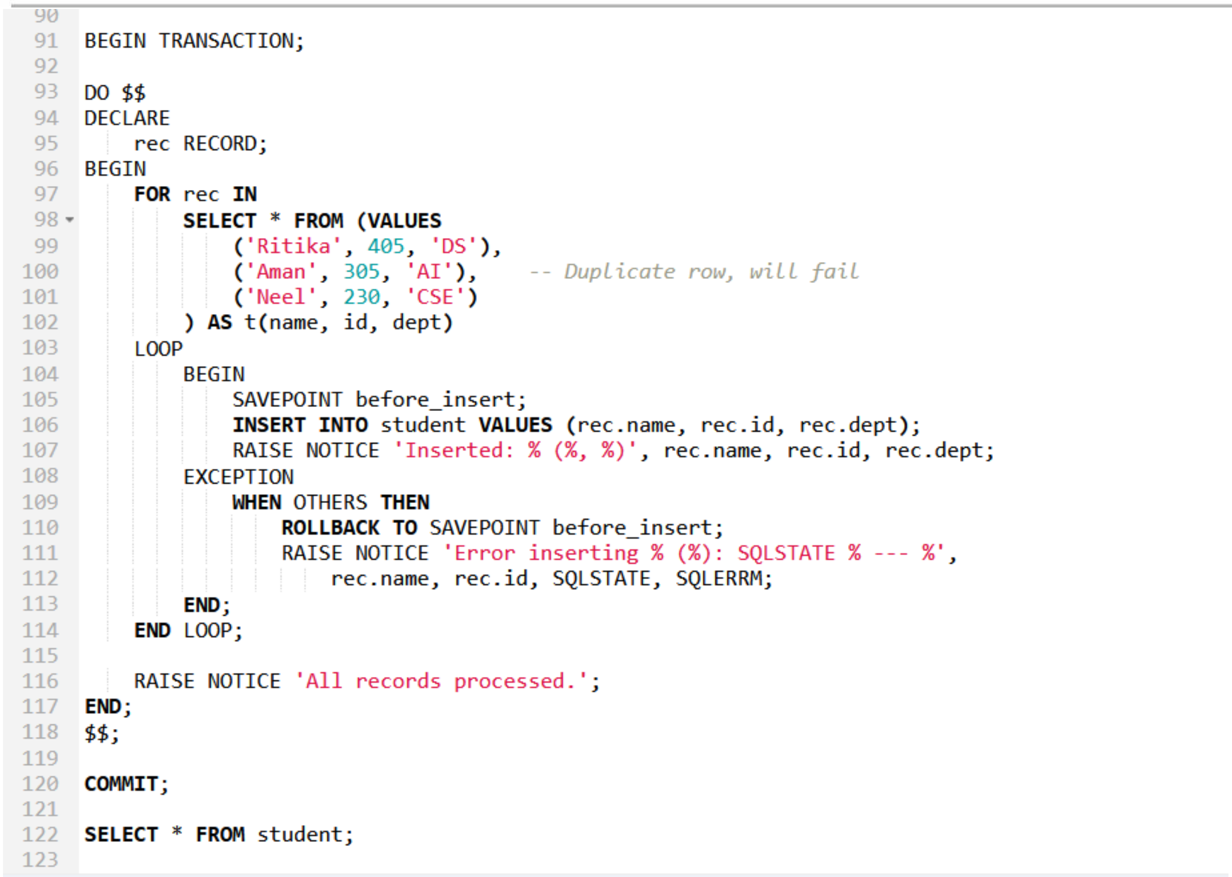
SQL Quries



```sql
------------------------------Experiment 08-------------------------------
--------------------Hard Level Problem----------------------------
/*
Design a robust PostgreSQL transaction system for the students table where multiple student
records are inserted in a single transaction.

If any insert fails due to invalid data, only that insert should be rolled back while preserving the
previous successful inserts using savepoints.

The system should provide clear messages for both successful and failed insertions, ensuring data integrity
and controlled error handling.

HINT: YOU HAVE TO USE SAVEPOINTS
*/

-- Create the student table first
CREATE TABLE student (
    name VARCHAR(100),
    id INT PRIMARY KEY,
    dept VARCHAR(50)
);


SELECT * FROM student;

SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE table_name = 'student';


SELECT
    trigger_name,
    event_manipulation AS event,
    action_timing AS timing,
    action_statement AS trigger_function
FROM information_schema.triggers
WHERE event_object_table = 'student';

SELECT * FROM INFORMATION_SCHEMA.TRIGGERS WHERE event_object_table = 'student';

DROP TRIGGER IF EXISTS trg_student ON student;


SELECT * FROM student;
```

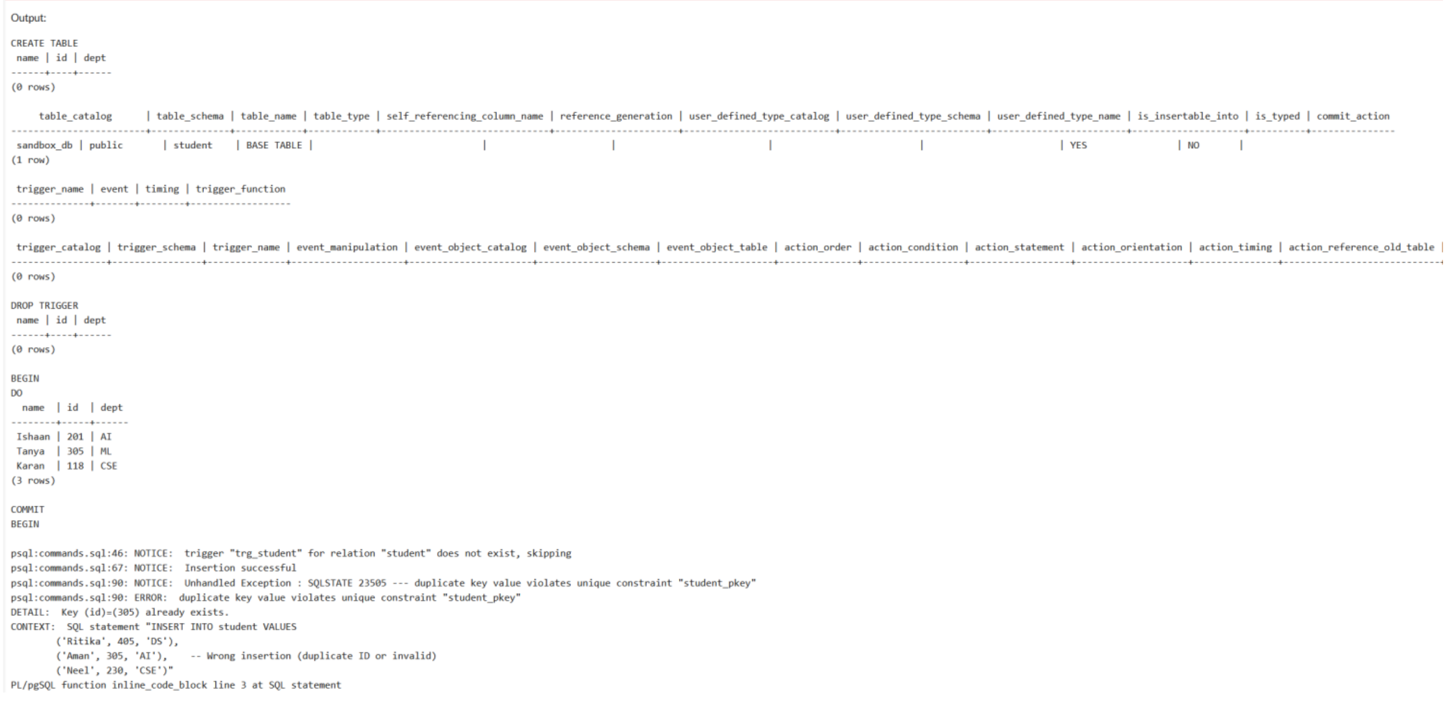


```sql

BEGIN TRANSACTION;
DO $$
BEGIN
    INSERT INTO student VALUES
        ('Ishaan', 201, 'AI'),
        ('Tanya', 305, 'ML'),
        ('Karan', 118, 'CSE');

    RAISE NOTICE 'Insertion successful';

EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Unhandled Exception : SQLSTATE % --- %', SQLSTATE, SQLERRM;
        RAISE;
END;
$$;

SELECT * FROM student;

COMMIT;



BEGIN TRANSACTION;
DO $$
BEGIN
    INSERT INTO student VALUES
        ('Ritika', 405, 'DS'),
        ('Aman', 305, 'AI'),     -- Wrong insertion (duplicate ID or invalid)
        ('Neel', 230, 'CSE');

    RAISE NOTICE 'Insertion successful';

EXCEPTION
    WHEN OTHERS THEN
        RAISE NOTICE 'Unhandled Exception : SQLSTATE % --- %', SQLSTATE, SQLERRM;
        RAISE;
END;
$$;

ROLLBACK;

SELECT * FROM student;

COMMIT;
```



```
BEGIN TRANSACTION;

DO $$
DECLARE
    rec RECORD;
BEGIN
    FOR rec IN
        SELECT * FROM (VALUES
            ('Ritika', 405, 'DS'),
            ('Aman', 305, 'AI'),      -- Duplicate row, will fail
            ('Neel', 230, 'CSE')
        ) AS t(name, id, dept)
    LOOP
        BEGIN
            SAVEPOINT before_insert;
            INSERT INTO student VALUES (rec.name, rec.id, rec.dept);
            RAISE NOTICE 'Inserted: % (%, %)', rec.name, rec.id, rec.dept;
        EXCEPTION
            WHEN OTHERS THEN
                ROLLBACK TO SAVEPOINT before_insert;
                RAISE NOTICE 'Error inserting % (%): SQLSTATE % --- %',
                    rec.name, rec.id, SQLSTATE, SQLERRM;
        END;
    END LOOP;

    RAISE NOTICE 'All records processed.';
END;
$$;

COMMIT;

SELECT * FROM student;
```

Output:



```
Output:

CREATE TABLE
 name | id | dept
------+----+------
(0 rows)

    table_catalog      | table_schema | table_name | table_type | self_referencing_column_name | reference_generation | user_defined_type_catalog | user_defined_type_schema | user_defined_type_name | is_insertable_into | is_typed | commit_action
-----------------------+--------------+------------+------------+------------------------------+----------------------+---------------------------+--------------------------+------------------------+--------------------+----------+---------------
 sandbox_db | public       | student    | BASE TABLE |                              |                      |                           |                          |                        | YES                | NO       |
(1 row)

 trigger_name | event | timing | trigger_function
--------------+-------+--------+------------------
(0 rows)

 trigger_catalog | trigger_schema | trigger_name | event_manipulation | event_object_catalog | event_object_schema | event_object_table | action_order | action_condition | action_statement | action_orientation | action_timing | action_reference_old_table |
-----------------+----------------+--------------+--------------------+----------------------+---------------------+--------------------+--------------+------------------+------------------+--------------------+---------------+----------------------------+
(0 rows)

DROP TRIGGER
 name | id | dept
------+----+------
(0 rows)

BEGIN
DO
  name  | id  | dept
--------+-----+------
 Ishaan | 201 | AI
 Tanya  | 305 | ML
 Karan  | 118 | CSE
(3 rows)

COMMIT
BEGIN

psql:commands.sql:46: NOTICE:  trigger "trg_student" for relation "student" does not exist, skipping
psql:commands.sql:67: NOTICE:  Insertion successful
psql:commands.sql:90: NOTICE:  Unhandled Exception : SQLSTATE 23505 --- duplicate key value violates unique constraint "student_pkey"
psql:commands.sql:90: ERROR:  duplicate key value violates unique constraint "student_pkey"
DETAIL:  Key (id)=(305) already exists.
CONTEXT:  SQL statement "INSERT INTO student VALUES
        ('Ritika', 405, 'DS'),
        ('Aman', 305, 'AI'),    -- Wrong insertion (duplicate ID or invalid)
        ('Neel', 230, 'CSE')"
PL/pgSQL function inline_code_block line 3 at SQL statement
```

Learning Outcome:

- Learned how to **create and manage triggers** in PostgreSQL to automate database operations.
- Understood how to use **RAISE NOTICE** for displaying trigger-based actions and debugging.
- Gained practical knowledge of handling **constraint violations** such as duplicate primary keys.
- Learned how to **verify and drop existing triggers** before creating new ones to avoid conflicts.
- Understood how triggers help in **maintaining data consistency and enforcing business rules** automatically.