Experiment - 8

Student Name: Armaan Atri
UID: 23BAI70302
Branch: BE-AIT-CSE
Section/Group: 23AIT_KRG-G1_A
Semester: 5th
Date of Performance: 15 Oct, 2025
Subject Name: ADBMS
Subject Code: 23CSP-333

1. Aim:

HARD LEVEL PROBLEM:
To implement a robust PostgreSQL transaction system using savepoints that allows multiple student records to be inserted within a single transaction, ensuring that any failed insert operation is rolled back individually while preserving successful inserts. The system should use proper exception handling to display messages for successful and failed insertions, maintaining data integrity and consistency.

2. Objective:

- To implement a robust transaction system in PostgreSQL using savepoints for partial rollbacks.
- To allow multiple insert operations to execute within a single transaction safely.
- To ensure that if any insert fails due to invalid or duplicate data, only that specific operation is rolled back.
- To retain all successful inserts even if one or more fail within the same transaction.
- To maintain data integrity and consistency throughout the transaction process.
- To use exception handling to display clear success and failure messages for each operation.
- To demonstrate effective control of commit, rollback, and savepoint commands in practical database scenarios.

3. Theory:

PostgreSQL transactions are ACID-compliant units of work; savepoints let you undo only the failing part while keeping earlier successful statements, supporting safe multi-insert flows with controlled error handling.

Essentials

- ACID guarantees atomicity, consistency, isolation, and durability, ensuring reliable changes even with errors or crashes.
- With autocommit, each statement is its own transaction unless wrapped in an explicit BEGIN/COMMIT block.

Save points:

- SAVEPOINT marks a point inside a transaction; ROLLBACK TO SAVEPOINT reverts only the work done after that point, and RELEASE SAVEPOINT removes the marker without committing anything.
- This enables partial rollback for per-row error handling in batch inserts, preserving prior successful rows.

Exceptions:

- PL/pgSQL EXCEPTION blocks use subtransactions internally, allowing localized error recovery for a block of statements, but COMMIT is not allowed inside the exception block.

4. Procedure:

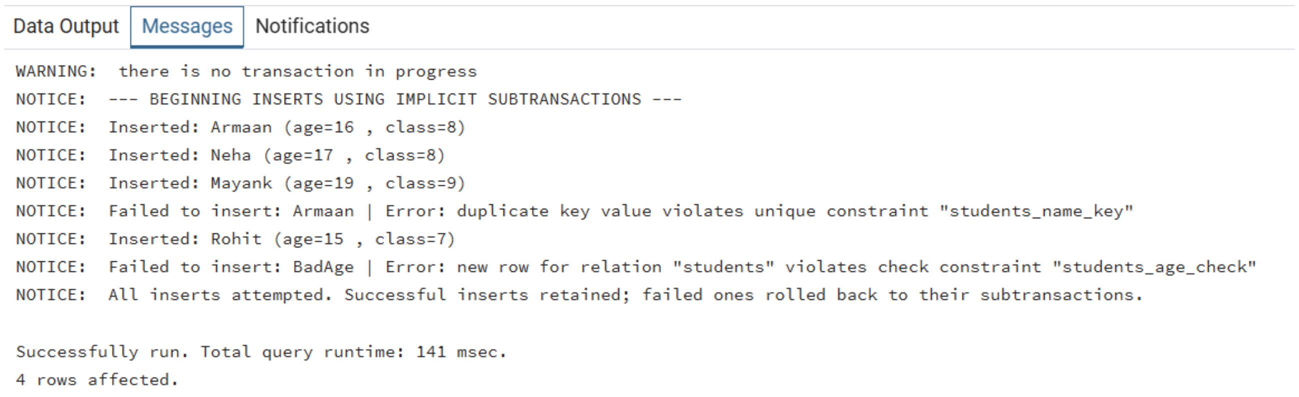
Hard Level Solution:

- Prepare a clean students table with a unique constraint on the name column to intentionally trigger duplicate-data errors during testing.
- Start an explicit transaction to group multiple inserts into one unit of work.
- Before each individual insert, set a savepoint to create a local rollback point.
- Attempt the insert; if it succeeds, proceed to the next step; if it fails, roll back to the most recent savepoint to undo only that failing insert.
- Continue with the remaining inserts, repeating the savepoint-then-insert pattern so prior successful inserts remain intact despite later failures.
- Optionally release savepoints after successful steps to keep the transaction stack tidy.
- After processing all inserts, commit the outer transaction so that all successful rows become permanent.
- Verify the final table contents to confirm that only valid, non-duplicate rows were preserved.
- Avoid committing inside exception-handling scopes; commit only at the outer transaction level.
- If using server-side blocks for looping, wrap each insert attempt in a try/catch style section that rolls back to a per-row savepoint on error and logs a clear message, then continues with the next row.

5. Code:

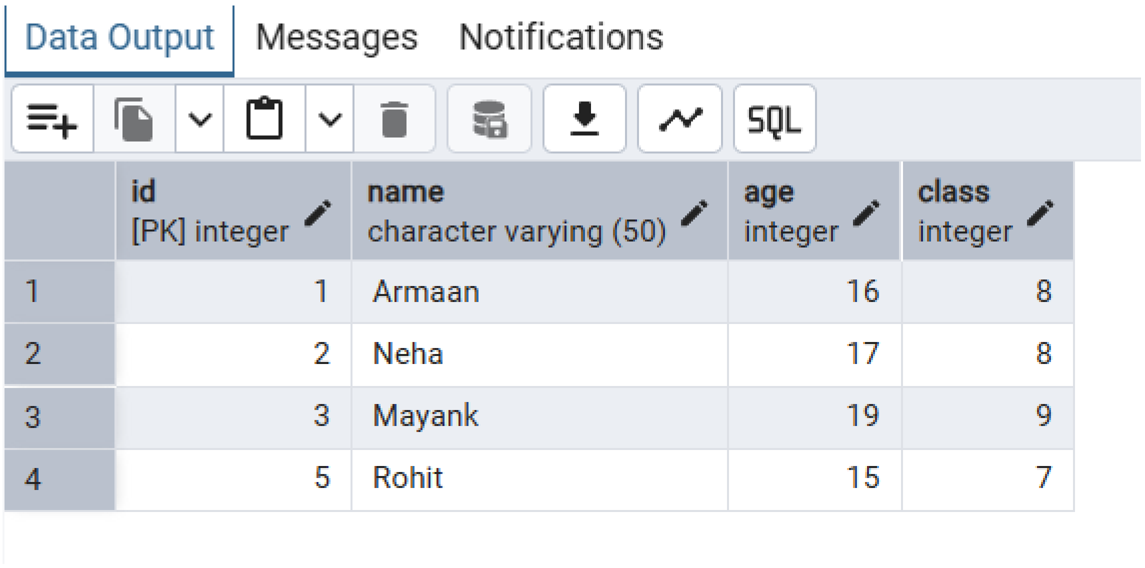
```
                    -----Experiment 8 (Hard Level Solution)-----
ROLLBACK;

DROP TABLE IF EXISTS students;

CREATE TABLE students (
    id SERIAL PRIMARY KEY,
    name VARCHAR(50) UNIQUE,
    age INT CHECK (age > 0),
    class INT CHECK (class BETWEEN 1 AND 12)
);

SELECT 'DONE CREATING TABLE' AS status;

DO $$
DECLARE
    student_row RECORD;
BEGIN
    RAISE NOTICE '--- BEGINNING INSERTS USING IMPLICIT SUBTRANSACTIONS ---';

    FOR student_row IN
        SELECT * FROM (
            VALUES
                ('Armaan', 16, 8),
                ('Neha',   17, 8),
                ('Mayank', 19, 9),
                ('Armaan', 18,10),  -- duplicate name -> will fail
                ('Rohit',  15, 7),
                ('BadAge', -1,  5)  -- invalid age -> will fail due to CHECK
        ) AS t(name, age, class)
    LOOP
        -- The inner BEGIN/EXCEPTION forms an implicit subtransaction.
        -- If INSERT fails, only this inner block is rolled back.
        BEGIN
            INSERT INTO students(name, age, class)
            VALUES (student_row.name, student_row.age, student_row.class);

            RAISE NOTICE 'Inserted: % (age=% , class=%)', student_row.name,
student_row.age, student_row.class;

        EXCEPTION WHEN OTHERS THEN
            -- The error caused the inner subtransaction to be rolled back
automatically.
            -- Log the error (do NOT re-raise) so outer transaction continues.
            RAISE NOTICE 'Failed to insert: % | Error: %', student_row.name,
SQLERRM;
        END;
    END LOOP;

    RAISE NOTICE 'All inserts attempted. Successful inserts retained; failed
ones rolled back to their subtransactions.';
END;
$$;

-- 3) Verify results
SELECT * FROM students ORDER BY id;
```

6. Output:



Data Output Messages Notifications

```
WARNING:  there is no transaction in progress
NOTICE:  --- BEGINNING INSERTS USING IMPLICIT SUBTRANSACTIONS ---
NOTICE:  Inserted: Armaan (age=16 , class=8)
NOTICE:  Inserted: Neha (age=17 , class=8)
NOTICE:  Inserted: Mayank (age=19 , class=9)
NOTICE:  Failed to insert: Armaan | Error: duplicate key value violates unique constraint "students_name_key"
NOTICE:  Inserted: Rohit (age=15 , class=7)
NOTICE:  Failed to insert: BadAge | Error: new row for relation "students" violates check constraint "students_age_check"
NOTICE:  All inserts attempted. Successful inserts retained; failed ones rolled back to their subtransactions.

Successfully run. Total query runtime: 141 msec.
4 rows affected.
```



Data Output Messages Notifications

	id [PK] integer	name character varying (50)	age integer	class integer
1	1	Armaan	16	8
2	2	Neha	17	8
3	3	Mayank	19	9
4	5	Rohit	15	7

7. Learning Outcomes:

- Understood the concept of transactions and ACID properties in PostgreSQL.
- Learned how to handle errors using `BEGIN ... EXCEPTION ... END;` blocks.
- Understood the use of `SAVEPOINT` for partial rollbacks within a transaction.
- Learned that explicit `COMMIT` and `ROLLBACK` are not allowed inside PL/pgSQL `DO` blocks.
- Gained knowledge of implicit subtransactions for handling individual insert failures.
- Ensured data integrity by rolling back only failed operations while preserving successful ones.
- Learned to use `RAISE NOTICE` for displaying custom success and error messages.
- Practiced creating robust transaction logic for bulk inserts with controlled error handling.
- Gained practical understanding of how constraints (e.g., `UNIQUE`, `CHECK`) trigger exceptions.
- Developed skills to design reliable, fault-tolerant database systems.