Laboratory Work №10

SQL Transactions and Isolation Levels

1. Objective

To study the concepts of database transactions, understand ACID properties, learn how to use COMMIT, ROLLBACK, and SAVEPOINT statements, and explore different transaction isolation levels in SQL.

2. Theoretical Background

2.1 What is a Transaction?

A transaction is a sequence of one or more SQL operations that are executed as a single logical unit of work. Database systems are normally accessed by many users or processes simultaneously, and transactions help manage concurrent access while maintaining data integrity.

2.2 ACID Properties

Property	Description
Atomic	Either the whole process is done or none is. All operations succeed or all fail.
Consistent	Database constraints are preserved. The database moves from one valid state to another.
Isolated	It appears to the user as if only one process executes at a time.
Durable	Effects of a process do not get lost if the system crashes.

2.3 Transaction Control Statements

BEGIN - Starts a new transaction:

```
BEGIN;
-- or
BEGIN TRANSACTION;
```

COMMIT - Makes all changes permanent:

```
COMMIT;
```

ROLLBACK - Undoes all changes since the transaction began:

```
ROLLBACK;
```

SAVEPOINT - Creates a point within a transaction to which you can roll back:

```
SAVEPOINT savepoint_name;
```

```
ROLLBACK TO savepoint_name;
RELEASE SAVEPOINT savepoint_name;
```

2.4 Isolation Levels

Level	Description	Phenomena Allowed
SERIALIZABLE	Highest isolation. Transactions appear to execute serially.	None
REPEATABLE READ	Data read is guaranteed to be the same if read again.	Phantom reads
READ COMMITTED	Only sees committed data, but may see different data on re-read.	Non-repeatable reads, Phantoms
READ UNCOMMITTED	Can see uncommitted changes from other transactions.	Dirty reads, Non-repeatable, Phantoms

Setting isolation level:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
-- or within BEGIN:
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

3. Practical Tasks

3.1 Setup: Create Test Database

Create the following tables for the exercises:

```
CREATE TABLE accounts (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100) NOT NULL,
    balance DECIMAL(10, 2) DEFAULT 0.00
);

CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    shop VARCHAR(100) NOT NULL,
    product VARCHAR(100) NOT NULL,
    price DECIMAL(10, 2) NOT NULL
);

-- Insert test data
INSERT INTO accounts (name, balance) VALUES
    ('Alice', 1000.00),
    ('Bob', 500.00),
    ('Wally', 750.00);

INSERT INTO products (shop, product, price) VALUES
```

```
    ('Joe''s Shop', 'Coke', 2.50),
    ('Joe''s Shop', 'Pepsi', 3.00);
```

3.2 Task 1: Basic Transaction with COMMIT

Objective: Transfer money between accounts using a transaction.

Execute the following transaction:

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
COMMIT;
```

Questions:

- a) What are the balances of Alice and Bob after the transaction?
- b) Why is it important to group these two UPDATE statements in a single transaction?
- c) What would happen if the system crashed between the two UPDATE statements without a transaction?

3.3 Task 2: Using ROLLBACK

Objective: Understand how ROLLBACK undoes changes.

Execute these commands:

```
BEGIN;
UPDATE accounts SET balance = balance - 500.00
    WHERE name = 'Alice';
SELECT * FROM accounts WHERE name = 'Alice';
-- Oops! Wrong amount, let's undo
ROLLBACK;
SELECT * FROM accounts WHERE name = 'Alice';
```

Questions:

- a) What was Alice's balance after the UPDATE but before ROLLBACK?
- b) What is Alice's balance after ROLLBACK?
- c) In what situations would you use ROLLBACK in a real application?

3.4 Task 3: Working with SAVEPOINTs

Objective: Learn to use savepoints for partial rollbacks.

```
BEGIN;
UPDATE accounts SET balance = balance - 100.00
    WHERE name = 'Alice';
SAVEPOINT my_savepoint;
```

```
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Bob';
-- Oops, should transfer to Wally instead
ROLLBACK TO my_savepoint;
UPDATE accounts SET balance = balance + 100.00
    WHERE name = 'Wally';
COMMIT;
```

Questions:

- a) After COMMIT, what are the balances of Alice, Bob, and Wally?
- b) Was Bob's account ever credited? Why or why not in the final state?
- c) What is the advantage of using SAVEPOINT over starting a new transaction?

3.5 Task 4: Isolation Level Demonstration

Objective: Observe how different isolation levels affect concurrent transactions.

This task requires two separate database sessions (Terminal 1 and Terminal 2).

Scenario A: READ COMMITTED

Terminal 1:

```
BEGIN TRANSACTION ISOLATION LEVEL READ COMMITTED;
SELECT * FROM products WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2 to make changes and COMMIT
-- Then re-run:
SELECT * FROM products WHERE shop = 'Joe''s Shop';
COMMIT;
```

Terminal 2 (while Terminal 1 is still running):

```
BEGIN;
DELETE FROM products WHERE shop = 'Joe''s Shop';
INSERT INTO products (shop, product, price)
    VALUES ('Joe''s Shop', 'Fanta', 3.50);
COMMIT;
```

Scenario B: SERIALIZABLE

Repeat the above scenario but use:

```
BEGIN TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Questions:

- a) In Scenario A, what data does Terminal 1 see before and after Terminal 2 commits?
- b) In Scenario B, what data does Terminal 1 see?
- c) Explain the difference in behavior between READ COMMITTED and SERIALIZABLE.

3.6 Task 5: Phantom Read Demonstration

Objective: Understand phantom reads with REPEATABLE READ isolation level.

Terminal 1:

```
BEGIN TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT MAX(price), MIN(price) FROM products
    WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2
SELECT MAX(price), MIN(price) FROM products
    WHERE shop = 'Joe''s Shop';
COMMIT;
```

Terminal 2:

```
BEGIN;
INSERT INTO products (shop, product, price)
    VALUES ('Joe''s Shop', 'Sprite', 4.00);
COMMIT;
```

Questions:

- a) Does Terminal 1 see the new product inserted by Terminal 2?
- b) What is a phantom read?
- c) Which isolation level prevents phantom reads?

3.7 Task 6: Dirty Read Demonstration

Objective: Understand dirty reads with READ UNCOMMITTED isolation level.

Terminal 1:

```
BEGIN TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT * FROM products WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2 to UPDATE but NOT commit
SELECT * FROM products WHERE shop = 'Joe''s Shop';
-- Wait for Terminal 2 to ROLLBACK
SELECT * FROM products WHERE shop = 'Joe''s Shop';
COMMIT;
```

Terminal 2:

```
BEGIN;
UPDATE products SET price = 99.99
    WHERE product = 'Fanta';
-- Wait here (don't commit yet)
-- Then:
ROLLBACK;
```

Questions:

- a) Did Terminal 1 see the price of 99.99? Why is this problematic?
- b) What is a dirty read?
- c) Why should READ UNCOMMITTED be avoided in most applications?

4. Independent Exercises

Exercise 1

Write a transaction that transfers $200 from Bob to Wally, but only if Bob has sufficient funds. Use appropriate error handling.

Exercise 2

Create a transaction with multiple savepoints that:

- Inserts a new product
- Creates a savepoint
- Updates the price
- Creates another savepoint
- Deletes the product
- Rolls back to the first savepoint
- Commits

Document the final state of the products table.

Exercise 3

Design and implement a banking scenario where two users simultaneously try to withdraw money from the same account. Demonstrate how different isolation levels handle this situation.

Exercise 4

Given the Sells(shop, product, price) relation from the lecture, write queries to demonstrate the problem where Sally sees MAX < MIN when she and Joe don't use transactions properly. Then show how transactions fix this issue.

5. Questions for Self-Assessment

1. Explain each ACID property with a practical example.
2. What is the difference between COMMIT and ROLLBACK?
3. When would you use a SAVEPOINT instead of a full ROLLBACK?
4. Compare and contrast the four SQL isolation levels.
5. What is a dirty read and which isolation level allows it?
6. What is a non-repeatable read? Give an example scenario.
7. What is a phantom read? Which isolation levels prevent it?
8. Why might you choose READ COMMITTED over SERIALIZABLE in a high-traffic application?
9. Explain how transactions help maintain database consistency during concurrent access.
10. What happens to uncommitted changes if the database system crashes?

6. Lab Report Requirements

Your laboratory report should include:

- Screenshots of SQL commands and their outputs for all tasks
- Answers to all questions in each task
- Complete solutions to the independent exercises
- Written answers to the self-assessment questions
- A conclusion summarizing what you learned about transactions

7. References

1. PostgreSQL Documentation: Transaction Isolation
2. SQL Standard: ISO/IEC 9075 - Transaction Processing
3. Database Systems: The Complete Book by Garcia-Molina, Ullman, and Widom