Experiment 8: Concurrency Control Simulation

Prerequisites

Knowledge of transactions, ACID properties, and transaction control commands (COMMIT, ROLLBACK)

Lab Objective

To understand and simulate concurrency control problems in database systems and analyze how isolation levels prevent data inconsistency during simultaneous transaction execution.

Learning Outcomes

By completing this experiment, students will be able to:

- Understand concurrency issues in database systems.
- Simulate problems such as lost updates, dirty reads, non-repeatable reads, and phantom reads.
- Demonstrate different transaction isolation levels.
- Analyze how isolation levels control data consistency.
- Understand the importance of concurrency control mechanisms.

Introduction

In multi-user database systems, multiple transactions may execute simultaneously. Without proper concurrency control, this can lead to data inconsistencies and anomalies.

Concurrency control ensures that transactions execute safely without interfering with each other. Database systems implement isolation levels to manage how and when changes made by one transaction become visible to others.

This experiment simulates common concurrency problems using multiple sessions.

Key Concepts Covered

- **Concurrency:** Simultaneous execution of multiple transactions.
- **Lost Update Problem**
- **Dirty Read**
- **Non-Repeatable Read**
- **Phantom Read**
- **Isolation Levels:**
    - READ UNCOMMITTED
    - READ COMMITTED
    - REPEATABLE READ
    - SERIALIZABLE

Experiment Steps

⚠️ Open **two separate SQL sessions (Session A and Session B)** to simulate concurrent transactions.

---

1. Simulate Dirty Read

Step 1 – Session A

START TRANSACTION;
UPDATE Students SET age = 30 WHERE student_id = 1;

Do not commit.

Step 2 – Session B

```
SET SESSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;
SELECT age FROM Students WHERE student_id = 1;
```

Observation:

- Session B may read uncommitted data (dirty read).

Step 3 – Session A

```
ROLLBACK;
```

Discuss inconsistency observed.

---

2. Simulate Lost Update

Session A

```
START TRANSACTION;
SELECT age FROM Students WHERE student_id = 2;
```

Session B

```
START TRANSACTION;
UPDATE Students SET age = age + 1 WHERE student_id = 2;
COMMIT;
```

Session A

```
UPDATE Students SET age = age + 2 WHERE student_id = 2;
COMMIT;
```

Observe how one update may overwrite another.

---

3. Simulate Non-Repeatable Read

Session A

```
SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
```

START TRANSACTION;
SELECT age FROM Students WHERE student_id = 3;

Session B

UPDATE Students SET age = 35 WHERE student_id = 3;
COMMIT;

Session A

SELECT age FROM Students WHERE student_id = 3;

Observation:

- Value changes within same transaction.

---

4. Simulate Phantom Read

Session A

SET SESSION TRANSACTION ISOLATION LEVEL READ COMMITTED;
START TRANSACTION;
SELECT * FROM Students WHERE age > 20;

Session B

INSERT INTO Students VALUES (10, 'NewStudent', 22);
COMMIT;

Session A

SELECT * FROM Students WHERE age > 20;

Observation:

- New row appears (phantom row).

---

5. Test Higher Isolation Level

Repeat experiments using:

SET SESSION TRANSACTION ISOLATION LEVEL SERIALIZABLE;

Observe how anomalies are prevented.

How It Works

- Concurrent transactions can lead to inconsistencies.
- Isolation levels control visibility of changes between transactions.
- Higher isolation levels increase consistency but may reduce performance.
- Concurrency control mechanisms maintain database integrity in multi-user systems.

Submission Instructions

Students must submit:

- Screenshots of Session A and Session B outputs.
- Demonstration of at least two concurrency anomalies.
- Comparison of results under different isolation levels.
- Brief explanation of observed behavior.

Conclusion

This experiment demonstrates how concurrent transactions can cause data inconsistencies and how isolation levels prevent such anomalies. Understanding concurrency control is essential for designing reliable and multi-user database systems.