EXPERIMENT – 12

Name : Ankit Kumar

UID : 23BDO10013

Subject : ADBMS

Section & Group : 23BCC-BDO-2

QUESTION :
Demonstrate deadlocks, MVCC (Multiversion Concurrency Control), and transaction concurrency control in a student enrollment system.

Solution:

Aim:

To illustrate how concurrency control is managed in DBMS through deadlocks, MVCC, and the comparison of traditional locking with MVCC using a student enrollment database.

Theory:

Part A: Deadlocks in DBMS

A deadlock arises when two or more transactions wait indefinitely for each other's locked resources.

Example:

Transaction 1 locks row A and waits for row B.
Transaction 2 locks row B and waits for row A.

Modern DBMS engines (MySQL InnoDB, PostgreSQL) automatically detect deadlocks and roll back one transaction to break the cycle.

Deadlocks can be prevented by:

- Following consistent ordering of transactions.
- Using row-level locks carefully.

Part B: MVCC (Multiversion Concurrency Control)

MVCC enables readers and writers to work concurrently without blocking one another.

Readers always view a snapshot of the data as it existed at the start of their transaction.

Writers update by creating a new version of the record; old versions remain visible to active readers until they commit.

This mechanism allows high concurrency while preserving consistency.

Part C: Locking vs. MVCC

Traditional Locking: Readers may be blocked if a writer holds a lock (e.g., using SELECT FOR UPDATE).

MVCC: Readers continue to see a consistent snapshot while writers perform updates, avoiding blocks.

MVCC thus improves performance, scalability, and user experience in environments with heavy concurrency.

```
-- Part A: Simulating a Deadlock
DROP TABLE IF EXISTS StudentEnrollments;

CREATE TABLE StudentEnrollments (
 student_id INT PRIMARY KEY,
 student_name VARCHAR(100),
 course_id VARCHAR(10),
 enrollment_date DATE
);

INSERT INTO StudentEnrollments VALUES
(1, 'Ashish', 'CSE101', '2024-06-01'),
(2, 'Smaran', 'CSE102', '2024-06-01'),
(3, 'Vaibhav', 'CSE103', '2024-06-01');

-- Part B: Using MVCC for Non-Blocking Read/Write
-- Session 1 (User A) reads record
START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT * FROM StudentEnrollments WHERE student_id = 1;
-- Output: enrollment_date = 2024-06-01

-- Session 2 (User B) updates concurrently
START TRANSACTION;
UPDATE StudentEnrollments
SET enrollment_date = '2024-07-10'
WHERE student_id = 1;
COMMIT;

-- Session 1 still sees enrollment_date = 2024-06-01
COMMIT;
-- After commit, Session 1 sees updated value (2024-07-10)
```

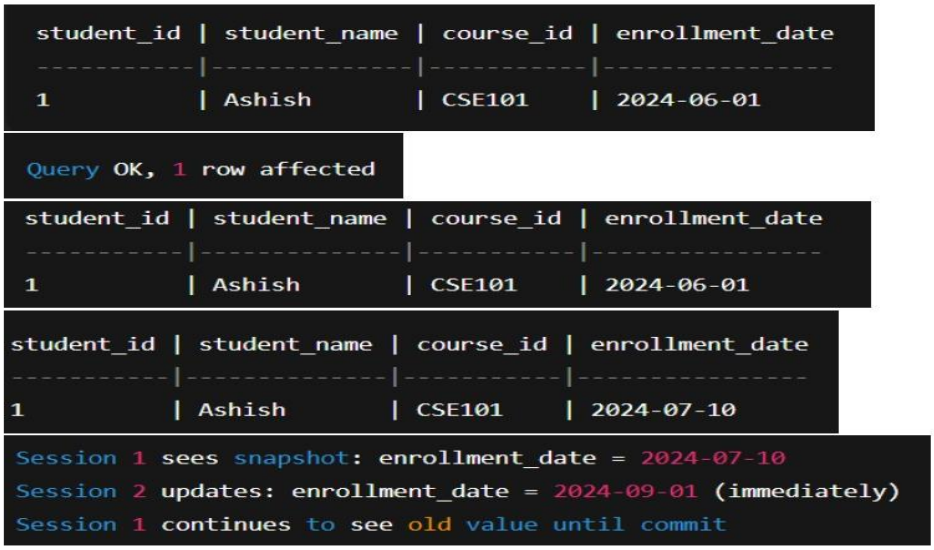
```
-- Part C: Comparing Locking vs. MVCC

-- Without MVCC
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_id = 1 FOR UPDATE;
-- Session 2 tries:
UPDATE StudentEnrollments SET enrollment_date = '2024-08-01' WHERE
student_id = 1;
-- Session 2 is blocked until Session 1 commits.

-- With MVCC
START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT * FROM StudentEnrollments WHERE student_id = 1;

-- Session 2 updates concurrently
UPDATE StudentEnrollments SET enrollment_date = '2024-09-01' WHERE
student_id = 1;
COMMIT;
-- Session 1 still sees old value until commit
COMMIT;
```

Outputs:



Learning Outcomes:

Gained understanding of deadlocks and how modern DBMS automatically resolve them.

Learned how MVCC allows concurrent read and write operations without blocking.

Compared the differences between traditional locking and MVCC.

Observed how concurrency control preserves consistency and improves performance.

Strengthened knowledge of ACID principles in high-concurrency database environments.