NAME- TUSHAR, UID-23BCC70034

ADBMS

EXP-12

- **AIM:** To demonstrate deadlocks, MVCC, and transaction concurrency control in a student enrollment system.
- **THEORY:**
- Part A: Deadlocks in DBMS A deadlock occurs when two or more transactions wait indefinitely for resources locked by each other.
- Example:
- Transaction 1 locks row A and waits for row B.
- Transaction 2 locks row B and waits for row A.
- Most modern DBMS (MySQL InnoDB, PostgreSQL) detect deadlocks automatically and roll back one transaction to resolve it.
- Deadlocks can be avoided by consistent transaction ordering or using row-level locks carefully.
- Part B: MVCC (Multiversion Concurrency Control) MVCC allows readers and writers to work concurrently without blocking each other.
- Readers see a snapshot of data at the start of the transaction, unaffected by concurrent writes.
- Writers create a new version of the data; old versions remain visible to readers until their transactions commit.
- Part C: Comparing Locking vs MVCC Traditional Locking: Readers may block if a writer holds a lock (e.g., SELECT FOR UPDATE).
- MVCC: Readers see consistent snapshots; writers update without blocking readers.

- MVCC improves concurrency, performance, and user experience in high-concurrency environments.

➢ **<u>CODES:</u>**

- Part A: Simulating a deadlock

```
-- Drop table if exists

DROP TABLE IF EXISTS StudentEnrollments;


-- Create table

CREATE TABLE StudentEnrollments (

    student_id INT PRIMARY KEY,

    student_name VARCHAR(100),

    course_id VARCHAR(10),

    enrollment_date DATE

);


-- Insert sample data

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 R/W

```sql
-- Session 1 (User A) reads the record
START TRANSACTION ISOLATION LEVEL
REPEATABLE READ;

SELECT * FROM StudentEnrollments WHERE student_id
= 1;
-- Output: enrollment_date = 2024-06-01
-- This snapshot is maintained even if other transactions
update

-- Session 2 (User B) updates the same record 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
-- Until User A commits or restarts the transaction
COMMIT;

-- Session 1 sees the updated value after commit
```

SELECT * FROM StudentEnrollments WHERE student_id = 1;

-- Output: enrollment_date = 2024-07-10

- Part C: Comparing Locking vs MVC
- Without MVCC

-- Session 1

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

-- Session 1

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 (2024-07-10) until commit

COMMIT;

OUTPUTS:

```
student_id | student_name | course_id | enrollment_date
-----------|--------------|-----------|----------------
1          | Ashish       | CSE101    | 2024-06-01
```

```
Query OK, 1 row affected
```

```
student_id | student_name | course_id | enrollment_date
-----------|--------------|-----------|----------------
1          | Ashish       | CSE101    | 2024-06-01
```

```
student_id | student_name | course_id | enrollment_date
-----------|--------------|-----------|----------------
1          | Ashish       | CSE101    | 2024-07-10
```

```
Session 1 sees snapshot: enrollment_date = 2024-07-10
Session 2 updates: enrollment_date = 2024-09-01 (immediately)
Session 1 continues to see old value until commit
```

➢ LEARNING OUTCOMES:

1. Learned to enforce unique constraints to prevent duplicate student enrollments.
2. Understood row-level locking using SELECT FOR UPDATE to handle concurrent transactions.
3. Observed how transactions preserve Atomicity and Consistency in a multi-user environment.
4. Practiced handling blocked transactions and understanding isolation effects.

5. Gained hands-on experience with ACID principles in a practical enrollment scenario.