Practice 3 – Transactions & Concurrency Control

Part A: Simulating a Deadlock Between Two Transactions

Setup SQL:

```
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');
```

Steps to Reproduce Deadlock:

```
-- Session 1
START TRANSACTION;
UPDATE StudentEnrollments SET course_id = 'CSE201' WHERE student_id = 1;
UPDATE StudentEnrollments SET course_id = 'CSE202' WHERE student_id = 2;

-- Session 2
START TRANSACTION;
UPDATE StudentEnrollments SET course_id = 'CSE301' WHERE student_id = 2;
UPDATE StudentEnrollments SET course_id = 'CSE302' WHERE student_id = 1;
```

Expected Output:

One transaction will fail with error:
ERROR 1213 (40001): Deadlock found when trying to get lock; try restarting transaction.

Explanation:

Both transactions try to lock each other's rows in reverse order, creating a deadlock. The database detects the deadlock and rolls back one transaction automatically.

Part B: Applying MVCC to Prevent Conflicts

Steps:

```
-- Session 1 (User A)
START TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT enrollment_date FROM StudentEnrollments WHERE student_id = 1;
-- Sees: 2024-06-01

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

-- Session 1 (User A continues)
SELECT enrollment_date FROM StudentEnrollments WHERE student_id = 1;
-- Still sees: 2024-06-01

COMMIT;

-- After commit, new transaction sees updated value 2024-07-10
```

Explanation:

MVCC ensures that User A reads a consistent snapshot of data from the start of the transaction. User B can update concurrently without blocking User A.

Part C: Comparing Behavior With and Without MVCC

Without MVCC (using SELECT FOR UPDATE):

```
-- Session 1
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_id=1 FOR UPDATE;

-- Session 2
SELECT * FROM StudentEnrollments WHERE student_id=1;
-- Blocks until Session 1 commits
```

With MVCC (normal SELECT):

```
-- Session 1
START TRANSACTION;
UPDATE StudentEnrollments SET enrollment_date='2024-08-01' WHERE student_id=1;

-- Session 2
SELECT * FROM StudentEnrollments WHERE student_id=1;
-- Sees old value, no blocking
```

Explanation:

In locking systems, readers block until writers commit. With MVCC, readers see a consistent snapshot while writers can still update, avoiding blocking.