NAME	RAMENDRA SHUKLA
UID	23BCS12146
CLASS	622-B

PRACTICE-4.3

Part A: Simulating a Deadlock Between Two Transactions

Description:

Given a table StudentEnrollments containing student records, simulate a situation where two concurrent transactions (from different users) try to update overlapping records in different orders, resulting in a deadlock. Demonstrate how such deadlocks are detected and how they can be avoided using proper transaction ordering.

Code

```
START TRANSACTION;

UPDATE StudentEnrollments
SET enrollment_date = '2024-07-01'
WHERE student_id = 1;
```

```
UPDATE StudentEnrollments
SET enrollment_date = '2024-07-02'
WHERE student_id = 2;

COMMIT;

START TRANSACTION;

UPDATE StudentEnrollments
SET enrollment_date = '2024-07-03'
WHERE student_id = 2;

UPDATE StudentEnrollments
SET enrollment_date = '2024-07-04'
WHERE student_id = 1;

COMMIT;
```

Input:

StudentEnrollments

student_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-06-01
2	Smaran	CSE102	2024-06-01
3	Vaibhav	CSE103	2024-06-01

Output:

Transaction 2 is aborted due to a detected deadlock.

Part B: Applying MVCC to Prevent Conflicts During Concurrent Reads/Writes

Description:

Use the **MVCC (Multiversion Concurrency Control)** approach to allow **User A to read** a record and **User B to update** the same record concurrently without blocking or conflict. Demonstrate how MVCC provides a consistent snapshot to the reader while allowing the writer to update.

Code

```
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

SELECT enrollment_date
FROM StudentEnrollments
WHERE student_id = 1;

START TRANSACTION;

UPDATE StudentEnrollments
SET enrollment_date = '2024-07-10'
WHERE student_id = 1;

COMMIT;
```

SELECT enrollment_date

FROM StudentEnrollments

WHERE student_id = 1;

COMMIT;

SELECT enrollment_date

FROM StudentEnrollments

WHERE student_id = 1;

Input:

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

Output:

- User A sees:

enrollment_date = 2024-06-01

- User B updates to:

2024-07-10

- User A continues to see the old value in the transaction until commit.

Part C: Comparing Behavior With and Without MVCC in High-Concurrency

Description:

Evaluate how **MVCC vs. traditional locking** behaves when multiple users access the same row for read and write. Use

SELECT FOR UPDATE

to demonstrate blocking in a non-MVCC system and contrast that with MVCC-based reads and updates.

Code

```
-- Scenario 1: With Locking
START TRANSACTION;

SELECT enrollment_date
FROM StudentEnrollments
WHERE student_id = 1
FOR UPDATE;

UPDATE StudentEnrollments
SET enrollment_date = '2024-07-10'
WHERE student_id = 1;

COMMIT;

START TRANSACTION;

SELECT enrollment_date
FROM StudentEnrollments
WHERE student_id = 1
FOR UPDATE;

COMMIT;
```

```
-- Scenario 2: With MVCC
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
START TRANSACTION;

SELECT enrollment_date
FROM StudentEnrollments
WHERE student_id = 1;

START TRANSACTION;

UPDATE StudentEnrollments
SET enrollment_date = '2024-07-10'
WHERE student_id = 1;

COMMIT;

SELECT enrollment_date
FROM StudentEnrollments
WHERE student_id = 1;

COMMIT;
```

Input:

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

Output:

- **Without MVCC**: Reader blocks until writer commits.
- **With MVCC**: Reader sees 2024-06-01 even while the writer updates to 2024-07-10.