NAME	Chirag
UID	23BCS14318
CLASS	622-B

❖ ADBMS PRACTISE 4.3

- CODE (Part-A):- **Simulating a Deadlock Between Two Transactions**

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

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

START TRANSACTION;

UPDATE StudentEnrollment
```

```
SET enrollment_date = DATE '2024-09-01'
WHERE student_id = 1;

UPDATE StudentEnrollment
SET enrollment_date = DATE '2024-09-02'
WHERE student_id = 2;

START TRANSACTION;

UPDATE StudentEnrollment
SET enrollment_date = DATE '2024-09-03'
WHERE student_id = 2;

UPDATE StudentEnrollment
SET enrollment_date = DATE '2024-09-04'
WHERE student_id = 1;
```

- OUTPUT
  Both transactions try to lock each other's rows in reverse order. This causes a deadlock, and the database automatically rolls back one transaction (usually the one that waited longest) to break the cycle.

- CODE (Part-B):- **Applying MVCC to Prevent Conflicts During Concurrent Reads/Writes**

```
INSERT INTO StudentEnrollment
VALUES (1, 'Ashish', 'CSE101', '2024-06-01');
```

```
COMMIT;

START TRANSACTION;

SELECT student_id, student_name, course_id, enrollment_date
FROM StudentEnrollment
WHERE student_id = 1;

START TRANSACTION;

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

COMMIT;

SELECT student_id, student_name, course_id, enrollment_date
FROM StudentEnrollment
WHERE student_id = 1;

COMMIT;

SELECT student_id, student_name, course_id, enrollment_date
FROM StudentEnrollment
WHERE student_id = 1;
```

- OUTPUT

student_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-07-10T00:00:00.000Z
2	Smaran	CSE102	2024-09-02T00:00:00.000Z
3	Vaibhav	CSE103	2024-06-01T00:00:00.000Z

- CODE (Part-C):- **Comparing Behavior With and Without MVCC in High-Concurrency**

- **Using Traditional Locking::**

```
--Session 1(Writer):
START TRANSACTION;

UPDATE StudentEnrollment
SET enrollment_date = '2024-07-11'
WHERE student_id = 1;

--Session 2(Reader):
START TRANSACTION;

SELECT enrollment_date
FROM StudentEnrollment
WHERE student_id = 1
FOR UPDATE;
```

  - Ouput:
    - Session 2 blocks until Session 1 commits.

- Reader cannot access the row because SELECT FOR UPDATE acquires a lock.

- **Using MVCC::**

```
--Session 1(Writer):
START TRANSACTION;

UPDATE StudentEnrollment
SET enrollment_date = '2024-07-11'
WHERE student_id = 1;

COMMIT;

--Session 2(Reader):
START TRANSACTION;

SELECT enrollment_date
FROM StudentEnrollment
WHERE student_id = 1;
```

- Ouput:
  - Reader does not block.
  - Sees the old value if its transaction started before the writer committed.

❖ **Traditional Locking**: The reader is blocked until the writer commits because SELECT … FOR UPDATE acquires an exclusive lock. The writer holds the lock on the row until it commits.

- **MVCC (Multiversion Concurrency Control):** The reader sees a consistent snapshot of the data without being blocked, even while the writer updates the row concurrently. The writer can commit the update without affecting the reader's transaction.