Experiment 12

Subject: ADBMS

Subject Code:23CSP-333

Date: 26th September 2025

Name: Aditya Sivam Kashyap

UID : 23BCC70036

Section: 23BCC-1

1. Aim:

Transactions and Concurrency Control

1. **Part A:** To simulate a **deadlock** between two concurrent transactions that lock resources in a conflicting order, and to demonstrate how the database automatically detects and resolves the issue.
2. **Part B:** To demonstrate how **Multiversion Concurrency Control (MVCC)** allows a transaction to read a consistent snapshot of data without being blocked by another transaction that is concurrently writing to the same data.
3. **Part C:** To directly compare the behavior of traditional **explicit locking** (which causes blocking) with the non-blocking nature of **MVCC** reads to highlight the performance advantages in a high-concurrency environment.

2. Theory:

Databases use sophisticated techniques to manage simultaneous user actions. A **deadlock** is a critical issue where two or more transactions are stuck in a circular wait, each holding a lock that the other needs. Because neither can proceed, modern databases have a **deadlock detector** that automatically identifies these cycles and resolves them by terminating one transaction (the "victim"), allowing the other to continue.

Mult version Concurrency Control (MVCC) is a more advanced technique to increase concurrency. Instead of using locks for reading, MVCC maintains multiple "versions" of a row. When a transaction begins, it is given a consistent "snapshot" of the database at that point in time. All reads within that transaction see only that version, completely isolated from changes being made by other concurrent transactions. This powerful principle means **readers don't block writers, and writers don't block readers**, significantly improving performance.

3. SQL Queries:

1. Part A: Simulating a Deadlock Between Two Transactions
   - Create the table and insert sample data.

```
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');
INSERT INTO StudentEnrollments VALUES (2, 'Smaran',
'CSE102', '2024-06-01');
```

   - **User A's Session (Step 1):** Locks the first record.

```
BEGIN;
UPDATE StudentEnrollments SET course_id = 'CSE111'
WHERE student_id = 1;
```

   - **User B's Session (Step 1): Locks the second record.**

```
BEGIN;

UPDATE StudentEnrollments SET course_id = 'CSE222'
WHERE student_id = 2;
```

   - **User A's Session (Step 2): Tries to lock the second record and is blocked.**

```
UPDATE StudentEnrollments SET course_id = 'CSE112'
WHERE student_id = 2;
```

   - **User B's Session (Step 2):** Tries to lock the first record, triggering a deadlock.

```
UPDATE StudentEnrollments SET course_id = 'CSE221'
WHERE student_id = 1;
```

   - **Result:**

     The database's deadlock detector immediately identifies the cycle. It will choose one transaction as the victim and terminate it with an error (e.g., `ERROR: deadlock detected`). The other transaction (e.g., User A's) will unblock and can then be committed.

2. Part B: Applying MVCC to Prevent Conflicts
   Simulate a reader and a writer accessing the same record concurrently.

   - **User A (Reader): Starts a transaction and reads the data.**

     ```
     BEGIN;

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

   - **User B (Writer): Updates and commits the record *while User A's transaction is still open*.**

     ```
     BEGIN;

     UPDATE StudentEnrollments SET enrollment_date = '2025-
     07-10'
     WHERE student_id = 1;
     COMMIT;
     ```

   - **User A (Reader): Reads the same data again *within its original transaction*.**

     ```
     SELECT enrollment_date FROM StudentEnrollments
     WHERE student_id = 1;
     COMMIT;
     ```

     After Committing: *If User A starts a new query, it will see the updated value (2025-07-10), demonstrating that MVCC provides consistency within a transaction and visibility of changes after it ends.*

3. Part C: Comparing Behavior With and Without MVCC
   Scenario 1: With Traditional Locking (Blocking Behavior) We simulate this using SELECT FOR UPDATE.

   - **User A (Writer):** Locks a row for updating.

     ```
     BEGIN;
     UPDATE StudentEnrollments SET enrollment_date = '2025-
     07-10' WHERE student_id = 1;
     ```

   - User **B (Reader):** Tries to read the same row with a lock.

     ```
     BEGIN;
     SELECT * FROM StudentEnrollments WHERE student_id = 1
     FOR UPDATE;
     ```

   - **Conclusion:** Traditional locking forces operations to wait, serializing access but reducing concurrency.

   Scenario 2: With MVCC (Non-Blocking Behavior) We simulate this using a standard SELECT statement.

   - **User A (Writer): Updates a row.**

     ```
     BEGIN;
     UPDATE StudentEnrollments SET enrollment_date = '2025-
     07-10' WHERE student_id = 1;
     COMMIT;
     ```

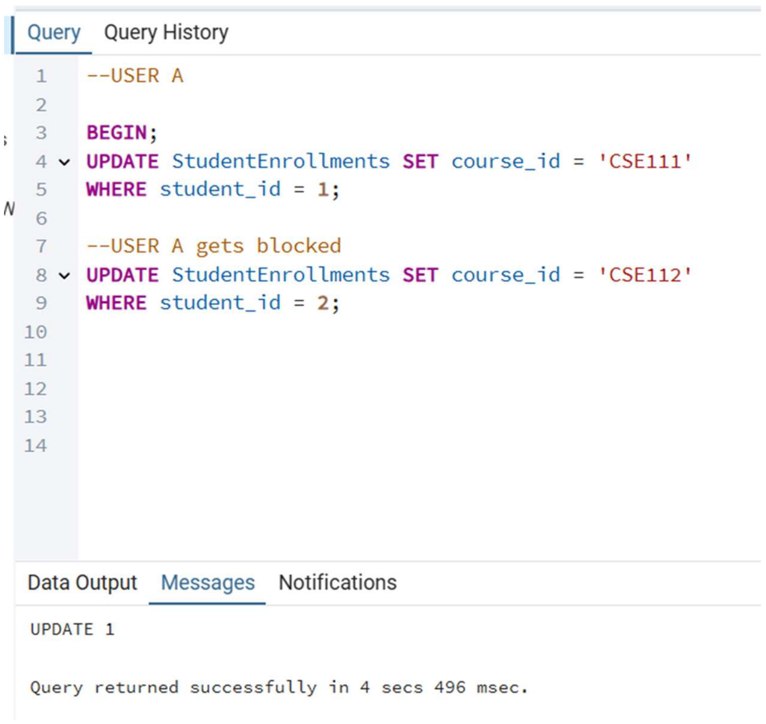
   - **User B (Reader): Reads the data concurrently.**

     ```
     BEGIN;

     SELECT enrollment_date FROM StudentEnrollments WHERE
     student_id = 1;

     COMMIT;
     ```

- **Conclusion:** MVCC allows readers to get consistent data without waiting for writers, dramatically improving performance and scalability in read-heavy applications.

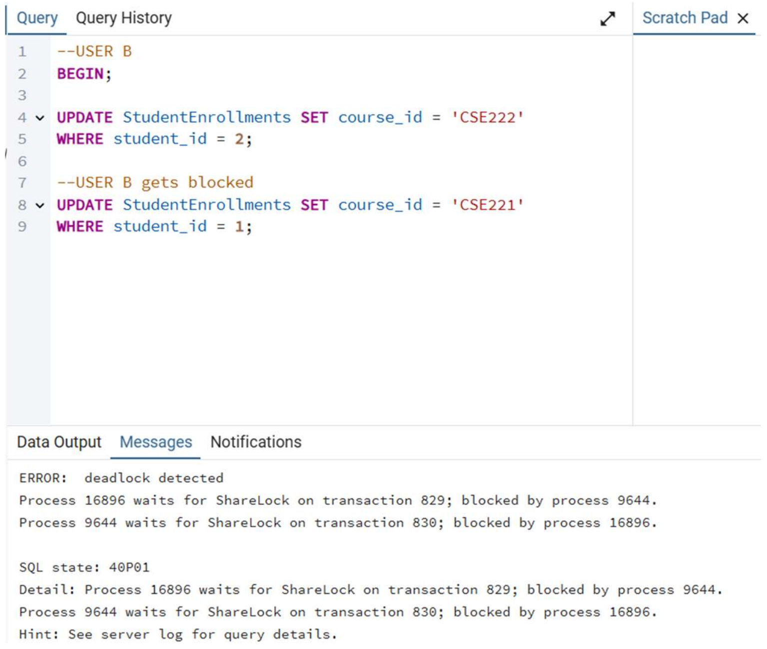
4. Result:

Part-A:



```
--USER A

BEGIN;
UPDATE StudentEnrollments SET course_id = 'CSE111'
WHERE student_id = 1;

--USER A gets blocked
UPDATE StudentEnrollments SET course_id = 'CSE112'
WHERE student_id = 2;
```

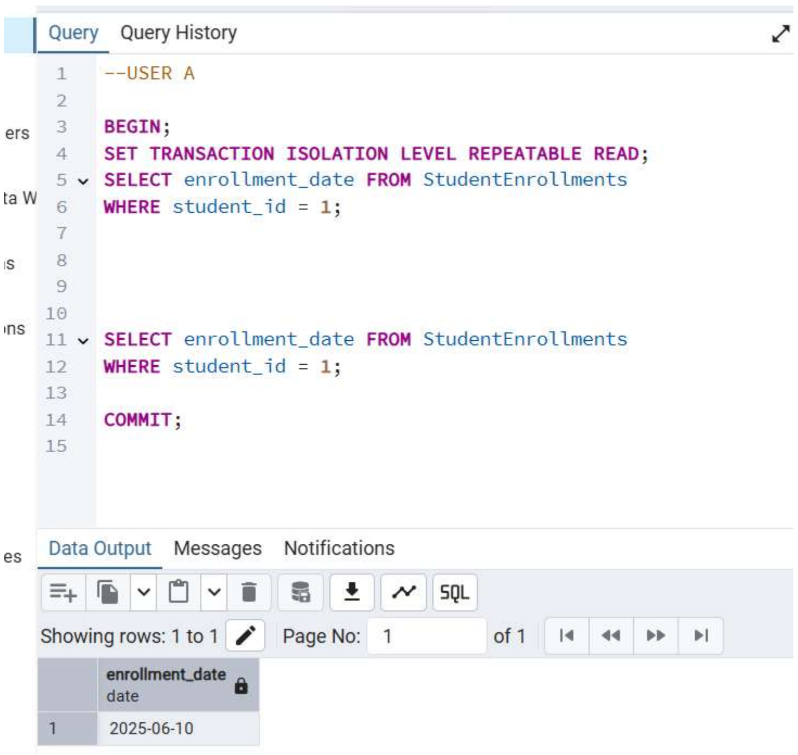
```
UPDATE 1

Query returned successfully in 4 secs 496 msec.
```



```
--USER B
BEGIN;

UPDATE StudentEnrollments SET course_id = 'CSE222'
WHERE student_id = 2;

--USER B gets blocked
UPDATE StudentEnrollments SET course_id = 'CSE221'
WHERE student_id = 1;
```

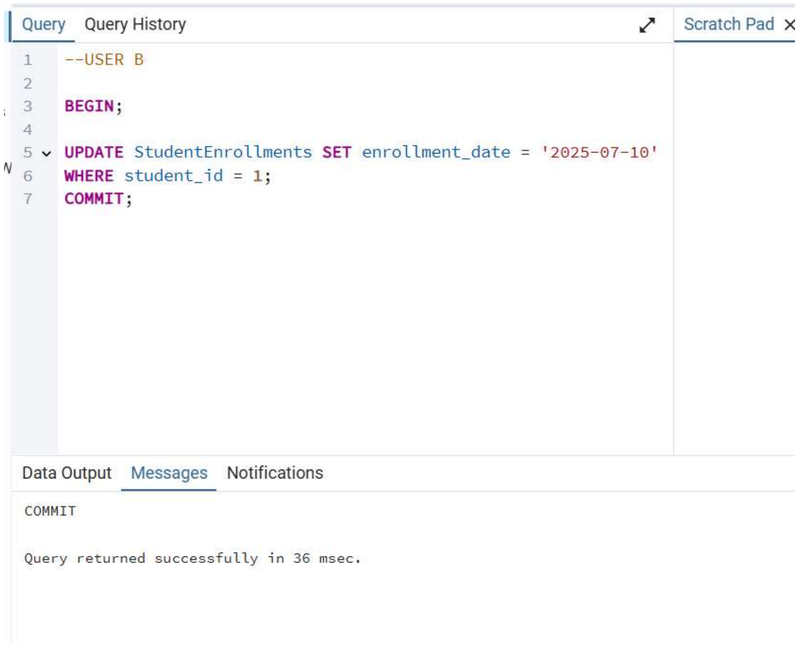
```
ERROR:  deadlock detected
Process 16896 waits for ShareLock on transaction 829; blocked by process 9644.
Process 9644 waits for ShareLock on transaction 830; blocked by process 16896.

SQL state: 40P01
Detail: Process 16896 waits for ShareLock on transaction 829; blocked by process 9644.
Process 9644 waits for ShareLock on transaction 830; blocked by process 16896.
Hint: See server log for query details.
```

Part-B:



```
--USER A

BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT enrollment_date FROM StudentEnrollments
WHERE student_id = 1;




SELECT enrollment_date FROM StudentEnrollments
WHERE student_id = 1;

COMMIT;
```

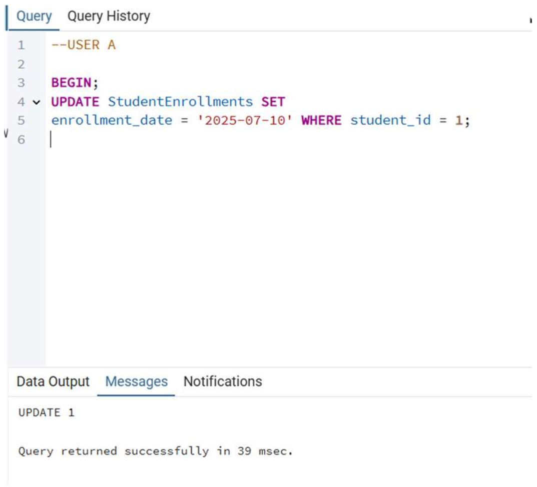
```
   enrollment_date
   date
1  2025-06-10
```



```
--USER B

BEGIN;

UPDATE StudentEnrollments SET enrollment_date = '2025-07-10'
WHERE student_id = 1;
COMMIT;
```

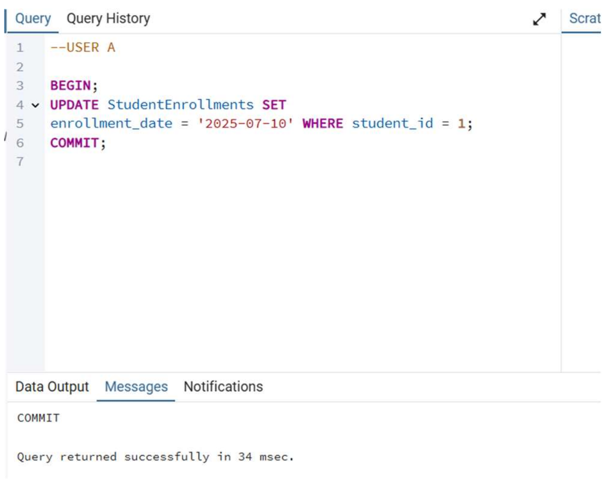
```
COMMIT

Query returned successfully in 36 msec.
```

Part-C:



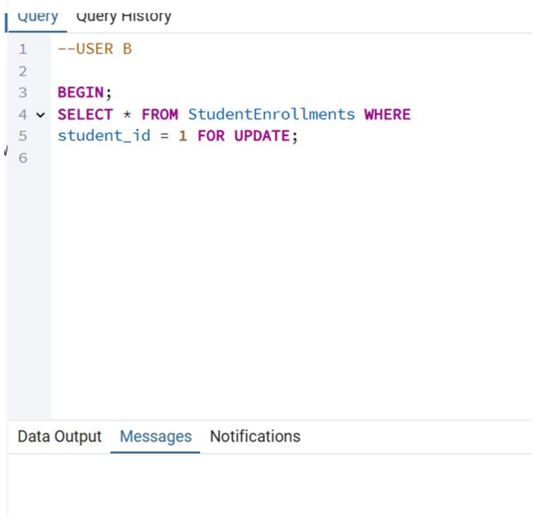
```
--USER A

BEGIN;
UPDATE StudentEnrollments SET
enrollment_date = '2025-07-10' WHERE student_id = 1;
```

```
UPDATE 1

Query returned successfully in 39 msec.
```

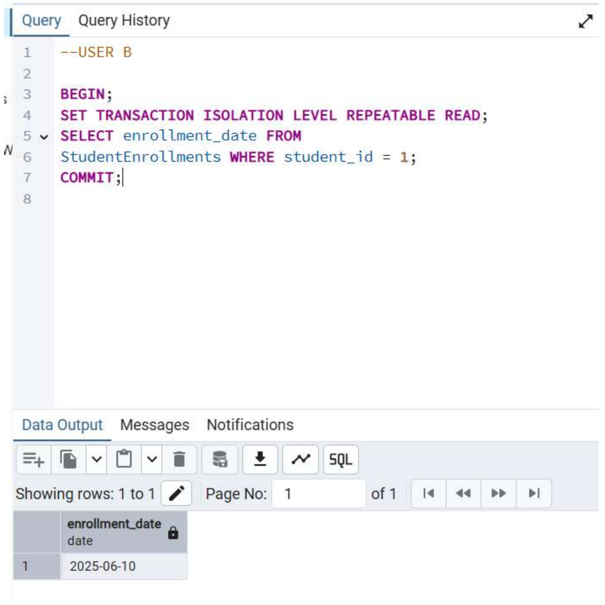


```
--USER A

BEGIN;
UPDATE StudentEnrollments SET
enrollment_date = '2025-07-10' WHERE student_id = 1;
COMMIT;
```

```
COMMIT

Query returned successfully in 34 msec.
```



```
--USER B

BEGIN;
SELECT * FROM StudentEnrollments WHERE
student_id = 1 FOR UPDATE;
```



```
--USER B

BEGIN;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
SELECT enrollment_date FROM
StudentEnrollments WHERE student_id = 1;
COMMIT;
```

```
     enrollment_date
     date
1    2025-06-10
```