Experiment 11

Subject: ADBMS

Name: Aditya Sivam Kashyap

Subject Code:23CSP-333

UID : 23BCC70036

Date: 26th September 2025

Section: 23BCC-1

1. Aim:

Transactions and Concurrency Control

1. Part A: To use a UNIQUE constraint to automatically prevent concurrent users from enrolling the same student in the same course, ensuring data integrity.
2. Part B: To demonstrate explicit row-level locking with SELECT FOR UPDATE to block other transactions from modifying a specific record while it's being processed.
3. Part C: To show how explicit locking serializes concurrent updates, preserving data consistency and preventing common issues like "lost updates."

2. Theory:

Databases provide two main ways to handle concurrency. First, through **constraints** like `UNIQUE`, which enforce data rules directly. The database engine uses internal, short-lived locks to guarantee these rules are never violated, even when multiple users try to insert conflicting data at the same time. This is the best approach for preventing simple duplicates.

For more complex scenarios, like needing to read a record, perform calculations, and then update it, you need explicit control. The **`SELECT FOR UPDATE`** command allows a transaction to place an **exclusive lock** on one or more rows. Any other transaction attempting to modify or lock the same rows will be forced to wait until the first transaction finishes. This powerful technique prevents race conditions and ensures that operations happen in a predictable, serialized order, preserving data consistency.

3. SQL Queries:

1. Part A: Prevent Duplicate Enrolments Using Locking
   - Create the table and add a UNIQUE constraint**.** This constraint is the core mechanism that prevents duplicate enrolments.

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

ALTER TABLE StudentEnrollments
ADD CONSTRAINT uq_student_course UNIQUE (student_name,
course_id);
```

- Simulate concurrent enrollment attempts.

User A's Transaction (Succeeds): This transaction is processed first

```
BEGIN;

INSERT INTO StudentEnrollments (enrollment_id, student_name,
course_id, enrollment_date)
VALUES (1, 'Ashish', 'CSE101', '2024-07-01');

COMMIT;
```

User B's Transaction (Fails): This transaction attempts to insert the same student-course pair.

```
BEGIN;

INSERT INTO StudentEnrollments (enrollment_id, student_name,
course_id, enrollment_date)
VALUES (2, 'Ashish', 'CSE101', '2024-07-02');

COMMIT;
```

2. Part B: Use SELECT FOR UPDATE to Lock Student Record
   - **User A (Acquires Lock):** Starts a transaction to verify a record and locks it to prevent changes.

```
BEGIN;

SELECT * FROM StudentEnrollments
WHERE student_name = 'Ashish' AND course_id = 'CSE101'
FOR UPDATE;
```

- **User B (Is Blocked):** While User A's transaction is open, User B tries to update the same record.

```
UPDATE StudentEnrollments
SET enrollment_date = '2025-09-26'
WHERE student_name = 'Ashish' AND course_id =
'CSE101';
```

- **User A (Releases Lock): Finishes their work and commits the transaction.**

```
COMMIT;
```

As soon as User A commits, User B's blocked UPDATE command automatically unblocks and completes its execution.

3. Part C: Demonstrate Locking Preserving Consistency
   - User A (Starts Update, Acquires Lock):.

```
-- USER A
BEGIN;


SELECT * FROM StudentEnrollments
WHERE student_name = 'Ashish' AND course_id = 'CSE101'
FOR UPDATE;
```

   - User B (Starts Conflicting Update, Is Blocked):

```
--USER B
BEGIN;


UPDATE StudentEnrollments
SET enrollment_date = '2025-10-15'
WHERE student_name = 'Ashish' AND course_id =
'CSE101';
```

- User A (Finishes and Commits):

```
UPDATE StudentEnrollments
SET enrollment_date = '2025-09-30'
WHERE student_name = 'Ashish' AND course_id =
'CSE101';

COMMIT;
```

- **User B (Unblocks and Finishes):** User B's UPDATE now proceeds, operating on the data that User A just committed.

```
COMMIT;
```

- **Final Result:**

  The final `enrollment_date` is `2025-10-15`. The locking mechanism forced the updates to happen sequentially (serialized), preventing User A's update from being "lost" and ensuring the final state of the data is consistent and correct.

```
SELECT student_name, course_id, enrollment_date FROM
StudentEnrollments WHERE student_name = 'Ashish';
```

4. Result:

Part-A:

Query | Query History

```
1  -- CREATE TABLE StudentEnrollments (
2  --     enrollment_id SERIAL PRIMARY KEY,
3  --     student_name VARCHAR(100) NOT NULL,
4  --     course_id VARCHAR(10) NOT NULL,
5  --     enrollment_date DATE NOT NULL,
6  --     CONSTRAINT uq_student_course UNIQUE (student_name, course_id)
7  -- );
8  BEGIN;
9
10 INSERT INTO StudentEnrollments (student_name, course_id, enrollment_date)
11 VALUES ('Ashish', 'CSE101', '2024-07-01');
12 -- Don't COMMIT yet!
13 COMMIT;
14
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 39 msec.
```

Query | Query History

```
1  BEGIN;
2
3  INSERT INTO StudentEnrollments (student_name, course_id, enrollment_date)
4  VALUES ('Ashish', 'CSE101', '2024-07-01');
5
```

Data Output | Messages | Notifications

```
ERROR:  duplicate key value violates unique constraint "uq_student_course"
Key (student_name, course_id)=(Ashish, CSE101) already exists.

SQL state: 23505
Detail: Key (student_name, course_id)=(Ashish, CSE101) already exists.
```

Part-B:

Query Query History

```
1
2  BEGIN;
3
4  SELECT * FROM StudentEnrollments
5  WHERE student_name = 'Ashish' AND course_id = 'CSE101'
6  FOR UPDATE;
7
8  Commit;
9
```

Data Output Messages Notifications

```
COMMIT

Query returned successfully in 60 msec.
```

Query Query History

```
1
2  UPDATE StudentEnrollments
3  SET enrollment_date = '2025-09-26'
4  WHERE student_name = 'Ashish' AND course_id = 'CSE101';
```

Data Output Messages Notifications

```
UPDATE 1

Query returned successfully in 34 secs 77 msec.
```

Part-C:

Query | Query History

```sql
-- USER A
BEGIN;


SELECT * FROM StudentEnrollments
WHERE student_name = 'Ashish' AND course_id = 'CSE101'
FOR UPDATE;


-- After User B is blocked
UPDATE StudentEnrollments
SET enrollment_date = '2025-09-30'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';

COMMIT;
```

Data Output | Messages | Notifications

```
COMMIT

Query returned successfully in 48 msec.
```

Query | Query History | Scratc

```sql
--USER B
BEGIN;


UPDATE StudentEnrollments
SET enrollment_date = '2025-10-15'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';

--After lock is released
COMMIT;

select * from studentEnrollments;
```

Data Output | Messages | Notifications

SQL

Showing rows: 1 to 1 | Page No: 1 of 1

	enrollment_id [PK] integer	student_name character varying (100)	course_id character varying (10)	enrollment_date date
1	1	Ashish	CSE101	2025-10-15