

**EXP-11**

➤ **AIM:** To demonstrate how **row-level locking and transactions** can prevent duplicate enrollments and preserve **data consistency** when multiple users attempt concurrent operations on the same student record.

➤ **THEORY:**

- Transactions in DBMS: Transactions ensure a sequence of operations executes as a single unit, maintaining Atomicity, Consistency, Isolation, and Durability (ACID).
- Concurrency Problems: Without proper locking, two users could insert or update the same student record simultaneously, causing duplicate enrolments or inconsistent data.
- Unique Constraints: Define (student\_name, course\_id) as UNIQUE to prevent duplicate enrollments.
- Row-Level Locking with SELECT FOR UPDATE: Locks specific rows during a transaction, blocking other users from updating the same rows until the transaction is committed or rolled back.
- Prevents race conditions in concurrent environments.
- Locking Preserves Consistency: Ensures no duplicate enrollments occur.
- Conflicting operations are serialized automatically.

➤ **CODES:**

- Part A: Prevent Duplicate Enrollments Using Unique Constraint
- 
- -- Drop table if exists

```
DROP TABLE IF EXISTS StudentEnrollments;
```

```
-- Create table with unique constraint
```

```
CREATE TABLE StudentEnrollments (  
    enrollment_id INT PRIMARY KEY,  
    student_name VARCHAR(100) NOT NULL,  
    course_id VARCHAR(10) NOT NULL,  
    enrollment_date DATE NOT NULL,  
    UNIQUE(student_name, course_id)  
);
```

```
-- Begin transaction to insert multiple records
```

```
START TRANSACTION;
```

```
INSERT INTO StudentEnrollments (enrollment_id,  
student_name, course_id, enrollment_date)
```

```
VALUES
```

```
(1, 'Ashish', 'CSE101', '2024-07-01'),
```

```
(2, 'Smaran', 'CSE102', '2024-07-01'),
```

```
(3, 'Vaibhav', 'CSE101', '2024-07-01');
```

COMMIT;

-- Verify inserted records

SELECT \* FROM StudentEnrollments;

Part B: Use SELECT FOR UPDATE to Lock a Student Record

-- User A locks a row for Ashish in CSE101

START TRANSACTION;

SELECT \* FROM StudentEnrollments

WHERE student\_name = 'Ashish' AND course\_id =  
'CSE101'

FOR UPDATE;

-- At this point, User A keeps transaction open

-- User B attempts to update the same row:

-- UPDATE StudentEnrollments SET enrollment\_date =  
'2024-08-01'

-- WHERE student\_name = 'Ashish' AND course\_id =  
'CSE101';

-- User B will be blocked until User A commits or rolls back

-- User A then commits

COMMIT;

-- After commit, User B can proceed

- Part C: Demonstrate Locking Preserving Consistency

-- Simulate concurrent updates

-- User A starts transaction

START TRANSACTION;

SELECT \* FROM StudentEnrollments

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101'

FOR UPDATE;

-- User A updates enrollment\_date

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-15'

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101';

-- User B (simulated concurrently) tries to update same row

-- UPDATE StudentEnrollments SET enrollment\_date = '2024-08-01'

```
-- WHERE student_name = 'Ashish' AND course_id =  
'CSE101';
```

```
-- This will be blocked until User A commits
```

```
-- User A commits
```

```
COMMIT;
```

```
-- Verify final state
```

```
SELECT * FROM StudentEnrollments;
```

**OUTPUTS:**

enrollment_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-07-01
2	Smaran	CSE102	2024-07-01
3	Vaibhav	CSE101	2024-07-01
enrollment_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-07-01
enrollment_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-07-01
enrollment_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-07-15
2	Smaran	CSE102	2024-07-01
3	Vaibhav	CSE101	2024-07-01
payment_id	student_name	amount	payment_date
1	Ashish	5000.00	2024-06-01
2	Smaran	4500.00	2024-06-02
3	Vaibhav	5500.00	2024-06-03

## ➤ **LEARNING OUTCOMES:**

1. Learned to enforce unique constraints to prevent duplicate student enrollments.
2. Understood row-level locking using SELECT FOR UPDATE to handle concurrent transactions.
3. Observed how transactions preserve Atomicity and Consistency in a multi-user environment.

4. Practiced handling blocked transactions and understanding isolation effects.
5. Gained hands-on experience with ACID principles in a practical enrollment scenario.