# Name: Aaryan Ranaut UID: 23BCC70032

# ADBMS EXPERIMENT-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

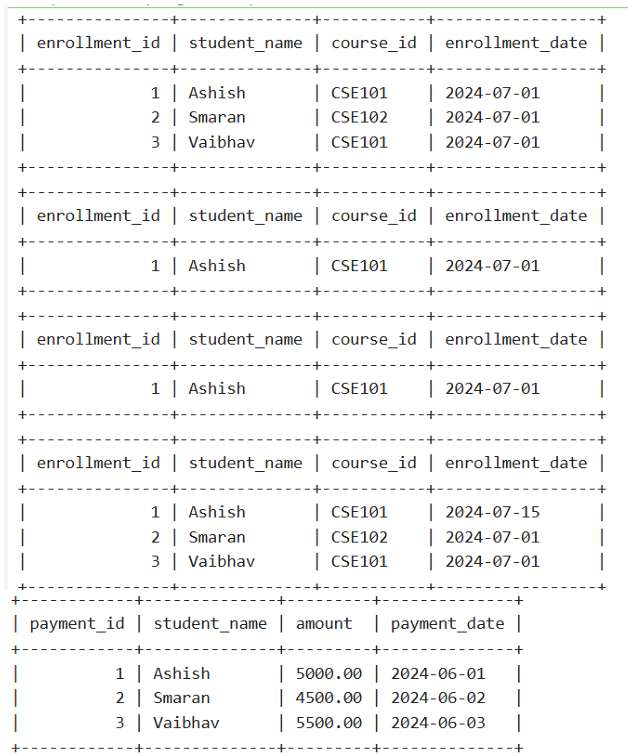
-- User A locks a row for Ashish in CSE101  
START TRANSACTION;  
  
SELECT \* FROM StudentEnrollments  
WHERE student\_name = 'Ashish' AND course\_id = 'CSE101'  
FOR UPDATE;  
  
-- At this point, User A keeps transaction open  
-- User B attempts to update the same row:  
-- UPDATE StudentEnrollments 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:

Outputs as per SQL execution verifying prevention of duplicates and row-level locking effects.



## LEARNING OUTCOMES:

Learned to enforce unique constraints to prevent duplicate student enrollments.

Understood row-level locking using SELECT FOR UPDATE to handle concurrent transactions.

Observed how transactions preserve Atomicity and Consistency in a multi-user environment.

Practiced handling blocked transactions and understanding isolation effects.

Gained hands-on experience with ACID principles in a practical enrollment scenario.