EXPERIMENT-11
ADVANCED DATABASE MANAGEMENT SYSTEMS

Aim:

To demonstrate how transactions, unique constraints, and row-level locking prevent duplicate enrollments, maintain data consistency, and handle concurrent operations safely in a multi-user environment.

Theory:

1. **Unique Constraint:** Ensures each student can enroll in a course only once.
2. **Transactions:** Provide atomicity—either all operations succeed, or none do.
3. **Row-Level Locking (SELECT FOR UPDATE):** Prevents race conditions by locking rows during verification or updates, ensuring consistent and conflict-free operations.
4. **Concurrency Handling:** Proper locking and transactions serialize conflicting operations, avoiding data corruption or duplication.

Part A: Prevent Duplicate Enrollments Using Locking

```
-- Create table with UNIQUE constraint
CREATE TABLE StudentEnrollments (
   enrollment_id INT PRIMARY KEY,
   student_name VARCHAR(100),
   course_id VARCHAR(10),
   enrollment_date DATE,
   UNIQUE(student_name, course_id)
);

-- Insert sample data
INSERT INTO StudentEnrollments VALUES
(1, 'Ashish', 'CSE101', '2024-07-01'),
(2, 'Smaran', 'CSE102', '2024-07-01'),
(3, 'Vaibhav', 'CSE101', '2024-07-01');

-- Simulate concurrent enrollment in a transaction
START TRANSACTION;
INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id, enrollment_date)
VALUES (4, 'Ashish', 'CSE101', '2024-07-02'); -- This will fail due to UNIQUE constraint
COMMIT;
```

Part B: Use SELECT FOR UPDATE to Lock Student Record

```
-- User A: Lock the row for verification
START TRANSACTION;
SELECT * FROM StudentEnrollments
WHERE student_name = 'Ashish' AND course_id = 'CSE101'
```

```
FOR UPDATE;

-- User A performs verification or update here
-- Transaction remains open, row is locked

-- User B: Attempt to update same row while User A's transaction is open
START TRANSACTION;
UPDATE StudentEnrollments
SET enrollment_date = '2024-07-03'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';
-- This will wait until User A commits
COMMIT; -- User B executes after User A finishes
```

Part C: Demonstrate Locking Preserving Consistency

```
-- Initial Data
SELECT * FROM StudentEnrollments;

-- User A starts transaction and locks the row
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 = '2024-07-05'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';

-- User B tries to update same row concurrently
START TRANSACTION;
UPDATE StudentEnrollments
SET enrollment_date = '2024-07-10'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';
-- This will wait until User A commits

COMMIT; -- User A commits
COMMIT; -- User B commits safely
```

OUTPUT:

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

Caption

Learning Outcomes:

1. Understand how to **prevent duplicate entries** using unique constraints.
2. Learn to **use transactions** (`START TRANSACTION`, `COMMIT`) to maintain atomicity.
3. Understand **row-level locking** to handle concurrent updates (`SELECT FOR UPDATE`).
4. Learn to **simulate multi-user environments** and see blocking behavior in practice.