Practice 2 – Transactions & Concurrency Control

Assignment Solution

This document contains solutions to the given assignment with SQL code, explanations, dry runs, and sample outputs.

Part A: Prevent Duplicate Enrollments Using Locking

Objective: Prevent two users from enrolling the same student into the same course simultaneously using unique constraints and transactions.

SQL Code:

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

-- User 1 transaction
START TRANSACTION;
INSERT INTO StudentEnrollments(student_name, course_id, enrollment_date)
VALUES('Ashish', 'CSE101', '2024-07-01');
COMMIT;

-- User 2 transaction (will fail due to duplicate pair)
START TRANSACTION;
INSERT INTO StudentEnrollments(student_name, course_id, enrollment_date)
VALUES('Ashish', 'CSE101', '2024-07-01');
COMMIT;
```

Explanation:

• The UNIQUE constraint ensures each (student_name, course_id) pair is unique.
• User 1 inserts successfully.
• User 2’s transaction fails with a constraint violation, preventing duplicate enrollment.

Sample Output:

```
User 1: Record inserted successfully.
User 2: ERROR 1062 (23000): Duplicate entry 'Ashish-CSE101' for key
'student_course_unique'
```

Part B: Use SELECT FOR UPDATE to Lock Student Record

Objective: Use row-level locking via SELECT FOR UPDATE to prevent conflicts while verifying a student before enrollment.

SQL Code:

```
-- User A
START TRANSACTION;
SELECT * FROM StudentEnrollments
WHERE student_name='Ashish' AND course_id='CSE101'
FOR UPDATE;

-- (Row is now locked until commit/rollback)

-- User B (while User A has not committed yet)
UPDATE StudentEnrollments
SET enrollment_date='2024-07-02'
WHERE student_name='Ashish' AND course_id='CSE101';
-- This query will be blocked until User A commits or rollbacks.
```

Explanation:

• User A locks the row with SELECT FOR UPDATE.
• User B’s update is blocked until User A finishes.
• Ensures consistency and avoids simultaneous conflicting updates.

Sample Output:

```
User A: Row selected and locked.
User B: Query waiting... (blocked until User A commits).
```

Part C: Demonstrate Locking Preserving Consistency

Objective: Show how locking ensures consistent data when multiple users attempt concurrent updates.

SQL Code:

```
-- Without Locking (Problematic)
User A: UPDATE StudentEnrollments
    SET enrollment_date='2024-07-03'
    WHERE student_name='Ashish' AND course_id='CSE101';

User B (simultaneous): UPDATE StudentEnrollments
    SET enrollment_date='2024-07-04'
    WHERE student_name='Ashish' AND course_id='CSE101';

-- Result: Last update overwrites previous one (race condition).

-- With Locking
User A:
START TRANSACTION;
SELECT * FROM StudentEnrollments
WHERE student_name='Ashish' AND course_id='CSE101' FOR UPDATE;
UPDATE StudentEnrollments SET enrollment_date='2024-07-03'
WHERE student_name='Ashish' AND course_id='CSE101';
COMMIT;

User B (waits until User A finishes):
START TRANSACTION;
UPDATE StudentEnrollments SET enrollment_date='2024-07-04'
WHERE student_name='Ashish' AND course_id='CSE101';
COMMIT;
```

Explanation:

- Without locking, both users’ updates conflict, and only the last one persists.
- With locking, transactions are serialized: User B waits until User A finishes, preserving

consistency.

Sample Output:

Final Table Row:
student_name = Ashish, course_id = CSE101, enrollment_date = 2024-07-04

Conclusion

This assignment demonstrates how transactions and locking mechanisms prevent race conditions, duplicate entries, and inconsistent data. Using UNIQUE constraints, SELECT FOR UPDATE, and proper transaction management ensures that database operations follow ACID properties, preserving integrity and consistency in concurrent environments.