Experiment :11

Part A: Prevent Duplicate Enrollments Using Locking Description:

Simulate concurrent users attempting to enroll students in courses. Implement a mechanism that prevents two users from enrolling the same student into the same course simultaneously by using transactions and unique constraints.

Input Format:

- Table StudentEnrollments with columns:
  - `enrollment_id` (INT, Primary Key)
  - `student_name` (VARCHAR(100))
  - `course_id` (VARCHAR(10))
  - `enrollment_date` (DATE)

Output Format:

Only one user should be able to insert the record successfully for a given `(student_name, course_id)` pair.

Constraints:

- Each student can enroll in a course only once.
- The pair `(student_name, course_id)` must be unique.
- Use transactions to handle concurrent access.

Sample Input:

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

Sample Output:If two users try to enroll `'Ashish'` in `'CSE101'`, only the first will succeed; the second will get a constraint violation.

Query:

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

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')
ON DUPLICATE KEY UPDATE enrollment_id = enrollment_id;

START TRANSACTION;

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

INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id,
enrollment_date)
SELECT 4, 'Ashish', 'CSE101', '2024-07-02'
WHERE NOT EXISTS (
   SELECT 1 FROM StudentEnrollments
   WHERE student_name = 'Ashish' AND course_id = 'CSE101'
);

COMMIT;
```

Output:

queries.sql

AI NEW MYSQL RUN

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

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')
ON DUPLICATE KEY UPDATE enrollment_id = enrollment_id;

START TRANSACTION;

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

INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id, enrollment_date)
SELECT 4, 'Ashish', 'CSE101', '2024-07-02'
WHERE NOT EXISTS (
    SELECT 1 FROM StudentEnrollments
    WHERE student_name = 'Ashish' AND course_id = 'CSE101'
);

COMMIT;
```

STDIN

Input for the program (Optional)

Output:

```
+---------------+--------------+-----------+-----------------+
| enrollment_id | student_name | course_id | enrollment_date |
+---------------+--------------+-----------+-----------------+
|             1 | Ashish       | CSE101    | 2024-07-01      |
+---------------+--------------+-----------+-----------------+
```

Part B: Use SELECT FOR UPDATE to Lock Student Record

Description:

Use row-level locking via `SELECT FOR UPDATE` to prevent conflicts. Simulate a situation where a student is being verified before enrollment and locked until confirmation, preventing other users from updating it simultaneously.

Input Format:

- Same table: StudentEnrollments

Output Format:

The selected row will be locked until the transaction is committed or rolled back. Other users trying to access that row will be blocked.

Constraints:

- Use `START TRANSACTION` and `SELECT FOR UPDATE`.
- Locking should block conflicting transactions on the same record.

Sample Input:

Simulation Steps (Using Row-Level Locking with `SELECT FOR UPDATE`) User

A:

1. Start a transaction.
2. Use a `SELECT FOR UPDATE` query to lock the specific row where:
    - Student name is `'Ashish'`
    - Course ID is `'CSE101'`
3. Keep the transaction open (do not commit or rollback yet).

This locks the row so that no one else can update it until User A finishes.

User B (while User A's transaction is still open):

1. Try to update the same row (`student_name = 'Ashish'` and `course_id = 'CSE101'`).

This update will be blocked (it will wait) because the row is locked by User A.

Sample Output:

User B will be blocked until User A finishes the transaction.

Query:

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

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')
ON DUPLICATE KEY UPDATE enrollment_id = enrollment_id;

START TRANSACTION;

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

UPDATE StudentEnrollments
SET enrollment_date = '2024-08-01'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';

COMMIT;
```

Output:

Part C: Demonstrate Locking Preserving Consistency in Concurrent Transactions Description:

Demonstrate how locking preserves data consistency when multiple users attempt concurrent updates. Show how update conflicts are avoided when row-level locks are used appropriately in transactions.

Input Format:

Same StudentEnrollments table as above.

Output Format:

Conflicting operations are serialized due to locking, and data remains consistent without corruption or duplication.

Constraints:

- Each user must use transactions with locking.
- Show that without locking, both users could overwrite each other's changes.

Sample Input:

enrollment_id	student_name	course_id	enrollment_date
1	Ashish	CSE101	2024-07-01

Sample Output:

After both users run their updates one after the other, only the last committed update is reflected — no race condition or inconsistent data.

Output: