

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

```
1 CREATE TABLE IF NOT EXISTS StudentEnrollments (  
2   enrollment_id INT PRIMARY KEY,  
3   student_name VARCHAR(100),  
4   course_id VARCHAR(10),  
5   enrollment_date DATE,  
6   UNIQUE(student_name, course_id)  
7 );  
8  
9 INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id, enrollment_date)  
10 VALUES  
11 (1, 'Ashish', 'CSE101', '2024-07-01'),  
12 (2, 'Smaran', 'CSE102', '2024-07-01'),  
13 (3, 'Vaibhav', 'CSE101', '2024-07-01')  
14 ON DUPLICATE KEY UPDATE enrollment_id = enrollment_id;  
15  
16 START TRANSACTION;  
17  
18 SELECT * FROM StudentEnrollments  
19 WHERE student_name = 'Ashish' AND course_id = 'CSE101'  
20 FOR UPDATE;  
21  
22 INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id, enrollment_date)  
23 SELECT 4, 'Ashish', 'CSE101', '2024-07-02'  
24 WHERE NOT EXISTS (  
25   SELECT 1 FROM StudentEnrollments  
26   WHERE student_name = 'Ashish' AND course_id = 'CSE101'  
27 );  
28  
29 COMMIT;  
30
```

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:



The screenshot shows a MySQL query editor with a list of queries on the left and a preview of the output on the right. The queries are as follows:

```
1 CREATE TABLE IF NOT EXISTS StudentEnrollments (  
2   enrollment_id INT PRIMARY KEY,  
3   student_name VARCHAR(100),  
4   course_id VARCHAR(10),  
5   enrollment_date DATE,  
6   UNIQUE(student_name, course_id)  
7 );  
8  
9 INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id, enrollment_date)  
10 VALUES  
11 (1, 'Ashish', 'CSE101', '2024-07-01'),  
12 (2, 'Smaran', 'CSE102', '2024-07-01'),  
13 (3, 'Vaibhav', 'CSE101', '2024-07-01')  
14 ON DUPLICATE KEY UPDATE enrollment_id = enrollment_id;  
15  
16 START TRANSACTION;  
17  
18 SELECT *  
19 FROM StudentEnrollments  
20 WHERE student_name = 'Ashish' AND course_id = 'CSE101'  
21 FOR UPDATE;  
22  
23 UPDATE StudentEnrollments  
24 SET enrollment_date = '2024-08-01'  
25 WHERE student_name = 'Ashish' AND course_id = 'CSE101';  
26  
27 COMMIT;
```

The output table on the right is titled "Output:" and shows the result of the SELECT query. It has four columns: enrollment_id, student_name, course_id, and enrollment_date. The data row shows enrollment_id 1, student_name Ashish, course_id CSE101, and enrollment_date 2024-07-01.

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

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:

The screenshot shows a SQL IDE interface. On the left, a query is written in a text editor. The query creates a table named 'StudentEnrollments' with columns 'enrollment_id' (INT PRIMARY KEY), 'student_name' (VARCHAR(100)), 'course_id' (VARCHAR(10)), and 'enrollment_date' (DATE). It includes a UNIQUE constraint on (student_name, course_id). The query then inserts a row with enrollment_id 1, student_name 'Ashish', course_id 'CSE101', and enrollment_date '2024-07-01'. It starts a transaction, updates the enrollment_date to '2024-07-15' for the row where student_name is 'Ashish' and course_id is 'CSE101', and finally commits the transaction.

```
1 CREATE TABLE IF NOT EXISTS StudentEnrollments (  
2   enrollment_id INT PRIMARY KEY,  
3   student_name VARCHAR(100),  
4   course_id VARCHAR(10),  
5   enrollment_date DATE,  
6   UNIQUE(student_name, course_id)  
7 );  
8  
9 INSERT INTO StudentEnrollments (enrollment_id, student_name, course_id, enrollment_date)  
10 VALUES  
11 (1, 'Ashish', 'CSE101', '2024-07-01')  
12 ON DUPLICATE KEY UPDATE enrollment_id = enrollment_id;  
13  
14 START TRANSACTION;  
15  
16 SELECT *  
17 FROM StudentEnrollments  
18 WHERE student_name = 'Ashish' AND course_id = 'CSE101'  
19 FOR UPDATE;  
20  
21 UPDATE StudentEnrollments  
22 SET enrollment_date = '2024-07-15'  
23 WHERE student_name = 'Ashish' AND course_id = 'CSE101';  
24  
25 COMMIT;  
26
```

On the right, the output is displayed. It shows the result of the query, which is a single row with enrollment_id 1, student_name 'Ashish', course_id 'CSE101', and enrollment_date '2024-07-01'.

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