Experiment– 11

Name : Meenakshi

UID : 23BDO10011

Subject : ADBMS

Section & Group : 23BCC-BDO-2

QUESTION :

Demonstrate how row-level locking and transactions help prevent duplicate enrollments and ensure consistency when multiple users try to modify the same student record concurrently.

Solution:

Aim: To illustrate how row-level locking and transaction control preserve consistency and prevent duplicate enrollments in a database, especially when multiple users access the same student record simultaneously.

Theory: A transaction in DBMS is a group of SQL operations executed as a single logical unit, following the ACID principles:
Atomicity: All steps of a transaction succeed together, or none take effect.
Consistency: Ensures the database remains valid before and after transactions.
Isolation: Multiple transactions can run without interfering with each other.
Durability: Once committed, changes are stored permanently.

In multi-user environments, concurrency problems arise if two users attempt to insert or update the same record at the same time.
Unique Constraints: By defining (student_name, course_id) as unique, the database avoids duplicate enrollments.
Row-Level Locking (SELECT FOR UPDATE): This command locks specific rows during a transaction so no other transaction can modify them until commit/rollback.
Consistency Preservation: Conflicting operations are serialized, ensuring reliable results.

```
-- Part A: Prevent Duplicate Enrollments Using Unique Constraint
DROP TABLE IF EXISTS StudentEnrollments;
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)
);

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;

SELECT * FROM StudentEnrollments;

-- Part B: Use SELECT FOR UPDATE to Lock a Student Record
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_name = 'Ashish' AND
course_id = 'CSE101' FOR UPDATE;

-- User A keeps transaction open
-- If User B tries to update the same row, they will be blocked until
User A commits

COMMIT; -- After commit, User B can proceed

-- Part C: Demonstrate Locking Preserving Consistency
START TRANSACTION;
SELECT * FROM StudentEnrollments WHERE student_name = 'Ashish' AND
course_id = 'CSE101' FOR UPDATE;

UPDATE StudentEnrollments SET enrollment_date = '2024-07-15'
WHERE student_name = 'Ashish' AND course_id = 'CSE101';

-- If User B tries a concurrent update, they will be blocked until User
A commits
COMMIT;

SELECT * FROM StudentEnrollments;
```

Outputs:

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

Learning Outcomes:

Learned how to use unique constraints to prevent duplicate enrollments.

Practiced applying row-level locking with SELECT FOR UPDATE.

Observed how transactions preserve atomicity and consistency in a multi-user setup.

Understood the impact of blocked transactions and isolation effects.

Gained practical exposure to ACID properties in concurrent environments.