ADBMS CODES [Transactions and concurrency control]

Practice-1

-- ===========================================

-- Create Table (if not exists)

-- ===========================================

CREATE TABLE FeePayments (

payment\_id INT PRIMARY KEY,

student\_name VARCHAR(100) NOT NULL,

amount DECIMAL(10,2) CHECK (amount > 0),

payment\_date DATE NOT NULL

);

-- ===========================================

-- Part A: Insert Multiple Fee Payments (COMMIT)

-- ===========================================

START TRANSACTION;

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES

(1, 'Ashish', 5000.00, '2024-06-01'),

(2, 'Smaran', 4500.00, '2024-06-02'),

(3, 'Vaibhav', 5500.00, '2024-06-03');

COMMIT;

SELECT \* FROM FeePayments;

-- ===========================================

-- Part B: Demonstrate ROLLBACK (duplicate ID)

-- ===========================================

START TRANSACTION;

-- Valid insert

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES (4, 'Kiran', 4800.00, '2024-06-05');

-- Invalid insert: duplicate ID = 1, negative amount

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES (1, 'Ashish', -3000.00, '2024-06-06');

-- Transaction fails → Rollback

ROLLBACK;

SELECT \* FROM FeePayments;

-- ===========================================

-- Part C: Partial Failure (NULL student\_name)

-- ===========================================

START TRANSACTION;

-- Valid insert

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES (5, 'Anita', 5200.00, '2024-06-07');

-- Invalid insert: NULL name not allowed

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES (6, NULL, 4000.00, '2024-06-07');

-- Transaction fails → Rollback

ROLLBACK;

SELECT \* FROM FeePayments;

-- ===========================================

-- Part D: Verify ACID Compliance

-- ===========================================

START TRANSACTION;

-- Valid inserts

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES

(7, 'Sneha', 4700.00, '2024-06-08'),

(8, 'Arjun', 4900.00, '2024-06-09');

-- Invalid insert: duplicate ID = 1

INSERT INTO FeePayments (payment\_id, student\_name, amount, payment\_date)

VALUES (1, 'Duplicate', 5100.00, '2024-06-10');

-- Rollback to undo all

ROLLBACK;

-- Final state: only original committed records (1,2,3)

SELECT \* FROM FeePayments;

Practice-2

-- ===========================================

-- Create Table with Constraints

-- ===========================================

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,

CONSTRAINT unique\_student\_course UNIQUE (student\_name, course\_id)

);

-- ===========================================

-- Insert Initial Data

-- ===========================================

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');

SELECT \* FROM StudentEnrollments;

-- ===========================================

-- Part A: Prevent Duplicate Enrollments

-- ===========================================

-- User 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 (duplicate pair)

-- Rollback since duplicate attempt

ROLLBACK;

-- Only unique combinations exist

SELECT \* FROM StudentEnrollments;

-- ===========================================

-- Part B: Row Locking with SELECT FOR UPDATE

-- ===========================================

-- User A

START TRANSACTION;

SELECT \* FROM StudentEnrollments

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101'

FOR UPDATE;

-- Locks this row until commit/rollback

-- (At this point, User A keeps transaction open...)

-- User B (in another session) tries to update:

-- This query will BLOCK until User A commits/rolls back

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-05'

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101';

-- Once User A executes COMMIT or ROLLBACK,

-- User B’s update will proceed.

-- ===========================================

-- Part C: Demonstrate Locking Preserves Consistency

-- ===========================================

-- Suppose both User A and User B try to update the same row.

-- User A

START TRANSACTION;

SELECT \* FROM StudentEnrollments

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101'

FOR UPDATE;

-- Locks the row

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-10'

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101';

COMMIT;

-- Unlocks row

-- User B (runs after A commits)

START TRANSACTION;

SELECT \* FROM StudentEnrollments

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101'

FOR UPDATE;

-- Now it can lock safely

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-15'

WHERE student\_name = 'Ashish' AND course\_id = 'CSE101';

COMMIT;

-- Final state: enrollment\_date = 2024-07-15 (last committed update)

SELECT \* FROM StudentEnrollments;

Practice-3

-- ===========================================

-- Create Table

-- ===========================================

CREATE TABLE StudentEnrollments (

student\_id INT PRIMARY KEY,

student\_name VARCHAR(100) NOT NULL,

course\_id VARCHAR(10) NOT NULL,

enrollment\_date DATE NOT NULL

);

-- Insert initial data

INSERT INTO StudentEnrollments (student\_id, student\_name, course\_id, enrollment\_date)

VALUES

(1, 'Ashish', 'CSE101', '2024-06-01'),

(2, 'Smaran', 'CSE102', '2024-06-01'),

(3, 'Vaibhav', 'CSE103', '2024-06-01');

SELECT \* FROM StudentEnrollments;

-- ===========================================

-- Part A: Simulate Deadlock Between Two Users

-- ===========================================

-- (Run in two sessions to cause a deadlock)

-- User A

START TRANSACTION;

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-01'

WHERE student\_id = 1;

-- Locks row 1

-- Later in same transaction

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-02'

WHERE student\_id = 2;

-- Will wait for User B

-- User B

START TRANSACTION;

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-05'

WHERE student\_id = 2;

-- Locks row 2

-- Later in same transaction

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-06'

WHERE student\_id = 1;

-- Deadlock occurs here

-- Result: DB detects deadlock and rolls back one transaction automatically.

-- ===========================================

-- Part B: MVCC – Concurrent Reads and Writes

-- ===========================================

-- User A (Reader)

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- Sees old value: enrollment\_date = '2024-06-01'

-- User B (Writer)

START TRANSACTION;

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-10'

WHERE student\_id = 1;

COMMIT;

-- User A (still in same transaction, sees old snapshot)

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- Still sees '2024-06-01'

COMMIT;

-- New session/User A after commit

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- Now sees updated '2024-07-10'

-- ===========================================

-- Part C: Compare Locking vs MVCC

-- ===========================================

-- Without MVCC (Locking with SELECT FOR UPDATE)

-- User A

START TRANSACTION;

SELECT \* FROM StudentEnrollments

WHERE student\_id = 1

FOR UPDATE;

-- Locks row

-- User B

START TRANSACTION;

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- This blocks until User A commits

-- With MVCC (Non-blocking Reads)

-- User A

SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;

START TRANSACTION;

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- Sees old value (snapshot)

-- User B

START TRANSACTION;

UPDATE StudentEnrollments

SET enrollment\_date = '2024-07-15'

WHERE student\_id = 1;

COMMIT;

-- User A (still in same transaction)

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- Still sees old snapshot value

COMMIT;

-- User A (new transaction after commit)

SELECT \* FROM StudentEnrollments WHERE student\_id = 1;

-- Now sees updated '2024-07-15'