Name : Akshit Sharma
UID – 23BCS10929
SEC – 622(B)

Practice 1 – Transactions & Concurrency Control

Part A: Insert Multiple Fee Payments in a Transaction

Objective: Insert multiple payments as a single transaction to demonstrate Atomicity. Either all rows are inserted or none are.

SQL Code:

```
CREATE TABLE FeePayments (
   payment_id INT PRIMARY KEY,
   student_name VARCHAR(100) NOT NULL,
   amount DECIMAL(10,2) CHECK (amount > 0),
   payment_date DATE
);

-- Insert multiple rows in one transaction
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;
```

Simulated Output (Terminal):

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> 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');
Query OK, 3 rows affected (0.02 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
```

Simulated Output (Clean Table):

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

Part B: Demonstrate ROLLBACK for Failed Payment Insertion

Objective: Show that when a transaction contains an invalid operation, ROLLBACK undoes all changes.

SQL Code:

```
-- Assume FeePayments table exists as above and already has entries 1,2,3
-- Start a transaction that will fail due to constraint violation
START TRANSACTION;
INSERT INTO FeePayments(payment_id, student_name, amount, payment_date)
VALUES (4, 'Kiran', 4800.00, '2024-06-04'),
    (1, 'Ashish', -100.00, '2024-06-05'); -- duplicate ID and negative amount
-- Error occurs, so rollback
ROLLBACK;
```

Simulated Output (Terminal - Rollback):

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO FeePayments(payment_id, student_name, amount, payment_date)
    -> VALUES (4, 'Kiran', 4800.00, '2024-06-04'),
    ->        (1, 'Ashish', -100.00, '2024-06-05');
ERROR 1406 (22003): Data out of range or CHECK constraint violated
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

Table after Rollback (Clean):

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

Part C: Simulate Partial Failure and Ensure Consistent State

Objective: Demonstrate that a transaction with one valid and one invalid insert is fully rolled back.

SQL Code:

```
START TRANSACTION;
INSERT INTO FeePayments(payment_id, student_name, amount, payment_date)
VALUES (4, 'Kiran', 4800.00, '2024-06-04'),
    (5, NULL, 5000.00, '2024-06-05'); -- NULL student_name causes NOT NULL violation
ROLLBACK;
```

Simulated Output (Terminal - NULL error):

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO FeePayments(payment_id, student_name, amount, payment_date)
    -> VALUES (4, 'Kiran', 4800.00, '2024-06-04'),
    ->        (5, NULL, 5000.00, '2024-06-05');
ERROR 1048 (23000): Column 'student_name' cannot be null
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
```

Table after Rollback (Clean):

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

Part D: Verify ACID Compliance with Transaction Flow

Objective: Combine the techniques and explain how ACID properties are preserved.

SQL Code (Demonstrative):

```
-- Atomicity: group multiple operations
START TRANSACTION;
INSERT INTO FeePayments(payment_id, student_name, amount, payment_date) VALUES (6,
'Riya', 5200.00, '2024-06-06');
UPDATE FeePayments SET amount = amount + 100 WHERE payment_id = 2;
COMMIT;

-- Isolation: use SELECT ... FOR UPDATE to lock rows during complex operations
START TRANSACTION;
SELECT * FROM FeePayments WHERE payment_id = 6 FOR UPDATE;
-- perform checks and updates
COMMIT;

-- Durability: once committed, changes persist even after DB restart (demonstrated by
SELECT after COMMIT)
SELECT * FROM FeePayments;
```

Simulated ACID Timeline (Terminal):

```
Session 1 (Atomicity & Isolation):
START TRANSACTION;
SELECT * FROM FeePayments WHERE payment_id=6 FOR UPDATE;
-- holds lock, performs multiple inserts and updates
Session 2 (Blocked due to Isolation):
Attempting to INSERT/UPDATE the same payment_id=6 will wait until session1 commits.
COMMIT;
-- Changes durable (Durability)
```

Conclusion

This document demonstrated transactions for FeePayments table covering Atomicity, Consistency, Isolation, and Durability. Simulated terminal outputs and clean tables show that commits make changes persistent while errors trigger rollbacks to keep the database in a consistent state.