EXPERIMENT-10
ADVANCED DATABASE AND MANAGEMENT SYSTEMS

AIM:

- To demonstrate **ACID properties (Atomicity, Consistency, Isolation, Durability)** in SQL using transactions with the `FeePayments` table.
- To show how **COMMIT** and **ROLLBACK** control transaction success and failure.

THEORY:

Atomicity: Ensures all operations in a transaction succeed together, or none at all.

- **Consistency:** Database remains in a valid state before and after a transaction.
- **Isolation:** Transactions execute independently without interfering with each other.
- **Durability:** Once committed, changes persist even after a system crash.
- **Transactions:** A group of SQL statements executed as a single unit.
- **COMMIT:** Makes a transaction permanent.
- **ROLLBACK:** Undoes a transaction if any operation fails.

SQL Code / Implementation

Part A – Insert Multiple Fee Payments (Atomicity & Durability)

```
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;
```

Explanation:

- All inserts succeed together.

- COMMIT ensures data is durable.

Part B – Demonstrate ROLLBACK for Failed Payment

```
START TRANSACTION;

-- First valid insert
INSERT INTO FeePayments(payment_id, student_name, amount,
payment_date)
VALUES (4, 'Kiran', 4800.00, '2024-06-04');

-- Invalid insert (duplicate payment_id or negative amount)
INSERT INTO FeePayments(payment_id, student_name, amount,
payment_date)
VALUES (1, 'Ashish', -5000.00, '2024-06-05');

-- Transaction fails → rollback
ROLLBACK;

SELECT * FROM FeePayments;
```

Explanation:

- Second insert violates constraints → entire transaction rolled back.
- Atomicity & consistency preserved: no partial data inserted.

Part C – Partial Failure Simulation

```
START TRANSACTION;

-- First valid insert
INSERT INTO FeePayments(payment_id, student_name, amount,
payment_date)
VALUES (5, 'Riya', 5200.00, '2024-06-06');

-- Second invalid insert (NULL student_name)
INSERT INTO FeePayments(payment_id, student_name, amount,
payment_date)
VALUES (6, NULL, 5000.00, '2024-06-07');

ROLLBACK;
```

```
SELECT * FROM FeePayments;
```

Explanation:

- Even though the first insert was valid, second insert fails → rollback.
- Ensures table remains **consistent**.

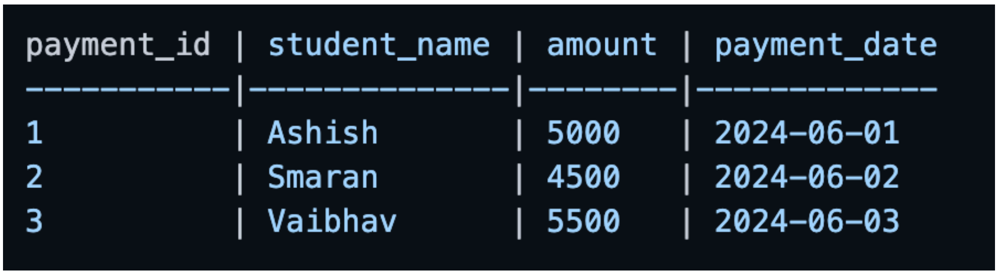
Part D – Verify ACID Compliance

```
-- Transaction showing ACID properties
START TRANSACTION;

-- Valid insert
INSERT INTO FeePayments(payment_id, student_name, amount,
payment_date)
VALUES (7, 'Aman', 6000.00, '2024-06-08');

-- Simulate isolation by using a separate session (optional
in DBMS)
-- Session 2 cannot see uncommitted record yet

-- Commit transaction
COMMIT;

SELECT * FROM FeePayments;
```

Explanation:

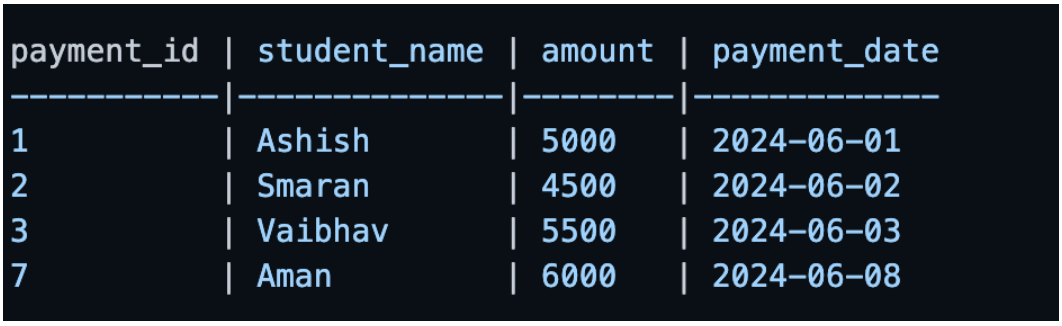
- Atomicity: All-or-nothing transaction
- Consistency: Database rules preserved
- Isolation: Uncommitted changes invisible to other sessions
- Durability: COMMIT ensures permanent storage

OUTPUT:



```
payment_id | student_name | amount | payment_date
-----------|--------------|--------|-------------
1          | Ashish       | 5000   | 2024-06-01
2          | Smaran       | 4500   | 2024-06-02
3          | Vaibhav      | 5500   | 2024-06-03
```

TABLE CREATED SUCCESSFULLY



```
payment_id | student_name | amount | payment_date
-----------|--------------|--------|-------------
1          | Ashish       | 5000   | 2024-06-01
2          | Smaran       | 4500   | 2024-06-02
3          | Vaibhav      | 5500   | 2024-06-03
7          | Aman         | 6000   | 2024-06-08
```

AMAN’S PAYMENT COMMITTED SUCCESSFULLY

Learning Outcomes (LOs)

1. **Understand transactions** in SQL and how they ensure **Atomicity**.
2. **Demonstrate rollback** to maintain database **Consistency** during failures.
3. **Learn Isolation techniques** to prevent interference between concurrent transactions.
4. **Ensure Durability** by committing valid transactions and verifying persistent data.