Deep Dive: Transactions in SQL

1. Why Use Transactions?

Use Case:

- Transfer money

- Update multiple related tables

- Ensure either all steps succeed or none happen

Without transactions, partial updates could leave your database in an inconsistent state.

2. Real-World Scenario: Bank Transfer

START TRANSACTION;

-- Deduct from sender

UPDATE accounts SET balance = balance - 1000 WHERE account_id = 101;

-- Add to receiver

UPDATE accounts SET balance = balance + 1000 WHERE account_id = 202;

-- Log the transfer

INSERT INTO transactions (from_id, to_id, amount) VALUES (101, 202, 1000);

COMMIT;

All three steps are needed. If any fails, we ROLLBACK.

Deep Dive: Transactions in SQL

3. What Happens Without COMMIT?

START TRANSACTION;

UPDATE products SET stock = stock - 1 WHERE id = 5;

-- COMMIT is not called

On disconnect, the change is lost.

Always end with COMMIT or ROLLBACK.

4. Handling Errors with ROLLBACK

START TRANSACTION;

UPDATE accounts SET balance = balance - 1000 WHERE account_id = 1;

-- Simulated error

INSERT INTO trnasactions (from_id, to_id, amount) VALUES (1, 2, 1000);

ROLLBACK;

The previous update is undone because of the error.

5. Using SAVEPOINT and ROLLBACK TO

START TRANSACTION;

```
UPDATE orders SET status = 'processing' WHERE order_id = 1;

SAVEPOINT step1;


UPDATE inventory SET quantity = quantity - 1 WHERE item_id = 5;

SAVEPOINT step2;


-- Error simulation

ROLLBACK TO step2;


COMMIT;
```

Only the second step is rolled back, not the first.

6. Isolation Levels (Conceptual Intro)

Defines how transactions interact when running concurrently.

Common Isolation Levels:

- READ UNCOMMITTED: Can see uncommitted changes (dirty reads)
- READ COMMITTED: Can only read committed data
- REPEATABLE READ: Same query returns same result
- SERIALIZABLE: Full isolation

Set with:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Deep Dive: Transactions in SQL

Practice Exercises

1. Create a transaction that updates both customer and order tables.

2. Use SAVEPOINT to rollback part of a shopping cart checkout.

3. Simulate an error (e.g., product out of stock) and recover with ROLLBACK.