What is a Transaction?

A **transaction** is a sequence of one or more SQL operations executed as a **single unit of work**. It ensures data integrity in situations like transfers, updates, or multi-step changes.

A transaction must be:

- **Atomic** – All steps succeed or none do.
- **Consistent** – Data moves from one valid state to another.
- **Isolated** – One transaction doesn't affect others running at the same time.
- **Durable** – Once committed, the changes persist even after a crash.

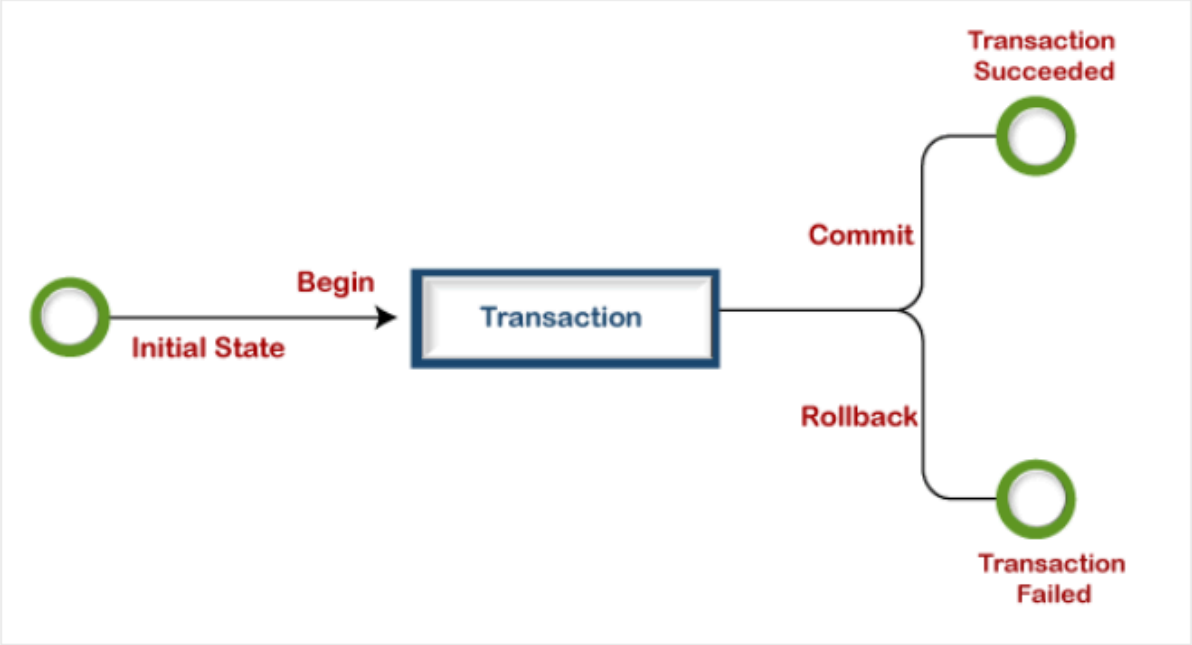
This is known as the **ACID properties**.

How to Implement SQL Transactions

To use SQL transactions, we use commands like `BEGIN`, `COMMIT`, and `ROLLBACK`, so we can manage transactions effectively, group operations together, and handle errors.

Using BEGIN, COMMIT, and ROLLBACK

1. `BEGIN`**:** Marks the start of a transaction. All subsequent operations will be part of this transaction.
2. `COMMIT`: Finalizes the transaction, making all changes permanent in the database.
3. `ROLLBACK`: Undoes all changes made during the transaction, reverting the database to its previous state in case of an error or failure.



Example-1:

```
BEGIN TRANSACTION;
-- Deduct $500 from account A
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;
-- Add $500 to account B
UPDATE accounts SET balance = balance + 500 WHERE account_id = 2;
-- Commit the transaction
COMMIT;
```

If an error occurs, such as insufficient funds, the transaction can be rolled back:

```
BEGIN TRANSACTION;
UPDATE accounts SET balance = balance - 500 WHERE account_id = 1;
-- Check for errors (pseudo-code for demonstration)
-- IF insufficient_balance THEN
ROLLBACK;
-- ELSE Commit the transaction
COMMIT;
```

Example-2

```
BEGIN TRANSACTION;

-- insert new order into orders table
INSERT INTO orders (customer_id, product_id, quantity, status)
VALUES (1, 2, 3, 'Pending');

-- update inventory level for relevant product
UPDATE inventory
SET quantity = quantity - 3
WHERE product_id = 2;

-- check if inventory level is now negative
IF EXISTS (SELECT * FROM inventory WHERE product_id = 2 AND quantity < 0)
BEGIN
 -- if inventory level is negative, rollback transaction
 ROLLBACK TRANSACTION;
 PRINT 'Error: inventory level is negative';
END
ELSE
BEGIN
 -- if inventory level is not negative, commit transaction
 COMMIT TRANSACTION;
 PRINT 'Order successfully placed';
END
```

Challenges:

1. Concurrency - use locks, serializability
2. Deadlock - prevention, avoidance, detection, and removal (recovery)
3. Partial update - use savepoints

Using Savepoints

```
START TRANSACTION;

-- Step 1: Deduct from Alice
UPDATE accounts SET balance = balance - 100 WHERE id = 1;

-- Step 2: Add to Bob
UPDATE accounts SET balance = balance + 100 WHERE id = 2;

-- Save state after Alice & Bob updates
SAVEPOINT transfer_complete;

-- Step 3: Attempt to credit Charlie (say this step fails or we change our mind)
```

```
UPDATE accounts SET balance = balance + 50 WHERE id = 3;

-- Now roll back just that step
ROLLBACK TO SAVEPOINT transfer_complete;

-- Commit the rest
COMMIT;
```