--------*LECTURE- 20*--------

◈ TRANSACTIONS

Transaction → Anything that changes data

Meaning (very important)

A **transaction** is:

A group of SQL statements that the database treats as **ONE single unit of work**.

So the database says:

- Either **everything succeeds**
- Or **nothing happens**

There is **no in-between state**.

Why transactions are needed

Imagine banking 💰
You transfer ₹10,000 from Account A to Account B.

Steps:

1. Deduct ₹10,000 from A
2. Credit ₹10,000 to B

Now imagine:

- Step 1 succeeds
- System crashes before step 2

👉 Money is **lost**

That's why **transactions exist**.

Transaction definition

A transaction is a sequence of SQL statements executed as one logical unit of work

Example:

```
START TRANSACTION;
```

```sql
UPDATE account SET balance = balance - 10000 WHERE acc_id = 1;

UPDATE account SET balance = balance + 10000 WHERE acc_id = 2;


COMMIT;
```

If **both updates succeed** → COMMIT
If **any one fails** → ROLLBACK

COMMIT vs ROLLBACK

COMMIT

- Makes changes **permanent**
- Data is saved to disk
- Cannot be undone

ROLLBACK

- Cancels all changes
- Database goes back to previous state

---

example: 10K

You wrote:

- 1 → Deduct
- 2 → Credit
- If both done → COMMIT
- If none done → ROLLBACK

✔ This is **exactly correct**

🔷 Updating Multiple Tables

Why transactions are REQUIRED here

Example:

```sql
UPDATE orders SET status='PAID';
UPDATE inventory SET stock = stock - 1;
```

If:

- Order updated
- Inventory update fails

👉 Data becomes **inconsistent**

Transaction ensures:

- Either both tables update
- Or neither updates

🔹 Handling Banking, Orders, Inventory

All these systems:

- Are **multi-user**
- Have **high concurrency**
- Cannot afford wrong data

That's why:

Transactions are the backbone of real-world databases

🔹 Preventing Partial Updates

Partial update = DANGEROUS ❌

Partial update means:

- Some rows updated
- Some rows not updated

Transactions **prevent this completely**.

🔹 ACID Properties (VERY IMPORTANT)

ACID explains the **nature of transactions**.

A → Atomicity (ALL or NOTHING)

Meaning:

- Either full transaction happens
- Or nothing happens

Example:

- Deduct + Credit
- If one fails → rollback everything

C → Consistency (Rules always followed)

Meaning:

- Database moves from **one valid state** to **another valid state**
- Constraints must hold

Example:

- Balance cannot go negative
- Foreign key must exist

If rule breaks → transaction fails

I → Isolation (transactions don't disturb each other)

Imagine:

- 100 users booking tickets at same time

Isolation ensures:

- One transaction does not see **half-completed** data of another

Each transaction behaves like it is **running alone**.

D → Durability (data survives crash)

Meaning:

- Once committed
- Even power failure cannot erase it

This is achieved using:
👉 **WAL (Write Ahead Logging)**

🔷 START TRANSACTION block (your syntax)

```
START TRANSACTION;

UPDATE ...
UPDATE ...

COMMIT;
-- or
ROLLBACK;
```

Between START and COMMIT:

- Changes are temporary
- Only visible to your session

🔷 Deadlock

What is a deadlock?

Deadlock happens when:

- Transaction A waits for B
- Transaction B waits for A
- Both are stuck forever

Simple example 🧠

Transaction 1:

- Locks Row A
- Wants Row B

Transaction 2:

- Locks Row B
- Wants Row A

👉 Neither can proceed
👉 DEADLOCK

Why deadlocks happen

- Multiple users
- Row-level locking
- Poor transaction order

How database handles deadlock

Database:

- Detects deadlock
- Kills one transaction
- Rolls it back
- Other continues

This is normal behavior.

◈ Thread Synchronization

Database internally:

- Manages multiple threads/processes
- Uses locks to synchronize access
- Prevents corruption

You don’t code this — DB engine handles it.

◈ Row-level locking (very important)

What is row-level locking?

When a transaction updates a row:

- That row is **locked**
- Other transactions must wait

Example:

```
UPDATE account SET balance = 500 WHERE acc_id = 1;
```

Until commit/rollback:

- No one else can modify acc_id = 1

Why row-level locking is good

- High concurrency
- Other rows still accessible
- Faster than table lock

🔶 “1 million transactions in 1 sec” (what it implies)

This means:

- Database can handle massive concurrency
- Locking is efficient
- Transactions are optimized

Modern databases are built for this.

🧠 FINAL BIG PICTURE (

- Transaction = safety boundary
- COMMIT = save
- ROLLBACK = undo
- ACID = rules database follows
- Locks = prevent conflicts
- Deadlock = unavoidable but manageable

one-liner

Transactions ensure data consistency and reliability by executing multiple SQL statements as a single unit of work following ACID properties.