Newton School of Technology

Transactions & ACID Properties

Recap

Newton School of Technology

- Concept of Database Normalisation
- Relationships between Tables (1:1, 1:M, M:N)
- Unnormalized Form (UNF)
- First Normal Form (1NF)
- Second Normal Form (2NF)
- Third Normal Form (3NF)
- Boyce-Codd Normal Form (BCNF)

Agenda

Newton School of Technology

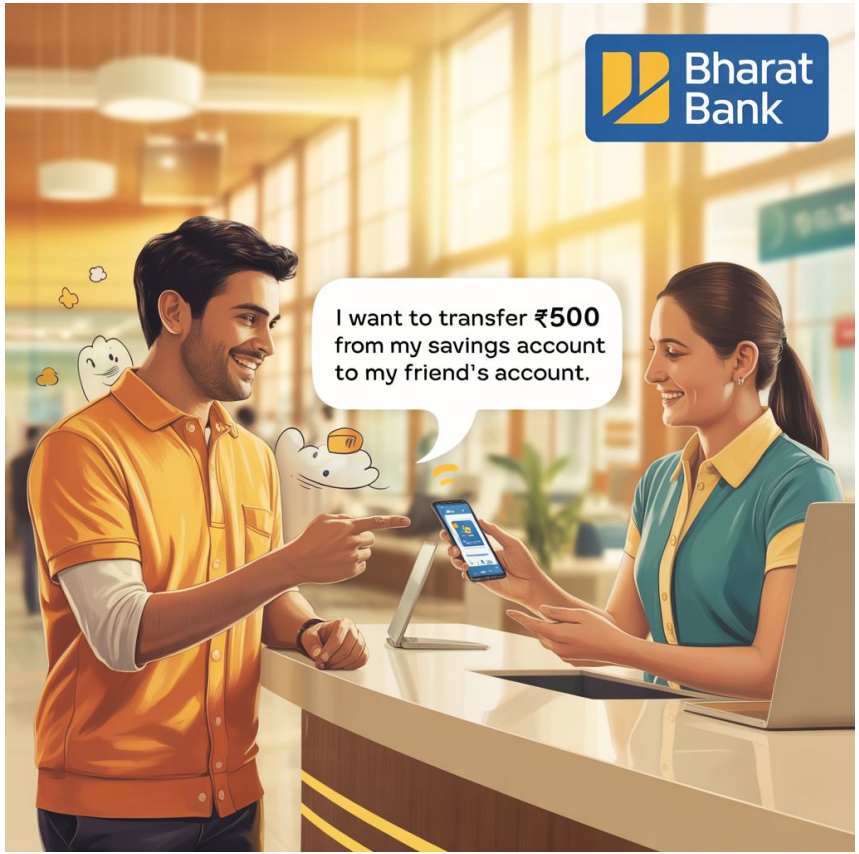
- **Fundamentals of Transactions**
  - Core concepts and ACID properties
- **Concurrency Challenges**
  - Understanding the problems that arise with multiple transactions
- **Isolation Levels in Detail**
  - From READ UNCOMMITTED to SERIALIZABLE
- **Hands-on Demonstrations**
  - Real-world examples and practical implementations
  - Best Practices & Performance Considerations
  - Making the right isolation level choices for your applications

Objective: Learn how to keep database transactions reliable and efficient.

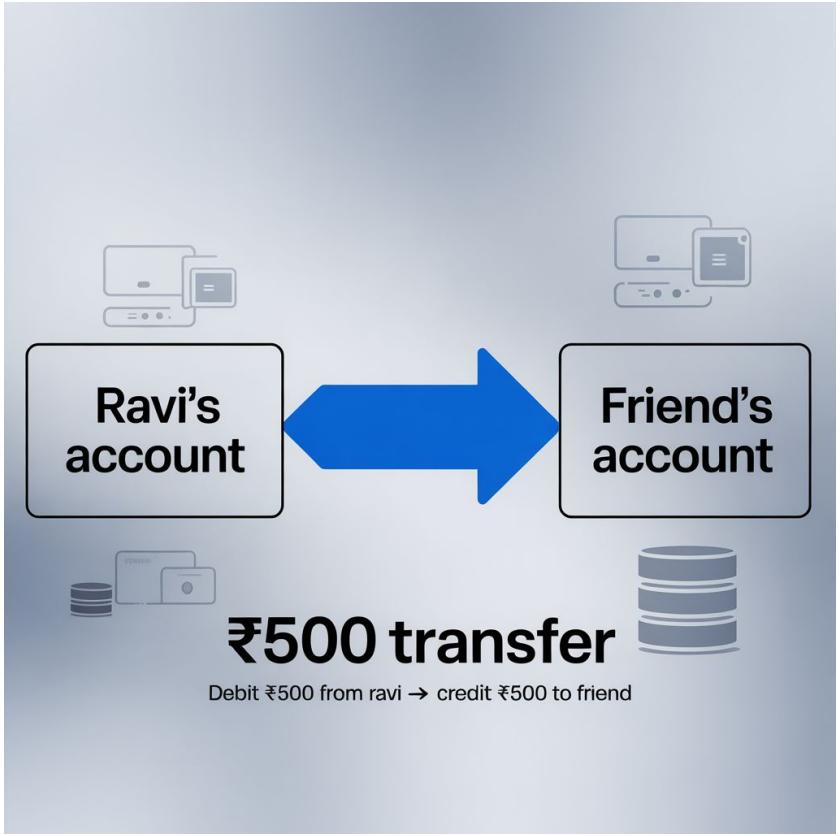
Fundamentals of Transactions

Newton School of Technology

- Ravi walks into his bank to send ₹500 to his friend.
- For him, it's just one click — a simple transfer.
- But for the bank, it's a little more than that.



Understanding What Happens Behind the Scenes



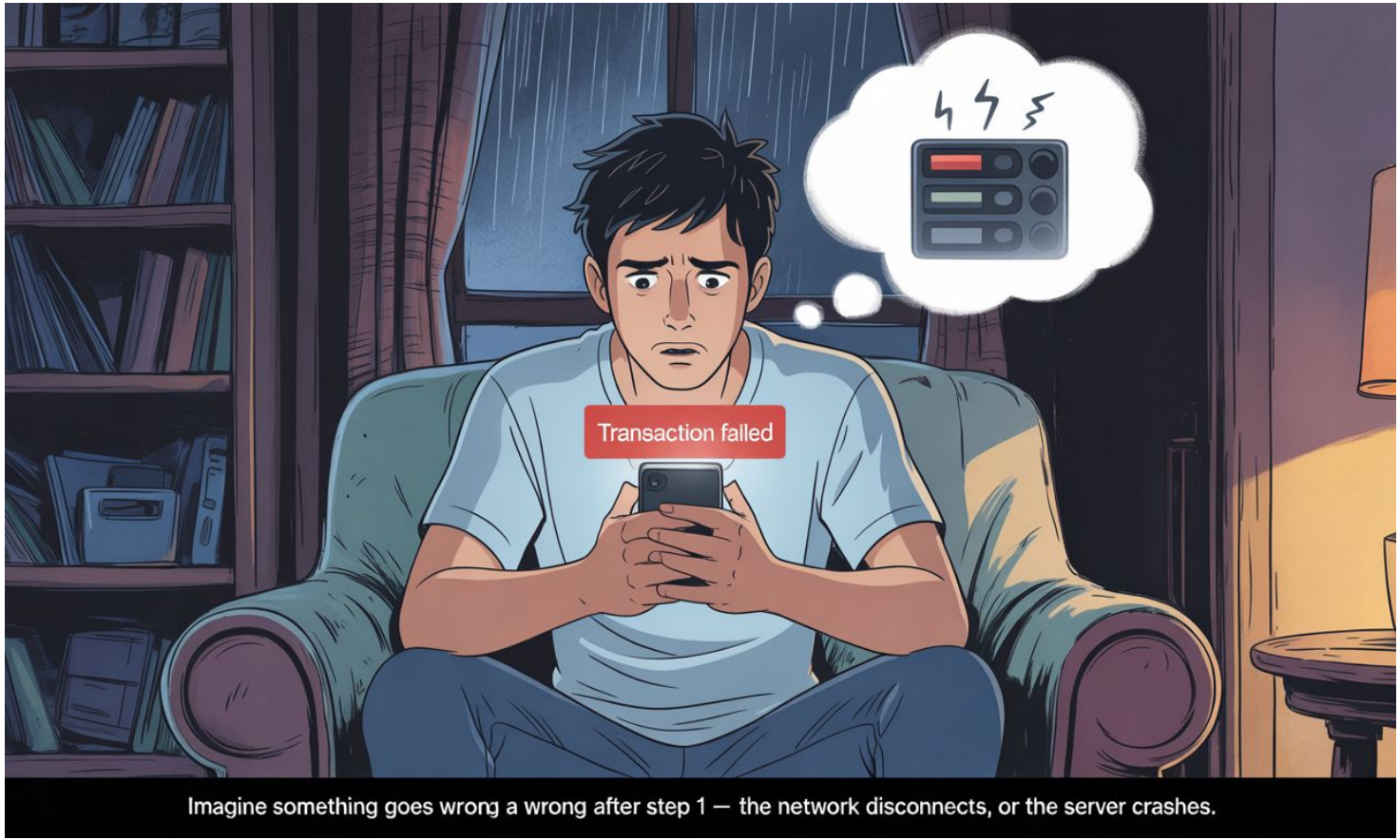
The bank must do two things:

1. Take ₹500 from Ravi's account.
2. Add ₹500 to his friend's account.

Both must happen together — not one without the other.

The Problem (What If Something Goes Wrong?)

Newton School of Technology



The Bank’s Secret Safety Net (The Concept of Transaction)

Newton School of Technology

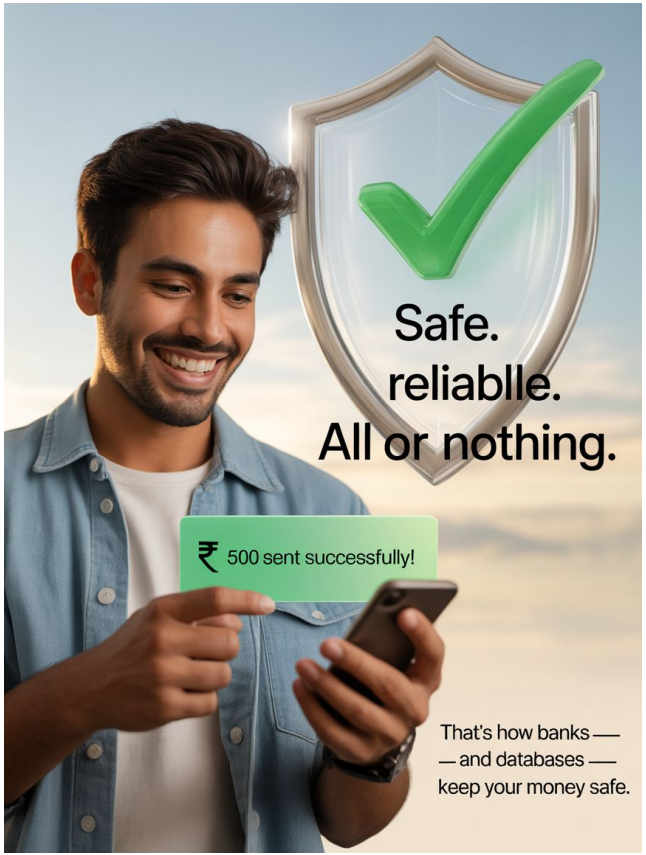
If Something Fails (Rollback in Real Life Words)

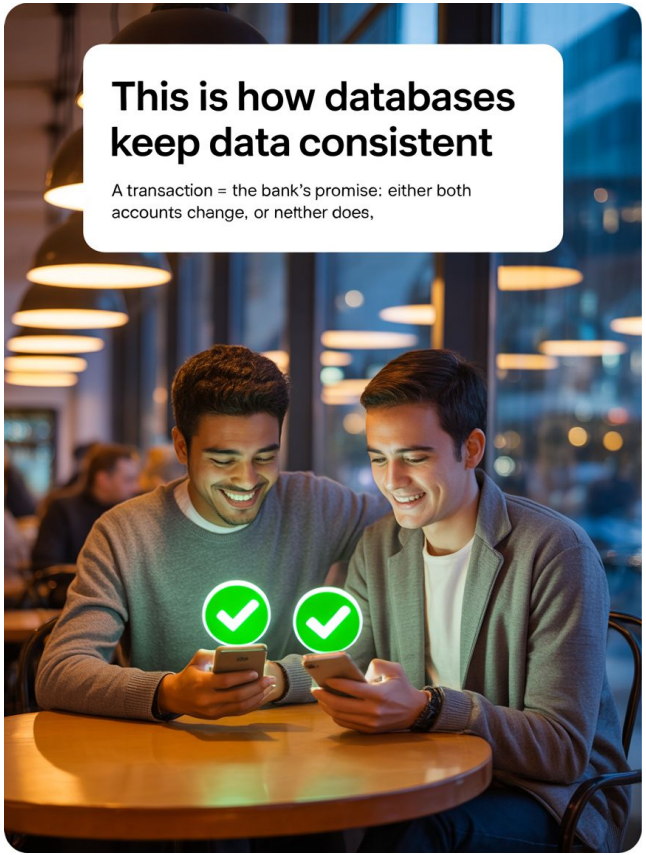
Newton School of Technology

If something goes wrong, the bank cancels the whole process — like it never happened.

The Promise (All or Nothing)

Newton School of Technology

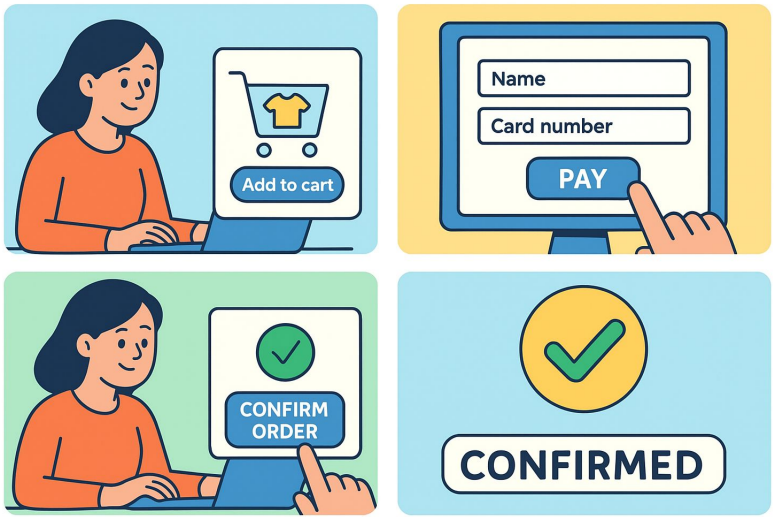




Database Transactions

Newton School of Technology

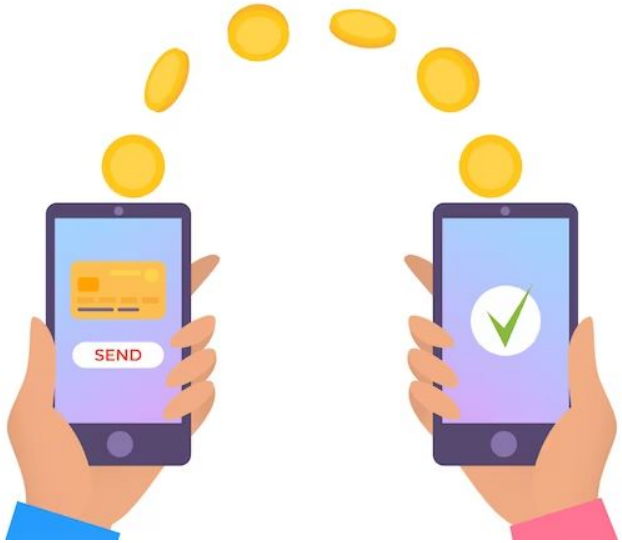
A transaction in a database is a set of operations that are executed as a single, indivisible unit.



A database transaction is like completing an online purchase:

- Adding items to your cart
- Entering payment details
- Confirming your order
- Receiving a confirmation

Importance of Transactions



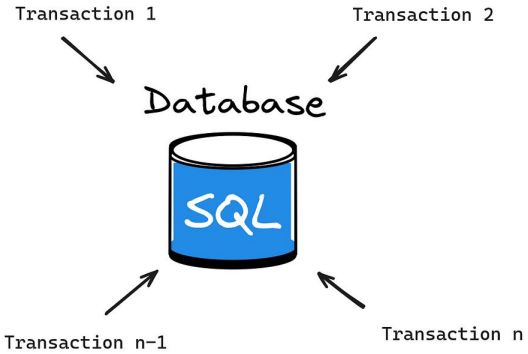
Bank amount transfer steps:

- Withdrawing from Account A
- Depositing to Account B

– Need to happen together, or the transfer isn't valid.

If only one step completes, then money is lost!

Database Transactions

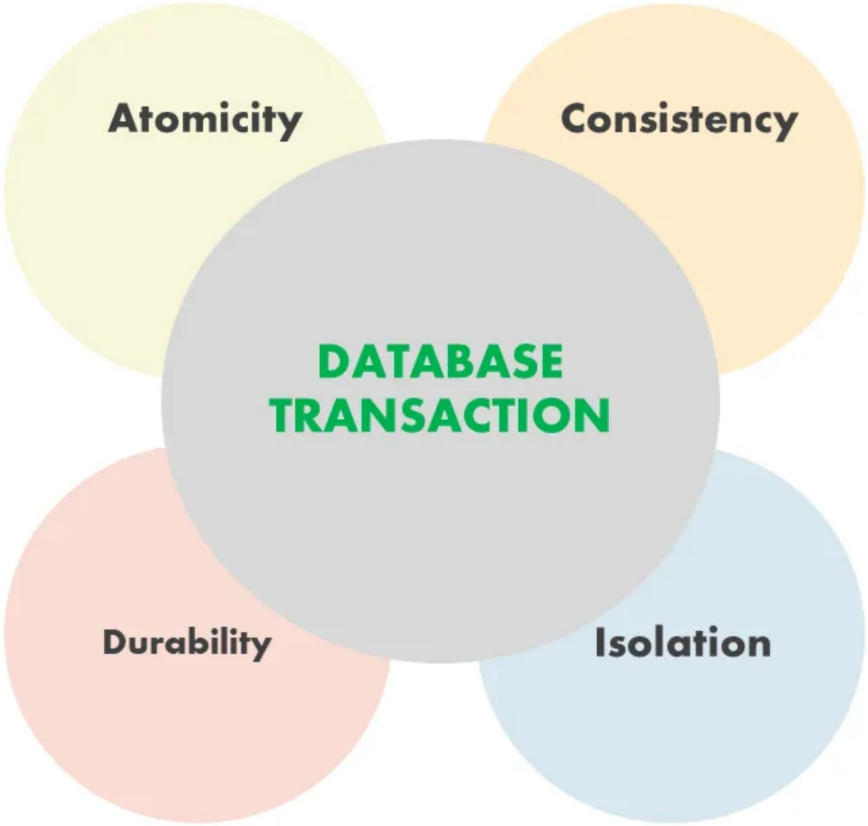


- What happens if you don't pay after adding items to the cart?
- What if the payment is processed, but the items are not shipped?
- How should the system behave if it crashes after payment but before confirming the order?

To ensure that each scenario is handled safely—that is, the transaction either completes in full or leaves the system unchanged—we must guarantee that every business operation obeys the four ACID properties

Properties of Database Transaction

Newton School of Technology



ACID Properties of Transactions

Atomicity

Transactions are all-or-nothing operations. Either all operations succeed, or none do. This prevents partial updates that could leave data in an inconsistent state.

Consistency

A transaction brings the database from one valid state to another valid state, enforcing all defined rules, constraints, and triggers.

Isolation

Concurrent transactions execute as if they were running sequentially, preventing interference between them. This is achieved through isolation levels.

Durability

Once a transaction is committed, its changes persist even in the event of system failures through techniques like write-ahead logging.

Basic Transaction Commands in MySQL

```
-- Start a new transaction
START TRANSACTION;
-- or
BEGIN;

-- Make your changes
UPDATE accounts SET balance = balance - 200 WHERE id = 1;

UPDATE accounts SET balance = balance + 200 WHERE id = 2;

-- Save all changes permanently
COMMIT;

-- Or discard all changes
ROLLBACK;
```

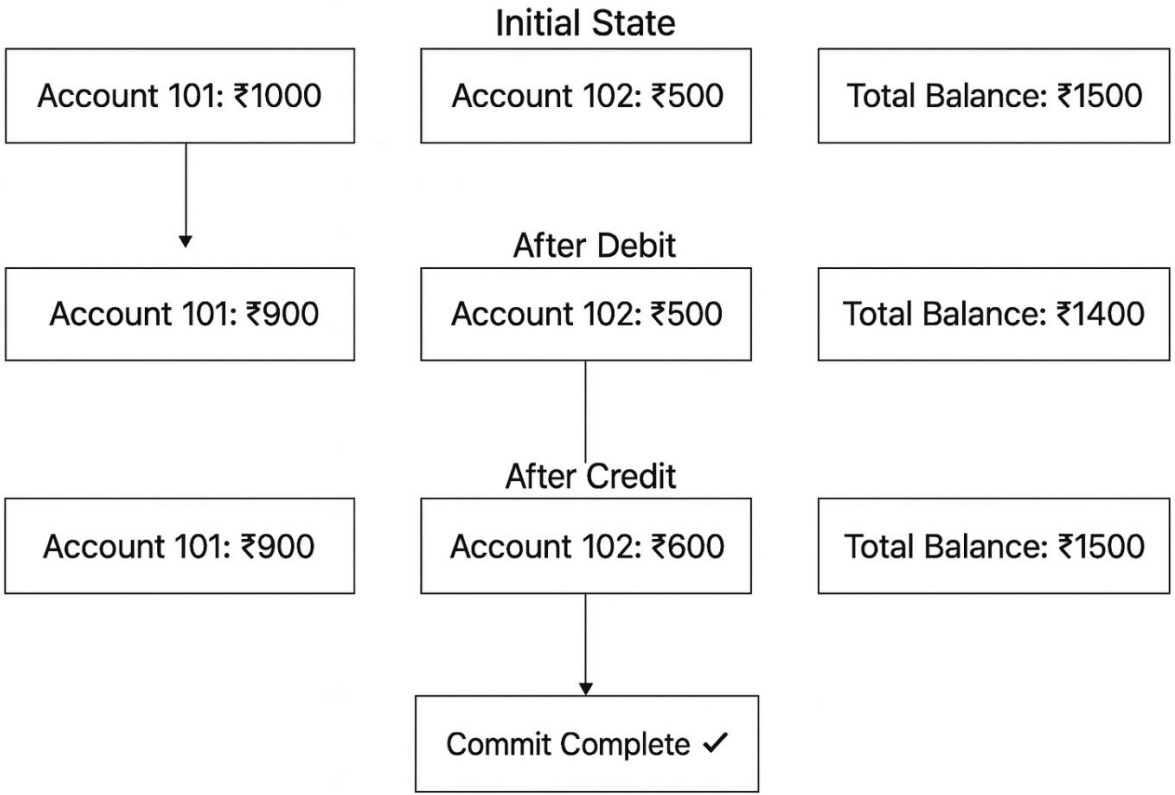
Key Points:

- Transactions must be explicitly started in MySQL
- Changes are temporary until COMMIT is executed
- ROLLBACK discards all changes since transaction start
- Auto-commit is enabled by default (each statement is its own transaction)
- SET autocommit=0; disables auto-commit mode

Example of Transactions

Newton School of Technology

How do I transfer Rs100 from account 101 to 102?



Atomicity

Newton School of Technology

All operations within a transaction either complete successfully or none at all.

- If any operation fails, the entire transaction is rolled back.
- No partial updates should be persisted.

Example:

```
BEGIN;
  UPDATE accounts SET balance = balance - 100 WHERE id = 101;
  -- System crash
  UPDATE accounts SET balance = balance + 100 WHERE id = 102;
ROLLBACK;
```

- Only deducting money but failing to credit violates atomicity.
- Database systems ensure rollback in such cases.

Example of Transactions

Newton School of Technology

How do I transfer Rs100 from acct 101 to 102?

```
BEGIN;
  UPDATE accounts SET balance = balance - 100 WHERE id = 101;
  UPDATE accounts SET balance = balance + 100 WHERE id = 102;
COMMIT;
```

Consistency

Newton School of Technology

Transactions must transition the database from one valid state to another, maintaining all integrity constraints.

- All integrity constraints (PK, FK, CHECK, UNIQUE) must remain valid before and after the transaction.
- Business rules must be satisfied (e.g., no negative balances).

Example:

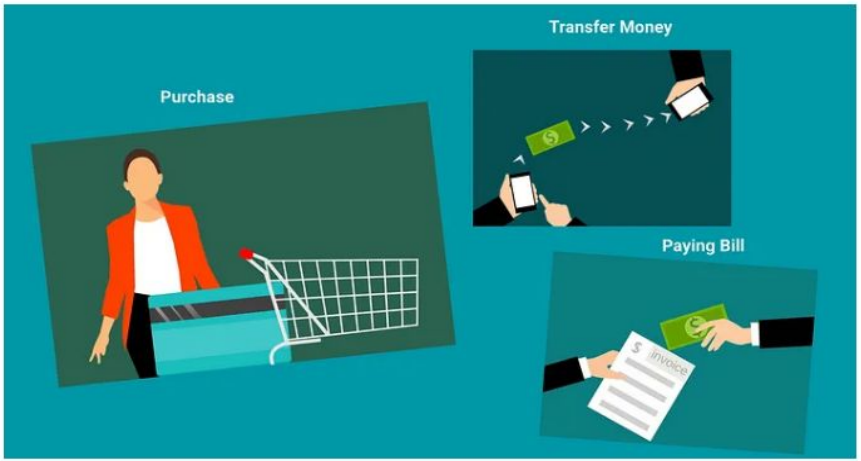
```
-- Starting balances: A=500, B=500
BEGIN;
  UPDATE accounts SET balance = balance - 100 WHERE id = 101;
  -- crash before crediting 102
ROLLBACK;
```

- Total balance dropped to 900 momentarily, which violates consistency.
- Only completing both sides of transfer restores consistent state.

Isolation

Newton School of Technology

- Ensures transactions are executed independently & don’t interfere with each other.
- Even when multiple transactions occur simultaneously.



Isolation

Newton School of Technology

Isolation means that each transaction should execute as if it's the only one in the system.

Bunty

I want to buy a sandwich (₹80)

Shunty

I want to buy a coffee (₹50)

Isolation

Both rush to place their order **at the same time**.

Isolation

Newton School of Technology

Time	Action	Wallet Balance (Observed)
t1	Bunty checks the wallet → sees ₹100	T1: R(A) = 100
t2	Shunty also checks the wallet → sees ₹100	T2: R(A) = 100
t3	Shunty deducts ₹50 → balance becomes ₹50	T2: W(A) = 50
t4	Bunty deducts ₹80 → balance becomes ₹20 ❌	T1: W(A) = 20
t5	Final Balance: ₹20 ❌ (Incorrect)	

Problem:

Shunty paid ₹50 for coffee, but the wallet says ₹20.
His ₹50 is lost. Bunty **overwrote** it because both didn't know the other was spending.

The Problem

- Shunty already paid ₹50 for his coffee.
- But the wallet now shows only ₹20 left!
- His ₹50 is **lost** because Bunty's transaction **overwrote** the balance.

This happened because both transactions ran **at the same time** — the system didn't make them wait for each other.

The database become **inconsistent** — it no longer shows the correct wallet amount.

Isolation

Newton School of Technology

If both transactions happen **without isolation**,
the system becomes inconsistent

Database must serialized
scheduling to ensure correctness

If the system enforced **Isolation**:

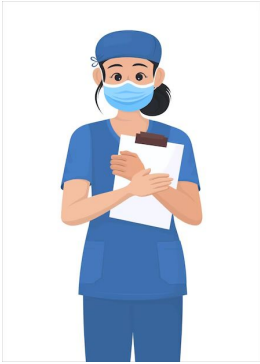
- After Bunty reads ₹100, the system **locks** it until he finishes the deduction.
- Shunty must **wait** until Bunty finishes → No overwriting, no confusion.

Isolation

Newton School of Technology

T1 (Doctor): Updates patient report to "Critical" (not yet committed).

T2 (Nurse): Reads that status and calls emergency staff.



T1: Realizes it was an error and rolls back.

This is a **Dirty Read:** Action taken based on an update that never existed officially.

Isolation

Newton School of Technology

1. Imagine Bunty and Shunty are ordering lunch on Zomato at the same time.
2. At **12:00 PM**, Bunty opens the app and sees:
   - 🍕 Pizza: ₹250
   - 🍔 Burger: ₹150

3. At **12:02 PM**, the restaurant updates prices:
   - 🍕 Pizza: ₹300
   - 🍔 Burger: ₹120

4. Shunty opens the app **after** the price change, so he sees the new prices.
5. But Bunty **still sees the old prices** in his snapshot!

That's MVCC — **everyone sees the version of the data from when they started**.

Should Bunty be affected by the new price if he *started earlier*?

Isolation

Newton School of Technology

But Wait — If Each Transaction Runs "Alone," How Can They Wait for Each Other?

When we say *"each transaction should act as if it's the only one running,"* we don't mean that no other transactions exist.

It means that **even if multiple transactions happen at the same time**, the **final result** should be the same as if they ran **one by one**.

Here's how the database ensures that:

- The system allows multiple transactions to start, but it **locks the shared data** (like the wallet) so others can't change it at the same time.
- Other transactions can still run, but they must **wait** when they try to touch the same data.

Think of it like a **canteen counter**:

- Bunty takes the wallet first → Shunty waits behind him.
- Once Bunty pays and returns the wallet, Shunty can use it next.

Both are active, but only **one can use the wallet** at a time — so everything stays correct and consistent.

Durability

Newton School of Technology

Once a transaction commits, its effects are permanent.

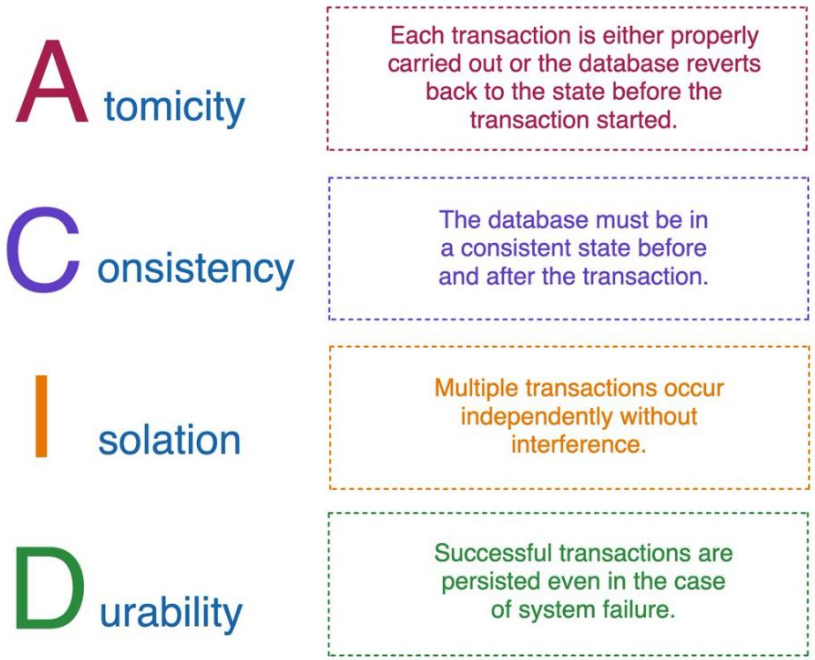
- Committed data must be stored persistently (e.g., logs, disk writes).
- Even if system crashes, committed state must be recoverable.

Example:

```
BEGIN;
  UPDATE orders SET status = 'confirmed' WHERE id = 1001;
COMMIT;
-- System crashes
-- On restart, status must remain 'confirmed'
```

- Ensures no committed data is lost.

Summary



Why ACID properties?

- **Data Integrity:** Keeps data correct across all actions.
- **Reliability:** Maintains stability even with errors.
- **Isolation and Concurrency:** Supports multiple users handling transactions at the same time.

Why Schedule Transactions?

Newton School of Technology

In a database, multiple transactions may run concurrently.

A schedule shows the order of read (R), write (W), and commit (C) operations from each transaction.

Think of it like a timeline of actions—who did what and when.

Like managing queues at an amusement park

Transactions Scheduling

Newton School of Technology

Example:

👨‍💼 T1: John transfers ₹100 from A to B

👩‍💼 T2: Priya deducts ₹10 as a processing fee from A

Imagine both operations happening around the same time.

Who writes first?

Who commits first?

The order matters. That’s where scheduling becomes important.

Schedule Representation

Newton School of Technology

Time	Transaction	Operation	Description
t1	T1	W1	Deduct ₹100 from A (balance - 100)
t2	T2	W1	Deduct ₹10 from A (balance - 10)
t3	T1	W2	Add ₹100 to B (balance + 100)
t4	T2	C	Commit processing fee deduction
t5	T1	C	Commit transfer operation

Transactions Scheduling

Newton School of Technology

-- T1: Transfer ₹100 from A to B

```
BEGIN;
UPDATE accounts SET balance = balance - 100 WHERE id = 1;
```

T1.w1

-- Interleaved here
-- T2: Deduct ₹10 fee from A

```
BEGIN;
UPDATE accounts SET balance = balance - 10 WHERE id = 1;
COMMIT;
```

T2.w1

T2.C

-- Back to T1

```
UPDATE accounts SET balance = balance + 100 WHERE id = 2;
COMMIT;
```

T1.w2

T1.C

Transactions Scheduling

Newton School of Technology

Does This Schedule Cause Problems?

- What if both T1 and T2 write to A's balance without knowing each other's updates?
- Could one update be lost?
- What's the final balance in A and B?

This naturally leads into the concepts of **serializability**, **conflicts**, and **concurrency control**

Serial vs Non-Serial Schedule

Newton School of Technology

Serial Schedule Timeline

T1: W(A) — W(B) — COMMIT
T2: R(A) — COMMIT

No interference. Fully isolated. Safe.

Non-Serial Schedule Timeline

T1: W(A)
T2: R(A)
T2: COMMIT
T1: W(B)
T1: COMMIT

Can this cause inconsistency?

🛤️ Serial: Like one train on a single-track tunnel
🚦 Non-Serial: Two trains on the same track—needs coordination!

Serial Scheduling

Newton School of Technology

T1

Alex withdraws Rs.100 from his account with balance of Rs.500

T2

The bank credits a Rs.50 bonus to Alex’s account.

Expected Serial Result:

- If T1 happens first, the balance is Rs.400, then T2 makes it Rs.450.
- If T2 happens first, the balance is Rs.550, then T1 makes it Rs.450.

In both cases, the final balance should be Rs.450.

Non Serial Scheduling

Newton School of Technology

T1

Alex withdraws Rs.100 from his account with balance of Rs.500

T2

The bank credits a Rs.50 bonus to Alex's account.

Problem with Concurrent Execution:

- T1 and T2 run at the same time without order.
- T1 reads the balance as Rs.500 and withdraws Rs.100, thinking the balance will be Rs.400.
- T2 also reads the balance as Rs.500 and adds Rs.50, thinking the balance will be Rs.550.

Final Result: The account ends up showing Rs.500 (or another incorrect value) instead of Rs.450.

Serial vs Non-Serial Schedule

Newton School of Technology

Feature	Serial Schedule	Non-Serial Schedule
Execution Order	One transaction at a time (no mixing)	Interleaved operations from multiple transactions
Example	T1.W1 → T1.W2 → T1.C → T2.W1 → T2.C (Only after T1 is done, T2 begins)	T1.W1 → T2.W1 → T2.C → T1.W2 → T1.C (Both operations from T1 and T2 are interleaved)
Performance	Safe but slow (no concurrency)	Fast but risky (requires careful control)
Risk	No risk	May cause dirty reads, lost updates, etc.

Conflict Serializability

Newton School of Technology

A **non-serial schedule** is **conflict-serializable** if it results in the same outcome as **some serial schedule**.

> If two transactions access the same data, and one of them writes, **the order must preserve dependencies.**

T1: Transfer ₹100 from A to B
T2: Read A's balance

S1: T1.W(A) → T1.W(B) → T1.C → T2.R(A) → T2.C

Effectively the same as running T1 first, then T2.

Not Conflict Serializability

Newton School of Technology

A schedule is not conflict-serializable if it cannot be rearranged into any serial schedule without violating operation order dependencies.

If two transactions access the same data, and one of them writes, the **order of conflicting operations must not be changed**.

T1: Transfer ₹100 from A to B

T2: Read A's balance

S2: T2.R(A) → T1.W(A) → T2.C → T1.W(B) → T1.C

📌 **Problem:**
T2 reads the balance of A **before** T1 updates it.
This means T2 sees an outdated value —May result in inconsistent reads.

Summary Description

Newton School of Technology

Concept	Description
Schedule	Timeline of interleaved operations from multiple transactions
Serial Schedule	One transaction finishes completely before the next starts
Non-Serial	Operations are mixed to improve concurrency
Conflict-Serializable	Interleaved, but safe—final state matches a serial schedule

Newton School of Technology

Any Questions

Summary

Newton School of Technology

- Transactions = logical units of work.
- ACID = reliability pillars.
- Isolation affects concurrency.
- Scheduling determines correctness when multiple users operate at once.

Newton School of Technology

Please fill the feedback form.

Reference

Newton School of Technology

- PostgreSQL Docs: Transactions
- MySQL Docs: Isolation Levels
- Stanford: Transaction Scheduling (Slides)

Newton School of Technology

Thanks for watching!