[ACID Transactions]

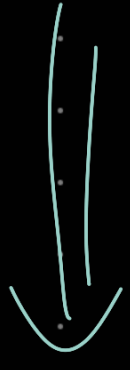
↳ We already learnt ACID in database

↳ Taken care by Transaction Manager + Recovery Manager + Concurrency Manager + Application programmer

* How DBMS ensure (Atomicity)?

↳ Log Based Recovery

↳ checkpoint based Recovery



Databases use various mechanisms to guarantee atomicity. One common method is **logging**—a transaction log (also known as a write-ahead log, or WAL) keeps track of all changes during a transaction.

If a transaction fails halfway through, the database consults this log and rolls back any incomplete operations.

Steps behind the scenes:

1. **Begin Transaction**: The system writes a "begin" entry to the log.
2. **Execute Operations**: Changes are made in-memory (debit your account, credit your friend's account).
3. **Write to Log**: The transaction writes all changes to the log, ensuring they can be rolled back if needed.
4. **Commit**: Once everything is successful, a "commit" entry is written to the log, making the changes permanent.
5. **Rollback if Error**: If there's an error, the system uses the log to undo any changes made.

* How Consistency in ensured?

Consistency in Distributed Systems

In a single database, consistency is often enforced through constraints and rules that are relatively simple to implement.

However, in distributed systems (where data is spread across multiple databases or regions), ensuring consistency can become more complex.

Strong Consistency vs. Eventual Consistency

In distributed databases like Amazon DynamoDB or Google Spanner, consistency can be more nuanced, especially when it comes to **strong consistency** vs. **eventual consistency**.

1. **Strong Consistency**: This guarantees that after a transaction, all users see the same data immediately. This is similar to traditional databases, where consistency is tightly enforced.
2. **Eventual Consistency**: In systems that prioritize availability and performance (e.g., distributed databases), data might take time to synchronize across multiple regions. Eventually, all nodes will be consistent, but in the short term, different nodes might see different versions of the data.

* How Isolation is ensured?

Isolation Levels:

Isolation comes with different **levels**, each providing a different balance between **data integrity** and **performance**.

Higher isolation levels provide stronger data consistency but can reduce system performance by increasing the wait times for transactions.

Let's explore the four common isolation levels:

- **Read Uncommitted:** The lowest level, where transactions can see uncommitted changes made by other transactions.
- **Read Committed:** Transactions only see committed changes made by other transactions, avoiding dirty reads.
- **Repeatable Read:** Ensures that if a transaction reads a value, it will see the same value throughout the transaction, preventing non-repeatable reads.
- **Serializable:** The highest isolation level, ensuring complete isolation by serializing all transactions. → SLOW

Isolation levels vs read phenomena [edit]

Isolation level	Dirty reads	Lost updates	Non-repeatable reads	Phantoms
Read Uncommitted	may occur	may occur	may occur	may occur
Read Committed	don't occur	may occur	may occur	may occur
Repeatable Read	don't occur	don't occur	don't occur	may occur
Serializable	don't occur	don't occur	don't occur	don't occur

→ This is good but locking increases transaction execution time.

* How Durability is ensured.
  - Making Recovery with Logs and checkpoint
  - Replicas of DBs