🔒 ACID Transactions Explained

The Pillar of Reliable Databases

When designing robust systems, one of the most critical topics you'll encounter is **ACID transactions**. If you've heard the term thrown around but never fully unpacked it — this page is for you. Let's break down what ACID means, why it matters, and how databases make it work.

🚦 What Are ACID Transactions?

ACID is an acronym that defines four key properties that ensure reliable processing of database transactions. These properties are abstractions that databases provide to guarantee that your data stays accurate, consistent, and safe, even when multiple users are reading and writing at the same time (a common scenario as databases are multi-threaded servers) — or when systems fail unexpectedly.

1️⃣ Atomicity

"All or nothing."

Atomicity means a transaction's operations are treated as a single, indivisible unit. Either every operation in the transaction succeeds, or none do.

Example: Transferring $10

- Your balance must decrease by $10.
- Your friend's balance must increase by $10.

If only one part succeeds, money would be created or destroyed. Atomicity guarantees this can't happen by rolling back the entire transaction if any part fails.

2️⃣ Consistency

"The database stays in a valid state."

Consistency ensures that a transaction brings the database from one valid state to another, always following defined rules, constraints, and invariants. It ensures the database doesn't fail partway through a write and end up in a corrupted, absurd state.

Example 1: Account Balance Rule

A bank rule states an account balance cannot be negative. A transaction attempting to withdraw $200 from an account with $100 will be blocked, preserving the database's consistency.

Example 2: Application Invariant

An application requires an on-duty security guard at all times. A transaction to swap guards (e.g., remove Larry, add Oscar) must be atomic. If it fails midway, consistency ensures the database rolls back to a state where there is still a guard on duty.

3️⃣ Isolation

"Transactions happen independently."

Isolation means that even when transactions are executed concurrently, they appear to run sequentially. It prevents race conditions where two operations interfere with each other and corrupt data. The strongest level of isolation is called **Serializability**.

Example: Concurrent Deposits

Two users deposit $100 into the same account (initial balance: $500). Without isolation, both might read $500, add $100, and write back $600, losing one deposit. With isolation, one transaction will complete first, and the second will see the updated balance, resulting in a correct final balance of $700.

4️⃣ Durability

"Committed data stays committed."

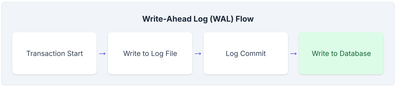
Durability guarantees that once a transaction is committed, it stays committed — even if the system crashes moments later. The data is now permanent, assuming the storage hardware (e.g., hard drive) remains functional.

Example: Power Failure

You transfer money and get a confirmation. Moments later, the server loses power. Durability ensures your transaction isn't lost. When the system reboots, the data reflects the completed transfer, often by replaying a transaction log. In-memory databases may require special replication or logging to ensure this property.

🗂️ How Databases Implement ACID

One popular tool to achieve atomicity, consistency, and durability is the **Write-Ahead Log (WAL)**. Before any changes are made to the database itself, the intended operations are first written to a persistent log file. Only after the log is successfully updated is the transaction considered "committed."



Example WAL Entry:

```
BEGIN_TRANSACTION ID=12;
UPDATE ACCOUNTS SET balance=balance-10 WHERE user='Jordan';
UPDATE ACCOUNTS SET balance=balance+10 WHERE user='Riley';
COMMIT ID=12;
```

If the system crashes before the 'COMMIT' record is written, the database knows to ignore the partial transaction upon recovery. If it crashes after, it can use the log to replay the transaction. Isolation, meanwhile, is often handled using separate, complex mechanisms like locks or multi-version concurrency control (MVCC) — a topic worth its own deep dive!

⚡ Why Doesn't Every Database Use Full ACID?

While ACID is the gold standard for correctness, guaranteeing it fully—especially strict serializable isolation—can come at a performance cost. The strict controls can slow down operations, creating a bottleneck in massive, distributed systems.

That's why some databases, particularly in the **NoSQL** world, relax certain properties for speed or scalability. For example, they might use **eventual consistency** instead of strict consistency or allow weaker isolation levels for higher throughput. The trade-off always depends on your system's needs: absolute correctness for financial data, or massive scalability for a social feed where temporary inconsistencies are acceptable?

📚 Wrapping Up & Detailed Summary

🧪 What are ACID transactions?

Effectively, ACID transactions are a way of writing to certain databases. The database provides abstractions that let us guarantee certain properties will always hold true. Those properties follow the acronym: ACID.

🔹 A — Atomicity

"All or nothing." Imagine you're buying Karnov's bath hat for $10: you lose $10, and Karnov gains $10. If only one of those writes succeeds, money vanishes from the system or is magically created. Atomicity ensures both updates either both succeed, or both fail. No in-between.

🔹 C — Consistency

Consistency means that after every transaction, the database moves from one valid state to another. When databases come back up after a crash, they shouldn't be in an absurd or corrupted state. For example, if a transaction to swap security guards fails midway, consistency ensures the database rolls back so we aren't left with no guard on duty.

🔹 I — Isolation

Isolation ensures that transactions appear to run independently, even if they're happening at the same time. This prevents race conditions. If two people add $1 to a counter at the same time, isolation ensures the final result is $2, not $1. This is powerful but comes with a performance cost.

🔹 D — Durability

Durability guarantees that once a transaction is committed, the data stays there — even if the system crashes right afterward. As long as the hardware is working, your data remains. In-memory databases need extra work (like writing to disk) to meet this principle.

⚙️ How databases achieve ACID: Write-Ahead Log

A classic technique for Atomicity, Consistency, and Durability is a Write-Ahead Log (WAL). The database writes what it plans to do into the log on disk, and only after writing a "commit" marker are the changes applied. If the system crashes, it replays the log, applying only transactions that were fully committed.

Example WAL entry:

```
Transaction #12:
- Riley Reid gets $10 for foot pictures
- Jordan gives her $10 for foot pictures
- Commit
```

If the crash happens before writing 'Commit', the transaction is ignored. If after, it's safely applied.

📌 Final Summary

- **Atomicity:** All or nothing.
- **Consistency:** Move from one valid state to another.
- **Isolation:** Transactions don't step on each other's toes.
- **Durability:** Committed data stays, even after crashes.

Together, these make databases reliable and trustworthy.