Acid: Features of Database

First of all, let's see what database transactions are.

- Database transactions

A database transaction is a unit of work against a database. Generally, a transaction represents any change in a database. Sometimes transactions are a single unit of work like adding a new record or made up of multiple operations.

ACID is a set of properties of database transactions intended to guarantee data validity despite errors, power failures.

- A: Atomicity

A property where operations either happen completely or don't happen at all.

Imagine that our database contains bank accounts as shown below

name	balance
Vec	100$
Cleopatra	100$

Now if Vec decides to transfer 10$ to Cleopatra, we will have to perform 2 steps:

1 Decrease Vec's balance by 10 dollars →

name	balance
Vec	90$
Cleopatra	100$

2 Increase Cleo's balance by 10 dollars →

name	balance
Vec	90$
Cleopatra	110$

But what happens if the system crashed during the second step?

1 Decrease Vec's balance by 10 dollars →

name	balance
Vec	90$
Cleopatra	100$

2 Increase Cleo's balance by 10 dollars →

SYSTEM CRASHED

If the system crashes after the decrement, then the increment won't happen and 10 dollars will disappear into nothing. This is called halfway state or partial success because part of the operation is complete and another part is not.

To solve this, some databases have features that provide atomicity. To use them, you wrap your operations to mark them as an atomic group which is called a database transaction if any operation in the transaction fails, the already finished operations before it will be automatically undone.

Atomic Group

1 Decrease Vec's balance by 10 dollars →

name	balance
Vec	90$
Cleopatra	100$

2 Increase Cleo's balance by 10 dollars →

SYSTEM CRASHED

undo step 1 ↓

name	balance
Vec	100$
Cleopatra	100$

To summary, with atomicity, either the whole transaction happens or none of it happens. There is not halfway state.

- C: Consistency

Many terms in the software engineering are overloaded and the term consistency is overloaded. So we will look at 2 definitions of consistency.

1) Data Consistency

When the data in a system is accurate, valid and meets all desired constraints.

Imagine you have a table of users and you have a constraint that every user in the table must have a first name longer than 4 characters.

users		
id	number	unique
first_name	string	> 4 chars

The database is considered consistent if:

- All desired constraints are met

There are 0 records that violate the constraints

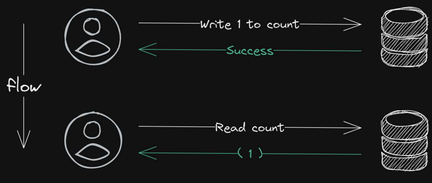
- It enforces such constraints

The database will throw an error when trying to insert a record that violates a constraint like a user with a first name shorter than 4 chars

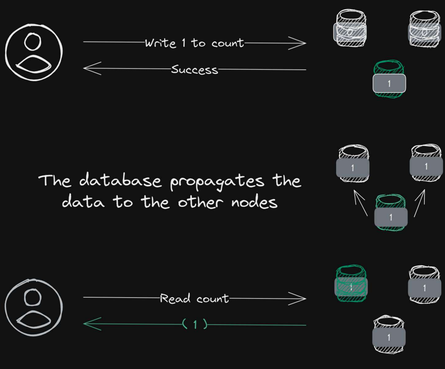
2) Freshness Consistency

When reading from the database always retrieves the latest value

In simple cases when write some data to the database then read it, the retrieved data is always fresh.

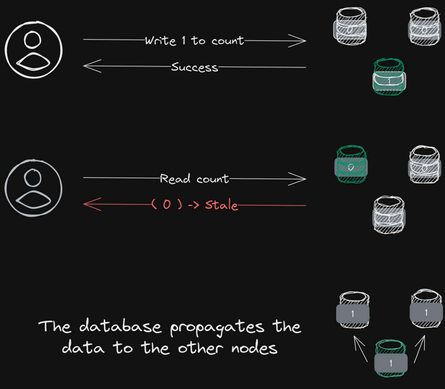


At scale, the database usually consists of many nodes so that each node handles a fraction of the load



In the example above, the user writes 1 to a database node, then it propagates the data to the other nodes.

But there is a case where the user could reads `count` before the propagation happens and user will get a stale value. due to this, the database does not have freshness consistency



Some large-scale databases offer full-freshness consistency known as strong consistency while others offer a limited kind of consistency known as eventual consistency

- Strong Consistency

When reading from a database `always` retrieves the latest value

- Eventual Consistency

When reading from a database gives the latest value if the nodes are given enough time to sync

There is a trade off between consistency and performace because providing strong consistency create a huge coordination bottle-neck where all the database nodes have to be aware of each other all the time.

Which one is good depends on the kind of application you're building. For example strong consistency is critical for many areas like payments. On the other hand, eventual consistency will be more than enough for a social network where a like now showing immediately is not a problem at all

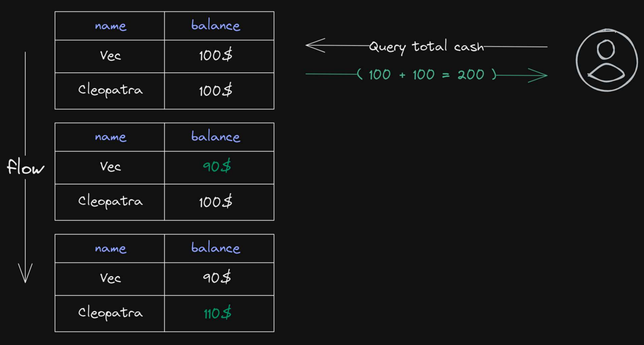
- I: Isolation

A property of a database where simultaneous operations on the same data never see each other's in-progress work.

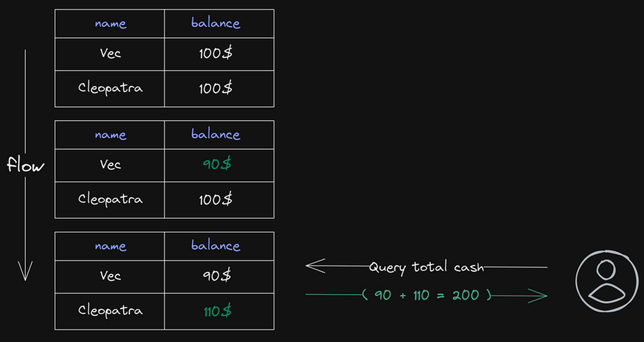
Remember the bank accounts example. If Vec wants to transfer 10 dollars to Cleo, we first need to remove 10 dollars from Vec's account then we need to add 10 dollars to Cleo's account. Let's say while this transfer happens, we asked the system to query the total amount of cash in the system. The answer should be always 200 dollars.

If we are performing a transfer and querying the total amount of cash in parallel, there are 3 ways these operations can be carried out.

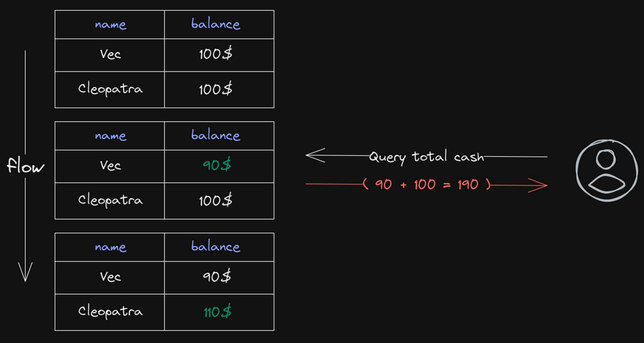
1) Before the transfer begins



2) After the transfer finishes



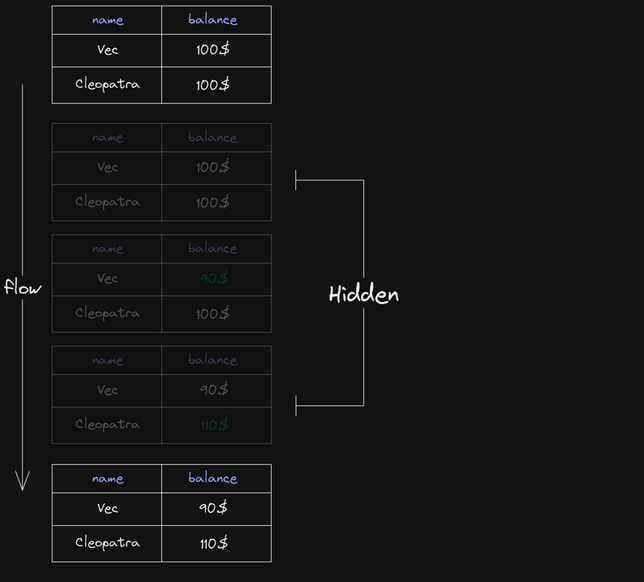
3) In the middle of the transfer



As you can see in the third case, There is a critical moment between the first and the second operation where the system is not in a valid state because 10 dollars are missing.

- How isolation works

In a database with isolation guarantees, when the transaction begins, the changes are only visible inside the same transaction and hidden from other transactions. This let's other transactions query Vec's and Cleo's accounts as if there is no on-going transfer.

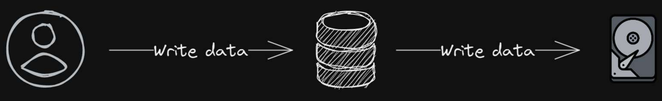


As you can see, there is never a moment where parallel transactions can see each other's in progress work

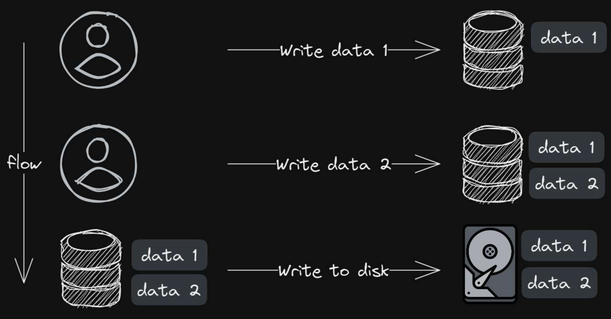
- D: Durability

A property of a database that guarantees that completed updates will not be lost if the system crashes.

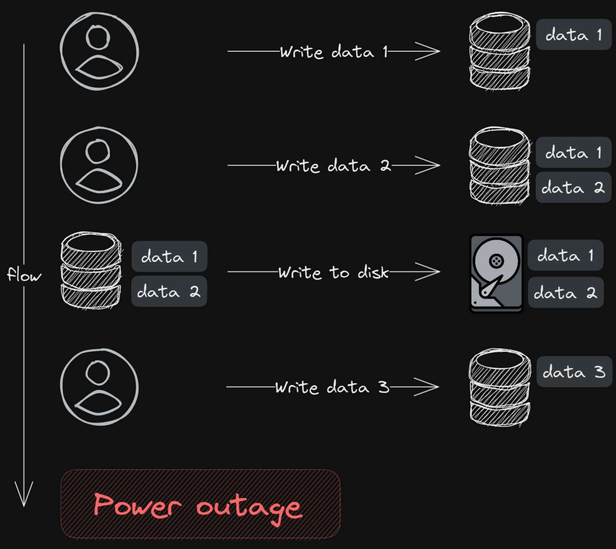
In general, when you ask the database to write some data, it writes that data to a persistent storage such as the hard disk. It is called persistent because if the system crashes or suffers a power cut, the data will not be lost.



Some databases will delay writing to a persistent storage so that they can write several updates at once, thus increasing performance.



This technique is called batching and it increases performance, however, if the database has some data that has not been written yet and there is a power outage, this data will be lost



Even if the database does not use batching and writes to the persistent storage on every change, it won't be 100% be durable. That is because the hard disk itself queues writes to be batched together.

As a result, if a power outage happens, data queued in the operating system layer will also be lost because they were not encoded in the actual atoms of the hard disk. To avoid batching, it's possible to tell the operating system to write everything currently queued which is called flushing.

A database with durability both executes a write to the disk drive on every change and tells the operating system to flush any pending writes.

In summary, a database is considered durable if it can guarantee that whenever you ask it to write some data and it responds successfully, the data is encoded in real physical atoms.