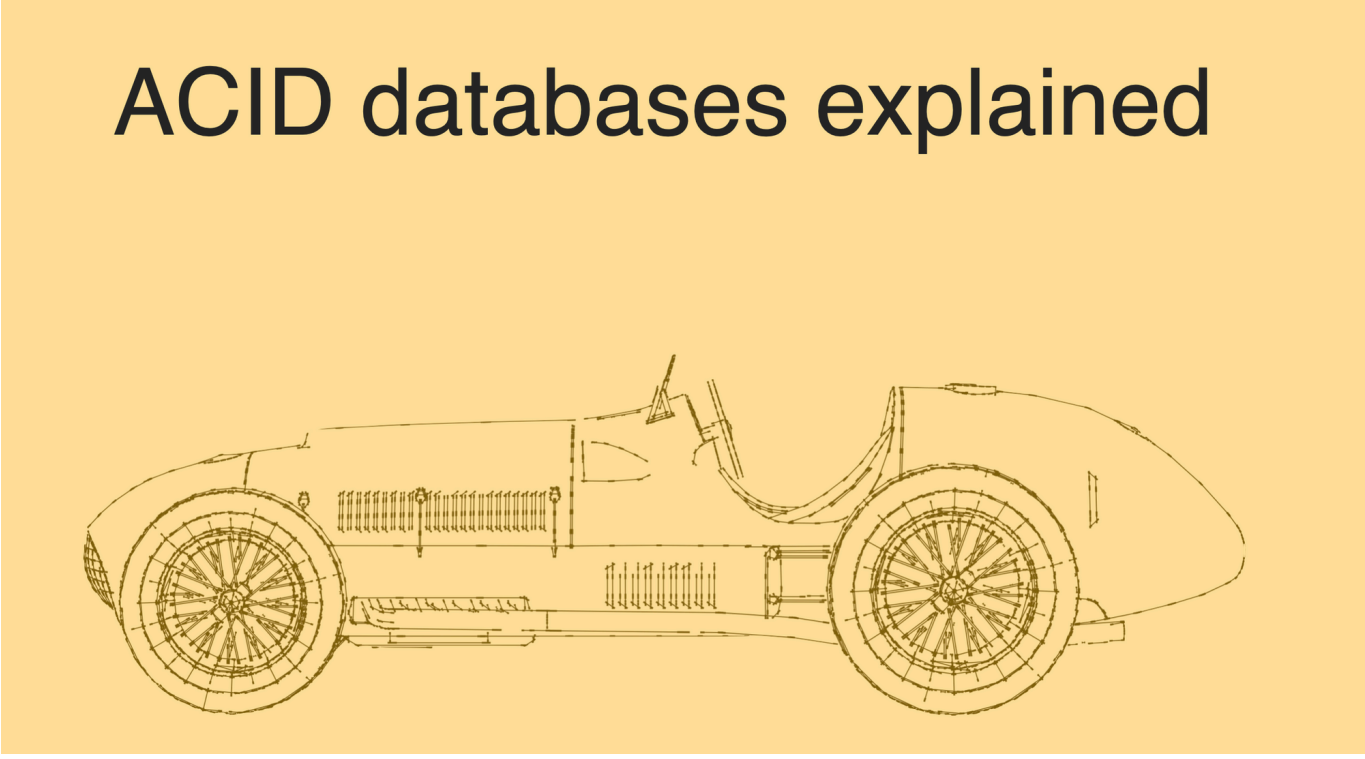
JANUARY 17, 2024 / #DATABASE

ACID Databases – Atomicity, Consistency, Isolation & Durability Explained

D Daniel Adetunji



Durability. These are four key properties that most database management systems (DBMS) offer as guarantees when handling transactions.

Most popular DBMS like MySQL, PostgresSQL and Oracle have ACID guarantees out of the box. Others have partial ACID guarantees like Redis, DynamoDB, and Cassandra. The trend, however, seems to be that more and more DBMS are offering ACID compliance.

It is important to note that while a lot of DBMS may say they are ACID compliant, the implementation of this compliance can vary.

So, for example, if isolation is a key property that you need for an application you are building, you need to understand how exactly your chosen DBMS implements isolation.

This article will explain what transactions are, and go through, in detail, what atomicity, consistency, isolation and durability mean, using analogies and real world examples.

Table of Contents:

What are Transactions?

Lots of things can go wrong when using a database:

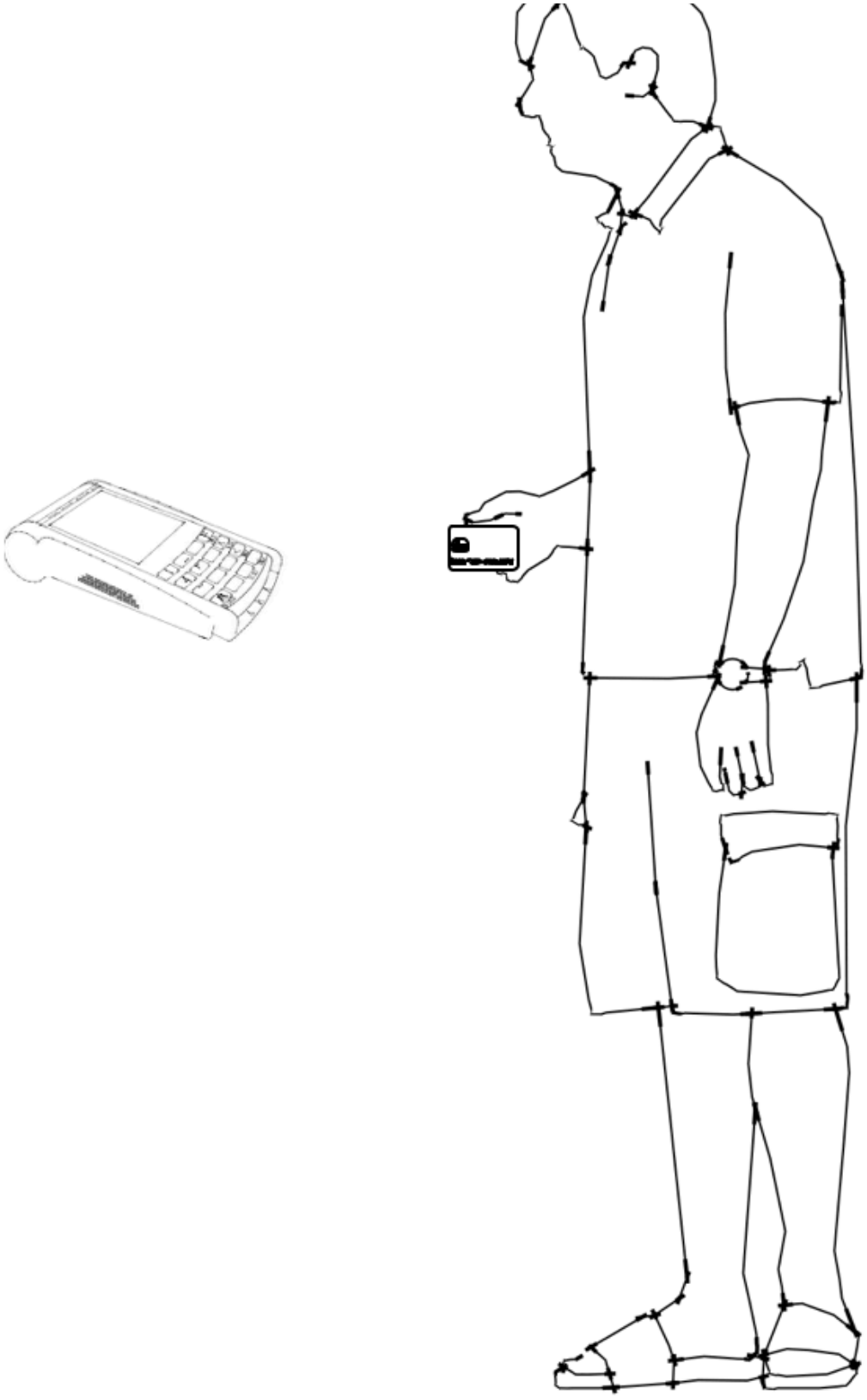
- the database hardware or software can fail
- the application calling the database can fail mid-operation
- the network can be flooded with more traffic can it can handle (rendering it inoperable)
- several clients can make writes at the same time that overwrite the other's changes
- clients can read phantom data that should not be in the database

And so on – this is in no way an exhaustive list of things that can go wrong.

Since things can fail in more ways than we can possibly anticipate, trying to prevent every possible failure can become unnecessarily expensive and complicated. Instead, it is better to design a system that can continue to operate in spite of a failure. Transactions allow us to do this.

Transactions serve a single purpose: they make sure a system is **fault tolerant.** If a failure in a system occurs, can the system continue to operate without complete catastrophe? Phrased differently, can the system tolerate faults? An answer of 'yes' to this question means that such a system is fault tolerant.

So, what exactly is a transaction?



and writes) that are treated as a single logical operation.

Imagine you want to buy a single book from an online store, say amazon.com. The steps below show a simplified view of what needs to happen:

1. First, you select the book, which adds the item to your basket.
2. The inventory quantity of the book is checked to ensure it is valid (that is, the inventory value for the title you are buying needs to be greater than 0).
3. You click 'buy', which updates Amazon's inventory for the book and decreases it by 1 (since you are buying a single book).
4. Also, your bank account balance is updated to account for the cost of the book.

A transaction ensures that all operations related to the purchase are treated as a single operation. If any part of the transaction fails, the entire transaction is rolled back, leaving the database in a state as if the customer had never attempted the purchase, thus maintaining the integrity of the data.

The transaction is committed when all the operations within the transaction are successfully completed and their results are permanently recorded. This permanence is typically achieved by writing the changes to the database's storage, which could be on disk for traditional databases or in memory for in-memory databases like Redis.

isolation and **durability**.

What Does Atomicity Mean?

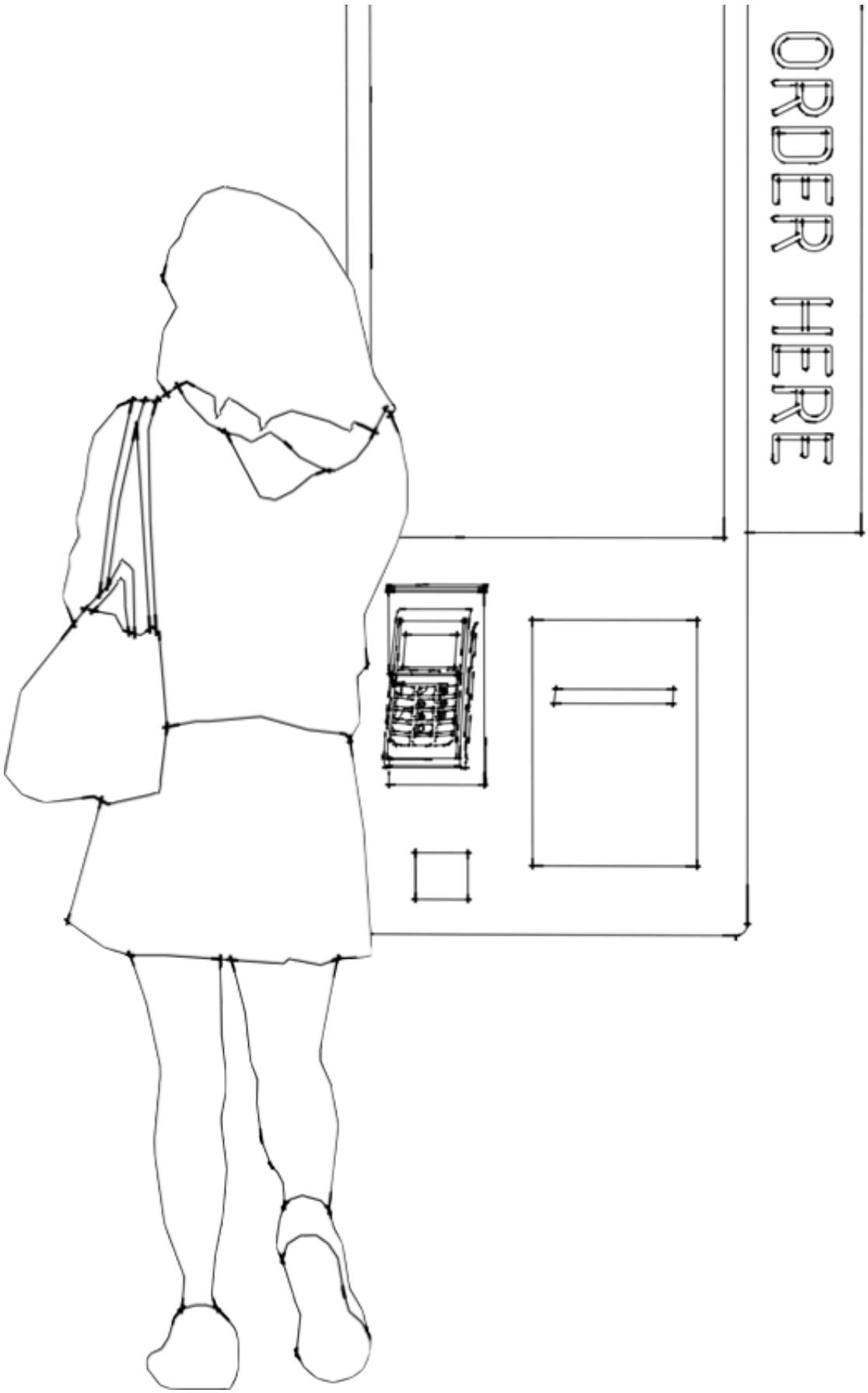
Atomicity simply means that all queries in a transaction must succeed for the transaction to succeed. If one query fails, the entire transaction fails.

An Atomic Restaurant

Imagine using a self-service machine at a fast-food restaurant. The transaction in this case is ordering food, and consists of two separate operations:

1. Select food
2. Make payment

Both of these must succeed for the transaction to succeed. If either fails, the transaction fails.



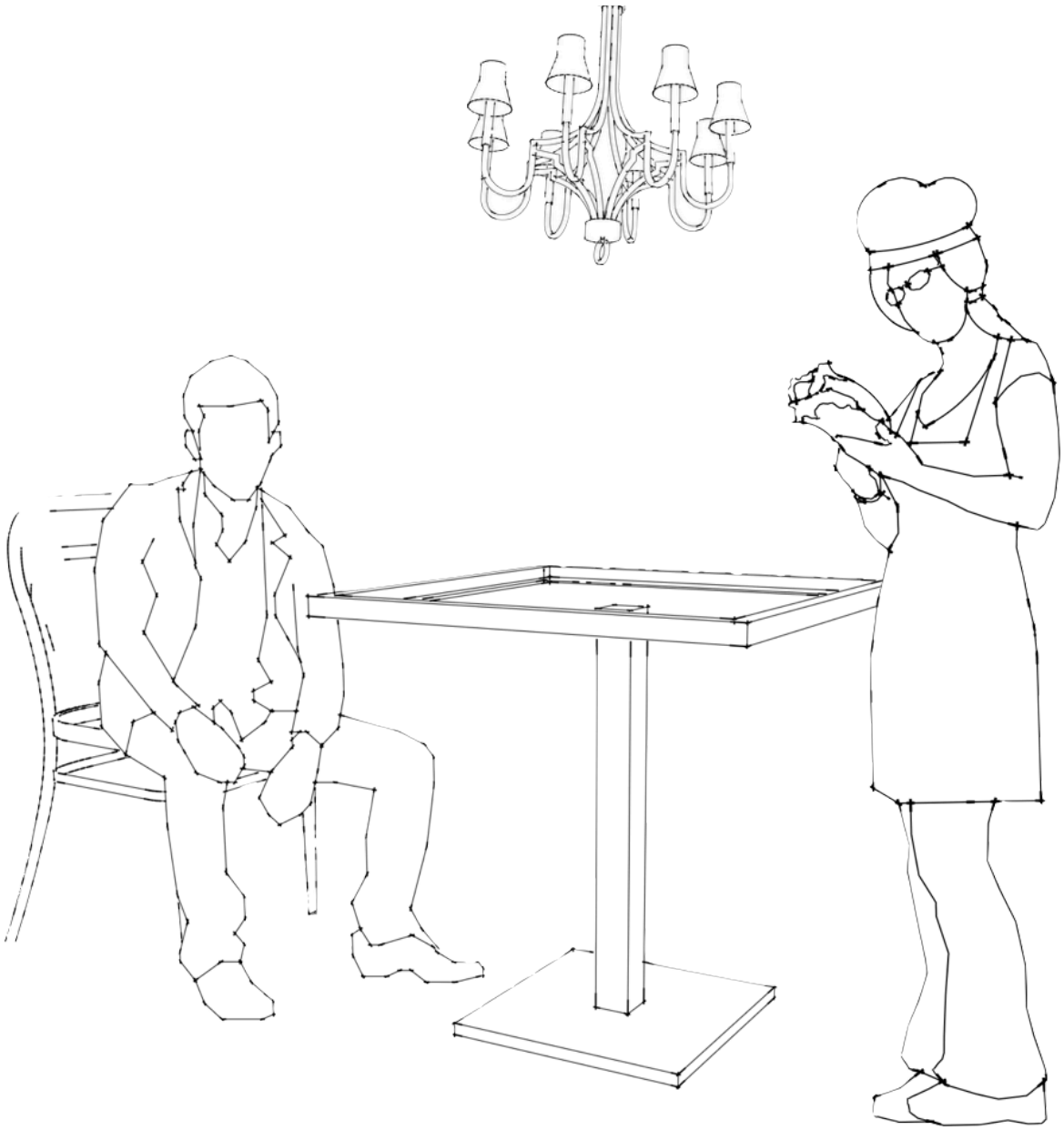
You select your burger, fries, and a drink from the touchscreen menu. The machine prompts you to pay, and only after your payment is processed successfully, it sends your order to the kitchen. Moments later, your entire order is ready, and you pick it up from the counter.

This is an atomic operation: the transaction (ordering food) is either entirely completed (if you select your food item and make a payment) or not completed at all.

Either part of the transaction failing means the entire transaction will fail. If your payment fails, the machine won't process any part of the order, so the transaction fails. If you make a payment without selecting a food item, the transaction also fails, as there is nothing for the kitchen to prepare.

A Non-Atomic Restaurant

Now consider the alternative, a traditional sit-down restaurant where you order several dishes. As each dish is prepared, it is brought to your table.



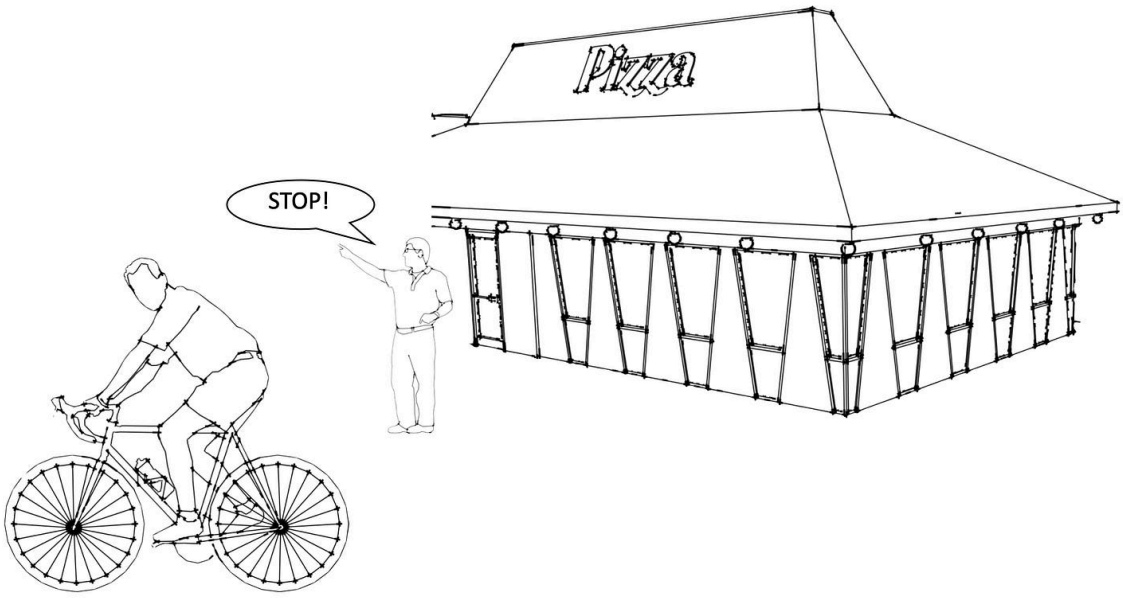
customer making an order in a "non-atomic" restaurant

Again, the transaction is ordering food, and consists of two separate operations:

1. Select food
2. Make payment

your meal. Partial failures do not cause a transaction to fail.

This creates a risk for the restaurant. Customers that choose to dine and dash can order food to their heart's delight and then simply leave without paying, causing a financial loss for the restaurant.



non-atomic restaurants are at risk of customers doing a dine and dash

Atomic Transactions

If several SQL queries are grouped together in a transaction, atomicity is a guarantee that, should any of the queries fail for any reason (hardware, application or networking problems) then the transaction is aborted and the database returns to its previous state, as if nothing had happened.

failure can compound the problem, since you risk introducing incorrect data to the database by re-running queries that previously succeeded.

Atomic transactions prevent such uncertainty, since you know that if the previous transaction failed, it failed in its entirety, and you can simply retry without worrying about introducing inconsistent data.

What Does Consistency Mean?

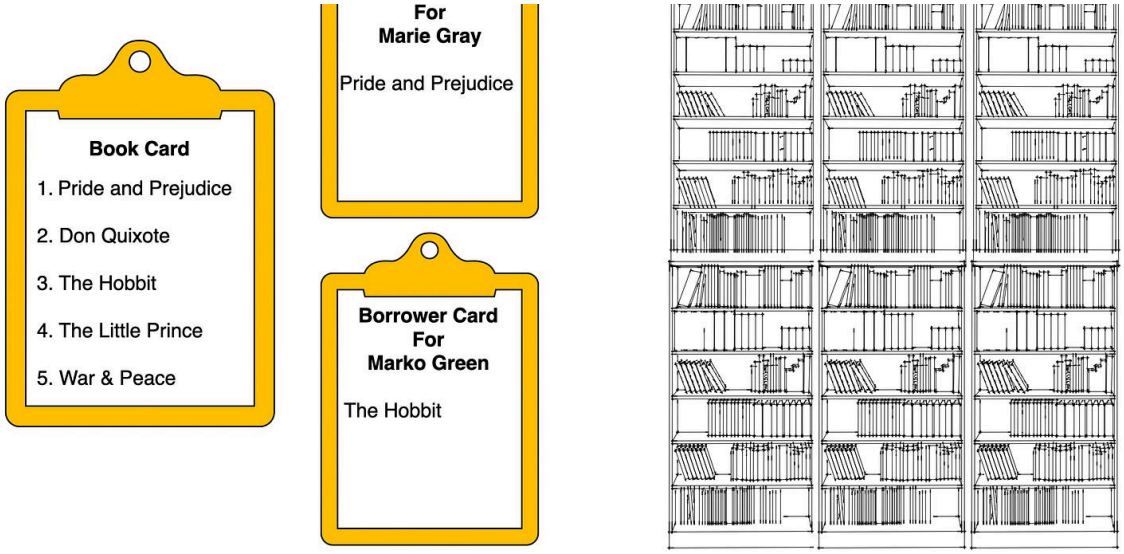
Consistency can mean different things in cloud/software engineering, depending on the context. In the case of ACID, the “C” was most likely added to make the acronym work.

Consistency in the context of ACID means *consistency in data*, which is defined by the creator of the database. The technical term for consistency in data is called referential integrity. Referential integrity is a method of ensuring that relationships between tables remain consistent. It's usually enforced through the use of foreign keys.

To understand referential integrity, consider the following.

Imagine a library system with two types of cards: a book card and a borrower's card.

- The book card lists all the books available in the library.
- The borrower's card tracks which books are borrowed by which members.



A book card and borrower card for a library

The rule of the library is that a book can only be listed on a borrower's card if it exists on a book card. This is referential integrity. If someone tries to list a book on a borrower's card that isn't on the book card (that is a book that doesn't exist in the library), the system will not allow it.

While atomicity, isolation and durability are properties intrinsic to the database itself, consistency in data, or referential integrity, is not a property intrinsic to the database.

Consistency is defined by the creator of the database. The application calling the database relies on the atomicity and isolation properties of the database to maintain that consistency.

What Does Isolation Mean?

transactions trying to modify or read the same database record(s) at the same time.

There are three levels of transaction isolation. I'll just explain the two main ones below, arranged in order from the least strict to most strict.

Read Committed

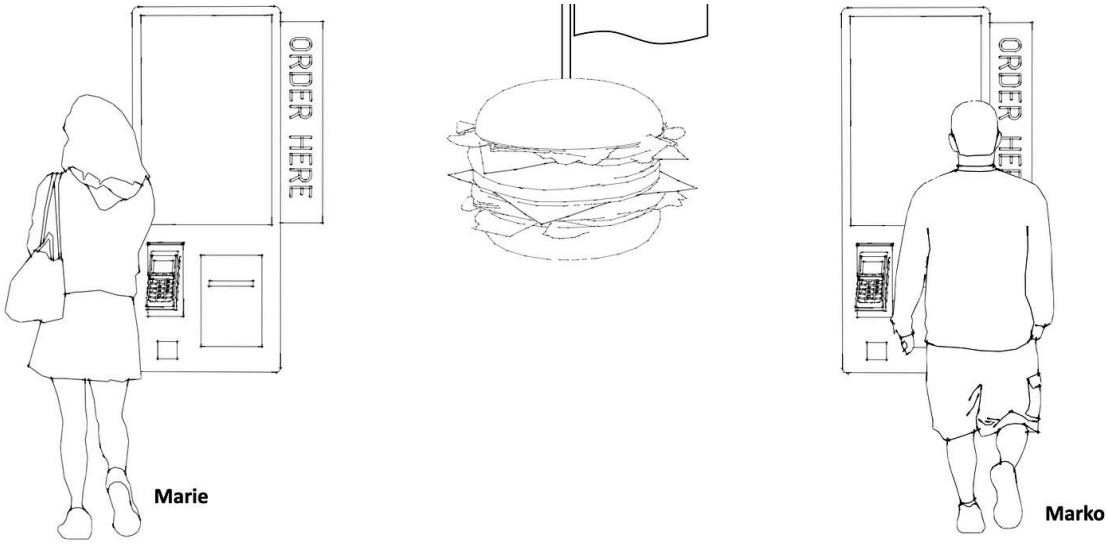
This gives two guarantees. It prevents dirty reads and dirty writes.

No Dirty Reads: Reading data from another transaction that has not yet been committed is called a dirty read. With the read committed isolation level, you will only see data that has been committed by another transaction.

No Dirty Writes: Overwriting data that has already been written by another transaction but not yet committed is called a dirty write.

To understand how read committed isolation works, consider the following example.

Imagine a fast-food restaurant with only one last special burger available, and two hungry customers, Marie and Marko, are trying to buy it simultaneously.



Two customers ordering a burger at the same time

1. Marie checks the availability of burgers and sees the last one available. Unknown to her, Marko's order is being processed but hasn't been finalised in the system, as he has not paid. Since his order has not yet been finalised, Marie is not aware that his order conflicts with her own. This is similar to a transaction reading the most recently committed data, where it does not see uncommitted changes (like Marko's pending order).
2. Marie places an order based on this incomplete information, thinking a burger is available.
3. Once Marko pays, the system updates to show that there are no burgers left. This is similar to a transaction being committed
4. Marie's order will have to be aborted since there are no burgers left.

still be a burger available for Marie.

In this example, read committed isolation ensures that Marie is not prematurely excluded from buying the burger just because someone else said they wanted it. Only committed transactions can be read. Therefore, the burger is available to be ordered as long as no one has paid for it.

Repeatable Read

The repeatable read is a more strict isolation level, in that it has the same guarantees as read committed isolation – plus it guarantees that reads are repeatable.

A repeatable read guarantees that if a transaction reads a row of data, any subsequent reads of that same row of data within the same transaction will yield the same result, regardless of changes made by other transactions. This consistency is maintained throughout the duration of the transaction.

When a transaction reads the same data twice, but sees a different value in each read because a committed transaction has updated the value between the two reads, this is called a fuzzy read. The repeatable read isolation level prevents fuzzy reads.

Fuzzy reads are neither inherently good nor bad. It all depends on what you are trying to achieve.

Fuzzy reads are bad for long-running, read-only transactions, since new writes are likely to occur during the transaction and this can

used in a data warehouse.

Repeatable reads are usually implemented by the DBMS by reading from a snapshot of the database which remains unchanged for the duration of the transaction, thereby ignoring any new committed writes in that period.

What Does Durability Mean?

Durability is a guarantee that changes made by a committed transaction must not be lost. All committed transactions must be persisted on durable, non-volatile storage, that is on disk. This ensures that any committed transactions are protected even if the database crashes.

Naturally, durability cannot protect against destruction of the disk which stores the data. Additional redundancy can be added by having backups of your database stored separately from the original.

Bringing it Together

ACID (Atomicity, Consistency, Isolation, and Durability) provides a set of guarantees when working with a DBMS. While most relational DBMS are ACID compliant, the implementation of this compliance can vary.

Atomicity ensures that all parts of a transaction are completed or none at all. Partial failures are not allowed.

application calling the database relies on the atomicity and isolation properties of the database to maintain consistency.

Isolation is a guarantee that concurrently running transactions should not interfere with each other. This is arguably the most important property because a DBMS can often have different default isolation levels, which may need to be changed based on what is needed for your application.

Finally, durability is a guarantee that changes made by a committed transaction must not be lost.

Daniel Adetunji

Read more posts.
