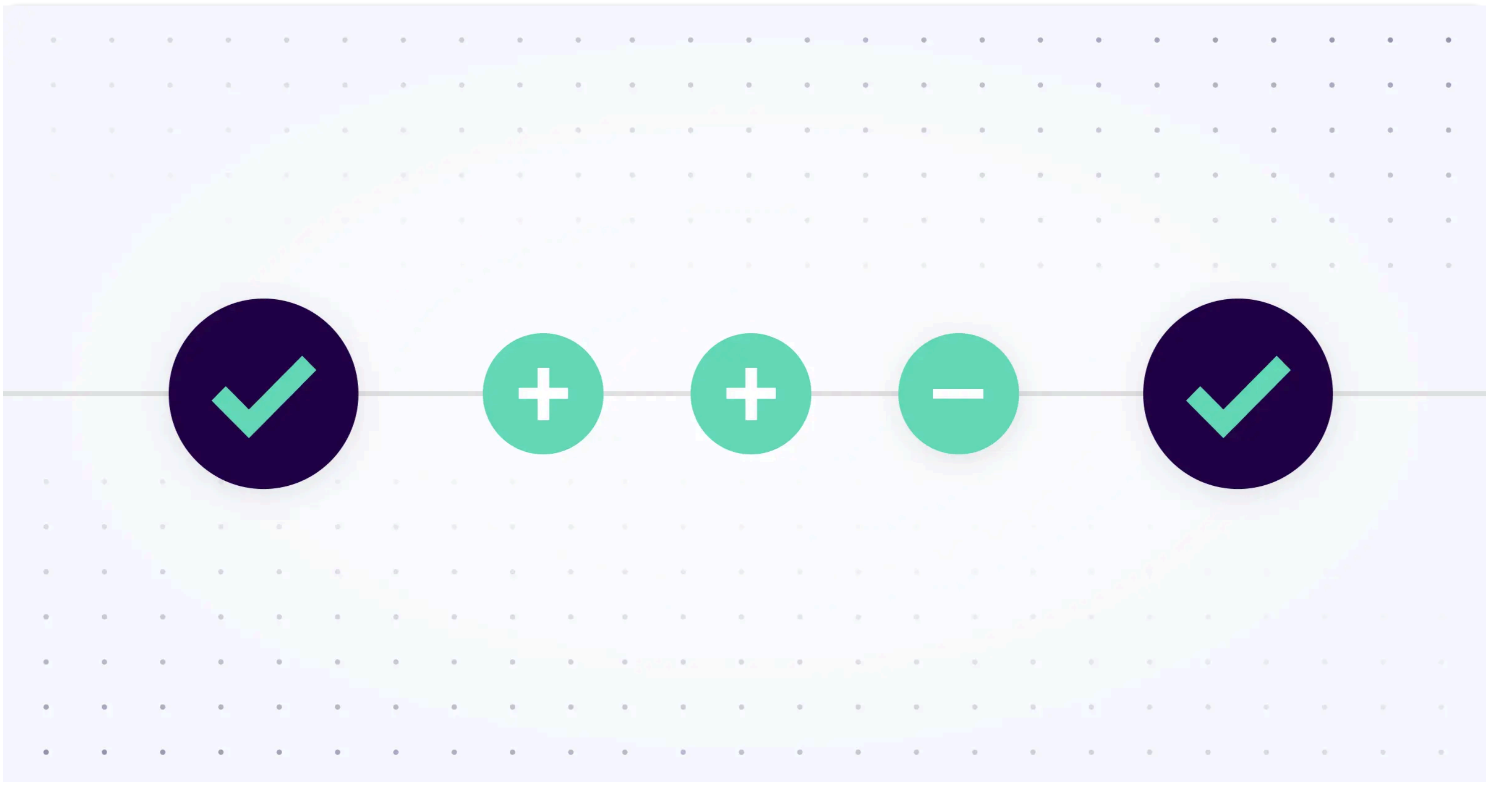


What is a database transaction?

Aug 2nd, 2021 | Categories: Data Consistency

Databases are known for processing millions of concurrent requests per second. In many cases, these requests touch the same item in the database. Imagine, for instance, that you are trying to purchase a limited supply of your favorite game on an online e-commerce site. Suppose everyone in the online store puts the game in their carts at the same time and proceed to checkout. In that case, the remaining inventory needs to be calculated accurately — neither over nor under. Typically, a database transaction is used in such scenarios.

So, what is a database transaction?

In short, a database transaction is a sequence of multiple operations performed on a database, and all served as a single logical unit of work — taking place wholly or not at all. In other words, there's never a case that *only half of the operations* are performed and the results saved.

When a database transaction is in flight, the database state may be temporarily inconsistent, but when the transaction is committed or ends, the changes are applied.

To explain the concept of a database transaction, let us use a typical example of transferring money between Account A and Account B. Let's say you want to move 5 dollars from Account A to Account B. This action can be broken down into the following simple operations:

1. Create a record to transfer 5 dollars from Account A to Account B. This is typically called the *begin* of a database transaction.
2. Read the balance from Account A.
3. Subtract 5 dollars from the balance of Account A.
4. Read the balance from Account B.
5. Add 5 dollars credit to Account B.

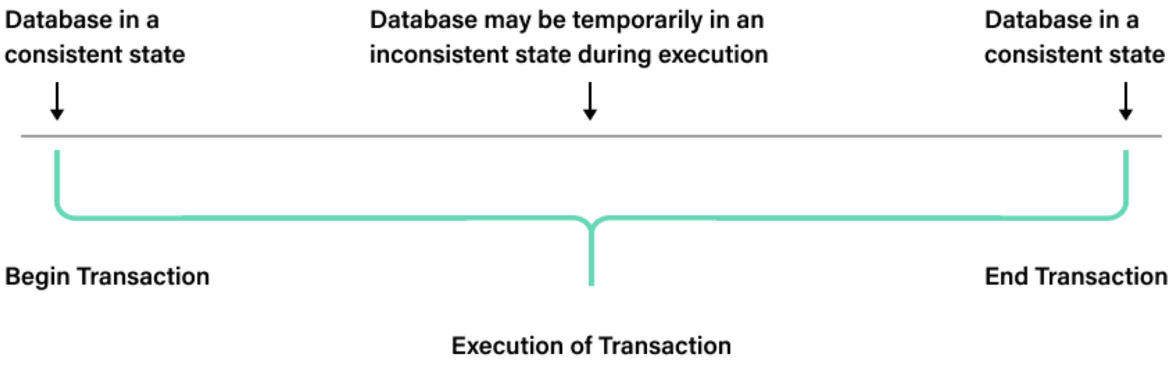
Now, if your database is running this transaction as one whole atomic unit, and the system fails due to a power outage, the transaction can be undone, reverting your database to its original state. Typically, a term like *rollback* refers to the process that undoes any changes made by the transaction, and the term *commit* is used to refer to a permanent change made by the transaction.

How do database transactions work?

Before we learn about how database transactions work, let's explore why they are needed in the first place.

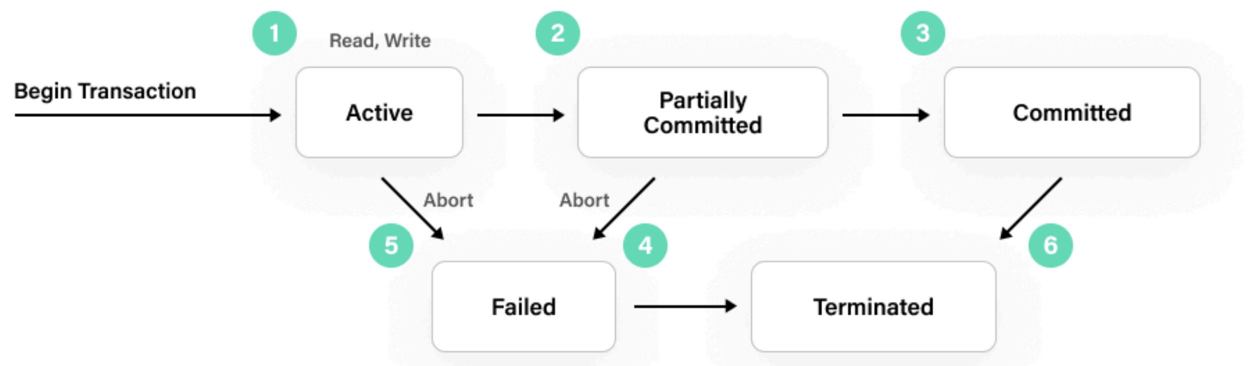
1. System failures are inevitable, and in these cases, a transaction provides a way to ensure that the outcome is reliable and consistent. This means that the state of the database reflects all transactional changes committed before the point of failure and that transactions that were in-flight at the failure point are cleanly rolled back.

2. When multiple concurrent requests are hitting the database server, changing the same underlying data simultaneously, the transaction must isolate requests from each other to avoid conflicts.


Database in a consistent state
Database may be temporarily in an inconsistent state during execution
Database in a consistent state
Begin Transaction
End Transaction
Execution of Transaction

During its lifecycle, a database transaction goes through multiple states. These states are called **transaction states** and are typically one of the following:

1. **Active states:** It is the first state during the execution of a transaction. A transaction is *active* as long as its instructions (read or write operations) are performed.
2. **Partially committed:** A change has been executed in this state, but the database has not yet committed the change on disk. In this state, data is stored in the memory buffer, and the buffer is not yet written to disk.
3. **Committed:** In this state, all the transaction updates are permanently stored in the database. Therefore, it is not possible to rollback the transaction after this point.
4. **Failed:** If a transaction fails or has been aborted in the active state or partially committed state, it enters into a *failed* state.
5. **Terminated state:** This is the last and final transaction state after a committed or aborted state. This marks the end of the database transaction life cycle.



What are ACID properties, and why are they important?

In **relational databases**, transactions must be atomic, consistent, isolated and durable. These properties are commonly abbreviated as ACID. **ACID properties** ensure that a database transaction is processed reliably. In this section, let's learn a bit more about what these properties mean for the application.

Atomicity

Atomicity in terms of a transaction means *all or nothing*. When a transaction is committed, the database either completes the transaction successfully or rolls it back so that the database returns to its original state. For example, in an online ticketing application, a booking may consist of two separate actions that form a transaction — reserving the seat for the customer and paying for the seat. A transaction guarantees that when a booking is completed, both these actions, although independent, happen within the same transaction. If any of the actions fail, the entire transaction is rolled back, and the booking is freed up for another transaction attempting to take it.

Consistency