ACID & BASE properties

19 August 2023 11:16 PM

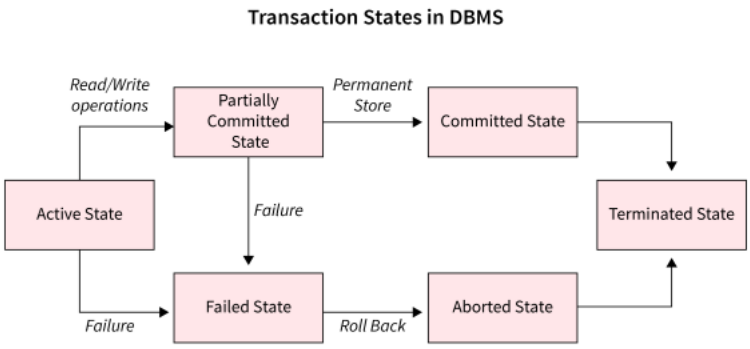
ACID Data Base Model :

The ACID database transaction model ensures that a performed transaction is always consistent. This makes it a good fit for businesses which deal with online transaction processing (e.g., finance institutions) or online analytical processing (e.g., data warehousing). These organizations need database systems which can handle many small simultaneous transactions. There must be zero tolerance for invalid states.

What is a Transaction?

So basically in layman terms, a Transaction is a way for an application to group several reads and writes (CRUD) together into a logical unit. And with transactions, error handling becomes much simpler for an application.

So technically everything between a BEGIN TRANSACTION and a COMMIT statement is considered to be part of the same transaction.



State	Description
Active	Transactions are in progress.
Partially Committed	Operations completed, but data not yet saved.
Failed	Transaction fails database recovery system checks.
Committed	Successful completion, changes permanently saved.
Aborted	Transaction fails tests, rolled back or aborted.
Terminated	Transaction terminated, system ready for new transactions.

Atomicity :

- All operations in a transaction succeed or every operation is rolled back.
- It states that database modifications must follow an all-or-nothing rule. Each transaction is said to be *atomic*. If one part of the transaction fails, the entire transaction fails. It is critical that the database management system maintains the atomic nature of transactions in spite of any DBMS, operating system, or hardware failure.

Example :-

i. Sometimes, a current operation will be running and then, an operation with a higher priority enters. This discontinues the current operation and the current operation will be aborted.
ii. In the given scenario, if two users simultaneously try to book the only available seat on a train, the transaction is considered incomplete. According to atomicity, the first user who successfully clicks the booking button will reserve the seat and receive a notification, while the second user's transaction will be rolled back, and they will be notified that no more seats are available.
iii. In a simpler example, if a person tries to book a ticket, selects a seat, and proceeds to the payment gateway but encounters a failure due to bank server issues, their booked seat will not be reserved for them. A complete transaction involves reserving the seat and completing the payment. If any step fails, the operation is aborted, and the user is brought back to the initial state without their seat being reserved.

Atomicity in DBMS is often referred to as the ‘all or nothing’ rule.

Consistency :

- On the completion or rollback of a transaction, the database is in a consistent state; there are no partial updates or logical corruptions.

- It states that only valid data will be written to the database. If a transaction is executed that violates the database's consistency rules, the entire transaction is rolled back, and the database is restored to a state consistent with those rules. On the other hand, if a transaction successfully executes, it takes the database from one state that is consistent with the rules to another state that is also consistent with the rules.

  Example :-
  Let us consider an example where one person is trying to book a ticket. They are able to reserve their seat but their payment hasn't gone through due to bank issues. In this case, their transaction is rolled back. But just doing that isn't sufficient. The number of available seats must also be updated. Otherwise, if it isn't updated, there will be an inconsistency where the seat given up by the person is not accounted for. Hence, the total sum of seats left in the train + the sum of seats booked by users would not be equal to the total number of seats present in the train if not for consistency.

Isolation :

- It ensures that multiple transactions can run concurrently without interfering with each other.
- Each transaction is independent of the other. A transaction does not have access to other transactions which had not finished yet. The changes occurring during a transaction will not be visible to others until it is written in the main memory.
- It requires that multiple transactions occurring at the same time not impact each other's execution. For example, if Joe issues a transaction against a database at the same time that Mary issues a different transaction, both transactions should operate on the database in an isolated manner. The database should either perform Joe's transaction before executing Mary's or vice-versa. This prevents Joe's transaction from reading intermediate data produced as a side effect of part of Mary's transaction that will not eventually be committed to the database. The isolation property does not ensure which transaction executes first—only that transactions will not interfere with each other.

  Example :-
  Suppose two people try to book the same seat simultaneously. Transactions are serialized to maintain data consistency. The first person's transaction succeeds, and they receive a ticket. The second person's transaction fails as the seat is already booked. They receive an error message indicating no available seats.

Durability :

- It ensures that any transaction committed to the database is not lost. Durability is ensured by using database backups and transaction logs that facilitate the restoration of committed transactions despite any subsequent software or hardware failures or crashes. There is a continuous backup of data for disaster recovery.
- However, if the database is lost, the recovery manager is responsible for guaranteeing the database's long-term viability. Every time we make a change, we must use the COMMIT command to commit the values.

  By the way, perfect durability does not exist. if all of your hard disks and all your backups are destroyed at the same time, there's nothing your database can do to save you.

  Example :-
  Suppose that there is a system failure in the railway management system resulted in the loss of all booked train details. Millions of users who had paid for their seats are now unable to board the train, causing significant financial losses and eroding trust in the company. The situation is particularly critical as these trains are needed for important reasons, causing widespread panic and inconvenience.

How ACID Works in Practice :

Database administrators use several strategies to enforce ACID.

- One strategy used to enforce atomicity and durability is write-ahead logging, in which any transaction detail is first written to a log that includes both redo and undo information. This approach ensures that, given a database failure, the database can check the log and compare its contents to the state of the database.
- Another method used to address atomicity and durability is shadow-paging, in which a shadow page is created when data is to be modified. The query's updates are written to the shadow page rather than to the real data in the database. The database is modified only when the edit is complete.
- Another strategy is called the two-phase commit protocol, especially useful in distributed database systems. This protocol separates a request to modify data into two phases: a commit-request phase and a commit phase. In the request phase, all DBMSs on a network that are affected by the transaction must confirm that they received it and have the capacity to perform the transaction. Once confirmation is received from all relevant DBMSs, the commit phase completes in which the data is modified.

Use Cases for ACID Databases :

- ACID-compliant databases are widely used in industries like finance, healthcare, and government administration, where data storage is highly regulated. ACID compliance makes it so that, for example, a bank's customers don't have

to worry about their account balances displaying incorrectly since an ACID-compliant database will always retrieve up-to-date data.

- Many small-to-medium-scale enterprises also use ACID-compliant databases for their ease of use and stability. The relatively small scale of the data these organizations process means that they don't mind the overhead ACID compliance adds to database operations, such as time to process long JOINs.

Which Databases Use ACID?

- One safe way to make sure your database is ACID compliant is to choose a relational database management system. These include MySQL, PostgreSQL, Oracle, SQLite, and Microsoft SQL Server.
- Some NoSQL DBMSs, such as Apache's CouchDB or IBM's Db2, also possess a certain degree of ACID compliance. However, the philosophy behind the NoSQL approach to database management goes against the strict ACID rules. Hence, NoSQL databases are not the recommended choice for those who need strict environments.

Advantages of ACID Properties in DBMS:

1. Data Consistency: ACID properties ensure that the data remains consistent and accurate after any transaction execution.
2. Data Integrity: ACID properties maintain the integrity of the data by ensuring that any changes to the database are permanent and cannot be lost.
3. Concurrency Control: ACID properties help to manage multiple transactions occurring concurrently by preventing interference between them.
4. Recovery: ACID properties ensure that in case of any failure or crash, the system can recover the data up to the point of failure or crash.

Disadvantages of ACID Properties in DBMS:

1. Performance: The ACID properties can cause a performance overhead in the system, as they require additional processing to ensure data consistency and integrity.
2. Scalability: The ACID properties may cause scalability issues in large distributed systems where multiple transactions occur concurrently.
3. Complexity: Implementing the ACID properties can increase the complexity of the system and require significant expertise and resources.
   Overall, the advantages of ACID properties in DBMS outweigh the disadvantages. They provide a reliable and consistent approach to data
4. management, ensuring data integrity, accuracy, and reliability. However, in some cases, the overhead of implementing ACID properties can cause performance and scalability issues. Therefore, it's important to balance the benefits of ACID properties against the specific needs and requirements of the system.

BASE Data Base Model :

For many domains and use cases, ACID transactions are far more pessimistic (i.e., they're more worried about data safety) than the domain actually requires.

In some NoSQL databases where performance relies on large-scale sharding and horizontal scale-out (MPP) for performance, maintaining ACID compliance is extremely costly (for example, the overhead of 2 Phased Commit/2PC protocols) when a transaction spans possibly dozens of instances. As a result, NoSQL databases that rely heavily on horizontal scaling for performance often use the BASE transaction model.

BASE consists of three principles:

Basic Availability :

- It means that the system is always available, even if there is a network partition or a node failure.

- The NoSQL database approach focuses on the availability of data even in the presence of multiple failures. It achieves this by using a highly distributed approach to database management. Instead of maintaining a single large data store and focusing on the fault tolerance of that store, NoSQL databases spread data across many storage systems with a high degree of replication. In the unlikely event that a failure disrupts access to a segment of data, this does not necessarily result in a complete database outage.

Soft State :

- The state of the data could change without application interactions due to eventual consistency.
- BASE databases abandon the consistency requirements of the ACID model pretty much completely. One of the basic concepts behind BASE is that data consistency is the developer's problem and should not be handled by the database.

Eventual Consistency :

- It means that the system will eventually become consistent, although there may be some inconsistency in the meantime.
- The only requirement that NoSQL databases have regarding consistency is to require that at some point in the future, data will converge to a consistent state. No guarantees are made, however, about when this will occur. That is a complete departure from the immediate consistency requirement of ACID that prohibits a transaction from executing until the prior transaction has completed and the database has converged to a consistent state.

Use Cases for BASE Databases :

- BASE-compliant databases are almost exclusively used by large companies in relatively unregulated spaces that process several terabytes or more of data every day. The large scale of these companies means that they reach a point where the overhead from enforcing ACID compliance starts to demonstratively hurt operations. Thus, they look for alternative models without such overhead. In fact, many BASE-compliant databases originated as projects by these companies to address this very issue, like DynamoDB (Amazon) and BigTable (Google).
- BASE databases are used in modern, highly-available, and scalable systems that handle large amounts of data. Examples of such systems include online shopping websites, social media platforms, and cloud-based services.
- There may also be some niche use cases for BASE-compliant databases for smaller organizations. For example, a startup that predicts that the amount of data it processes will grow quickly may want to use a BASE-compliant database to avoid a potential migration process down the line.

Which Databases Use BASE?

- Just as SQL databases are almost uniformly ACID compliant, NoSQL databases tend to conform to BASE principles. MongoDB, Cassandra and Redis are among the most popular NoSQL solutions, together with Amazon DynamoDB and Couchbase.
- In addition to the MPP NoSQL use cases, other stores that use the BASE model are ones that are not considered to be the primary system of record and can be rebuilt easily from the original sources. These include aggregate stores, key-value stores (often used to cache data), and, in some cases, document stores.

Advantages of BASE Model:

1. High availability
2. Good scalability
3. Can handle large volumes of data and high concurrency
4. Allows for flexibility in handling data

Disadvantages of BASE Model:

1. Data may be inconsistent for a short period of time
2. Can lead to conflicts in data
3. Can be difficult to maintain

Comparison of ACID vs BASE Database Models :

	ACID Model	BASE Model
Data Consistency	Ensures data consistency and accuracy	Data may be inconsistent for a short period of time
Transactional integrity	Provides transactional integrity	No transactional integrity guarantees
Locking and synchronization	Requires locking and synchronization mechanisms	No locking and synchronization mechanisms required
Performance	Can lead to performance issues	Good scalability and high availability
Availability	Not suitable for systems that require high availability	High availability
Handling large volumes of data	Limited handling of large volumes of data	Can handle large volumes of data and high concurrency
Flexibility	Limited flexibility in handling data	Allows for flexibility in handling data
Application	Banking systems, financial systems, airline reservation systems	Social media platforms, e-commerce websites

Which is better among Acid vs Base Database Models?

In the comparison of the base vs acid database model, there is no straightforward answer to which model is better, ACID or BASE, as both have their own advantages and disadvantages.

The ACID model is suited for applications that require data consistency, accuracies, and transactional integrity, such as financial systems and banking applications. However, it can be challenging to scale ACID-compliant systems, and it can be difficult to maintain synchronization mechanisms. Moreover, ACID transactions may lead to performance issues and are not well-suited for systems that require high availability.

On the other hand, the BASE model prioritizes availability, scalability, and handling large volumes of data. It is a better fit for applications such as social media platforms and e-commerce websites, where data consistency is not as crucial as availability and scalability. However, this model may sacrifice some data consistency for a short period of time, and there are no transactional integrity guarantees.

In summary, the choice between ACID and BASE models depends on the specific needs of the application. For applications that require strict data consistency and transactional integrity, the ACID model is a better fit. For applications that require high availability and scalability, the BASE model is a better fit.