A Guide to ACID Properties in Database Management Systems

The use of real-time applications and Internet of Things (IoT) devices, plus the exponential growth of unstructured data assets, has organizations transitioning to NoSQL databases in ever-increasing numbers. In fact, the NoSQL market is anticipated to reach US $74.5 billion by 2032, exhibiting a growth rate (CAGR) of 24.9% from 2024 to 2032 (IMARC group, 2024).

This growth isn't surprising given the ability of NoSQL database management systems (DBMS) to effectively manage large, diverse datasets and the associated big data analysis required. However, many still believe that NoSQL DBMS aren't able to address a key requirement of many organizations – ACID transactions management and compliance. The good news is that some NoSQL DBMS absolutely can!

In this article, we'll discuss what ACID transactions are, ACID transaction properties, why these transactions are important, and an ACID transaction example in a NoSQL DBMS.

Table of contents

What are ACID transactions?

Transactions explained

At their most basic level, database transactions are a group of database read and write operations that have been completed successfully according to DBMS definitions (e.g., defined transaction criteria). There are two main types of transactions:

- **Single transactions:** A single transaction refers to a series of one or more database operations resulting in one action, completed successfully. Once complete, the transaction is accepted and can be found in a transaction log. A common example of a single transaction is the withdrawal of money from an ATM.
- **Multi-transactions:** A multi-transaction, sometimes called a distributed transaction, consists of multiple, interdependent transactions spread across different databases and systems (e.g., distributed systems). Records of these transactions are also found in a transaction log. An example of these transactions includes transferring money from one account to another or an employer issuing a new employee a security badge with a photo.

It's important to note that some transactions are required to adhere to strict standards of data integrity (e.g., data is complete and correct) and data consistency (e.g., the value is the same across all tables/databases). This is often the case where fiduciary responsibility or regulatory compliance is involved. Examples include commercial banking, investment brokerage, and legal settlements. In these situations, a standard adherence to DBMS definitions is not enough, and ACID transactions are required.

ACID transactions

ACID is an acronym that stands for atomicity, consistency, isolation, and durability (ACID). Together, ACID properties ensure that a set of database operations (grouped together in a transaction) leave the database in a valid state even in the event of unexpected errors.

Further, ACID transactions provide the level of transactional guarantees that many regulatory agencies require.

Below is a general overview of each ACID transaction element, as well as a description of how a NoSQL document database is able to handle that ACID element. For the purposes of this article, MongoDB Atlas will be used.

Atomicity

Atomicity guarantees that all of the commands that make up a transaction are treated as a single unit and either succeed or fail together. This is important in the event of a system failure or power outage, in that if a transaction wasn't completely processed, it will be discarded and the database maintains its data integrity.

How MongoDB handles atomicity:

In MongoDB, a write operation is atomic on the level of a single document, even if the operation modifies multiple embedded documents within a single document. For situations that require atomicity of reads and writes to multiple documents (in a single collection or multiple collections), MongoDB supports distributed transactions, including transactions on replica sets and sharded clusters.

Consistency

Consistency guarantees that changes made within a transaction are populated across the database system (e.g., nodes) and in alignment with DBMS constraints. If data consistency is going to be negatively impacted by a transaction in an inconsistent state, the entire transaction will fail.

How MongoDB handles consistency:

MongoDB gives the flexibility to normalize or duplicate data to optimize applications. If data is duplicated in the schema, the developer must decide how to keep duplicated data consistent across multiple collections. Some applications require duplicated data to be made consistent immediately, whereas other applications can tolerate reading stale data. Examples are illustrated below:

Method

Description

Transactions	Updates to multiple collections occur in a single atomic operation
Embed related data	Modify the application schema to embed related data in a single collection.
Atlas Database Triggers	When an update occurs in one collection, triggers automatically update another collection.

Performance Impact

Transactions	Potentially high, due to read contention
Embed related data	Low to moderate, depending on document size and indexes
Atlas Database Triggers	Low to moderate, with potential delays in processing triggered events

Use

Transactions	Your application must always return up-to-date data and can tolerate potential negative performance impact during periods of heavy reads.
Embed related data	Your application always reads and updates the related data at the same time. This solution simplifies your schema and prevents the need for $lookup operations.
Atlas Database Triggers	Your application can tolerate reading slightly stale data. Users can potentially see outdated data if they run a query immediately after an update, but before the trigger finishes updating the second collection.

Note: Learn more about data consistency.

Isolation

Each transaction is isolated from the other transactions to prevent data conflicts. This also helps database operations in relation to managing multiple entries and multi-level transactions. For example, if two users are trying to modify the same data (or even the same transaction), the DBMS uses a mechanism called a lock manager to suspend other users until the changes being made by the first user are complete.

How MongoDB handles isolation:

MongoDB employs a technique called snapshot isolation (e.g., each transaction appears to operate on a personal snapshot of the database taken at the start of the transaction). Transactions can read data from the “snapshot” of data committed at the time the transaction starts, and any conflicting updates will cause the transaction to abort.

In addition, MongoDB transactions support a transaction-level read concern and transaction-level write concern. Clients can set an appropriate level of read and write concern, with the most rigorous being snapshot read concern combined with majority write concern. Majority write concern means that the write operations have been durably committed to a calculated majority of the data-bearing nodes (configurable by the developer).

Durability

Durability guarantees that once the transaction completes and changes are written to the database, they are persisted. This ensures that data within the system will persist even in the case of system failures like crashes or power outages. The concept of durability is a key element in data reliability.

How MongoDB handles durability:

MongoDB creates an OpLog containing the disk location and bytes changed for each "write." If there is an unforeseen event (e.g., a power outage) during the writing of the transaction, the OpLog can be used when the system starts again to replay any writes that were not flushed to disk before the shutdown. In addition, operations are changed before written in the OpLog so

they're idempotent and can be retried multiple times. Transactions, or "writes," are flushed to disk roughly every 60 seconds by default.

Why are ACID transactions important?

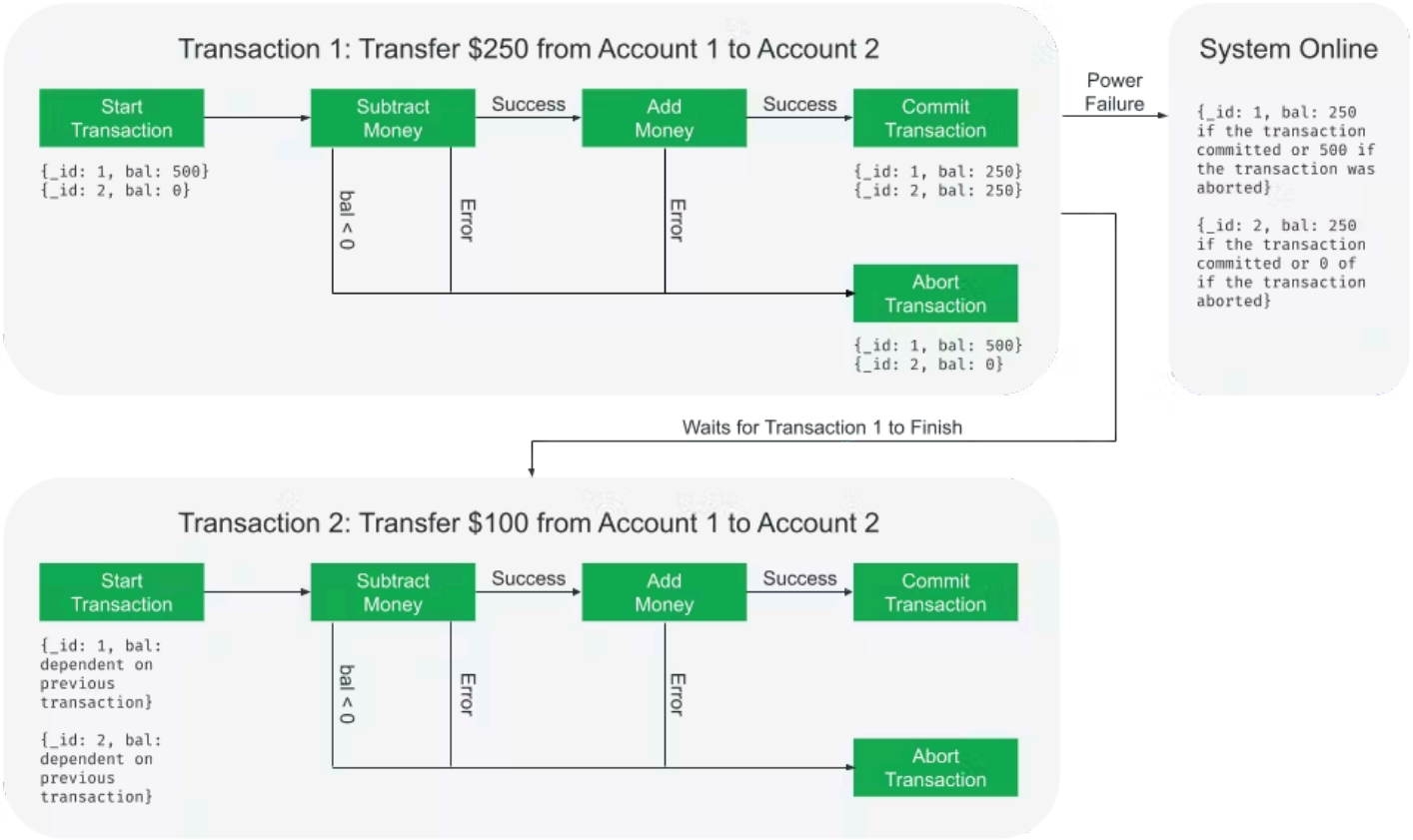
ACID transactions help maintain data integrity and reliability, while also ensuring that vital data which is subject to governmental or industry regulation (e.g., bank account, stock portfolio) meet required standards. Further, ACID compliance is often a prerequisite for implementing data replication and achieving high availability in distributed database systems.

ACID transaction example in DBMS

Using the NoSQL document database MongoDB Atlas, here is an example of how ACID multi-document transactions are handled and how ACID transactions guarantee alignment to minimum ACID property standards.

ACID multi-document transactions

Imagine you are building a function to transfer money from one bank account to another where each account is its own record. If money is successfully taken from the source account but never credited to the destination account, a serious accounting problem has been created. Conversely, if the destination account is credited but the source account is never debited, another serious accounting issue occurs.



The diagram demonstrates how the ACID properties impact the flow of transferring money from one bank account to another.

These two write operations have to either both happen or both not happen to keep the system and its data consistent. Further, this means that if any command in the transaction fails, the database must roll back (e.g., undo) all of the changes it had written in the course of the transaction.

```
await session.withTransaction(async () => {
   const subtractMoneyResults = await accountsCollection.updateOne(
       { _id: account1 },
       { $inc: { balance: amount * -1 } },
       { session });
   if (subtractMoneyResults.modifiedCount !== 1) {
       await session.abortTransaction();
       return;
   }

   const addMoneyResults = await accountsCollection.updateOne(
       { _id: account2 },
       { $inc: { balance: amount } },
       { session });
   if (addMoneyResults.modifiedCount !== 1) {
       await session.abortTransaction();
       return;
   }
});
```

Note: To learn more, visit the Node.js Quick Start GitHub repository to get a copy of the full code sample and run it yourself.

Impacts to remember

When dealing with multi-document transactions in a distributed system, remember that there is performance overhead impact which may affect resource constraints and performance goals. Further, since the database has to “lock” the involved resources to prevent concurrent writes from interfering with one another (e.g., transaction fails), other clients trying to write data may be stuck waiting for the transaction to complete which can impact application latency and user experience.

How do ACID transactions work in MongoDB?

MongoDB's document model allows related data to be stored together in a single document. The document model, combined with atomic document updates, obviates the need for transactions in a majority of use cases. Nonetheless, there are cases where true multi-document, multi-collection MongoDB transactions are the best choice.