# ACID PROPERTIES IN MONGODB

MongoDB uses a flexible, schema-less data model, which makes it particularly suitable for handling unstructured or semi-structured data. Two critical concepts in MongoDB for ensuring data integrity and performance are ACID properties and indexes. This report delves into these concepts, providing detailed explanations and examples of how they are implemented and used in MongoDB.
ACID stands for Atomicity, Consistency, Isolation, and Durability. These properties ensure reliable transactions in a database system. Understanding how MongoDB adheres to ACID principles helps in designing robust applications that require transactional integrity.

### 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 as in the case of an unwanted event, like a crash or power outage, we can be sure of the state of the database. The transaction would have either completed successfully or been rolled back if any part of the transaction failed.
If we continue with the above example, money is deducted from the source and if any anomaly occurs, the changes are discarded and the transaction fails.

### CONSISTENCY:

Consistency guarantees that changes made within a transaction are consistent with database constraints. This includes all rules, constraints, and triggers. If the data gets into an illegal state, the whole transaction fails.
Going back to the money transfer example, let’s say there is a constraint that the balance should be a positive integer. If we try to overdraw money, then the balance won’t meet the constraint. Because of that, the consistency of the ACID transaction will be violated and the transaction will fail.

### ISOLATION:

Isolation ensures that all transactions run in an isolated environment. That enables running transactions concurrently because transactions don’t interfere with each other.
For example, let’s say that our account balance is $200. Two transactions for a $100 withdrawal start at the same time. The transactions run in isolation which guarantees that when they both complete, we’ll have a balance of $0 instead of $100

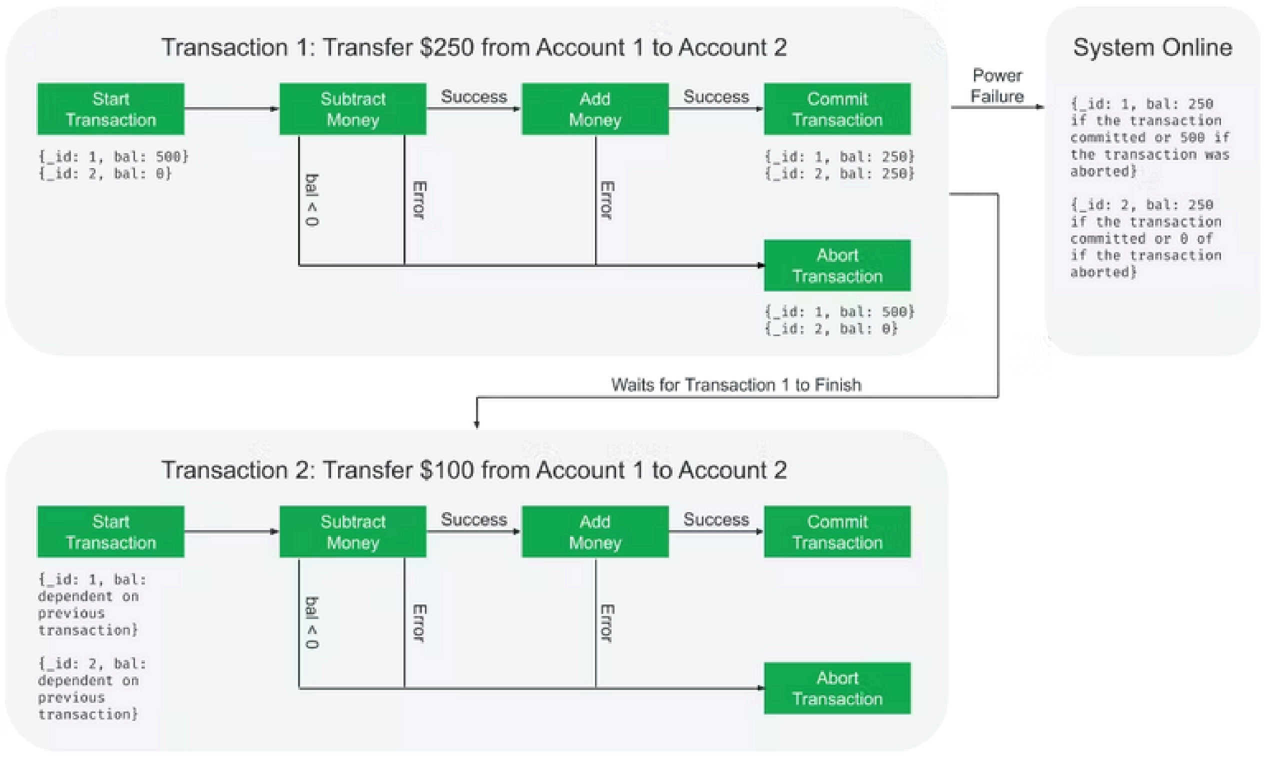
### DURABILITY:

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 ACID characteristics of transactions are what allow developers to perform complex, coordinated updates and sleep well at night knowing that their data is consistent and safely stored.

## WHAT IS AN ACID TRANSACTION EXAMPLE?

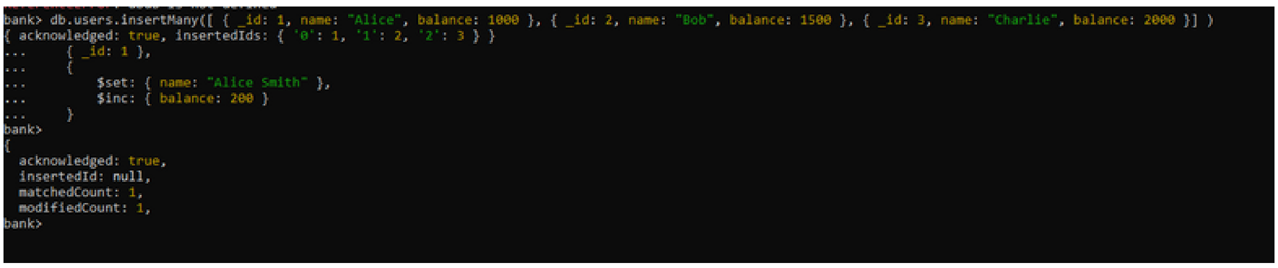
Let's continue with the banking example we discussed earlier in the article where money is being transferred from one account to another. Let's examine each of the ACID properties in this example:

- Atomicity: Money needs to both be removed from one account and added to the other, or the transaction will be aborted. Removing money from one account without adding it to another would leave the data in an inconsistent state.
- Consistency: Consider a database constraint that an account balance cannot drop below zero dollars. All updates to an account balance inside of a transaction must leave the account with a valid, non-negative balance, or the transaction should be aborted.
- Isolation: Consider two concurrent requests to transfer money from the same bank account. The final result of running the transfer requests concurrently should be the same as running the transfer requests sequentially.
- Durability: Consider a power failure immediately after a database has confirmed that money has been transferred from one bank account to another. The database should still hold the updated information even though there was an unexpected failure.



Transaction 1: Transfer $250 from Account 1 to Account 2
Start Transaction
Subtract Money
Success
Add Money
Success
Commit Transaction
{_id: 1, bal: 500}
{_id: 2, bal: 0}
bal < 0
Error
Error
{_id: 1, bal: 250}
{_id: 2, bal: 250}
Abort Transaction
{_id: 1, bal: 500}
{_id: 2, bal: 0}
Power Failure
System Online
{_id: 1, bal: 250 if the transaction committed or 500 if the transaction was aborted}
{_id: 2, bal: 250 if the transaction committed or 0 of if the transaction aborted}
Waits for Transaction 1 to Finish
Transaction 2: Transfer $100 from Account 1 to Account 2
Start Transaction
Subtract Money
Success
Add Money
Success
Commit Transaction
{_id: 1, bal: dependent on previous transaction}
{_id: 2, bal: dependent on previous transaction}
bal < 0
Error
Error
Abort Transaction

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



```
bank> db.users.insertMany([ { _id: 1, name: "Alice", balance: 1000 }, { _id: 2, name: "Bob", balance: 1500 }, { _id: 3, name: "Charlie", balance: 2000 }] )
{ acknowledged: true, insertedIds: { '0': 1, '1': 2, '2': 3 } }
...     { _id: 1 },
...     {
...         $set: { name: "Alice Smith" },
...         $inc: { balance: 200 }
...     }
bank>
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
bank>
```

This operation ensures that both the name update and the balance increment are performed as a single atomic operation. If either operation fails, the entire update is rolled back, and no changes are applied to the document.

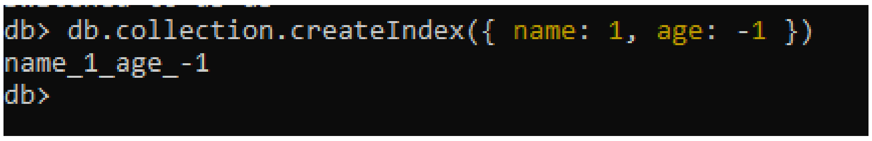
## INDEXES IN MONGODB

Indexes are special data structures that store a small portion of the data set in an easy-to-traverse form. They improve query performance and enable efficient execution of various operations.

**Types of Indexes:**

1. **Single Field Indexes:** Indexes on a single field.
2. **Compound Indexes:** Indexes on multiple fields.
3. **Multikey Indexes:** Indexes on array fields, creating an index key for each array element.
4. **Geospatial Indexes:** Indexes supporting geospatial queries.
5. **Text Indexes:** Indexes supporting text search on string content.
6. **Hashed Indexes:** Indexes with hashed values of the indexed field.

The default name for an index is the concatenation of the indexed keys and each key's direction in the index (1 or -1) using underscores as a separator. For example, an index created on



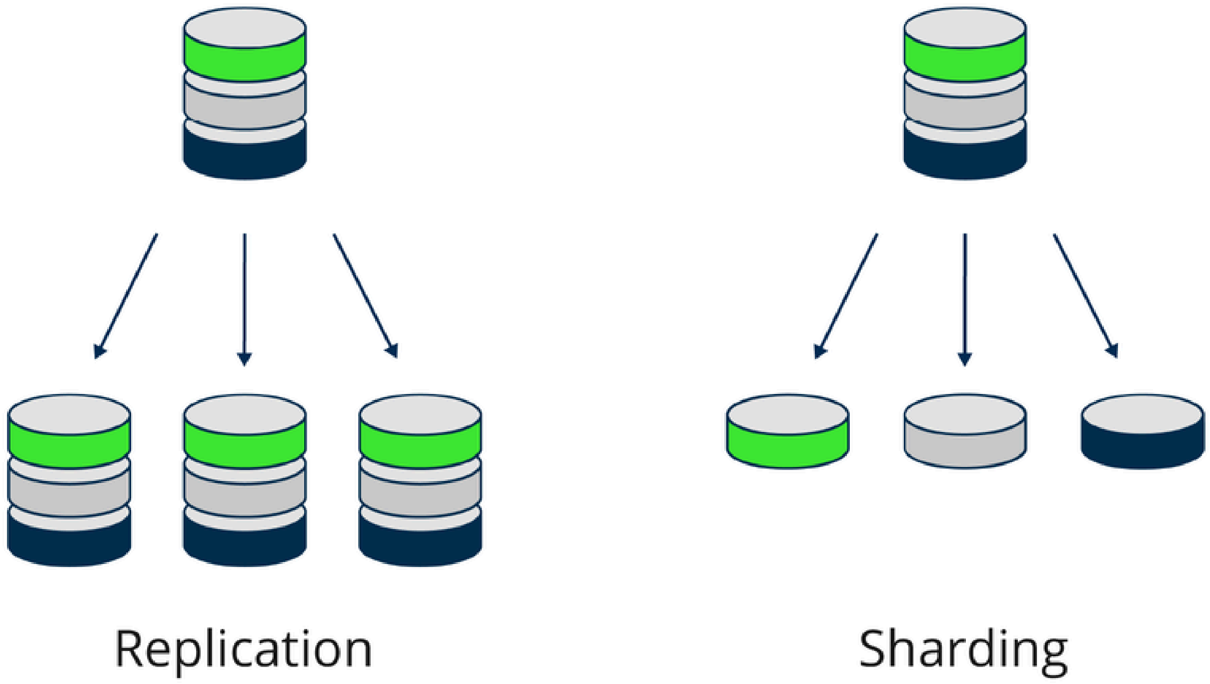
```
db> db.collection.createIndex({ name: 1, age: -1 })
name_1_age_-1
db>
```

You cannot rename an index once created. Instead, you must drop and recreate the index with a new name.

Incorporating indexes into your MongoDB collections is a fundamental practice for improving query efficiency and overall database performance. By carefully designing and maintaining indexes, developers can ensure that their applications remain responsive and capable of handling large volumes of data and complex queries. Understanding the different types of indexes and their appropriate use cases is essential for making informed decisions about index implementation and optimization.

## SHARDING AND REPLICATION

MongoDB is designed for scalability, high performance, and high availability. Two fundamental concepts that enable these capabilities are sharding and replication. Sharding allows MongoDB to distribute data across multiple servers, while replication ensures data redundancy and availability. This report explains these concepts in detail, including their mechanisms and how they can be implemented in MongoDB.



**Sharding in MongoDB:**
Sharding is the process of distributing data across multiple machines to support horizontal scaling. Sharding is crucial for handling large datasets and high throughput operations by distributing the load and storage across multiple servers.
**Benefits of Sharding:**

- Scalability: By adding more shards, the system can handle increased load and larger datasets.
- High Throughput: Distributing data across multiple shards allows for parallel processing of queries, improving performance.
- Fault Tolerance: Even if one shard fails, the rest of the system can continue to operate, provided the application can tolerate partial data unavailability.

**Replication in MongoDB:**

Replication is the process of synchronizing data across multiple servers to ensure redundancy, high availability, and disaster recovery. In MongoDB, replication is implemented through replica sets.

**Benefits of Replication:**

- High Availability: If the primary server fails, an automatic election takes place to select a new primary from the secondaries, ensuring continuous availability.
- Data Redundancy: Multiple copies of data are maintained across different servers, protecting against data loss.
- Read Scalability: Secondary servers can serve read operations, distributing the read load and improving performance.

Sharding and replication are critical components of MongoDB's architecture, enabling it to handle large-scale, high-performance, and highly available applications. Sharding distributes data across multiple servers for horizontal scaling, while replication ensures data redundancy and high availability. Properly implementing and managing these features allows MongoDB to efficiently support demanding applications and workloads.