ACID AND INDEXES

❖ DEFNATIONS:

ACID is an acronym that stands for atomicity, consistency, isolation, and durability. Together these 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.

❖ Let's look into the concepts:

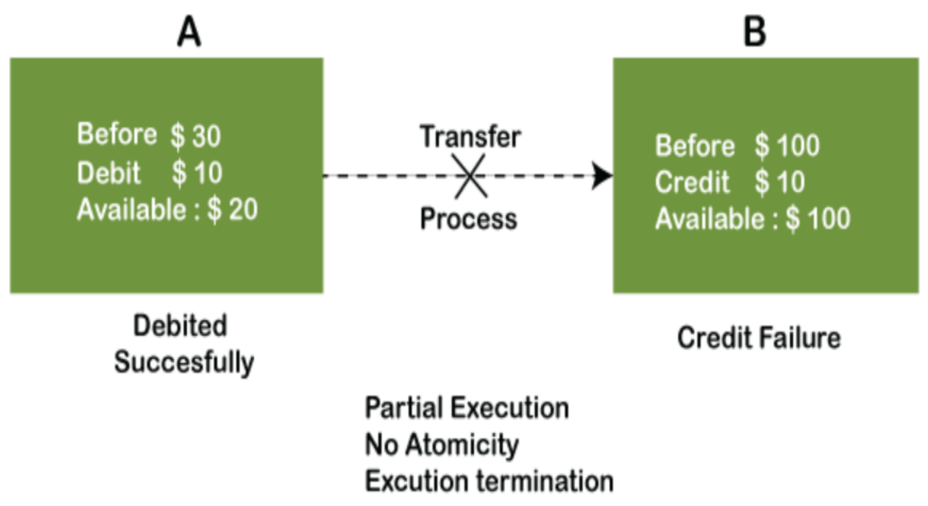
1.ATOMICITY:

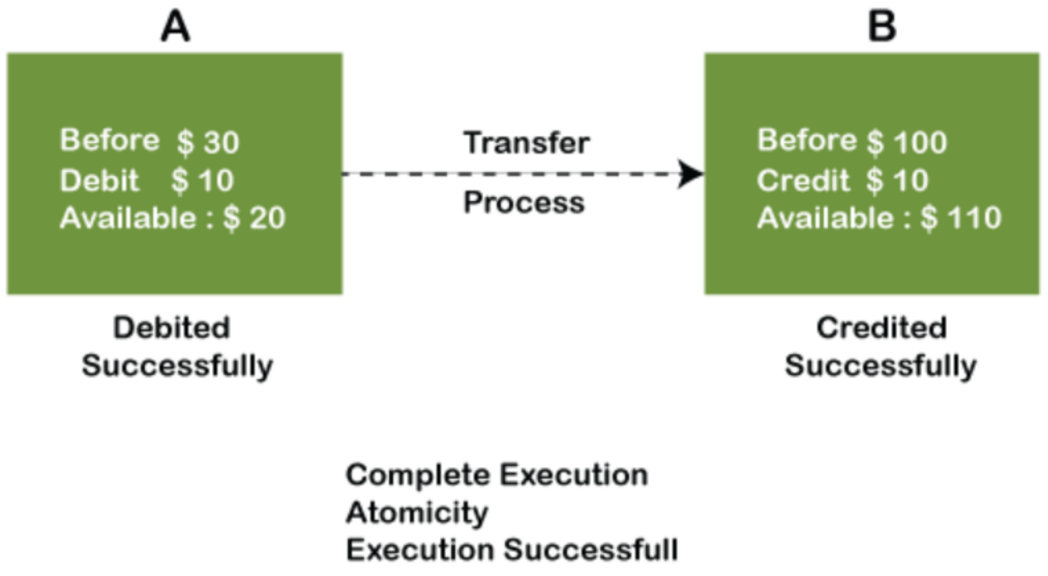
Atomicity is all based around this idea of togetherness.

When carrying out any kind of database transaction, it often consists of multiple operations. With atomicity, either every operation succeeds or none of them do.

This is important because the operations can have an impact on each other, so one failing can lead to unexpected results

Think of a financial transaction, for example. You are paying a friend $250 for a holiday you are going on. The whole transaction would consist of the money leaving your account and arriving in the recipient's account. If there was no atomicity, it is possible that money leaves your account but doesn't arrive at the other end, resulting in you being debited the money but still owing the recipient.

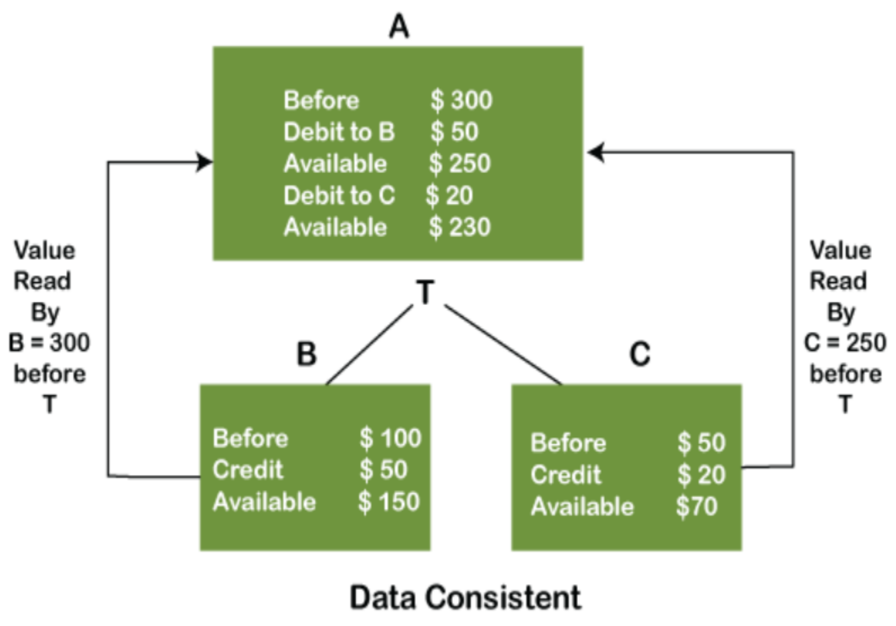




2.CONSISTENCE:

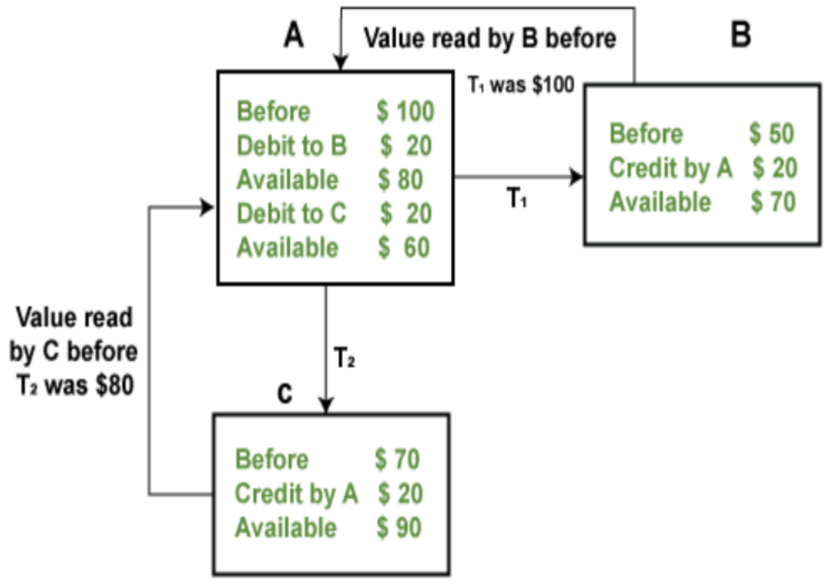
The word **consistency** means that the value should remain preserved always. In DBMS, the integrity of the data should be maintained, which means if a change in the database is made, it should remain preserved always. In the case of transactions, the integrity of the data is very essential so that the database remains consistent before and after the transaction. The data should always be correct.

Example:



3.ISOLATION:

Example: If two operations are concurrently running on two different accounts, then the value of both accounts should not get affected. The value should remain persistent. As you can see in the below diagram, account A is making T1 and T2 transactions to account B and C, but both are executing independently without affecting each other. It is known as Isolation.



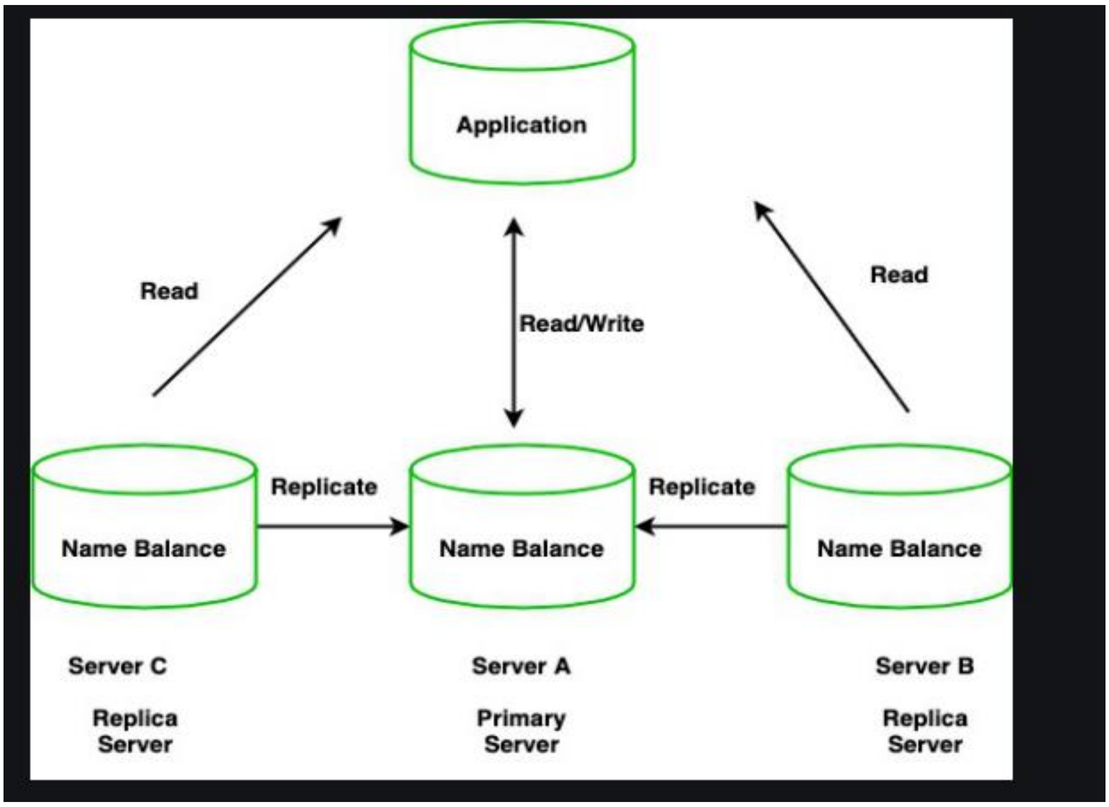
Isolation - Independent execution of T_1 & T_2 by A

4.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.

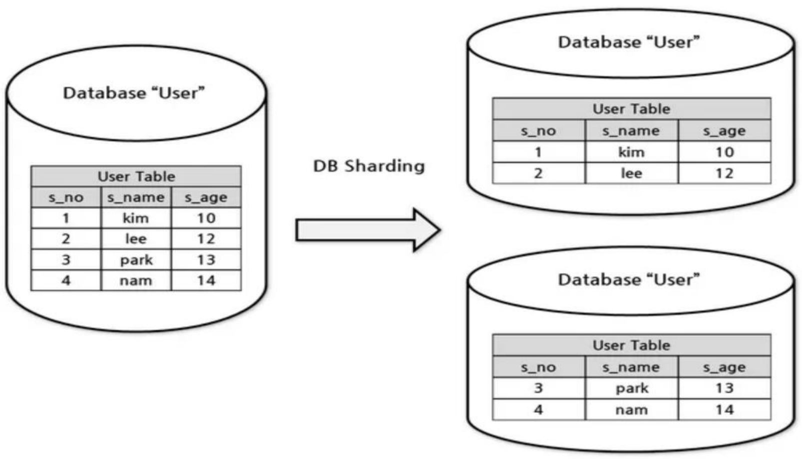
❖ REPLICATION (MASTER SLAVE):

Master-slave replication enables data from one database server (the master) to be replicated to one or more other database servers (the slaves). The master logs the updates, which then ripple through to the slaves.



❖ SHARDING:

Sharding is a method for distributing data across multiple machines. MongoDB uses sharding to support deployments with very large data sets and high throughput operations



User Table

s_no	s_name	s_age
1	kim	10
2	lee	12
3	park	13
4	nam	14

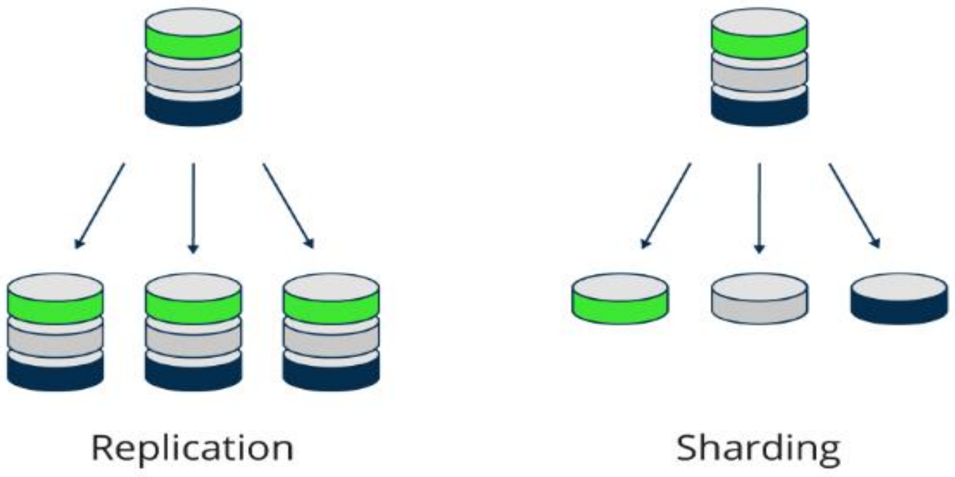
User Table

s_no	s_name	s_age
1	kim	10
2	lee	12

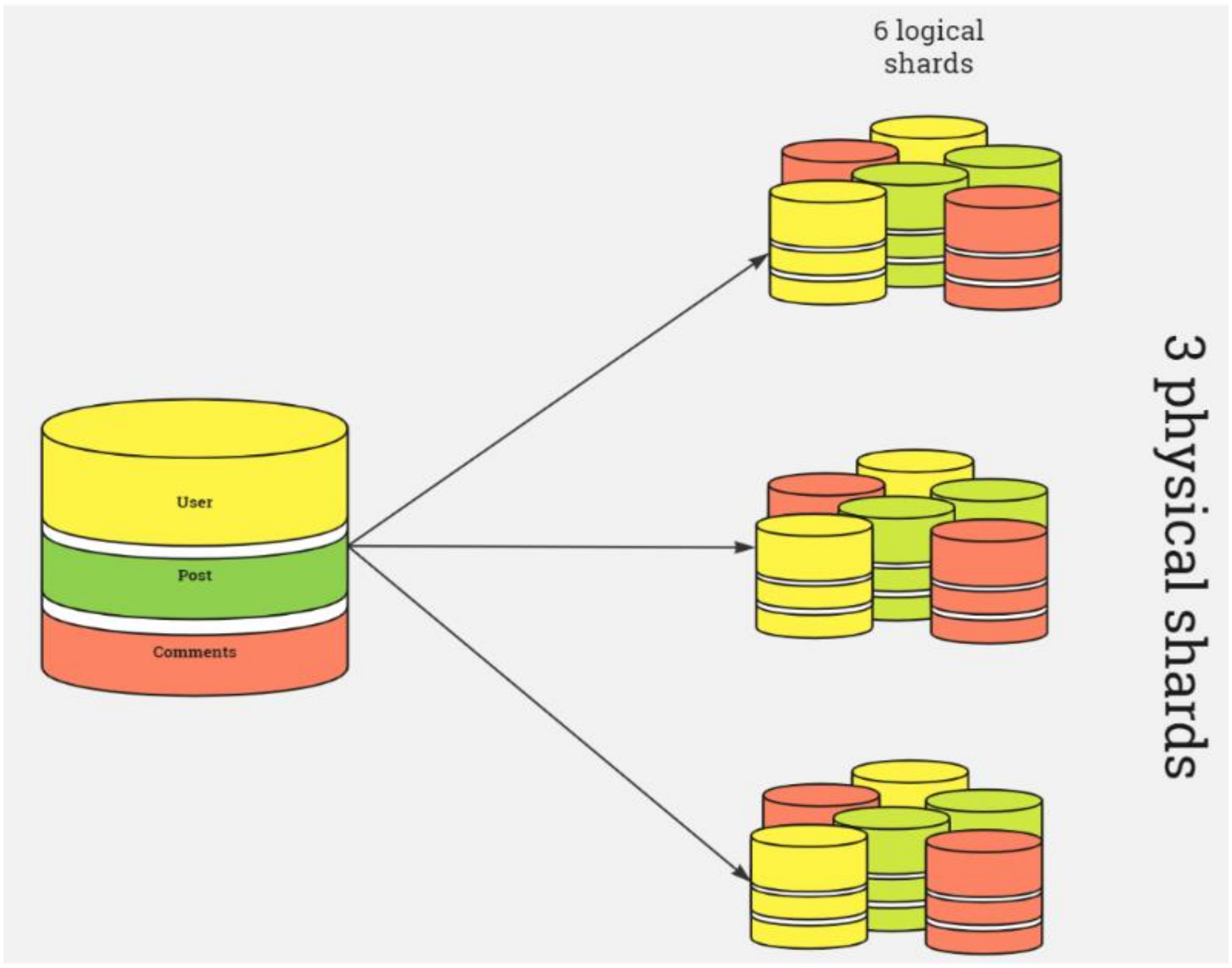
User Table

s_no	s_name	s_age
3	park	13
4	nam	14

❖ **REPLICATION VS SHARDING:**



❖ **REPLICATION + SHARDING:**



❖ INDEXES:

MongoDB provides various types of indexes to optimize query performance based on different data structures and query patterns.

Single Field Indexes

- Indexes a single field of a document.
- Improves performance for queries that filter, sort, or use equality matches on that field.
- Example: db.products.createIndex({ price: 1 })

Compound Indexes

- Indexes multiple fields in a specified order.
- Optimizes queries that involve multiple fields in the same order.
- Example: db.products.createIndex({ category: 1, price: -1 }) (category ascending, price descending)

Multikey Indexes

- Indexes each element of an array field individually.
- Useful for queries that involve filtering or searching within array elements.
- Example: db.products.createIndex({ tags: 1 })

Text Indexes

- Creates a full-text index on one or more text fields.
- Supports text search queries using operators like $text.
- Example: db.products.createIndex({ description: "text" })

Geospatial Indexes

- Indexes geographical coordinates for efficient spatial queries.
- Supports queries like finding points within a certain radius.
- Two types:
  - 2d indexes: For planar data.
  - 2dsphere indexes: For spherical data (lat/long coordinates).
- Example: db.places.createIndex({ loc: "2dsphere" })

Hashed Indexes

- Creates a hashed index of a specified field.
- Suitable for equality matches but not for range queries.
- Example: db.users.createIndex({ userId: "hashed" })

Unique Indexes

- Ensures that each value in the indexed field is unique.
- Useful for preventing duplicate values.
- Example: db.users.createIndex({ email: 1 }, { unique: true })

Sparse Indexes:

- Indexes only documents where the specified field exists.
- Can improve performance for queries involving sparse fields.
- Example: db.products.createIndex({ tags: 1 }, { sparse: true })

TTL Indexes

- Automatically deletes documents after a specified expiration time.
- Useful for implementing data expiration policies.
- Example: db.sessions.createIndex({ expiresAt: 1 }, { expireAfterSeconds: 3600 })

Wildcard Indexes

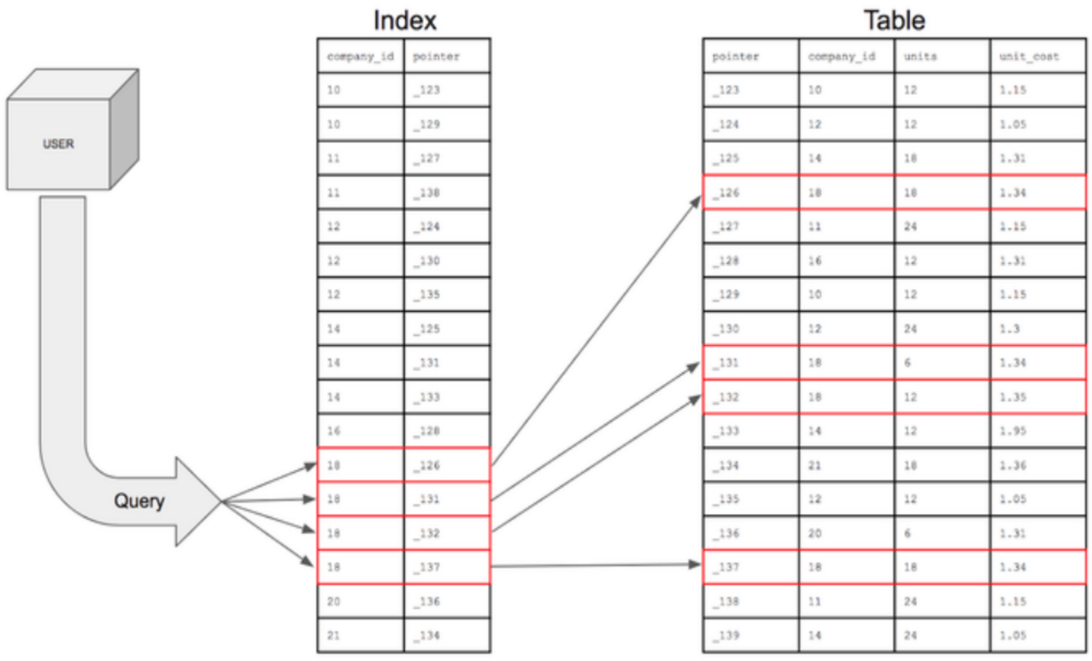
- Creates an index on a partial field path.
- Useful for indexing nested documents efficiently.
- Example: db.addresses.createIndex({ "location.city": 1 })

Choosing the Right Index

The choice of index depends on the specific query patterns and data characteristics of your application. Consider the following factors:

- **Query selectivity:** How specific are your queries?
- **Data distribution:** How are values distributed across the indexed field?
- **Index size:** The size of the index can impact performance.
- **Write operations:** Indexing can impact write performance.

By carefully selecting and creating appropriate indexes, you can significantly improve the performance of your MongoDB applications.



company_id	pointer
10	_123
10	_129
11	_127
11	_138
12	_124
12	_130
12	_135
14	_125
14	_131
14	_133
16	_128
18	_126
18	_131
18	_132
18	_137
20	_136
21	_134

pointer	company_id	units	unit_cost
_123	10	12	1.15
_124	12	12	1.05
_125	14	18	1.31
_126	18	18	1.34
_127	11	24	1.15
_128	16	12	1.31
_129	10	12	1.15
_130	12	24	1.3
_131	18	6	1.34
_132	18	12	1.35
_133	14	12	1.95
_134	21	18	1.36
_135	12	12	1.05
_136	20	6	1.31
_137	18	18	1.34
_138	11	24	1.15
_139	14	24	1.05

CHECK FOR MORE DETAILS ABOUT INDEXES:

https://g.co/gemini/share/7e3cd8a53fcf

db.collection.getIndexes()

db.products.getIndexes()