Scalability

How can we make our application scalable?

Content

- What is data vs Information
- History of Database
- Why scalability.
- How to make our application scalable
- Types of Scalability

What is data?

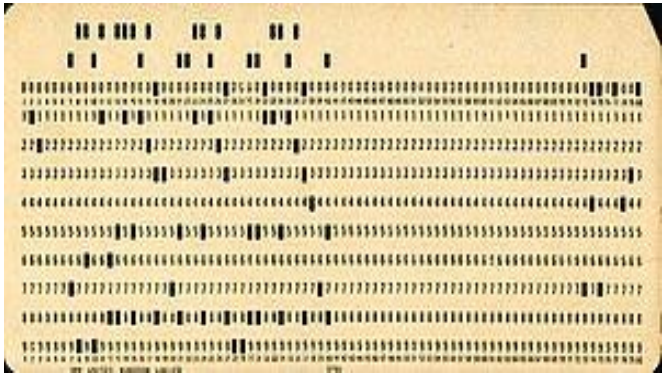
- People have used the term data to refer to computer information and this information was either transmitted or stored.
- But data can be texts or numbers written on papers,bytes or bits inside memory or it could also be thoughts stored inside a person’s mind.
- It’s unprocessed facts, figures, and symbols that are collected from various sources. Data on its own does not have meaning until it is interpreted.
- A good example could be a collection that holds productId,userId,and amount inside a database.

What is information?

- It’s information when data is processed, organized, interpreted, and structured.
- Information is essentially data made valuable and accessible.
- It provides context and insights.
- A good example could be the report generated from a table to provide a valuable insight on a product that is purchased most by customers.

History of data storage and processing

1950 and early 1960’s



contd....

- It was the time where magnetic tapes and punched cards were used as storage devices.
- **Punched cards** were physical cards with holes punched in specific positions to represent data. They were also processed in sequence, one card after another.
- Each punched card could store information in the form of holes punched in specific positions on the card.
- A standard punched card typically had 80 columns and 12 rows, meaning it could store 80 characters of data (one character per column) using a specific coding system like Hollerith or IBM's EBCDIC.
- Not only were punched cards used to store data records (e.g., employee information, inventory lists), but they were also used to store program instructions

contd……

- **Magnetic tapes** stored in a linear sequence, meaning you can only read or write data in the order it appears on the tape, from start to end.
- Data records are placed one after the other on the tape. Each record is stored in sequence, so you can only read or write data in the order it was placed unless you use additional indexing or marking techniques.’

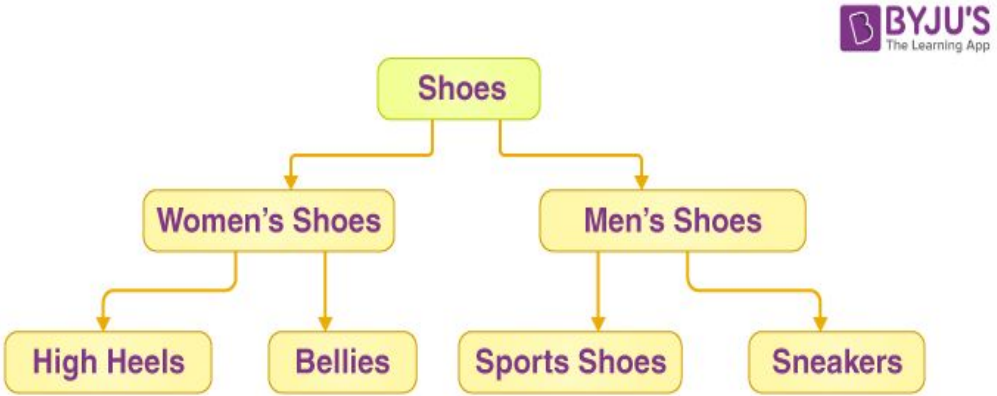
contd...

Late 1960s and 1970s

- The introduction of hard disk, where you can access data at specific point without traversing through the previous ones,changed the way data was stored and processed.
- **Hierarchical** and **Network** data models were introduced.

contd...

Hierarchical Data Model



contd...

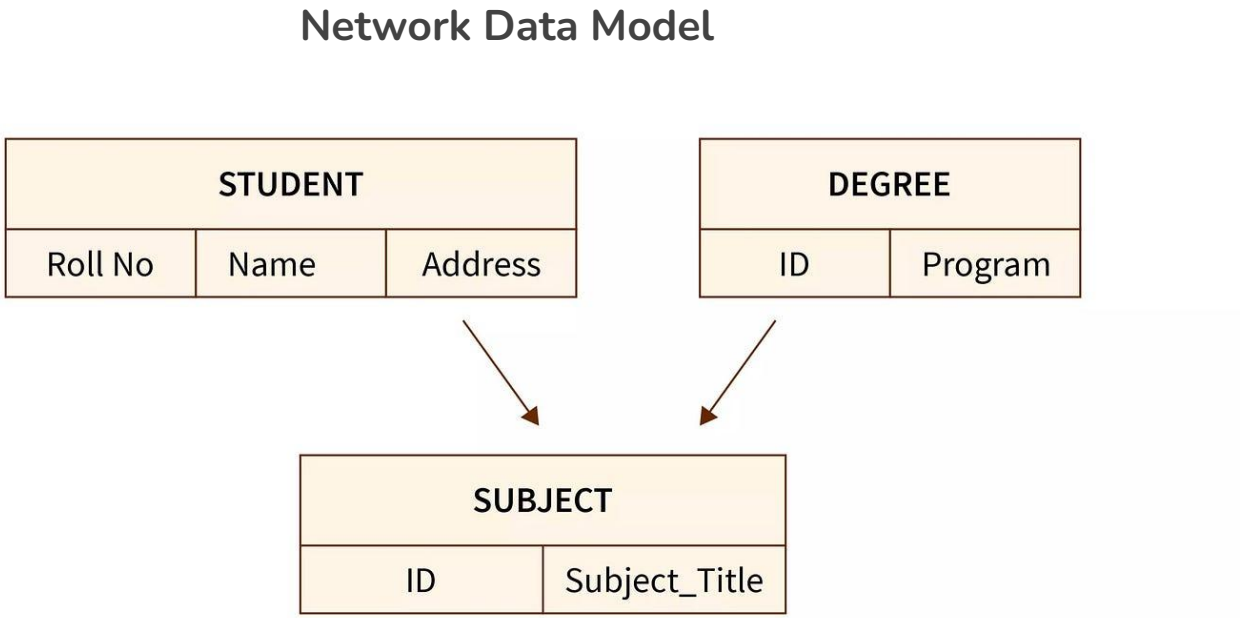
- This model organizes data in tree like structure.
- Within the hierarchy a higher layer is perceived as the parent of the segment that is immediately beneath it which is known as the child.
- Starts from root extends like a tree adding nodes to the lymph node.
- It only had one way approach i.e to get to a certain data the whole tree had to be traversed starting from roots to nodes.
- Depicts only one-one and one-many relation ships.

contd...

Cons of Hierarchical data model

- Inflexible structure, unsuitable for complex relationships.
- Potential for data redundancy and inconsistency
- Limited querying capabilities for non-hierarchical data.
- Difficult to update or reorganize data.
- Scalability issues with large or complex hierarchies.

contd...


Network Data Model
STUDENT
Roll No
Name
Address
DEGREE
ID
Program
SUBJECT
ID
Subject_Title

contd...

- Was developed to overcome the setbacks of the hierarchical data model
- It represented complex data relationships much more effectively than the hierarchical data model
- It organizes data using two fundamental concepts called **Records** and **Sets**
- It allows a record to have more than one parent.
- It allows one-many ,one-one and many to many relationships.

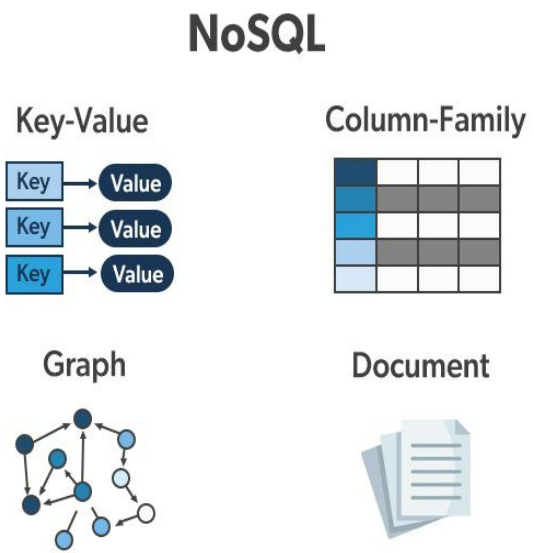
contd...

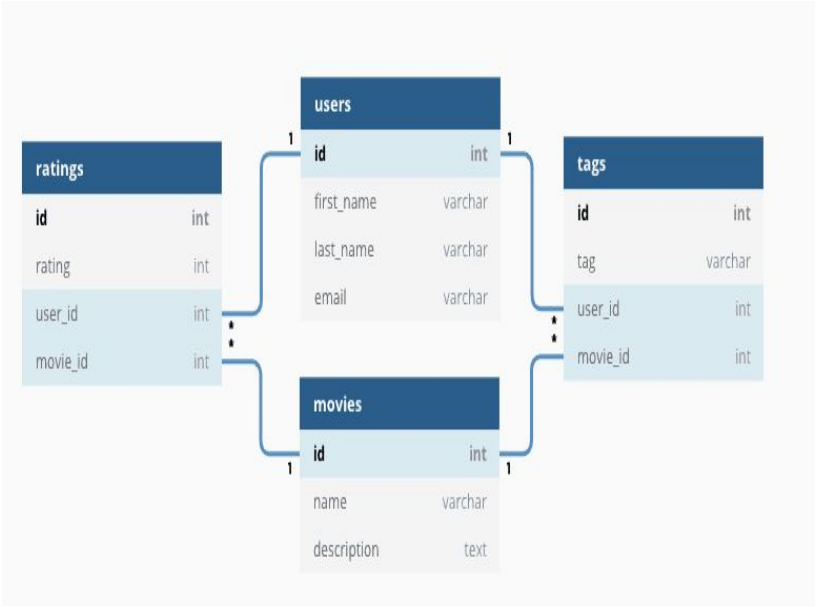
Cons of Network data model

- Complex Design Difficult to design, implement, and understand.
- Complex Query Complex and cumbersome querying; lacks ad-hoc querying capabilities.
- Difficult to modify Changes to the database structure are challenging and costly
- Scalability Issue Performance can degrade with large or complex databases.
- Lack of standardization

contd...

Relational(SQL) and NOSQL Data Model





contd...

- While relational(sql)databases store data in the form of tables with row and column,the nosql databases store data in different forms like document,key-value or graph.
- Came with a lot of advantages that are tailor-made for different use cases
- Data model and structure,schema flexibility,scalability.....

What is Scalability

- The above data base model changes came in action due to the ongoing increase of data exponentially .
- As data used by applications become immense,making the application scalable with the ongoing growth is a no brainer.
- Scalability is different from performance.A high performance application might be blazing fast to query a single user data but can’t handle multiple concurrent users.
- Scalability is about ensuring performance remains consistent even as the workload grows.

Application level scalability

- Here we are talking about ability of software to handle growing amount of work gracefully.
- The work could be **Increased user traffic**,**Higher data volume** or **more complex operations**

Architecture

- Let's say our application is monolithic-all the components of an application—such as the user interface, business logic, data access layer, and third-party integrations—are tightly integrated and deployed as a single unit.
- The application needs to be scaled due to high traffic from user requests to a specific service.

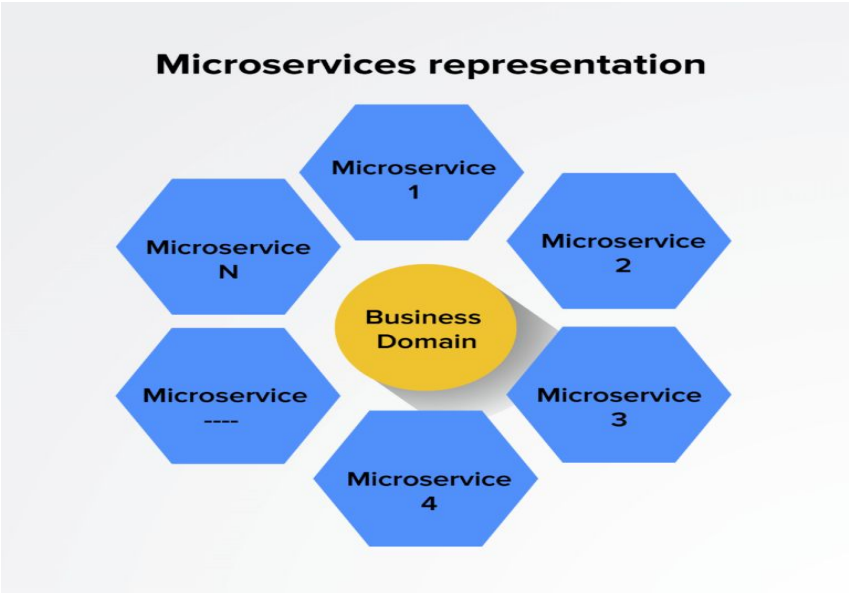
contd...

Setbacks

- Scaling a monolithic application horizontally involves creating multiple instances (or copies) of the entire application and distributing the incoming traffic among these instances.
- Inefficient use of resources-Resources like **RAM** due to duplicate cached datas and loading the entire application to memory,**File descriptors** due to duplicate file accesses and socket connections.Unnecessary request to third party integrations **...**
- It leads to Over Provisioning and High cost .Thus **Microservices.**

contd...

Microservices



contd...

- It is a design approach where an application is built as a collection of small, independent, loosely coupled services that communicate with each other through well-defined APIs.

Key characteristics

- **Independence -** It comprises loosely coupled components that can be easily developed replaced and scaled individually.Can be beneficial for
  - Team organization
  - Easier maintenance and updates
  - Independant deployment-CICD down time only for that one service
  - Independent scaling

contd...

- **Flexibility in Technology -** Teams can choose the most suitable technology stack for each service. This allows for using the best tool for each specific job, rather than being tied to a single technology across the entire application.
- **Resilience and Fault Isolation-**If one service fails, it does not necessarily bring down the entire application. Other services can continue to operate normally, making the system more resilient to failures.

Cons of microservice architecture

- **Operational Overhead -** Each microservice needs its own deployment, monitoring, logging, and management, which can lead to increased operational overhead. Use **kubernetes** to solve this.

Contd...

- **Data management challenges -**Ensuring data consistency across services can be difficult, especially in distributed systems. Transactions that span multiple services are more complex to manage compared to monolithic architectures.
    - Eventual consistency
    - Distributed Transactions
    - Network Failures
    - Event ordering issues
- **Inter-Service Communication-** Communication between services introduces latency and can be a point of failure. The network dependency can complicate the architecture, especially when services need to interact frequently.

contd...

- **Testing complexity -** End-to-end testing in a microservices architecture is more complex due to the need to test the interactions between multiple services. Automated testing, including integration tests, becomes essential but is more challenging to implement.
- **Security Concerns -** Each service exposes its API, increasing the potential attack surface. Securing each service and managing authentication and authorization across services adds to the security burden.

Contd...

When to use microservice architecture

- **Need for scalability-**Your application needs to scale to handle large volumes of traffic or data.
- Microservices allow you to scale specific services that need more resources without scaling the entire application.
- For example, you can scale the Product Catalog Service independently from the User Authentication Service, based on the specific needs of each component.
- **Heterogeneous technology stack**-Your application requires different technologies, programming languages, or databases for different components.

Contd...

- **High Availability and Fault Tolerance-**Microservices can increase fault tolerance by isolating failures to specific services. If one service goes down, it doesn't necessarily take down the entire system.
- In addition let's look at some application performance optimization methods before deep diving in to database scalability.

Indexing

- Indexing is a technique used in databases, including MongoDB, to improve the speed and efficiency of data retrieval operations

contd...

- An **index** is a data structure that stores a small portion of a collection’s or table's data in a way that makes it easier and faster for the database to find and retrieve specific records
- An index contains entries for the values of the indexed field(s) along with pointers (references) to the actual documents or rows in the database.
- When querying and indexed field Instead of scanning every document in the collection, the database can use the index to jump directly to the documents that match the query conditions.
- This makes data retrieval easy.

contd...

- But it comes with a trade-off.
- **Write Performance overhead-**Every time data is inserted, updated, or deleted, the database must also update the indexes, which can slow down write operations.
- For collections with a high write-to-read ratio, indexes are expensive because each insert must also update any indexes.
- **Storage cost**

contd...

Connection pooling

- It involves creating and managing a pool of database connections that can be reused, rather than creating a new connection every time a request is made
- This eliminates the overhead of establishing a new connection, which can be time-consuming and resource-intensive.
- useful in situations where database requests are made frequently, such as web applications, where multiple users simultaneously access the database.

contd...

What happens if we don't use connection pooling

- **Time Consumption**: Each new connection involves several steps that require time, from network communication to authentication and resource allocation.
- **Resource Consumption**: New connections consume server resources, such as memory and CPU, which can become a bottleneck if many connections are established frequently.
- **Performance Impact**: These factors combined can slow down application performance, particularly in high-traffic environments where connections are frequently opened and closed.

contd...

Advantage of connection pooling

- **Improved Performance**: Connection pooling significantly reduces the overhead of establishing new database connections, resulting in faster response times and better overall performance.
- **Efficient Resource Utilization:** By reusing connections, connection pooling reduces the number of connections needed, freeing up system resources and improving scalability.
- **Cost Savings**: Connection pooling reduces the need for expensive hardware upgrades by optimizing resource utilization, ultimately saving costs for businesses

contd...

Best Practices

- **Optimal Pool Size:** Determine the optimal pool size based on the expected number of concurrent connections. Having too few connections can cause performance bottlenecks, while having too many can lead to resource wastage.
- **Timeouts and Connection Lifetime**: Specify appropriate timeouts for connections to avoid keeping idle connections for too long, which can lead to resource wastage.

Types of scalability

Vertical Scaling(Scaling Up)

- Involves adding more resources (such as CPU, RAM, or storage) to a single server or system to handle increased load or performance demands
- While vertical scaling can be effective up to a certain point, it has inherent limitations due to hardware constraints.
- **CPU Cores**: There is a physical limit to the number of CPU cores that a single server can support. Even as processors with more cores become available, there is a maximum number of cores that can be efficiently utilized by a single system

contd...

- **Memory Capacity**: Servers have a maximum amount of RAM that can be installed.
- **Storage**: Although storage capacities continue to increase, there is a limit to how much storage can be physically added to a single server.
- **Heat and Power Consumption**: Increasing the power of a single server can lead to higher heat generation and power consumption, which can require more robust cooling systems and impact operational costs.
- **Software Limitations**: Not all software or databases are designed to take full advantage of increased vertical scaling.
- **Downtime and Migration**: Upgrading hardware typically requires system downtime or migration, which can impact availability and performance during the transition period.

contd...

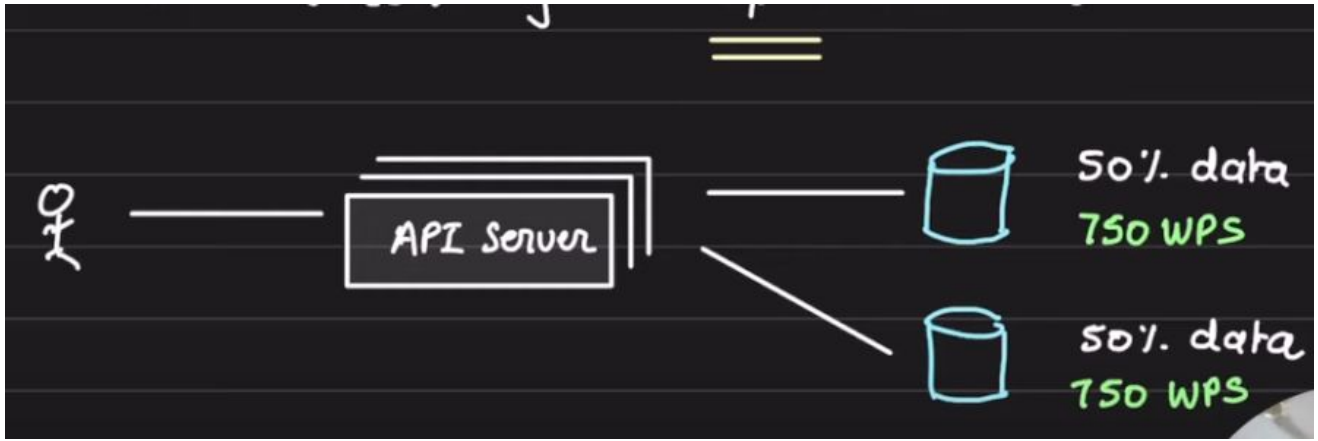
Horizontal scaling(Scaling out)

- It involves adding more nodes or servers to distribute the load and handle increased capacity
- For database systems, horizontal scaling may involve adding additional database servers to handle more queries or store more data. This can be done in various ways,
- **Sharding and partitioning** and **Replication.**

Sharding and partitioning

contd...

- When we reach at a point where vertical scaling cant save our life we scale out horizontally.
- It's basically creating separate database servers to handle requests.
- Each database server is called a Shard.



contd...

- On the above image we have created two database servers and partitioned the data in two 2 to be served by the database servers.
- We **Shard** a **database** and **partition** the **Data.**
- Sharded data are stored in chunks.
- **Shard key** is used to distribute the collection's documents across shards. The shard key consists of a field or multiple fields in the documents.

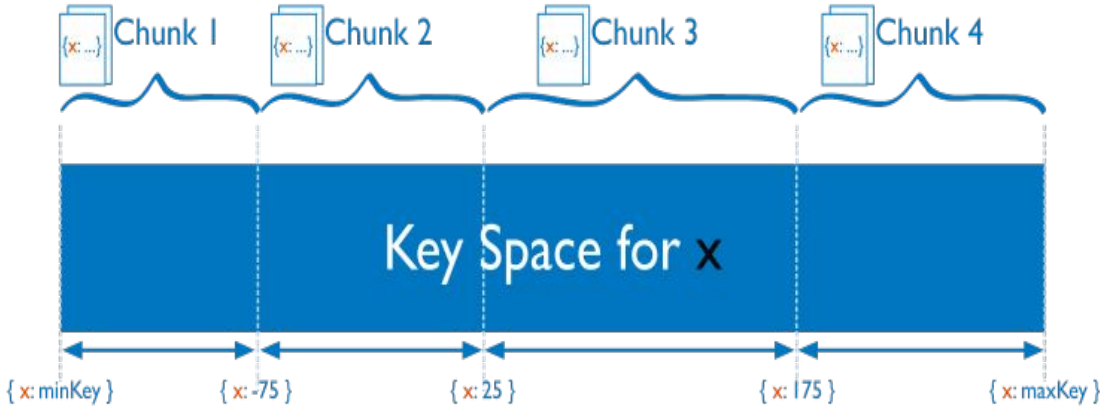
Sharding strategies

contd...

Ranged Sharding

- Ranged sharding involves dividing data into ranges based on the shard key values.

  Each chunkis then assigned a range based on the shard key values.
- Documents with “close” shard key values are likely to be in the same chunk or shard.



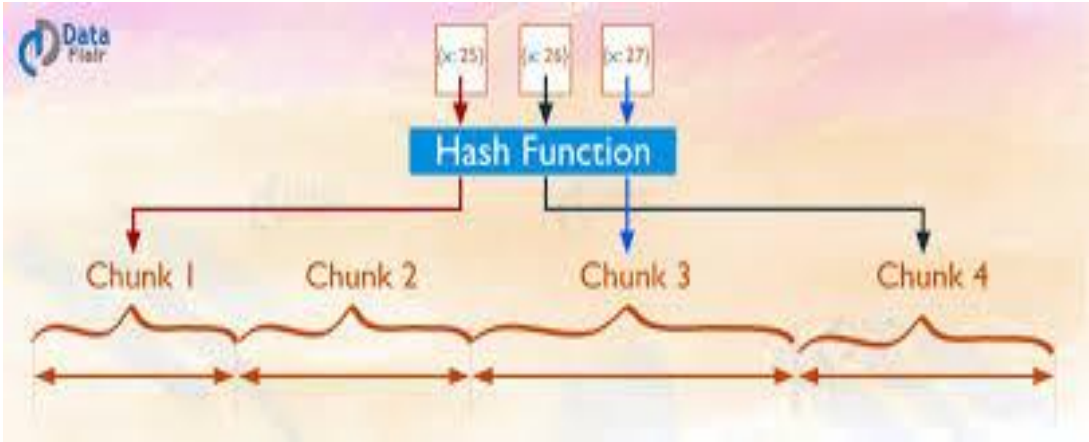
contd...

- Allows for targeted operations as a query can be routed to only the shards that contain the required data.
- However, both read and write performance may decrease with poor shard key selection.

contd...

Hashed Sharding

- Hashed Sharding involves computing a hash of the shard key field's value. Each chunk is then assigned a range based on the hashed shard key values.



contd...

- While a range of shard keys may be "close", their hashed values are unlikely to be on the same chunk.
- Data distribution based on hashed values facilitates more even data distribution, especially in data sets where the shard key changes monotonically.
- However, hashed distribution means that range-based queries on the shard key are less likely to target a single shard, resulting in more cluster wide.

Contd...

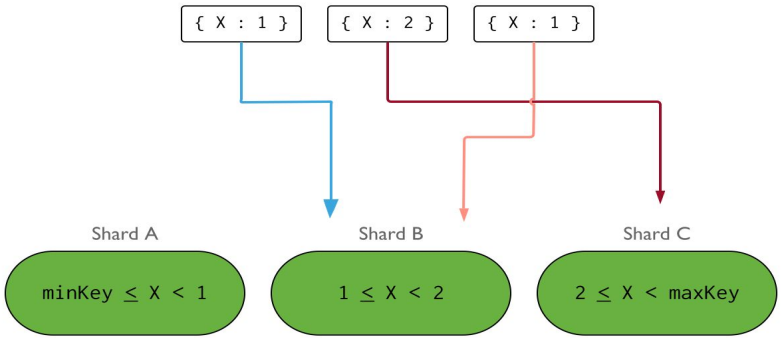
What should we consider when choosing sharding key?

- The choice of shard key affects the creation and distribution of chunks across the available shards.
- The distribution of data affects the efficiency and performance of operations within the sharded cluster.
- To avoid data inconsistency and work overload on one cluster on a shard let's look at the metrics we need to consider for choosing sharding key.

contd...

Cardinality

- Cardinality is the measure of the number of elements within a set of values of the shard key.
- It determines the maximum number of chunks that can be created.



contd...

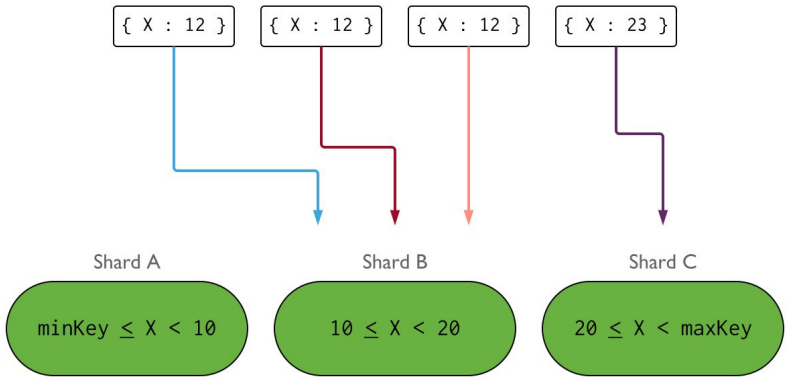
- A shard key with low cardinality reduces the effectiveness of horizontal scaling in the cluster.
- A shard key with high cardinality does not, on its own, guarantee even distribution of data across the sharded cluster. The frequency also matters.

Frequency

- Represents how often a given shard key value occurs in the data.
- If the majority of documents contain only a subset of the possible shard key values, then the chunks storing the documents with those values can become a bottleneck within the cluster.

contd...

- In some cases, chunks can grow beyond the specified chunk size but cannot undergo a split.
- This reduces the effectiveness of horizontal scaling within the cluster.



contd...

- If your data model requires sharding on a key that has high frequency values or low cardinality used compound shard keys.

Monotonically Changing Shard Keys

- A shard key on a value that increases or decreases monotonically is more likely to distribute inserts to a single chunk within the cluster.
- If the shard key value is always increasing, all new inserts are routed to the chunk with maxKey as the upper bound.

contd...

- If the shard key value is always decreasing, all new inserts are routed to the chunk with minKey as the lower bound.
- If your data model requires sharding on a key that changes monotonically, consider using Hashed Sharding.

Sharding Query Patterns

- When choosing a sharding key we need to carefully consider the queries that we run the most.Lets take a look at an example.

contd...

- Consider having an order table that holds a userId as a reference to the user who ordered the food.
- We chose the shard key for the order schema to be the order date and for the user schema to be the userId.
- The sharding strategies are range based sharding and hashing respectively
- Because there is a huge variation in our sharding keys and strategies the probability orderId 1 to be served by the same shard of its creator userId 1 is very unlikely.

contd...

- To avoid this problem one solution could be sharding both the user and the order collection with the same sharding key and strategy.
- We can use the **userId** as the sharding key.

contd...

Replication

- It is the process of creating and maintaining multiple copies of the same data in different location.
- It can happen **synchronously** or **Asynchronously**
- **Synchronously** - means data is constantly copied to the main server and replica servers simultaneously.

contd...

- **Asynchronously** - the data is copied to the main server and only copied to replica servers in batches.
- Synchronous replication ensures no data loss at the cost of network bandwidth.
- We can have:-
- **Full replication**-maintaining a full copy of the entire data set on every node in a distributed system.

contd...

Pros:

- Ensures high availability and fault tolerance.
- Improves read performance by distributing the load.
- Provides strong data redundancy and disaster recovery capabilities.

Cons:

- High storage requirements, as each node stores a full copy of the data.
- Potentially slower writes due to replication overhead.
- Scalability challenges as the data set grows.

contd...

- **partial replication**-maintaining a full copy of the entire data set on every node in a distributed system.

Pros:

- **Reduced Storage Requirements**: Each node only stores part of the data, making it more storage-efficient.
- **Improved Write Performance**: Writes are faster since fewer nodes are involved in the replication process.
- **Customizability**: You can tailor which data is replicated to which nodes, optimizing for performance and efficiency.

contd...

Cons:

- **Complexity in Querying**: Queries that need data from multiple nodes may be more complex and slower.
- **Data Availability**: If a node fails and it holds unique data not replicated elsewhere, that data might become unavailable.

contd...

In conclusion

- Data scalability is a critical aspect of modern application architecture, enabling systems to efficiently manage increasing amounts of data and user demands.
- Scalable databases are designed to grow in capacity and performance without compromising on reliability or speed
- Successful data scalability is about selecting the right combination of techniques and technologies to meet the specific needs of your application, ensuring that your database can grow alongside your business without sacrificing performance or stability