Sharding

Ashish Pratap Singh

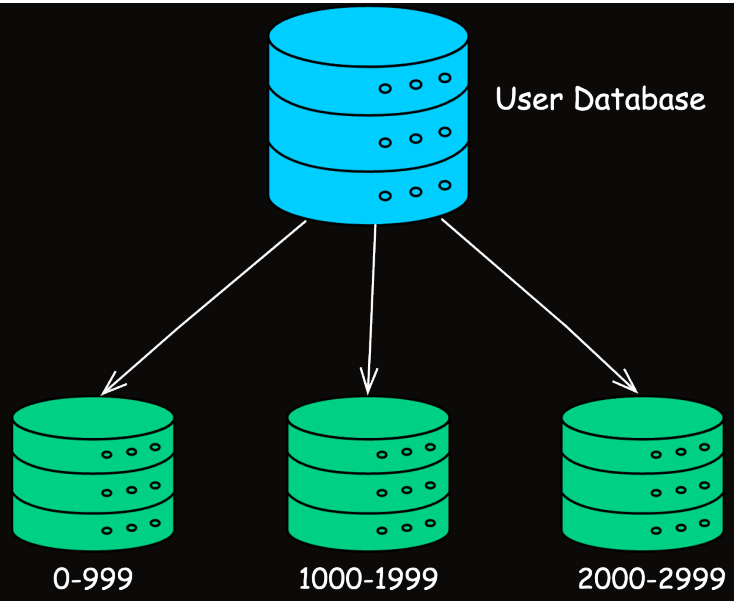
4 min read

Imagine a social media site like Instagram, which has over **1 billion** active users.

Think about what would happen if it tried to keep all the **user profile** data on a single server.

Due to limited scalability of a single machine, it would quickly run out of storage space and slow down leading to performance issues.

But what if we divided the user base into smaller groups based on a key like **userId** and stored each group on separate servers?



Example:

- Group 1: Users with IDs 0-999
- Group 2: Users with IDs 1000-1999
- Group 3: Users with IDs 2000-2999

Now, a single server only deals with subset of data.

Distributing data in this way makes it easier to scale and manage more users.

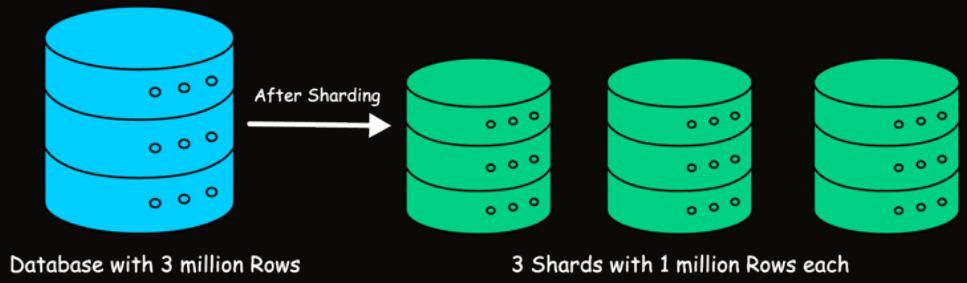
This is the idea behind **Database Sharding**.

In this article, we will explore what database sharding is, it's benefits, how it works, challenges that come with it and the best practices for implementing it.

1. What is Database Sharding?

Database sharding is a horizontal scaling technique used to split a large database into smaller, independent pieces called **shards**.

These shards are then distributed across multiple servers or nodes, each responsible for handling a specific subset of the data.



Sharding is widely used by large-scale applications and services that handle massive amounts of data, such as:

- **Social Networks:** Instagram uses sharding to manage billions of user profiles and interactions.
- **E-commerce Platforms:** Amazon employs sharding to handle massive product catalogs and customer data.
- **Search Engines:** Google relies on sharding to index and retrieve billions of web pages.
- **Gaming**: Online gaming platforms use sharding to handle

millions of players and vast amounts of game data.

2. Benefits of Sharding

1. **Improved Performance**: By distributing the data across multiple nodes, sharding can significantly reduce the load on any single server, resulting in faster query execution and improved overall system performance.
2. **Scalability**: Sharding allows databases to grow horizontally. As data volume increases, new shards can be added to spread the load evenly across the cluster.
3. **High Availability**: With data spread across multiple shards, the failure of a single shard doesn't bring down the entire system. Other shards can continue serving requests, ensuring high availability.
4. **Geographical Distribution:** Sharding allows you to strategically place shards closer to your users, reducing latency and improving the user experience.
5. **Reduced Cost**: Instead of scaling vertically by investing in more powerful and expensive hardware, sharding allows for cost-effective scaling by utilizing commodity hardware.

3. How Does Sharding Work?

The sharding process involves several key components including:

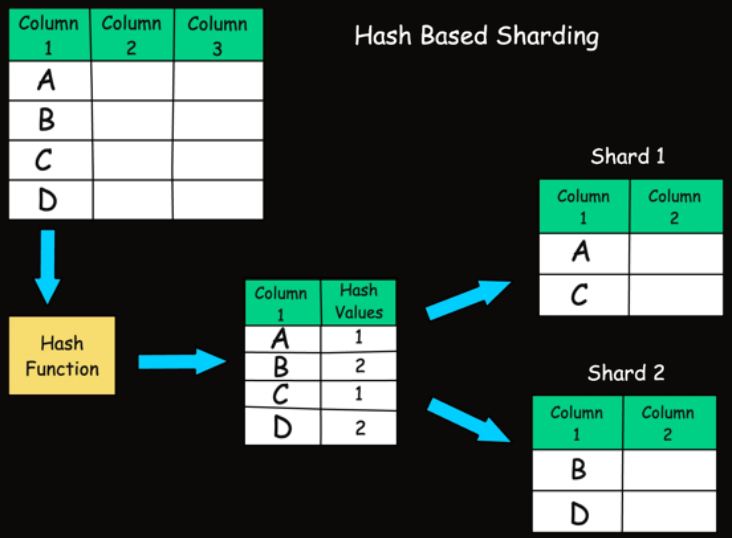
1. **Sharding Key**: The shard key is a unique identifier used to determine which shard a particular piece of data belongs to. It can be a single column or a combination of columns.
2. **Data Partitioning**: Partitioning involves splitting the data into shards based on the shard key. Each shard represents a portion of the total data set. Common strategies to partition database are **range-based, hash-based, and directory-based sharding.**
3. **Shard Mapping**: Creating a mapping of shard keys to shard locations.
4. **Shard Management:** The shard manager oversees the distribution of data across shards, ensuring data consistency and integrity.
5. **Query Routing**: The query router intercepts incoming queries and directs them to the appropriate shard(s) based on the shard key.

4. Sharding Strategies

Hash-Based Sharding

Data is distributed using a hash function, which maps data

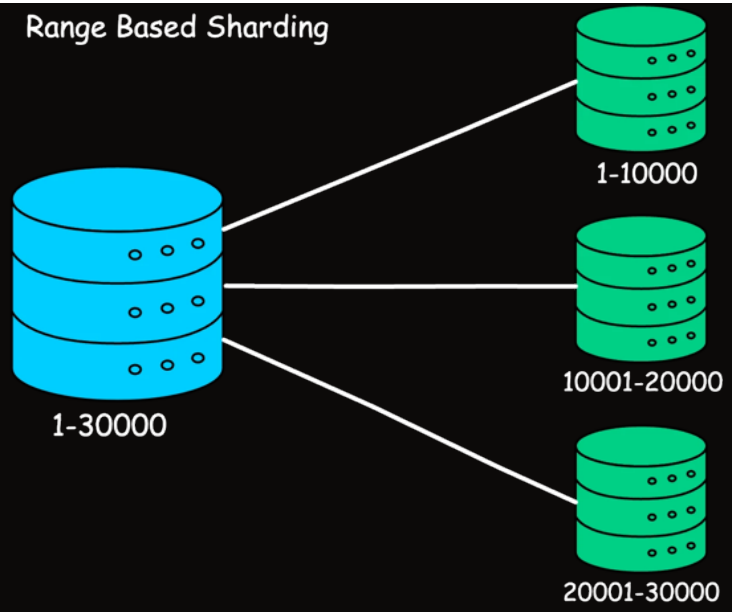
to a specific shard.



- **Example:** `Hash(user_id) % 2` determines the shard number for a user, distributing users evenly across 2 shards.

Range-Based Sharding

Data is distributed based on a range of values, such as dates or numbers.



- **Example:** Shard 1 contains records with IDs from 1 to 10000, Shard 2 contains records with IDs from 10001 to 20000, and so on.

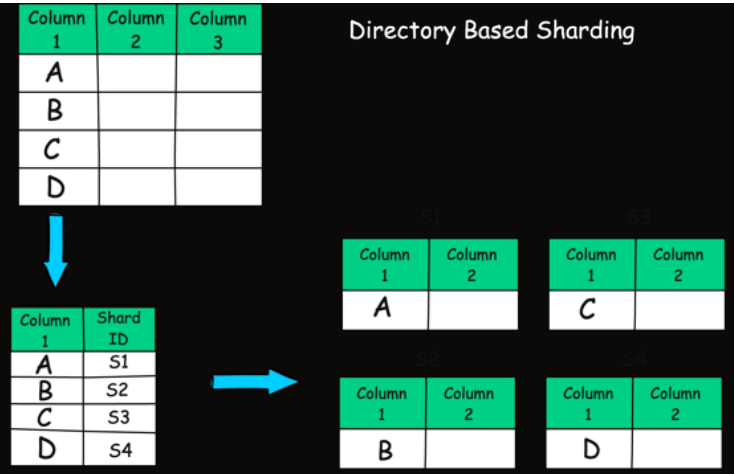
Geo-Based Sharding

Data is distributed based on geographic location.

- **Example:** Shard 1 serves users in North America, Shard 2 serves users in Europe, Shard 3 serves users in Asia.

Directory-Based Sharding

Maintains a lookup table that directly maps specific keys to specific shards.



5. Challenges with Sharding

1. **Complexity**: Sharding introduces additional complexity, requiring careful planning and management.
2. **Data Consistency**: Maintaining data consistency across shards can be challenging, especially when data needs to be joined or aggregated from multiple shards.
3. **Cross-shard Joins:** Performing joins across multiple shards can be complex and computationally expensive.
4. **Data Rebalancing:** As data volumes grow, shards may become unevenly distributed, requiring rebalancing to maintain optimal performance.

6. Best Practices for Sharding

- **Choose the Right Sharding Key**: Select a sharding key

that ensures an even distribution of data across shards and aligns with the application's access patterns.

- **Use Consistent Hashing**: Implement a consistent hashing algorithm to minimize data movement when adding or removing shards.
- **Monitor and Rebalance Shards**: Regularly monitor shard performance and data distribution. Rebalance shards as needed to ensure optimal performance and data distribution.
- **Handle Cross-Shard Queries Efficiently**: Optimize queries that require data from multiple shards by using techniques like query federation or data denormalization.
- **Plan for Scalability**: Design your sharding strategy with future growth in mind. Consider how the system will scale as data volume and traffic increase.

7. Conclusion

To summarize, database sharding is a powerful technique for scaling databases horizontally and handling large amounts of data, but it does come with additional complexity and requires thoughtful planning and understanding of application’s data access patterns.

Before implementing sharding, think about whether the benefits outweigh the costs or if there is a simpler solution.