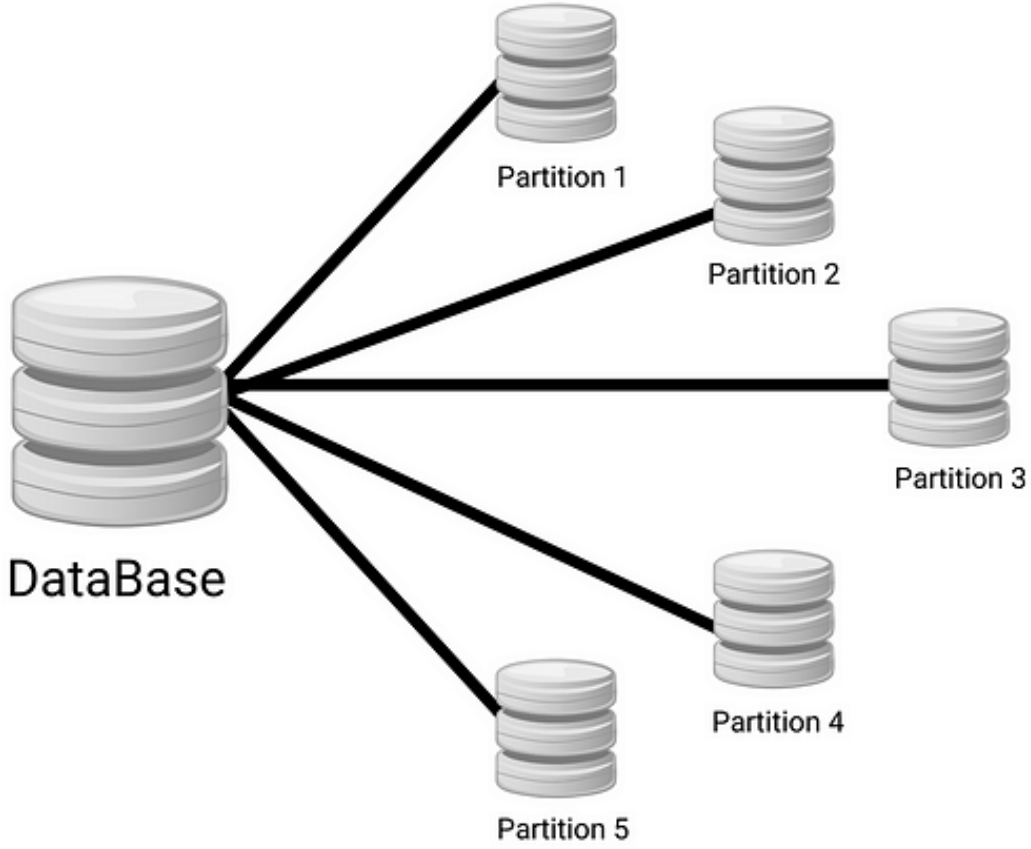
Database Partitioning (Sharding) in System Design

What is data partitioning?

Database partitioning is the backbone of modern distributed database management systems. It is a process of dividing a large dataset into several small partitions placed on different machines. In other words, It is a way of partitioning data like tables or index-organized tables into smaller pieces so that data can be easily accessed and managed.

- It distributes data across several partitions to improve database availability, scalability, and query processing performance. The combined data from all partitions is the same as the data from the original database.
- The partition architecture is transparent to the client application, where the client application keeps talking to the database partitions as if it was talking to a single database.



What are the problems solved by database partitioning?

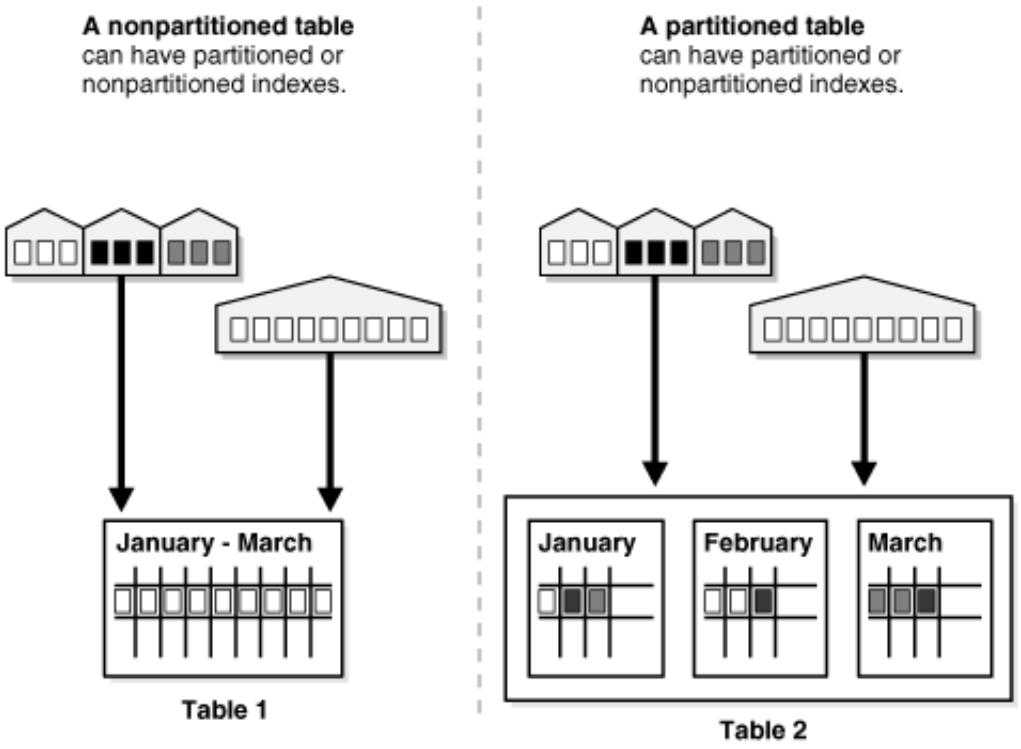
With the growth in services and user base, it becomes tricky for a single server or database to function efficiently. We may experience lower performance with the architecture of a single database server. Here is some situation that could arise:

- Database operations become slower.
- Network bandwidth starts reaching the saturation level.
- The database server starts running out of disk space at some point.

Database partition helps us fix all the above challenges by distributing data across several partitions. Each partition may reside on the same machine (coresident) or different machines (remote). The idea of co-resident partitioning is to reduce individual indexes size, and the amount of I/O needed to update records. Similarly, the concept of remote partitioning is to increase the bandwidth access to data by having more

RAM, avoiding disk access, or having more network interfaces and disk I/O channels available.

A view of partitioned tables



When to Partition a Table?

There are several scenarios when partitioning data can be beneficial:

- When tables are too large to fit in memory.
- When new data is added or updated on a daily basis, such as a table containing historical data where only the current month's data is updated and the other 11 months' data are read-only.
- When the table data needs to be distributed across different storage devices. It becomes easier to perform query tasks if the data is distributed and stored across different servers or systems.

However, not all cases require data partitioning. It is important to carefully consider the specific needs of the system before deciding whether or not to use partitioning.

Why do we need data partitioning?

There are several benefits to using data partitioning:

Improved Availability: By partitioning the database, we can ensure the high availability of our application. Individual partitions can be managed independently, so if one partition becomes unavailable, the other partitions can still execute database queries successfully. This helps to avoid a single point of failure for the entire dataset and increases the overall availability of the service.

Note: Keeping data in different partitions helps the database administrator do backup and recovery operations on each partition, independent of the other partitions. This could allow the active partition of the database to be made available sooner so access to the system can continue while the inactive data is still being restored.

Improved Scalability: Every hardware has certain capacity limitations. As traffic increases, the performance of the service can decrease. Data partitioning allows us to scale out the service by distributing the data across multiple partitions, removing any limitations on scalability.

Improved Security: Data partitioning can also improve security by storing sensitive and non-sensitive data in different partitions. This allows for better management and increased security for sensitive data.

Improved Query Performance: Instead of querying the entire database, data partitioning allows the system to query smaller components, improving overall performance.

Improved Data Manageability: Data partitioning divides tables and indexes into smaller, more manageable units. This "divide and conquer" approach to data management allows for easier maintenance of particular table partitions.

Data Partitioning Methods

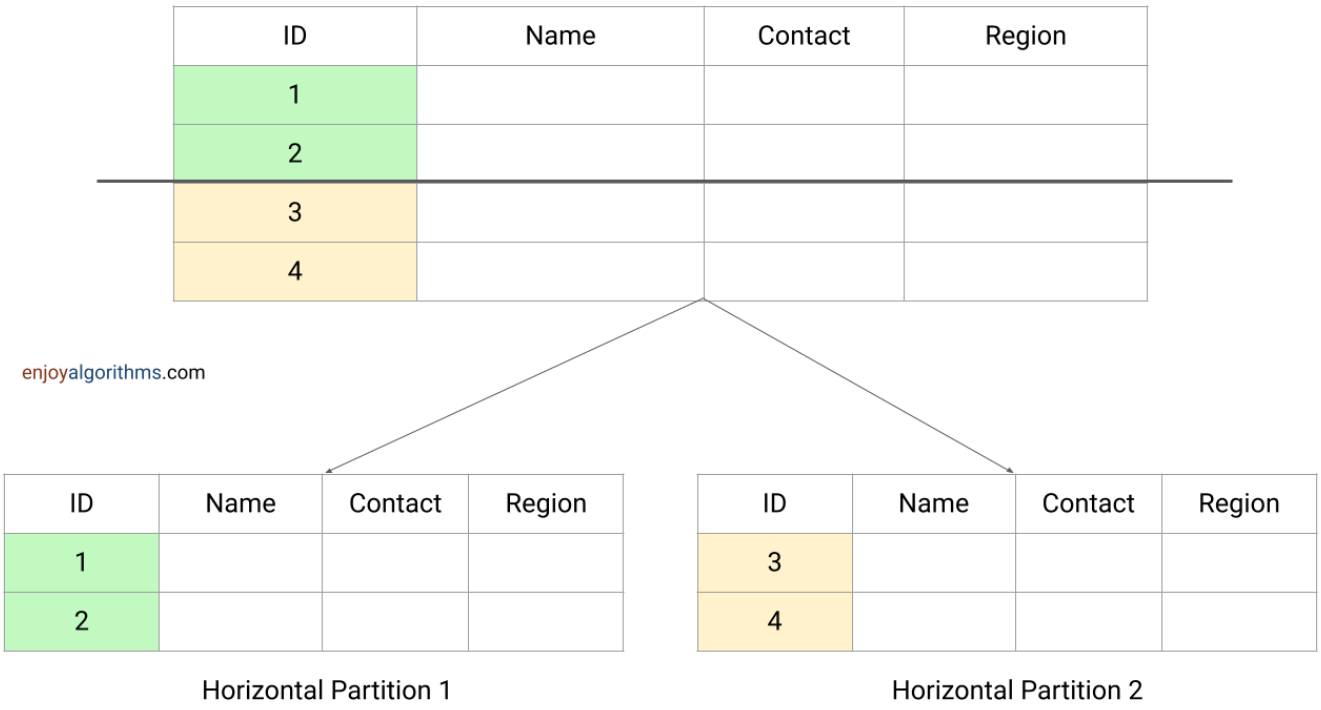
Data Partitioning can be done through various strategies to distribute the database into separate and smaller databases. Broadly there are three

different data partitioning strategies used. Let's have a look at each one of them.

Horizontal Partitioning or Database Sharding

Horizontal partitioning, also known as database sharding, is a strategy for splitting table data horizontally based on the range of values defined by a partition key. In this approach, the table is divided into smaller, more manageable tables, with each row of the table being assigned to a single partition. This allows each partition to be managed independently of the others.

- The partition key is responsible for distributing the data among all the partitions. When a query is made using the partition key, the database can determine which partition it needs to query. This ensures that the time required for a query is as minimal as possible, as the load is spread over more computers.
- It is important to balance the number of requests between partitions to ensure that none become overloaded or idle, as this can affect the performance of the service and make it more prone to failure.



Suppose a large database containing multiple rows of customer data has

a slow query performance. So we can think to partition the table into two separate tables horizontally. The first table would contain the first half of the customer data, and the second table would contain the second half. This allows us to query either partition 1 or partition 2, depending on the partition key. For example, suppose we store the contact info for customers. In that case, we can keep the contact info starting with name A-H on one partition and contact info starting with name I-Z on another partition.

The benefit of the horizontal partition: The horizontal partition scheme is the most straightforward partitioning method. It involves dividing the database into separate partitions that have the same schema as the original database. This makes it easy to answer queries without having to combine data from multiple partitions.

The disadvantage of the horizontal partition: Data may not be evenly distributed across the partitions. For example, if there are many more customers with names that fall in the range of A-H than in the range I-Z, the first partition may experience a much heavier load than the second partition.

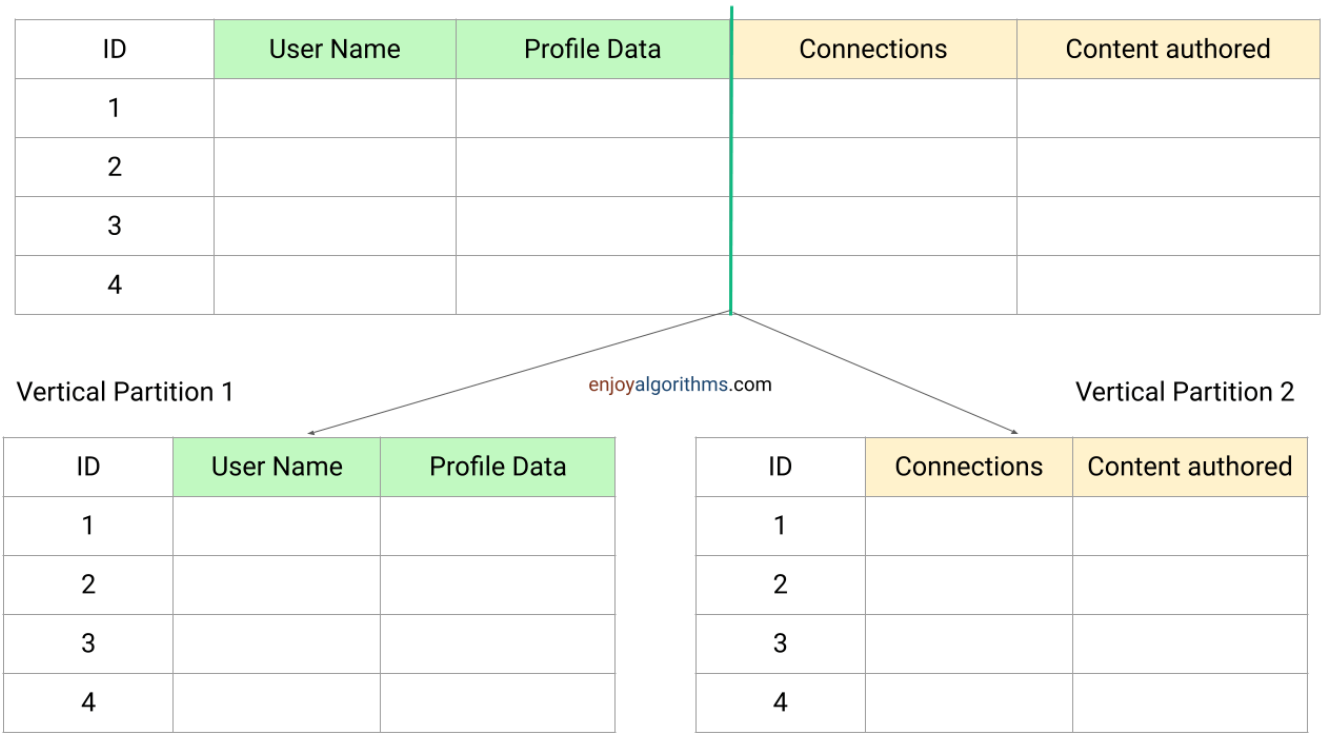
Vertical Partitioning

Vertical partitioning involves dividing a table into smaller tables based on columns. This is also known as **normalization**. In this method, each partition contains a smaller number of elements and is stored in a separate partition. For example, in a social media application like LinkedIn, a user's profile data, list of connections, and articles they have written can be placed on separate partitions using vertical partitioning. The user's profile data would be on one partition, the list of connections on a second partition, and the articles on a third partition. This can help to improve the performance and scalability of the database.

- Vertical partitioning allows us to store different types of data in separate partitions. This can be useful in situations where some data is more critical or sensitive than other data. For example, we could

store passwords, salary information, and other sensitive data in a separate partition to provide additional security controls.

- Vertical partitioning can also be beneficial when our database is stored on a solid-state drive (SSD). If certain columns in the database are not frequently queried, we can partition the table vertically and move those less frequently used columns to a different location. This can help to reduce the I/O and performance costs associated with fetching frequently accessed items.
- Overall, vertical partitioning allows us to separate slow-moving data from more dynamic data. Slow-moving data is a good candidate for caching in memory, which can improve the performance of the application.



There are a few disadvantages to using vertical partitioning:

- It may be necessary to combine data from multiple partitions to answer a query, which can increase the operational complexity of the system. For example, if a profile view request requires data from a user's profile, connections, and articles, this data will need to be retrieved from separate partitions and combined.
- If the website experiences additional growth, it may be necessary to further partition a feature-specific database across multiple servers.

This can be time-consuming and may require additional resources.

Functional Partitioning

In this type of partitioning strategy, data is organized based on the contextual dependency of a service. For example, a medical store system might store information about medicines in one partition and invoice data in another partition.

The choice of which type of partitioning to use depends on the structure of the data. In some cases, it may be useful to combine both horizontal and vertical partitioning to take advantage of both methods. For example, if we have a large dataset of customer information with different data types, we could use vertical partitioning to divide the database into string values and horizontal partitioning to divide the customer information.

Data Partitioning Criteria

There are a large number of criteria available for data partitioning. Most of them use partition keys and assign partitions on their basis. Some of the data partitioning criteria are range-partitioning, list-partitioning, hash partitioning, and many more.

Range Based Partitioning

In range partitioning, data is organized into partitions based on ranges of values for a partitioning key. This means that each partition contains rows with values for the partitioning key within a specific range. The ranges are typically contiguous and do not overlap, with each range specifying a non-inclusive lower and upper bound for the partition. Any partitioning key values equal to or higher than the upper bound of the range are added to the next higher partition.

Range partitioning is used in a few specific cases:

- When there is a need to organize data based on date and time. For

example, a table with a date column as the partitioning key might have a January-2022 partition that contains rows with partitioning key values from 01-Jan-2022 to 31-Jan-2022.
- When data is regularly added to the database and there is a need to remove old data. By dividing the partitions into date ranges, it becomes easier to remove old data. For example, we could delete all rows relating to employees who stopped working for the company before 1991. This can be more efficient for a table with many rows than running a delete query to remove employee data <= 1990.

Hash-Based Partitioning

In hash partitioning, rows are divided into different partitions based on a hashing algorithm. This is different from range partitioning, which groups database rows based on continuous indexes.

Hash partitioning can be used in a few different ways:

- We can use a client request's IP address or application ID as an input to a hash function to generate a hash value. This value determines which database partition to use for the request. For example, if we have 4 database partitions and each request contains an application ID, we could perform a modulo operation on the application ID with 4 and take the remainder to determine the partition to use.
- Hash partitioning is a good method for distributing data evenly across partitions.
- It is an easy-to-use alternative to range partitioning, especially when the partitioning data is not historical or does not have an obvious partitioning key.

One disadvantage of hash partitioning is that it can be expensive to dynamically add or remove database servers. For example, if we want to add more partitions, we may need to remap some of the keys and migrate them to a new partition, which requires changing the hash function. During this process, a large number of requests may not be able to be served, resulting in downtime until the migration is complete. This

problem can be addressed using consistent hashing!

List Based Partitioning

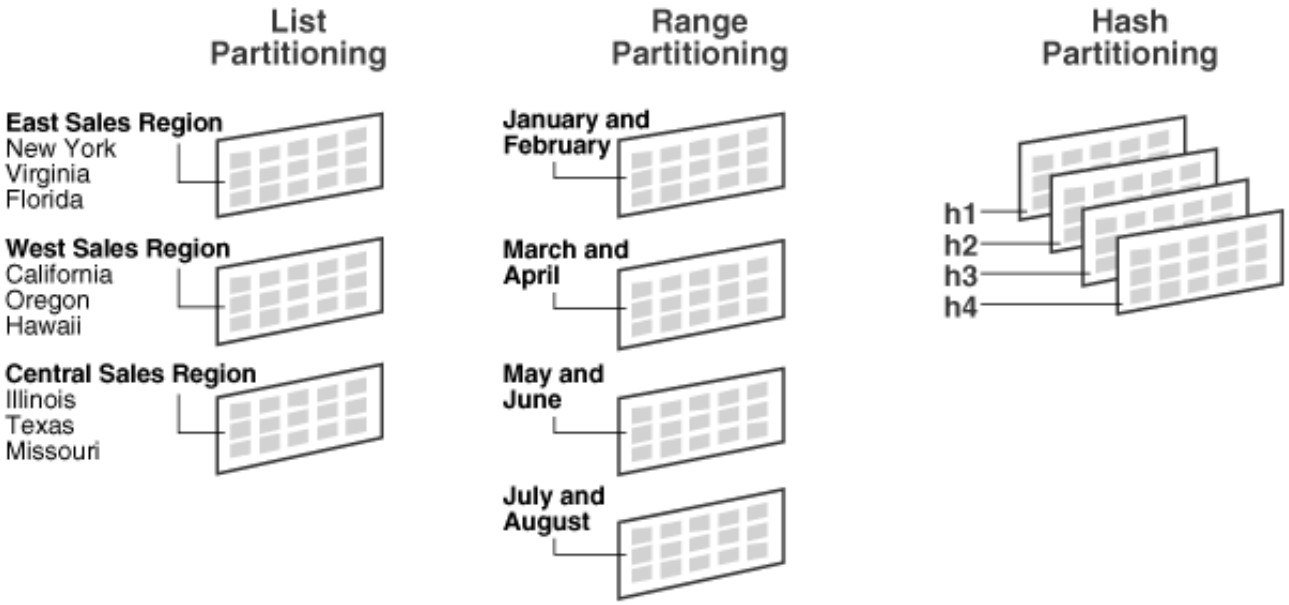
In list partitioning, each partition is defined and selected based on a list of values for a particular column, rather than a set of contiguous ranges of values. Some key points to consider when using list partitioning include:

1. The partitioning key can only consist of a single table column.
2. It is easy to group and organize unordered and unrelated data sets in different partitions.
3. We can partition data based on a specific column, such as a region column, to ensure that data for each region is stored in a single partition. For example, we could store all customers from India in one partition and customers from other countries in different partitions.

As an example, consider a table with data for 20 video stores distributed among 4 regions, as shown in the following table:

- Region: India, ID Numbers: 3, 5, 6, 9, 17
- Region: USA, ID Numbers: 1, 2, 10, 11, 19, 20
- Region: Japan, ID Numbers: 4, 12, 13, 14, 18
- Region: UK, ID Numbers: 7, 8, 15, 16

Using list partitioning, we could partition the table so that rows for stores belonging to the same region are stored in the same partition. This would allow us to easily add or drop records relating to specific regions from the table.



Composite Partitioning

Composite partitioning is a method of partitioning data based on two or more partitioning techniques. In this method, data is first partitioned using one technique, and then each partition is further divided into sub-partitions using the same or a different method.

Some key points to consider when using composite partitioning include:

- All sub-partitions of a given partition together represent a logical subset of the data.
- Composite partitioning supports operations such as adding new range partitions, and provides higher degrees of pruning and finer granularity of data placement through sub-partitioning.

Composite partitioning can be a useful technique for organizing and managing large datasets. It can help to improve the performance and scalability of the database by allowing for more precise control over data placement.

There are several types of composite partitioning:

Composite Range-Range Partitioning: This method performs range partitioning based on two table entries. For example, we could first partition the data by date and then sub-partition the range by price.