**Vertical and Horizontal scaling**

Scaling changes the size of a system. In this process, we can either make the system smaller or larger to meet expected needs. Scaling can be achieved by adding resources to an existing system, adding a new system, or doing both.

**Types of Scaling**

**Vertical Scaling**

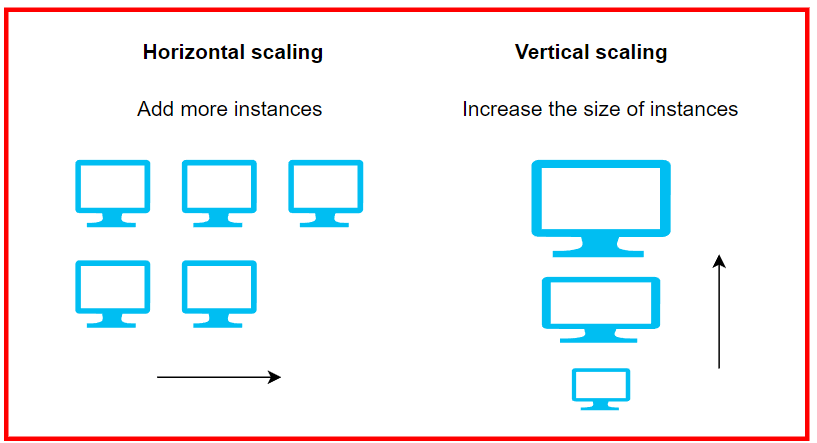
**Vertical scaling** means adding new resources to the existing system to meet expectations.

Example: Imagine a server rack that holds the current system. If it fails to meet the needs, and we can fix it by just adding more resources like CPU or RAM, that is vertical scaling. This type of scaling is usually easier and cheaper compared to horizontal scaling and takes less time to implement.

**Horizontal Scaling**

**Horizontal scaling** involves adding new server racks to the existing system to meet higher expectations.

Example: Using the same server rack scenario, if the system cannot be improved by just adding resources, we need to add new servers. This is horizontal scaling. It involves more machines and is typically more complex and costly than vertical scaling. It also takes more time to implement.



**Differences Between Horizontal and Vertical Scaling**

|  |  |
| --- | --- |
| **Horizontal Scaling** | **Vertical Scaling** |
| New server racks are added to meet higher expectations. | New resources are added to meet expectations. |
| Expands the system size horizontally. | Expands the system size vertically. |
| Easier to upgrade. | Harder to upgrade and may require downtime. |
| Difficult to implement. | Easy to implement. |
| Costly, as new server racks comprise many resources. | Cheaper, as only new resources need to be added. |
| Takes more time to implement. | Takes less time to implement. |
| High resilience and fault tolerance. | Single point of failure. |
| Examples: Cassandra, MongoDB, Google Cloud Spanner. | Examples: MySQL, Amazon RDS. |

**Advantages of Vertical Scaling**

* **Cost-Effective:** Cheaper than horizontal scaling.
* **Simplicity:** Easier to implement and manage.
* **Speed:** Faster to scale up by adding resources.

**Disadvantages of Vertical Scaling**

* **Limited Capacity:** Limited by the maximum capacity of a single machine.
* **Downtime:** May require downtime to upgrade.
* **Single Point of Failure:** If the system fails, all services may go down.

**Advantages of Horizontal Scaling**

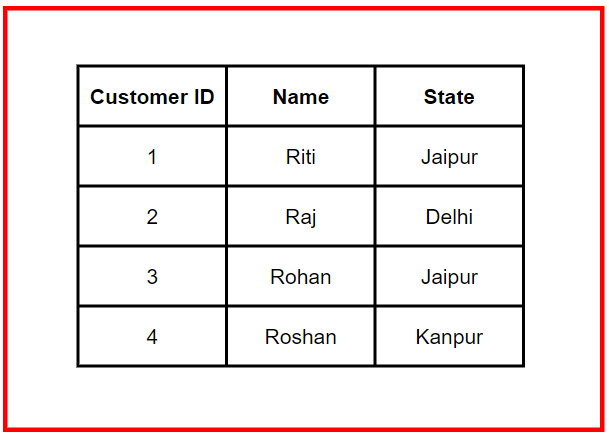
* **High Availability:** More resilient with no single point of failure.
* **Flexibility:** Can scale out indefinitely by adding more machines.
* **No Downtime:** New servers can be added without downtime.

**Disadvantages of Horizontal Scaling**

* **Complexity:** More complex to implement and manage.
* **Cost:** More expensive due to additional hardware and management overhead.
* **Time-Consuming:** Takes longer to add and configure new servers.

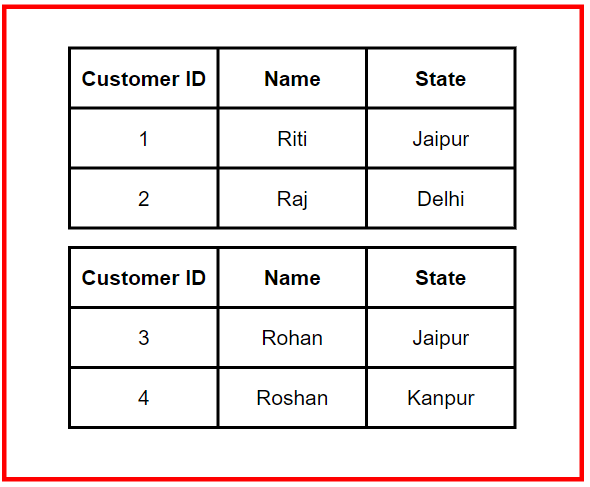
**Sharding**

Sharding in SQL refers to the practice of dividing a large database into smaller, more manageable pieces called "shards." Each shard is a separate database instance that can be hosted on different servers or clusters. Sharding is commonly used to improve the **performance**, **scalability**, and **availability** of a database system.



**Sharded Database**

Sharding involves distributing data rows across multiple nodes. For example:



**Components of Sharding**

* **Shards:** Logical partitions of data stored on separate machines. Each physical shard can contain multiple logical shards.
* **Shard Key:** A column in the dataset used to determine how rows are distributed across shards. It can be an existing column or a newly created one.
* **Shared-Nothing Architecture:** Each shard operates independently and is unaware of others. Only the shards with relevant data process requests in parallel.

**Methods of Database Sharding**

**1. Range-Based Sharding**

**Description:** Data is partitioned based on a range of values of the shard key.

**Example:** Sharding customer names by the first letter:

* Names starting with A-I → Shard A
* Names starting with J-S → Shard B
* Names starting with T-Z → Shard C

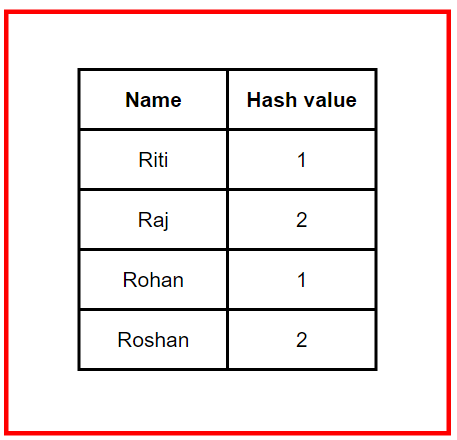
**Pros:** Easier to implement.

**Cons:** Can lead to uneven data distribution and hotspots.

**2. Hashed Sharding**

**Description:** Uses a hash function on the shard key to determine the shard.

**Example:** Using hash values to distribute customer records:



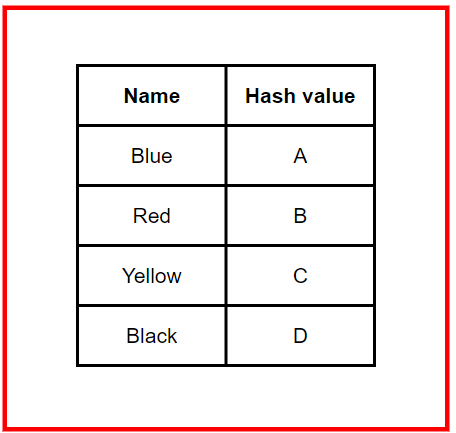
**Pros:** Even data distribution among shards.

**Cons:** Difficult to reassign hash values when adding more shards.

**3. Directory Sharding**

**Description:** Uses a lookup table to map shard keys to shards.

**Example:** Clothing colors mapped to shards:



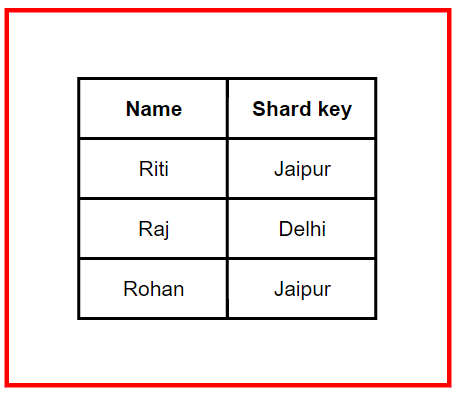
**Pros:** Flexible and meaningful data distribution.

**Cons:** Fails if the lookup table is incorrect.

**4. Geo Sharding**

**Description:** Partitions data based on geographical location.

**Example:** Using cities as shard keys for a dating service:



**Pros:** Faster data retrieval due to proximity.

**Cons:** Can result in uneven data distribution.

**Optimizing Database Sharding**

**Key Factors:**

* **Cardinality:** The number of unique values in the shard key. Higher cardinality allows for more shards.
* **Frequency:** The probability of data being stored in specific shards. Avoid high-frequency shard keys to prevent hotspots.
* **Monotonic Change:** The rate at which the shard key values change. Avoid monotonically increasing or decreasing shard keys to ensure balanced shards.

**Example of Poor Sharding**

A feedback database is split by the number of customer purchases:

* Shard A: 0-10 purchases
* Shard B: 11-20 purchases
* Shard C: 21+ purchases

As customers make more purchases, Shard C becomes overloaded, leading to imbalance.