What is NoSQL?

Traditionally, the software industries use relational databases to store and manage data persistently. Not only SQL or NoSQL is a new set of a database that has emerged in the recent past as an alternative solution to relational databases.

"In the early 1970, Flat File Systems are used. Data were stored in flat files and the biggest problems with flat files are each company implement their own flat files and there are no standards. It is very difficult to store data in the files, retrieve data from files because there is no standard way to store data.

Then the relational database was created by E.F. Codd and these databases answered the question of having no standard way to store data. But later relational database also get a problem that it could not handle big data, due to this problem there was a need of database which can handle every types of problems then NoSQL database was developed."

Carl Strozzi introduced the term NoSQL to name his file-based database 1998.

NoSQL refers to all databases and data stores that are not based on the Relational Database Management Systems or RDBMS principles. It relates to large data sets accessed and manipulated on a Web scale. NoSQL does not represent single product or technology. It represents a group of products and a various related data concepts for storage and management. "***NoSQL was a hashtag that was chosen for a tech meetup to discuss the new databases."*** NoSQL database doesn't use tables for storing data. It is generally used to store big data and real-time web applications. It is designed for distributed data stores where very large scale of data storing needs (for example Google or Facebook which collects terabits of data every day for their users). These type of data storing may not require fixed schema, avoid join operations and typically scale horizontally.

Database Features of NoSQL

NoSQL Databases can have a common set of features such as:

- Non-relational data model.
- Runs well on clusters.
- Mostly open-source.

- Built for the new generation Web applications.
- Is schema-less.

Why NoSQL?

With the explosion of social media, user-driven content has grown rapidly and has increased the volume and type of data that is produced, managed, analyzed, and archived. In addition, new sources of data, such as sensors, Global Positioning Systems or GPS, automated trackers, and other monitoring systems generate huge volumes of data on a regular basis. Personal user information, social graphs, geo location data, user-generated content and machine logging data are just a few examples where the data has been increasing exponentially. The evolution of NoSql databases is to handle these huge data properly.

These large volumes of data sets, also called big data, have introduced new challenges and opportunities for data storage, management, analysis, and archival. In addition, data is becoming increasingly semi-structured and sparse. To avail the above service properly, it is required to process huge amount of data, which SQL databases were never designed.

To resolve the problems related to large-volume and semi-structured data, a class of new database products has emerged. These new classes of database products consist of column-based data stores, key/value pair databases, and document databases. Together, these are called NoSQL. The NoSQL database consists of diverse products with each product having unique sets of features and value propositions.

Difference Between RDBMS and NoSQL Databases

RDBMS	NoSQL
• Data is stored in a relational model, with rows and columns. • A row contains information about an item while columns contain specific information, such as 'Model', 'Date of Manufacture', 'Color'.	• Data is stored in a host of different databases, with different data storage models. • Follows dynamic schemas. Meaning, you can add columns anytime. • Supports horizontal scaling. You can

• Follows fixed schema. Meaning, the columns are defined and locked before data entry. In addition, each row contains data for each column. • Supports vertical scaling. Scaling an RDBMS across multiple servers is a challenging and time-consuming process. • ACID Compliant. **Atomicity**: A database follows the all or nothing rule, i.e., the database considers all transaction operations as one whole unit or atom. **Consistency**: Ensures that only valid data following all rules and constraints is written in the database. **Isolation**: Ensures that transactions are securely and independently processed at the same time without interference, but it does not ensure the order of transactions. **Durability**: Durability ensures that any transaction committed to the database will not be lost.	scale across multiple servers. Multiple servers are cheap commodity hardware or cloud instances, which make scaling cost-effective compared to vertical scaling. • CAP theorem and BASE Transaction

CAP Theorem (Brewer's Theorem)

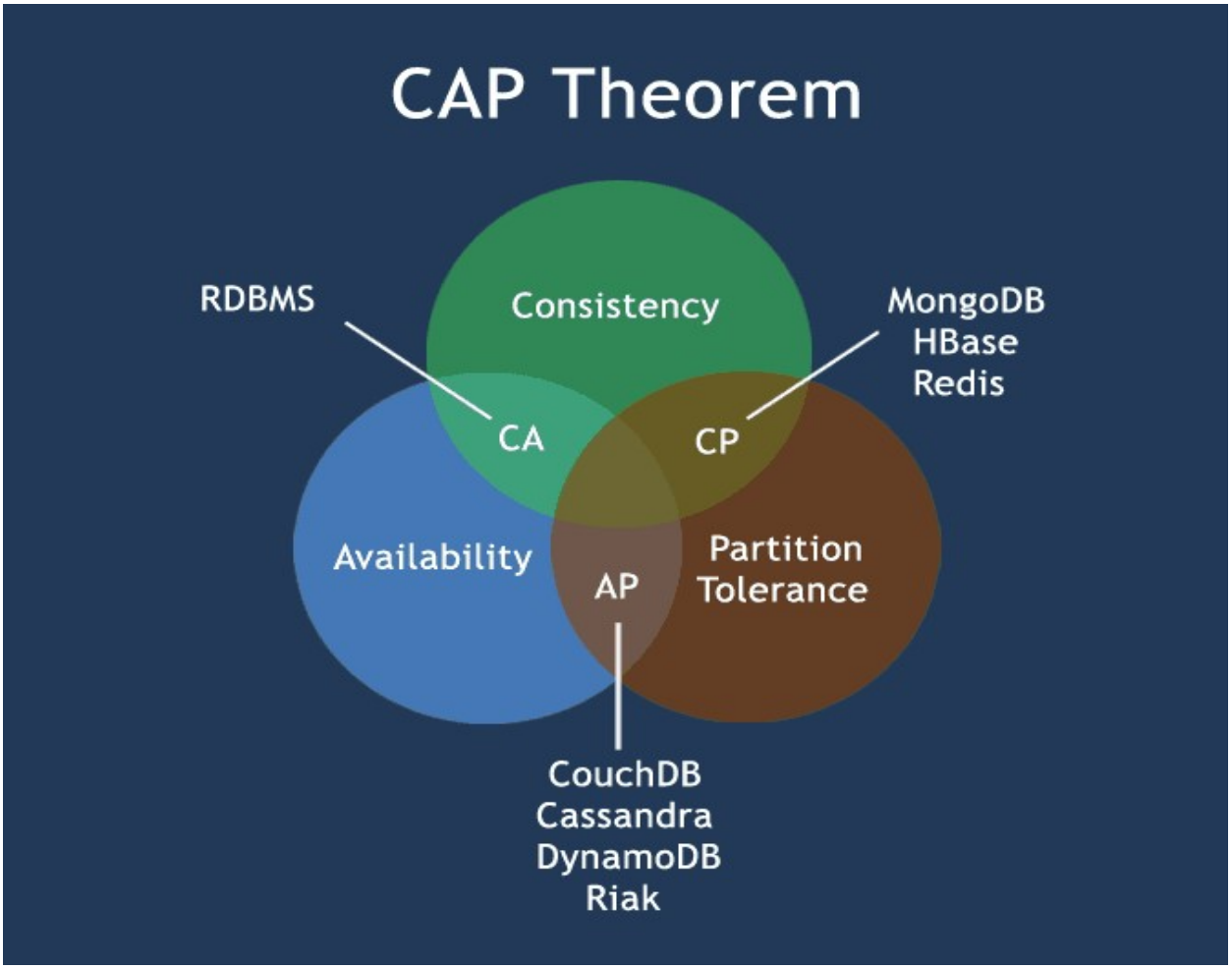
CAP theorem states that there are three basic requirements which exist in a special relation when designing applications for a distributed architecture.

Consistency - This means that the data in the database remains consistent after the execution of an operation. For example after an update operation all clients see the same data.

Availability - This means that the system is always on (service guarantee availability), no downtime.

Partition Tolerance - This means that the system continues to function even the communication among the servers is unreliable, i.e. the servers may be partitioned into multiple groups that cannot communicate with one another.

In theoretically it is impossible to fulfill all three requirements. CAP provides the basic requirements for a distributed system to follow two out of the three requirements.



.

The CAP theorem states that a distributed computer system cannot guarantee all of the following three properties at the same time:

- Consistency
- Availability
- Partition tolerance

A BASE system gives up on consistency.

- **Basically Available** indicates that the system *does* guarantee availability, in terms of the CAP theorem.
- **Soft** state indicates that the state of the system may change over time, even without input. This is because of the eventual consistency model.
- **Eventual** consistency indicates that the system will become consistent over time, given that the system doesn't receive input during that time.

Benefits of NoSQL

The desired technical characteristics of an enterprise-class NoSQL solution are as follows.

Primary and Analytic Data Source Capability

The first criterion of an enterprise-class NoSQL solution is it must serve as a primary or active data source that receives data from different business applications. It also must act as a secondary data source or analytic database that enhances business intelligence applications. From business perspective, the NoSQL database must be capable of quickly integrating all types of data viz structured, semi-structured, or unstructured. In addition, it must be able to execute high-performance queries. Once all the required data is collected in the database, data administrators may want to perform an analysis in real time and in map-reduce form. An enterprise-class NoSQL database can easily handle such requests using the same database. It does not require loading the data into a separate analytic database for analysis.

Big Data Capability

NoSQL databases are not restricted to working with big data. However, an enterprise-class NoSQL solution can scale to manage large volumes of data from

terabytes to petabytes. In addition to storing large volumes of data, it delivers high performance for data velocity, variety, and complexity.

Continuous Availability or No Single Point of Failure

To be considered enterprise-class, a NoSQL database must offer continuous availability, with no single point of failure. Moreover, rather than providing the continuous availability feature outside the software, the NoSQL solution delivers inherent continuous availability. The NoSQL databases must include the following key features:

- All nodes in a cluster must be able to serve read request even if some machines are down.
- Must be capable of easily replicating and segregating data between different physical shelves in a data center. This helps avoid hardware outages.
- Must be able to support data distribution designs that are multi-data centers, on-premises or in the cloud.

Multi-Data Center Capability

Typically, business enterprises own highly distributed databases that are spread across multiple data centers and geographic locales. Data replication is a feature offered by all legacy RDBMS. However, none can offer a simple model of data distribution between various data centers without any performance issue.

A simple method includes the ability to handle multiple data centers without concerning the occurrences of the read and write operations. A good NoSQL enterprise solution must support multi data-center deployment and must provide configurable option to maintain a balance between performance and consistency.

Easy Replication for Distributed Location-Independent Capabilities

To avoid data loss affecting an application, a good NoSQL solution provides strong replication abilities. These include a read-anywhere and write-anywhere capability with full location-independence support.

This means you can write data to any node in a cluster, have it replicated on other nodes, and make it available to all users irrespective of their location. In addition, the write capability on any node must ensure that the data is safe in the event of a power failure or any other incident.

No Need for Separate Caching Layer

A good NoSQL solution is capable of using multiple nodes and distributing data among all the participating nodes.

Thus, it does not require a specific caching layer to store data. The memory caches of all participating nodes can store data for quick input and output or I/O access. NoSQL database eliminates the problem of synchronizing cache data with the persistent database. Thus, it supports simple scalability with fewer management issues.

Cloud-Ready

As an adaptation of cloud infrastructure is increasing day by day, an enterprise-class NoSQL solution must be cloud-ready.

A NoSQL database cluster must be able to function in a cloud setting, such as Amazon EC2, and also must be able to expand and contract a cluster when necessary. It also must support a hybrid solution where part of the database is hosted within the enterprise premise and another part is hosted in a cloud setting.

High Performance with Linear Scalability

An enterprise-class NoSQL database can enhance performance by adding nodes to a cluster. Typically, the performance of database systems may go down when additional nodes are added to a cluster. However, a good NoSQL solution increase performance for both read and writes operations when additional nodes are added. These performance gains are linear in nature.

Flexible Schema Support

An enterprise-class NoSQL database offers a flexible or dynamic schema design to manage all types of data—structured, semi-structured, and non-structured.

Therefore, the need to have different vendors to support the different data types does not arise.

NoSQL databases may support various schema formats, such as columnar/Bigtable and document.

Therefore, choosing an appropriate database based on application requirement is a key design decision.

The flexible or dynamic schema support ensures that you can make schema changes to a structure without making the structure offline. This support is critical considering the near-zero downtime and round-the-clock availability for business applications.

Support Key Developer Languages and Platforms

Ideally, an enterprise-class NoSQL solution must support all major operating systems. In addition, it must run on a product hardware that does not require any tweaks or other proprietary add-ons.

The NoSQL database must provide client interfaces and drivers for all common developer languages. It must offer a structured query language or SQL or a similar language that helps store and access data in a NoSQL database.

Easy to Implement, Maintain, and Grow

A NoSQL database must be simple but robust. In other words, it must be easy to implement and use and must offer sturdy functionality to handle various enterprise applications.

In addition, the NoSQL vendor must supply good management tools that assist data professionals to perform various administrative tasks, such as adding capacity to a cluster, running utility tasks, and so on.

The NoSQL database must allow easy growth without making any change to the front-end of the business application.

Thriving Open Source Community

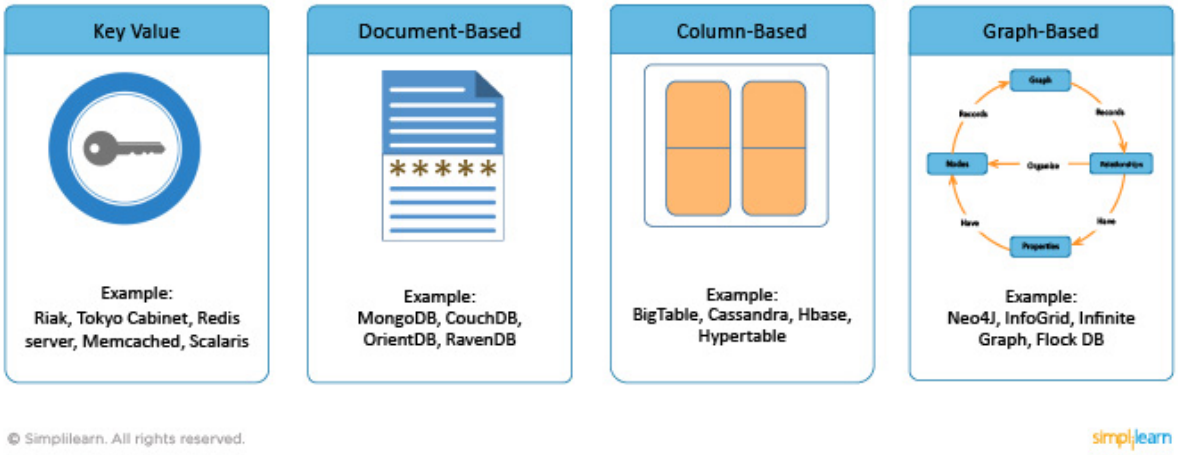
For an open source NoSQL database, having a vibrant community is essential to make a regular contribution to enhance the core software.

Moreover, open source communities generally provide excellent quality assurance or QA testing. This sometimes eliminates the need for software companies to hire, train, and retain a QA team.

To encourage a thriving open source community, including activities on mailing lists and technical forums, initiate technical discussions, and participate in conferences.

Types of NoSQL

There are four basic types of NoSQL databases.



Key-Value database

Key-Value database has a big hash table of keys and values. Riak (Pronounce as REE-awk), Tokyo Cabinet, Redis server, Memcached ((Pronounce as mem-cached), and Scalaris are examples of a key-value store.

Document-based database

Document-based database stores documents made up of tagged elements. Examples: MongoDB, CouchDB, OrientDB, and RavenDB .

Column-based database

Each storage block contains data from only one column, Examples: BigTable, Cassandra, Hbase, and Hypertable.

Graph-based database

A graph-based database is a network database that uses nodes to represent and store data. Examples are Neo4J, InfoGrid, Infinite Graph, and FlockDB. The availability of choices in NoSQL databases has its own advantages and disadvantages.

The advantage is, it allows you to choose a design according to your system requirements. However, because you have to make a choice based on requirements, there is always a chance that the same database product may not be used properly.

Key-Value Database

From an Application Program Interface or API perspective, a key-value database is the simplest NoSQL database. This database stores every single item as a key with a value. You can get the value for a key, add a value for a key, or delete a key. The value is a blob that the database stores without knowing its content. The responsibility lies with the application to understand what is stored. Typically, key-value databases use primary-key access. Therefore, they generally offer enhanced performance and scalability. All key-value databases may not have the same features.

For example, data is not persistent in Memcached while it is in Riak. These features are important when implementing certain solutions.

For example, you need to implement caching of user preferences. If you implement them in Memcached, you may lose all the data when the node goes down and may need to get them from the source database. If you store the same data in Riak, you may not lose data but must consider how to update the stale data. It is important to select a key-value database based on your requirements.

Advantages & Disadvantages of Key-Value Database

The key value store does not have a defined schema. It contains client defined semantics for understanding what the values are. A key value store is simple to build and easy to scale. It also tends to have great performance because the access pattern can be optimized to suit your requirement.

The advantages & disadvantages of the key-value store include the following.

Advantages	Disadvantages
• Queries: You can perform a query by using the key. Even range queries on the key are usually not possible. • Schema: Key value databases have the following schema - the key is a string, the value is a blob. The client determines how to parse data. • Usages: Key value databases can access data using a key. Key-value type database suffers from major weaknesses.	• It does not provide any traditional database capabilities, such as consistency when multiple transactions are executed simultaneously. • As the volume of data increases, maintaining unique values as keys become difficult.

Document Database

This NoSQL database type is based on the concept of documents.

- It stores and retrieves various documents in formats, such as XML, JavaScript Object Notation or JSON (Pronounce as JAY- Sahn), Binary JSON or BSON.
- These documents are self-descriptive, hierarchical tree data structures which consist of maps, collections, and scalar values.

- The stored documents can be similar to each other, but not necessarily the same. It stores documents in the value part of the key-value database. You can consider the document databases as key-value stores where you can examine the values.

Document Database Example

Examples of Document databases are:

- MongoDB: Provides a rich query language and many useful features such as built-in support for MapReduce-style aggregation and geospatial indexes.
- Apache CouchDB: Uses JSON for documents, JavaScript for MapReduce indexes, and regular HTTP for its API.

Column-Based Database

Column-based databases store data in column families as rows. These rows contain multiple columns associated with a row key. Column families are groups of related data that is accessed together.

For example, you may access customer profile information at the same time, but not their order history.

- Each column family is like a container of rows in an RDBMS table where the key identifies the rows.
- Each row consists of multiple columns. However, the various rows need not have the same columns. Moreover, you can add a column to any row at any time without adding it to other rows.

Goals of Column-Based Database

The goal of a Column-based database is to efficiently read and write data to and from hard disk storage to quickly return a query. In this database, all column one values are physically together, followed by all the column two values. The data is stored in record order so that the 100th entry for column one and the column two are from the same input record. This allows you to access individual data elements, such as customer name, as a group in columns.

The compression permits columnar operations like MIN, MAX, SUM, COUNT, and AVG— to be performed very rapidly. A column-based database management system or DBMS is self-indexing.

Therefore, these data can be retrieved fast in an efficient manner.

Column-Based Database Example

Cassandra is a column-based database.

- Cassandra is fast and easily scalable with write operations spread across the cluster.
- The cluster does not have a master node, hence, any node can handle the read and write operations.

Graph Database

A graph database lets you store data and its relationships with other data in the form of nodes and edges. Each relation can have a set of properties. Edges have a direction which has its own significance and enables you to explore the relationship in both the direction.

All the nodes in the graph are organized by relationships that help explore interesting and hidden patterns between the nodes.

Examples of Graph Database

- Examples of graph database are Neo4J (pronounce as Neo- four-J), Infinite Graph, OrientDB, and FlockDB.
- Neo4J is one of the most popular graph databases, which is ACID compliant. It is the product of the company Neo Technologies. It is Java based but has bindings for other languages, including Ruby and Python.
- FlockDB was created by Twitter for relationship related analytics.

In the graph database, the labeled property graph model is used for modeling the data. It is same as the entity relationships or ER model used in RDBMS. The property graph contains connected entities, such as the nodes which can hold any number of attributes or key-value-pairs.