**An Introduction to Big Data Concepts and Terminology**

[Scaling](https://www.digitalocean.com/community/tags/scaling)[Clustering](https://www.digitalocean.com/community/tags/clustering)[Big Data](https://www.digitalocean.com/community/tags/big-data)[Conceptual](https://www.digitalocean.com/community/tags/conceptual)

* [](https://www.digitalocean.com/community/users/jellingwood)
* By [Justin Ellingwood](https://www.digitalocean.com/community/users/jellingwood)

Published onSeptember 28, 2016 153.5kviews

Introduction

**Big data** is a blanket term for the non-traditional strategies and technologies needed to gather, organize, process, and gather insights from large datasets. While the problem of working with data that exceeds the computing power or storage of a single computer is not new, the pervasiveness, scale, and value of this type of computing has greatly expanded in recent years.

In this article, we will talk about big data on a fundamental level and define common concepts you might come across while researching the subject. We will also take a high-level look at some of the processes and technologies currently being used in this space.

What Is Big Data?

An exact definition of “big data” is difficult to nail down because projects, vendors, practitioners, and business professionals use it quite differently. With that in mind, generally speaking, **big data** is:

* large datasets
* the category of computing strategies and technologies that are used to handle large datasets

In this context, “large dataset” means a dataset too large to reasonably process or store with traditional tooling or on a single computer. This means that the common scale of big datasets is constantly shifting and may vary significantly from organization to organization.

Why Are Big Data Systems Different?

The basic requirements for working with big data are the same as the requirements for working with datasets of any size. However, the massive scale, the speed of ingesting and processing, and the characteristics of the data that must be dealt with at each stage of the process present significant new challenges when designing solutions. The goal of most big data systems is to surface insights and connections from large volumes of heterogeneous data that would not be possible using conventional methods.

In 2001, Gartner’s Doug Laney first presented what became known as the “three Vs of big data” to describe some of the characteristics that make big data different from other data processing:

Volume

The sheer scale of the information processed helps define big data systems. These datasets can be orders of magnitude larger than traditional datasets, which demands more thought at each stage of the processing and storage life cycle.

Often, because the work requirements exceed the capabilities of a single computer, this becomes a challenge of pooling, allocating, and coordinating resources from groups of computers. Cluster management and algorithms capable of breaking tasks into smaller pieces become increasingly important.

Velocity

Another way in which big data differs significantly from other data systems is the speed that information moves through the system. Data is frequently flowing into the system from multiple sources and is often expected to be processed in real time to gain insights and update the current understanding of the system.

This focus on near instant feedback has driven many big data practitioners away from a batch-oriented approach and closer to a real-time streaming system. Data is constantly being added, massaged, processed, and analyzed in order to keep up with the influx of new information and to surface valuable information early when it is most relevant. These ideas require robust systems with highly available components to guard against failures along the data pipeline.

Variety

Big data problems are often unique because of the wide range of both the sources being processed and their relative quality.

Data can be ingested from internal systems like application and server logs, from social media feeds and other external APIs, from physical device sensors, and from other providers. Big data seeks to handle potentially useful data regardless of where it’s coming from by consolidating all information into a single system.

The formats and types of media can vary significantly as well. Rich media like images, video files, and audio recordings are ingested alongside text files, structured logs, etc. While more traditional data processing systems might expect data to enter the pipeline already labeled, formatted, and organized, big data systems usually accept and store data closer to its raw state. Ideally, any transformations or changes to the raw data will happen in memory at the time of processing.

Other Characteristics

Various individuals and organizations have suggested expanding the original three Vs, though these proposals have tended to describe challenges rather than qualities of big data. Some common additions are:

* **Veracity**: The variety of sources and the complexity of the processing can lead to challenges in evaluating the quality of the data (and consequently, the quality of the resulting analysis). **Data Veracity** refers to the biases, noise and abnormality in **data**. Is the **data** that is being stored, and mined meaningful to the problem being analyzed. Inderpal feel **veracity** in **data** analysis is the biggest challenge when compares to things like volume and velocity
* **Variability**: Variation in the data leads to wide variation in quality. Additional resources may be needed to identify, process, or filter low quality data to make it more useful.
* **Value**: The ultimate challenge of big data is delivering value. Sometimes, the systems and processes in place are complex enough that using the data and extracting actual value can become difficult.

What Does a Big Data Life Cycle Look Like?

So how is data actually processed when dealing with a big data system? While approaches to implementation differ, there are some commonalities in the strategies and software that we can talk about generally. While the steps presented below might not be true in all cases, they are widely used.

The general categories of activities involved with big data processing are:

* Ingesting data into the system
* Persisting the data in storage
* Computing and Analyzing data
* Visualizing the results

Before we look at these four workflow categories in detail, we will take a moment to talk about **clustered computing**, an important strategy employed by most big data solutions. Setting up a computing cluster is often the foundation for technology used in each of the life cycle stages.

Clustered Computing

Because of the qualities of big data, individual computers are often inadequate for handling the data at most stages. To better address the high storage and computational needs of big data, computer clusters are a better fit.

Big data clustering software combines the resources of many smaller machines, seeking to provide a number of benefits:

* **Resource Pooling**: Combining the available storage space to hold data is a clear benefit, but CPU and memory pooling is also extremely important. Processing large datasets requires large amounts of all three of these resources.
* **High Availability**: Clusters can provide varying levels of fault tolerance and availability guarantees to prevent hardware or software failures from affecting access to data and processing. This becomes increasingly important as we continue to emphasize the importance of real-time analytics.
* **Easy Scalability**: Clusters make it easy to scale horizontally by adding additional machines to the group. This means the system can react to changes in resource requirements without expanding the physical resources on a machine.

Using clusters requires a solution for managing cluster membership, coordinating resource sharing, and scheduling actual work on individual nodes. Cluster membership and resource allocation can be handled by software like **Hadoop’s YARN** (which stands for Yet Another Resource Negotiator) or **Apache Mesos**.

The assembled computing cluster often acts as a foundation which other software interfaces with to process the data. The machines involved in the computing cluster are also typically involved with the management of a distributed storage system, which we will talk about when we discuss data persistence.

Ingesting Data into the System

Data ingestion is the process of taking raw data and adding it to the system. The complexity of this operation depends heavily on the format and quality of the data sources and how far the data is from the desired state prior to processing.

One way that data can be added to a big data system are dedicated ingestion tools. Technologies like **Apache Sqoop** can take existing data from relational databases and add it to a big data system. Similarly, **Apache Flume** and **Apache Chukwa** are projects designed to aggregate and import application and server logs. Queuing systems like **Apache Kafka** can also be used as an interface between various data generators and a big data system. Ingestion frameworks like **Gobblin** can help to aggregate and normalize the output of these tools at the end of the ingestion pipeline.

During the ingestion process, some level of analysis, sorting, and labelling usually takes place. This process is sometimes called ETL, which stands for extract, transform, and load. While this term conventionally refers to legacy data warehousing processes, some of the same concepts apply to data entering the big data system. Typical operations might include modifying the incoming data to format it, categorizing and labelling data, filtering out unneeded or bad data, or potentially validating that it adheres to certain requirements.

With those capabilities in mind, ideally, the captured data should be kept as raw as possible for greater flexibility further on down the pipeline.

Persisting the Data in Storage

The ingestion processes typically hand the data off to the components that manage storage, so that it can be reliably persisted to disk. While this seems like it would be a simple operation, the volume of incoming data, the requirements for availability, and the distributed computing layer make more complex storage systems necessary.

This usually means leveraging a distributed file system for raw data storage. Solutions like **Apache Hadoop’s HDFS** filesystem allow large quantities of data to be written across multiple nodes in the cluster. This ensures that the data can be accessed by compute resources, can be loaded into the cluster’s RAM for in-memory operations, and can gracefully handle component failures. Other distributed filesystems can be used in place of HDFS including **Ceph** and **GlusterFS**.

Data can also be imported into other distributed systems for more structured access. Distributed databases, especially NoSQL databases, are well-suited for this role because they are often designed with the same fault tolerant considerations and can handle heterogeneous data. There are many different types of distributed databases to choose from depending on how you want to organize and present the data. To learn more about some of the options and what purpose they best serve, read our [NoSQL comparison guide](https://www.digitalocean.com/community/tutorials/a-comparison-of-nosql-database-management-systems-and-models).

Computing and Analyzing Data

Once the data is available, the system can begin processing the data to surface actual information. The computation layer is perhaps the most diverse part of the system as the requirements and best approach can vary significantly depending on what type of insights desired. Data is often processed repeatedly, either iteratively by a single tool or by using a number of tools to surface different types of insights.

**Batch processing** is one method of computing over a large dataset. The process involves breaking work up into smaller pieces, scheduling each piece on an individual machine, reshuffling the data based on the intermediate results, and then calculating and assembling the final result. These steps are often referred to individually as splitting, mapping, shuffling, reducing, and assembling, or collectively as a distributed map reduce algorithm. This is the strategy used by **Apache Hadoop’s MapReduce**. Batch processing is most useful when dealing with very large datasets that require quite a bit of computation.

While batch processing is a good fit for certain types of data and computation, other workloads require more **real-time processing**. Real-time processing demands that information be processed and made ready immediately and requires the system to react as new information becomes available. One way of achieving this is **stream processing**, which operates on a continuous stream of data composed of individual items. Stream processing is the processing of *data in motion*, or in other words, computing on data directly as it is produced or received.

The majority of data are born as continuous streams: sensor events, user activity on a website, financial trades, and so on – all these data are created as a series of events over time.

Another common characteristic of real-time processors is in-memory computing, which works with representations of the data in the cluster’s memory to avoid having to write back to disk. In-memory computing means using a type of middleware software that allows one to store data in RAM, across a cluster of computers, and process it in parallel. Consider operational datasets typically stored in a centralized database which you can now store in “connected” RAM across multiple computers

**Apache Storm**, **Apache Flink**, and **Apache Spark** provide different ways of achieving real-time or near real-time processing. There are trade-offs with each of these technologies, which can affect which approach is best for any individual problem. In general, real-time processing is best suited for analyzing smaller chunks of data that are changing or being added to the system rapidly.

The above examples represent computational frameworks. However, there are many other ways of computing over or analyzing data within a big data system. These tools frequently plug into the above frameworks and provide additional interfaces for interacting with the underlying layers. For instance, **Apache Hive** provides a data warehouse interface for Hadoop, **Apache Pig** provides a high level querying interface, while SQL-like interactions with data can be achieved with projects like **Apache Drill**, **Apache Impala**, **Apache Spark SQL**, and **Presto**. For machine learning, projects like **Apache SystemML**, **Apache Mahout**, and **Apache Spark’s MLlib** can be useful. For straight analytics programming that has wide support in the big data ecosystem, both **R** and **Python** are popular choices.

Visualizing the Results

Due to the type of information being processed in big data systems, recognizing trends or changes in data over time is often more important than the values themselves. Visualizing data is one of the most useful ways to spot trends and make sense of a large number of data points.

Real-time processing is frequently used to visualize application and server metrics. The data changes frequently and large deltas in the metrics typically indicate significant impacts on the health of the systems or organization. In these cases, projects like **Prometheus** can be useful for processing the data streams as a time-series database and visualizing that information. **Prometheus** acts as the storage backend and **Grafana** as the interface for analysis and visualization

One popular way of visualizing data is with the  **Elastic Stack is a group of**[**open source**](https://whatis.techtarget.com/definition/open-source)**products from**[**Elastic**](https://searchitoperations.techtarget.com/definition/Elastic)**designed to help users take data from any type of source and in any format and search, analyze, and visualize that data in real time.**, formerly known as the ELK stack. Composed of Logstash for data collection, Elasticsearch for indexing data, and Kibana for visualization, the Elastic stack can be used with big data systems to visually interface with the results of calculations or raw metrics. A similar stack can be achieved using **Apache Solr** for indexing and a Kibana fork called **Banana** for visualization. The stack created by these is called **Silk**.

Another visualization technology typically used for interactive data science work is a data “notebook”. These projects allow for interactive exploration and visualization of the data in a format conducive to sharing, presenting, or collaborating. Popular examples of this type of visualization interface are **Jupyter Notebook** and **Apache Zeppelin**.

Big Data Glossary

While we’ve attempted to define concepts as we’ve used them throughout the guide, sometimes it’s helpful to have specialized terminology available in a single place:

* **Big data**: Big data is an umbrella term for datasets that cannot reasonably be handled by traditional computers or tools due to their volume, velocity, and variety. This term is also typically applied to technologies and strategies to work with this type of data.
* **Batch processing**: Batch processing is a computing strategy that involves processing data in large sets. This is typically ideal for non-time sensitive work that operates on very large sets of data. The process is started and at a later time, the results are returned by the system.
* **Cluster computing**: Clustered computing is the practice of pooling the resources of multiple machines and managing their collective capabilities to complete tasks. Computer clusters require a cluster management layer which handles communication between the individual nodes and coordinates work assignment.
* **Data lake**: Data lake is a term for a large repository of collected data in a relatively raw state. This is frequently used to refer to the data collected in a big data system which might be unstructured and frequently changing. This differs in spirit to data warehouses (defined below).
* **Data mining**: Data mining is a broad term for the practice of trying to find patterns in large sets of data. It is the process of trying to refine a mass of data into a more understandable and cohesive set of information.
* **Data warehouse**: Data warehouses are large, ordered repositories of data that can be used for analysis and reporting. In contrast to a *data lake*, a data warehouse is composed of data that has been cleaned, integrated with other sources, and is generally well-ordered. Data warehouses are often spoken about in relation to big data, but typically are components of more conventional systems.
* **ETL**: ETL stands for extract, transform, and load. It refers to the process of taking raw data and preparing it for the system’s use. This is traditionally a process associated with data warehouses, but characteristics of this process are also found in the ingestion pipelines of big data systems.
* **Hadoop**: Hadoop is an Apache project that was the early open-source success in big data. It consists of a distributed filesystem called HDFS, with a cluster management and resource scheduler on top called YARN (Yet Another Resource Negotiator). Batch processing capabilities are provided by the MapReduce computation engine. Other computational and analysis systems can be run alongside MapReduce in modern Hadoop deployments.
* **In-memory computing**: In-memory computing is a strategy that involves moving the working datasets entirely within a cluster’s collective memory. Intermediate calculations are not written to disk and are instead held in memory. This gives in-memory computing systems like Apache Spark a huge advantage in speed over I/O bound systems like Hadoop’s MapReduce.
* **Machine learning**: Machine learning is the study and practice of designing systems that can learn, adjust, and improve based on the data fed to them. This typically involves implementation of predictive and statistical algorithms that can continually zero in on “correct” behavior and insights as more data flows through the system.
* **Map reduce (big data algorithm)**: Map reduce (the big data algorithm, not Hadoop’s MapReduce computation engine) is an algorithm for scheduling work on a computing cluster. The process involves splitting the problem set up (mapping it to different nodes) and computing over them to produce intermediate results, shuffling the results to align like sets, and then reducing the results by outputting a single value for each set.
* **NoSQL**: NoSQL is a broad term referring to databases designed outside of the traditional relational model. NoSQL databases have different trade-offs compared to relational databases, but are often well-suited for big data systems due to their flexibility and frequent distributed-first architecture.
* **Stream processing**: Stream processing is the practice of computing over individual data items as they move through a system. This allows for real-time analysis of the data being fed to the system and is useful for time-sensitive operations using high velocity metrics.

Conclusion

Big data is a broad, rapidly evolving topic. While it is not well-suited for all types of computing, many organizations are turning to big data for certain types of work loads and using it to supplement their existing analysis and business tools. Big data systems are uniquely suited for surfacing difficult-to-detect patterns and providing insight into behaviors that are impossible to find through conventional means. By correctly implement systems that deal with big data, organizations can gain incredible value from data that is already available.

**Types Of Big Data**

Following are the types of Big Data:

1. **Structured**
2. **Unstructured**
3. **Semi-structured**

**Structured**

Any data that can be stored, accessed and processed in the form of fixed format is termed as a 'structured' data. Over the period of time, talent in computer science has achieved greater success in developing techniques for working with such kind of data (where the format is well known in advance) and also deriving value out of it. However, nowadays, we are foreseeing issues when a size of such data grows to a huge extent, typical sizes are being in the rage of multiple zettabytes.

***Do you know? 1021 bytes*** equal to ***1 zettabyte*** or ***one billion terabytes*** forms ***a zettabyte***.

Looking at these figures one can easily understand why the name Big Data is given and imagine the challenges involved in its storage and processing.

**Examples Of Structured Data**

An 'Employee' table in a database is an example of Structured Data

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee\_ID** | **Employee\_Name** | **Gender** | **Department** | **Salary\_In\_lacs** |
| 2365 | Rajesh Kulkarni | Male | Finance | 650000 |
| 3398 | Pratibha Joshi |  |  |  |

### ****Unstructured****

Any data with unknown form or the structure is classified as unstructured data. In addition to the size being huge, un-structured data poses multiple challenges in terms of its processing for deriving value out of it. A typical example of unstructured data is a heterogeneous data source containing a combination of simple text files, images, videos etc. Now day organizations have wealth of data available with them but unfortunately, they don't know how to derive value out of it since this data is in its raw form or unstructured format.

**Examples Of Un-structured Data**

The output returned by 'Google Search'

### ****Semi-structured****

Semi-structured data can contain both the forms of data. We can see semi-structured data as a structured in form but it is actually not defined with e.g. a table definition in relational [DBMS](https://www.guru99.com/what-is-dbms.html). Example of semi-structured data is a data represented in an XML file.

Examples Of Semi-structured Data

Personal data stored in an XML file-

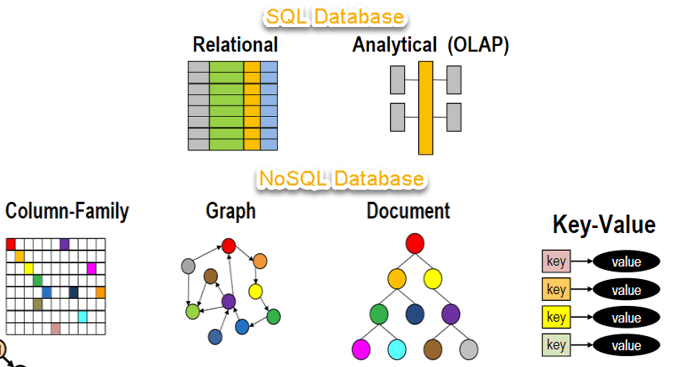
<rec><name>Prashant Rao</name><sex>Male</sex><age>35</age></rec>

**What is NoSQL?**

**NoSQL** Database is a non-relational Data Management System, that does not require a fixed schema. It avoids joins, and is easy to scale. The major purpose of using a NoSQL database is for distributed data stores with humongous data storage needs. NoSQL is used for Big data and real-time web apps. For example, companies like Twitter, Facebook and Google collect terabytes of user data every single day.

**NoSQL database** stands for "Not Only SQL" or "Not SQL." Though a better term would be "NoREL", NoSQL caught on. Carl Strozz introduced the NoSQL concept in 1998.

Traditional RDBMS uses SQL syntax to store and retrieve data for further insights. Instead, a NoSQL database system encompasses a wide range of database technologies that can store structured, semi-structured, unstructured and polymorphic data. Let's understand about NoSQL with a diagram in this NoSQL database tutorial:

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori1.png)

## Why NoSQL?

The concept of NoSQL databases became popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data. The system response time becomes slow when you use RDBMS for massive volumes of data.

To resolve this problem, we could "scale up" our systems by upgrading our existing hardware. This process is expensive.

The alternative for this issue is to distribute database load on multiple hosts whenever the load increases. This method is known as "scaling out."

## https://www.guru99.com/images/1/101818_0537_NoSQLTutori2.pngFeatures of NoSQL

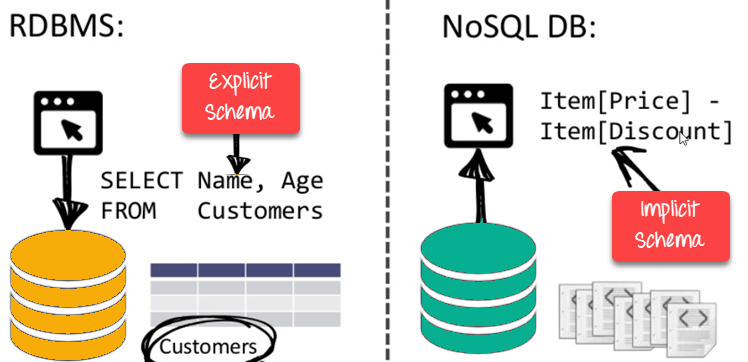
**Non-relational**

* NoSQL databases never follow the [relational model](https://www.guru99.com/relational-data-model-dbms.html)
* Never provide tables with flat fixed-column records
* Work with self-contained aggregates or BLOBs(A binary large object (**blob**) is concentrated binary data that's compressed into an individual **file** inside a database. The large size of the **file** means they need special storage treatment. **Blobs** are binary, which means they are usually images, audio or other media.)
* Doesn't require object-relational mapping and data normalization
* No complex features like query languages, query planners,

referential integrity joins, ACID

**Schema-free**

* NoSQL databases are either schema-free or have relaxed schemas
* Do not require any sort of definition of the schema of the data
* Offers heterogeneous structures of data in the same domain

**Simple API**

* Offers easy to use interfaces for storage and querying data provided
* APIs allow low-level data manipulation & selection methods
* Text-based protocols mostly used with HTTP REST with JSON(**REST** stands for REpresentational State Transfer. It means when a RESTful API is called, the server will transfer to the client a representation of the state of the requested resource. For example, when a developer calls Instagram API to fetch a specific user (the resource), the API will return the state of that user, including their name, the number of posts that user posted on Instagram so far, how many followers they have, and more. **JSON** stands for JavaScript Object Notation. **JSON** is a lightweight format for storing and transporting data. **JSON** is often used when data is sent from a server to a web page. **JSON** is "self-describing" and easy to understand.)
* Mostly used no standard based NoSQL query language
* Web-enabled databases running as internet-facing services

**Distributed**

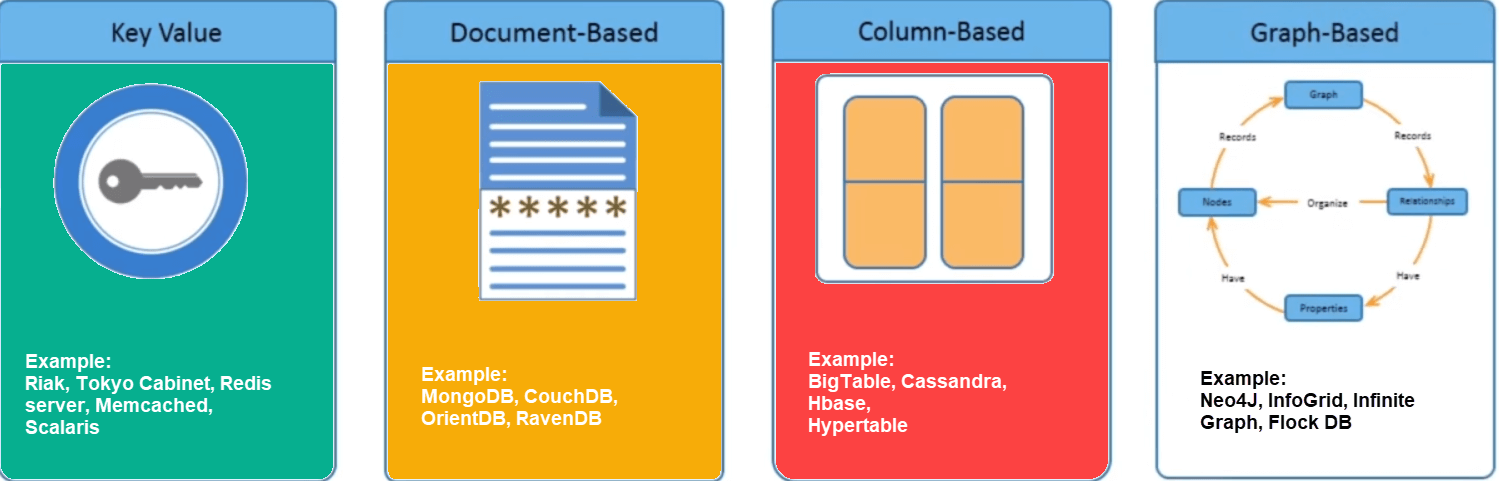
* Multiple NoSQL databases can be executed in a distributed fashion
* Offers auto-scaling and fail-over capabilities
* Often ACID concept can be sacrificed for scalability and throughput
* Mostly no synchronous replication between distributed nodes Asynchronous Multi-Master Replication, peer-to-peer, HDFS Replication
* Only providing eventual consistency
* Shared Nothing Architecture. This enables less coordination and higher distribution

## https://www.guru99.com/images/1/101818_0537_NoSQLTutori4.pngTypes of NoSQL Databases

**NoSQL Databases** are mainly categorized into four types: Key-value pair, Column-oriented, Graph-based and Document-oriented. Every category has its unique attributes and limitations. None of the above-specified database is better to solve all the problems. Users should select the database based on their product needs.

Types of NoSQL Databases:

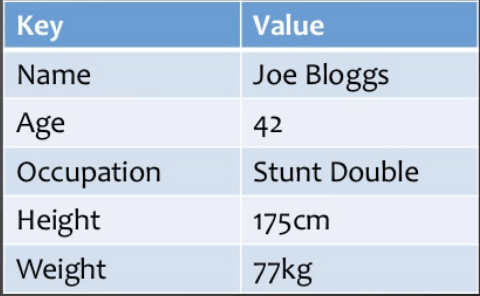
* Key-value Pair Based
* Column-oriented Graph
* Graphs based
* Document-oriented

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori5.png)

### Key Value Pair Based

Data is stored in key/value pairs. It is designed in such a way to handle lots of data and heavy load.

Key-value pair storage databases store data as a hash table where each key is unique, and the value can be a JSON, BLOB(Binary Large Objects), string, etc.

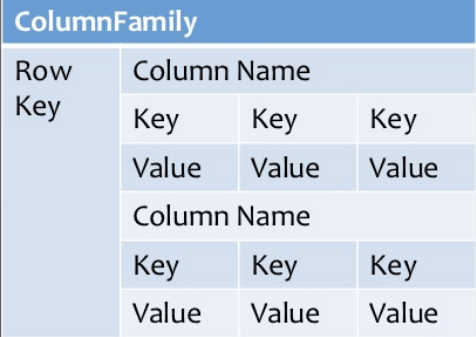
[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori6.png)

It is one of the most basic NoSQL database example. This kind of NoSQL database is used as a collection, dictionaries, associative arrays, etc. Key value stores help the developer to store schema-less data. They work best for shopping cart contents.

Redis, Dynamo, Riak are some NoSQL examples of key-value store DataBases. They are all based on Amazon's Dynamo paper.

### Column-based

Column-oriented databases work on columns and are based on BigTable paper by Google. Every column is treated separately. Values of single column databases are stored contiguously.

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori7.png)

Column based NoSQL database

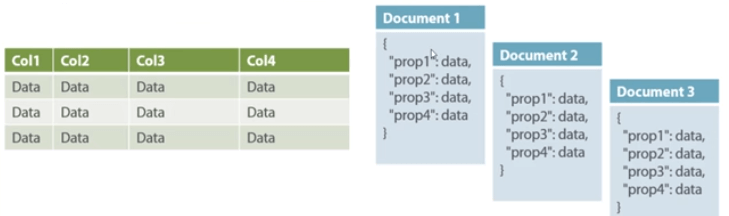
They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

Column-based NoSQL databases are widely used to manage data warehouses, [business intelligence](https://www.guru99.com/business-intelligence-definition-example.html), CRM, Library card catalogs,

HBase, Cassandra, HBase, Hypertable are NoSQL query examples of column based database.

### Document-Oriented:

Document-Oriented NoSQL DB stores and retrieves data as a key value pair but the value part is stored as a document. The document is stored in JSON or XML formats. The value is understood by the DB and can be queried.

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori8.png)

Relational Vs. Document

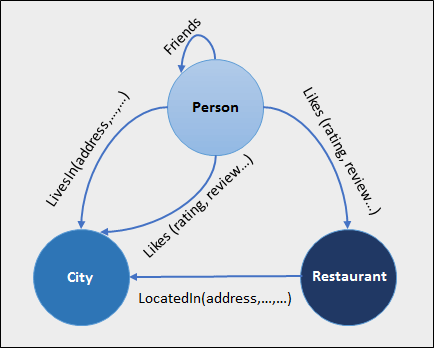
In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. You do not require to define which make it flexible.

The document type is mostly used for CMS systems, blogging platforms, real-time analytics & e-commerce applications. It should not use for complex transactions which require multiple operations or queries against varying aggregate structures.

Amazon SimpleDB, CouchDB, MongoDB, Riak, Lotus Notes, MongoDB, are popular Document originated DBMS systems.

### Graph-Based

A graph type database stores entities as well the relations amongst those entities. The entity is stored as a node with the relationship as edges. An edge gives a relationship between nodes. Every node and edge has a unique identifier.

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori9.png)

Compared to a relational database where tables are loosely connected, a Graph database is a multi-relational in nature. Traversing relationship is fast as they are already captured into the DB, and there is no need to calculate them.

Graph base database mostly used for social networks, logistics, spatial data.

Neo4J, Infinite Graph, OrientDB, FlockDB are some popular graph-based databases.

## Query Mechanism tools for NoSQL

The most common data retrieval mechanism is the REST-based retrieval of a value based on its key/ID with GET resource

Document store Database offers more difficult queries as they understand the value in a key-value pair. For example, CouchDB allows defining views with MapReduce

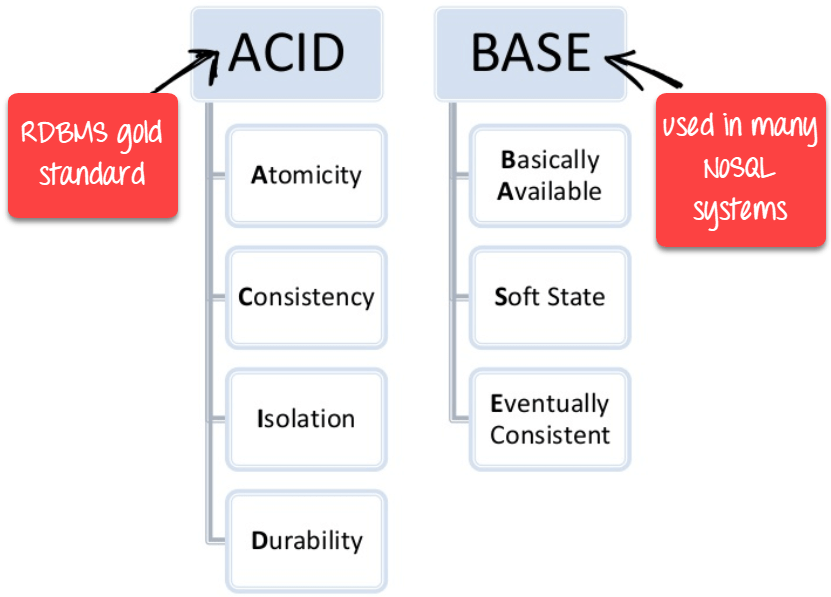
**Eventual Consistency**

The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability. Thus, changes made to any data item on one machine has to be propagated to other replicas.

Data replication may not be instantaneous as some copies will be updated immediately while others in due course of time. These copies may be mutually, but in due course of time, they become consistent. Hence, the name eventual consistency.

BASE: **B**asically **A**vailable, **S**oft state, **E**ventual consistency

* Basically, available means DB is available all the time as per CAP theorem
* Soft state means even without an input; the system state may change
* Eventual consistency means that the system will become consistent over time



**Advantages of NoSQL**

* Can be used as Primary or Analytic Data Source
* Big Data Capability
* No Single Point of Failure
* Easy Replication
* No Need for Separate Caching Layer
* It provides fast performance and horizontal scalability.
* Can handle structured, semi-structured, and unstructured data with equal effect
* Object-oriented programming which is easy to use and flexible
* NoSQL databases don't need a dedicated high-performance server
* Support Key Developer Languages and Platforms
* Simple to implement than using RDBMS
* It can serve as the primary data source for online applications.
* Handles big data which manages data velocity, variety, volume, and complexity
* Excels at distributed database and multi-data center operations
* Eliminates the need for a specific caching layer to store data
* Offers a flexible schema design which can easily be altered without downtime or service disruption

**Disadvantages of NoSQL**

* No standardization rules
* Limited query capabilities
* RDBMS databases and tools are comparatively mature
* It does not offer any traditional database capabilities, like consistency when multiple transactions are performed simultaneously.
* When the volume of data increases it is difficult to maintain unique values as keys become difficult
* Doesn't work as well with relational data
* The learning curve is stiff for new developers
* Open source options so not so popular for enterprises.

**Summary**

* NoSQL is a non-relational DMS, that does not require a fixed schema, avoids joins, and is easy to scale
* The concept of NoSQL databases beccame popular with Internet giants like Google, Facebook, Amazon, etc. who deal with huge volumes of data
* In the year 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
* NoSQL databases never follow the relational model it is either schema-free or has relaxed schemas
* Four types of NoSQL Database are 1).Key-value Pair Based 2).Column-oriented Graph 3). Graphs based 4).Document-oriented
* NOSQL can handle structured, semi-structured, and unstructured data with equal effect
* CAP theorem consists of three words Consistency, Availability, and Partition Tolerance
* BASE stands for **B**asically **A**vailable, **S**oft state, **E**ventual consistency
* The term "eventual consistency" means to have copies of data on multiple machines to get high availability and scalability
* NOSQL offer limited query capabilities

### What is Hadoop?

Apache Hadoop is an open source software framework used to develop data processing applications which are executed in a distributed computing environment.

 Applications built using HADOOP are run on large data sets distributed across clusters of commodity computers. Commodity computers are cheap and widely available. These are mainly useful for achieving greater computational power at low cost.

Similar to data residing in a local file system of a personal computer system, in Hadoop, data resides in a distributed file system which is called as a **Hadoop Distributed File system**. The processing model is based on **'Data Locality'** concept wherein computational logic is sent to cluster nodes(server) containing data. This computational logic is nothing, but a compiled version of a program written in a high-level language such as Java. Such a program, processes data stored in Hadoop HDFS.

## Hadoop EcoSystem and Components

