**What is Data?**

The quantities, characters, or symbols on which operations are performed by a computer, which may be stored and transmitted in the form of electrical signals and recorded on magnetic, optical, or mechanical recording media.

**What is Big Data?**

Big Data is also **data** but with a **huge size**. Big Data is a term used to describe a collection of data that is huge in size and yet growing exponentially with time. In short such data is so large and complex that none of the traditional data management tools are able to store it or process it efficiently.

[](https://www.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa1.jpg)

**Examples Of Big Data**

Following are some the examples of Big Data-

The **New York Stock Exchange** generates about ***one terabyte*** of new trade data per day.

[](https://www.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa2.jpg)

**Social Media**

The statistic shows that ***500+terabytes*** of new data get ingested into the databases of social media site **Facebook**, every day. This data is mainly generated in terms of photo and video uploads, message exchanges, putting comments etc.

[](https://www.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa3.jpg)

A single **Jet engine** can generate ***10+terabytes*** of data in ***30 minutes*** of flight time. With many thousand flights per day, generation of data reaches up to many ***Petabytes.***

[](https://www.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa4.jpg)

**Types Of Big Data**

BigData' could be found in three forms:

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.

***Do you know?*** Data stored in a relational database management system is one example of a **'structured'** data.

**Examples Of Structured Data**

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

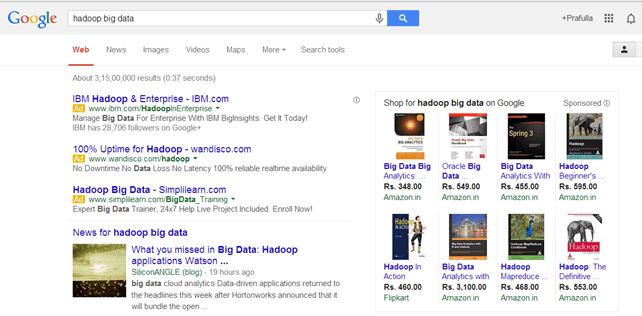
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Employee\_ID** | **Employee\_Name** | **Gender** | **Department** | **Salary\_In\_lacs** |
| 2365 | Rajesh Kulkarni | Male | Finance | 650000 |
| 3398 | Pratibha Joshi | Female | Admin | 650000 |
| 7465 | Shushil Roy | Male | Admin | 500000 |
| 7500 | Shubhojit Das | Male | Finance | 500000 |
| 7699 | Priya Sane | Female | Finance | 550000 |

**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'

[](https://www.guru99.com/images/Big_Data/061114_0759_WhatIsBigDa5.png)

**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. 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>

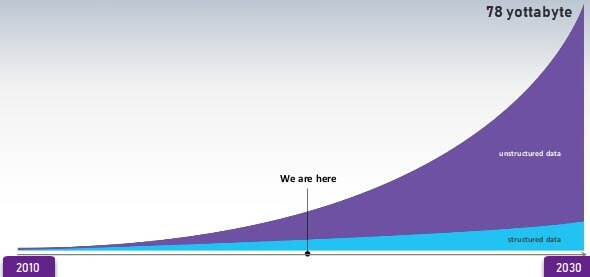
<rec><name>Seema R.</name><sex>Female</sex><age>41</age></rec>

<rec><name>Satish Mane</name><sex>Male</sex><age>29</age></rec>

<rec><name>Subrato Roy</name><sex>Male</sex><age>26</age></rec>

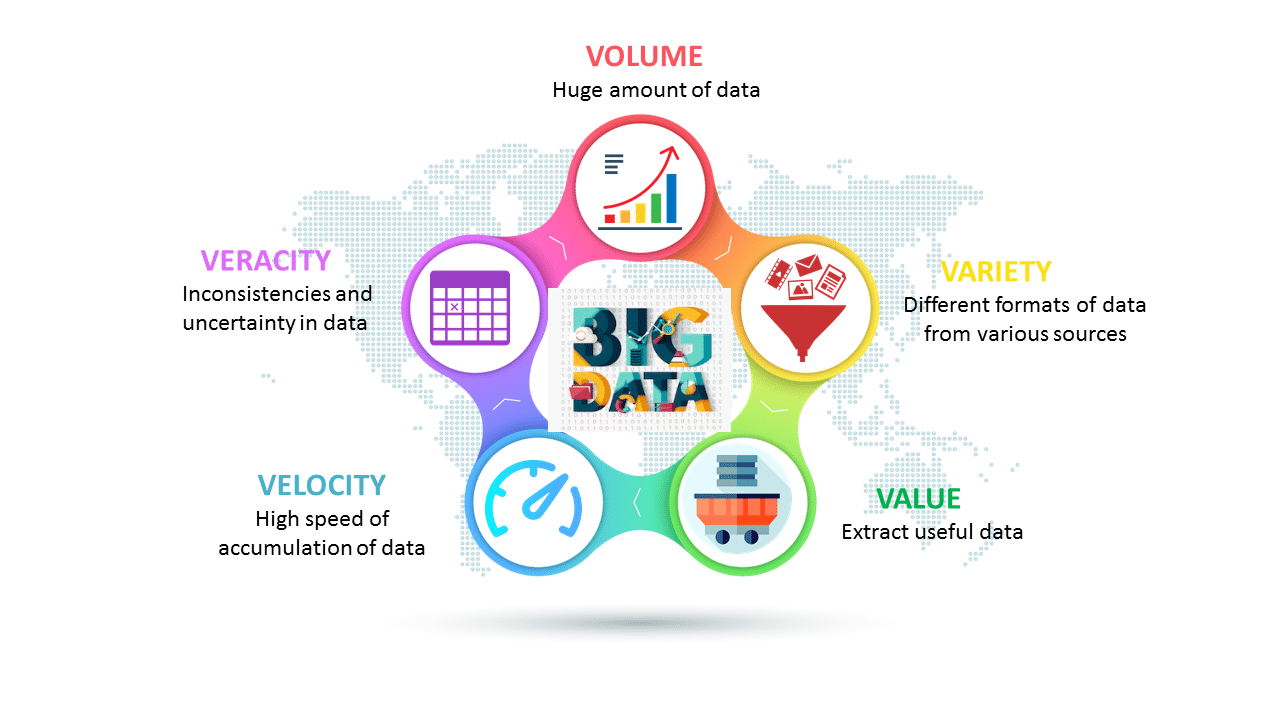
<rec><name>Jeremiah J.</name><sex>Male</sex><age>35</age></rec>

**Data Growth over the years**

[](https://www.guru99.com/images/1/big-data-growth.jpg)

Please note that web application data, which is unstructured, consists of log files, transaction history files etc. OLTP systems are built to work with structured data wherein data is stored in relations (tables).

**Characteristics Of Big Data**



***(i) Volume –*** The name Big Data itself is related to a size which is enormous. Size of data plays a very crucial role in determining value out of data. Also, whether a particular data can actually be considered as a Big Data or not, is dependent upon the volume of data. Hence, **'Volume'** is one characteristic which needs to be considered while dealing with Big Data.

***(ii) Variety –*** The next aspect of Big Data is its **variety**.

Variety refers to heterogeneous sources and the nature of data, both structured and unstructured. During earlier days, spreadsheets and databases were the only sources of data considered by most of the applications. Nowadays, data in the form of emails, photos, videos, monitoring devices, PDFs, audio, etc. are also being considered in the analysis applications. This variety of unstructured data poses certain issues for storage, mining and analyzing data.

***(iii) Velocity –*** The term **'velocity'** refers to the speed of generation of data. How fast the data is generated and processed to meet the demands, determines real potential in the data.

Big Data Velocity deals with the speed at which data flows in from sources like business processes, application logs, networks, and social media sites, sensors,[Mobile](https://www.guru99.com/mobile-testing.html)devices, etc. The flow of data is massive and continuous.

***(iv) Veracity –*** This refers to the inconsistency which can be shown by the data at times, thus hampering the process of being able to handle and manage the data effectively.

***(iv) Value –*** This refers to extract useful data .

**Benefits of Big Data Processing**

Ability to process Big Data brings in multiple benefits, such as-

* + Businesses can utilize outside intelligence while taking decisions

Access to social data from search engines and sites like facebook, twitter are enabling organizations to fine tune their business strategies.

* + Improved customer service

Traditional customer feedback systems are getting replaced by new systems designed with Big Data technologies. In these new systems, Big Data and natural language processing technologies are being used to read and evaluate consumer responses.

* + Early identification of risk to the product/services, if any
  + Better operational efficiency

Big Data technologies can be used for creating a staging area or landing zone for new data before identifying what data should be moved to the data warehouse. In addition, such integration of Big Data technologies and data warehouse helps an organization to offload infrequently accessed data.

**Summary**

* Big Data is defined as data that is huge in size. Bigdata is a term used to describe a collection of data that is huge in size and yet growing exponentially with time.
* Examples of Big Data generation includes stock exchanges, social media sites, jet engines, etc.
* Big Data could be 1) Structured, 2) Unstructured, 3) Semi-structured
* Volume, Variety, Velocity, and Variability are few Characteristics of Bigdata
* Improved customer service, better operational efficiency, Better Decision Making are few advantages of Bigdata

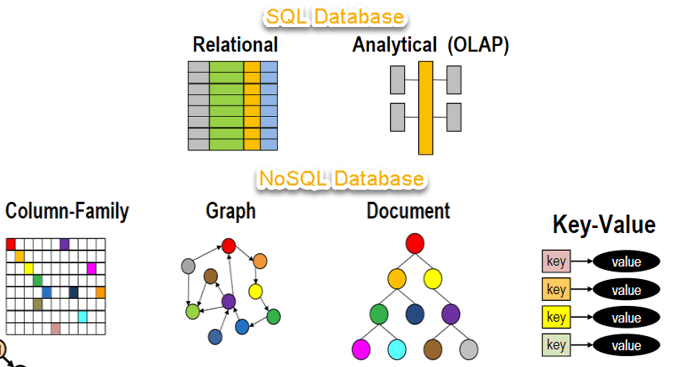
**What is NoSQL?**

NoSQL is a non-relational DMS, that does not require a fixed schema, avoids joins, and is easy to scale.

NoSQL database is used 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, Google that collect terabytes of user data every single day.

NoSQL database stands for "Not Only SQL" or "Not SQL."

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.

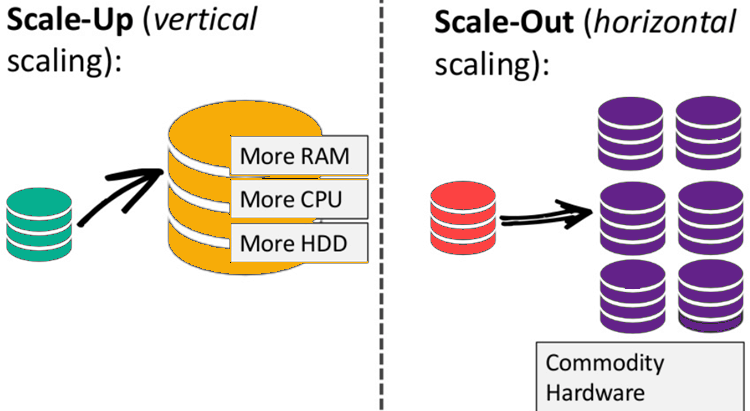
[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori1.png)

**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.png)

NoSQL database is non-relational, so it scales out better than relational databases as they are designed with web applications in mind.

**Brief History of NoSQL Databases**

* 1998- Carlo Strozzi use the term NoSQL for his lightweight, open-source relational database
* 2000- Graph database Neo4j is launched
* 2004- Google BigTable is launched
* 2005- CouchDB is launched
* 2007- The research paper on Amazon Dynamo is released
* 2008- Facebooks open sources the Cassandra project
* 2009- The term NoSQL was reintroduced

**Features of NoSQL**

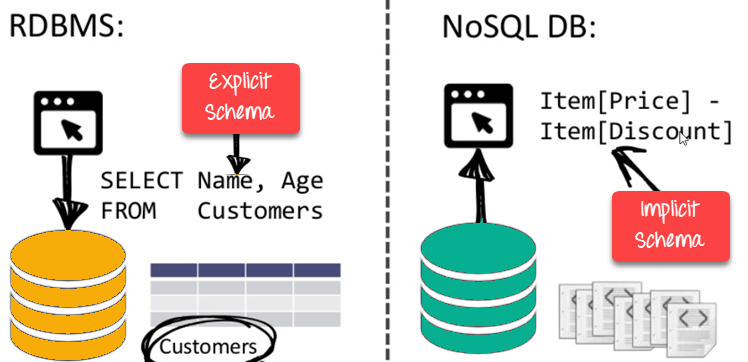
**Non-relational**

* NoSQL databases never follow the relational model
* Never provide tables with flat fixed-column records
* Work with self-contained aggregates or BLOBs
* 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

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori3.png)

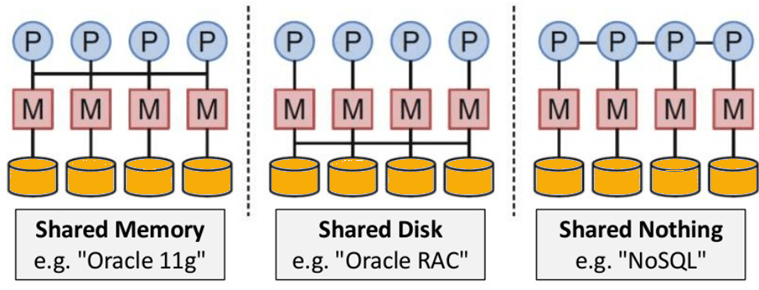
NoSQL is Schema-Free

**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
* Mostly used no standard based query language
* Web-enabled databases running as internet-facing services

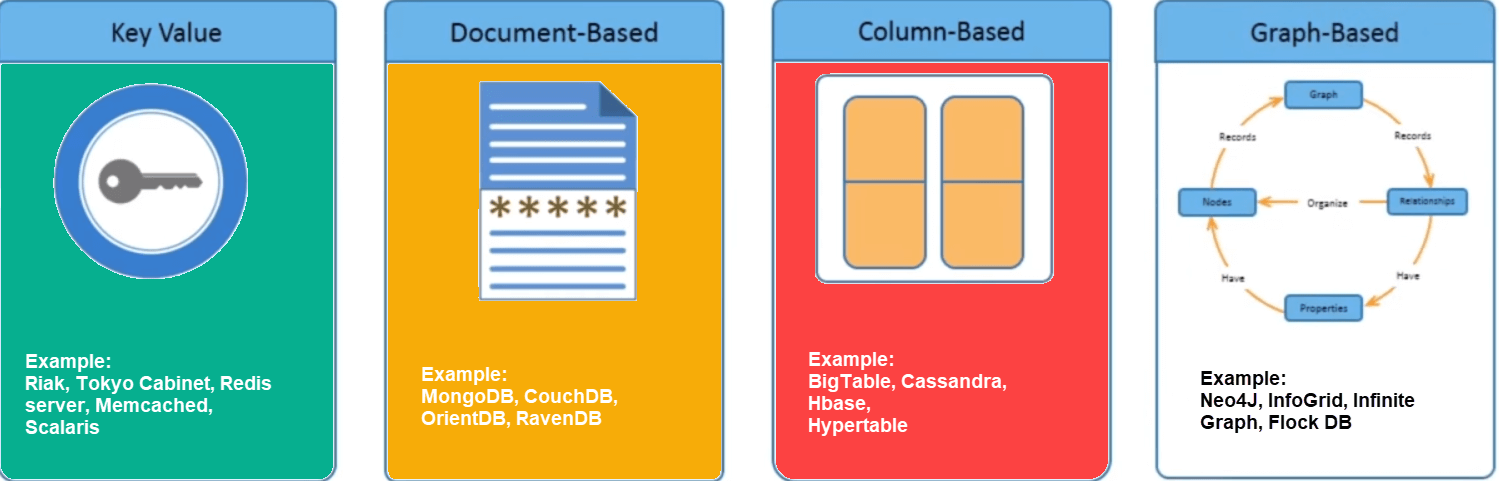
**Distributed**

* 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.png)

NoSQL is Shared Nothing.

**Types of NoSQL Databases**

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

There are mainly four categories of NoSQL databases. Each of these categories has its unique attributes and limitations. No specific database is better to solve all problems. You should select a database based on your product needs.

Let see all of them:

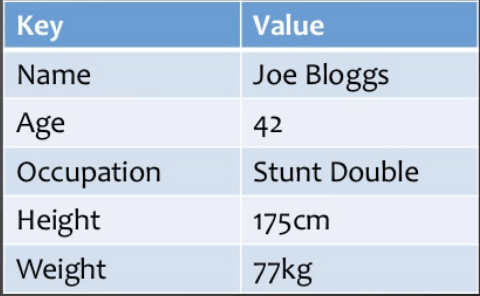
* Key-value Pair Based
* Column-oriented Graph
* Graphs based
* Document-oriented

**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.

For example, a key-value pair may contain a key like "Website" associated with a value like "Guru99".

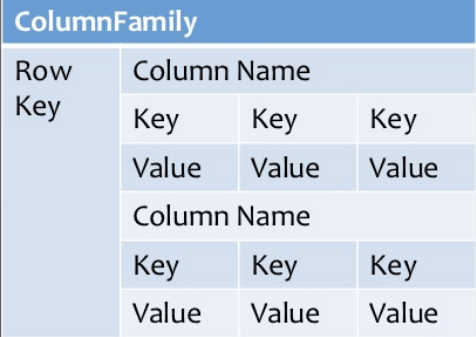
[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori6.png)

It is one of the most basic types of NoSQL databases. 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 examples of key-value store DataBases.

**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

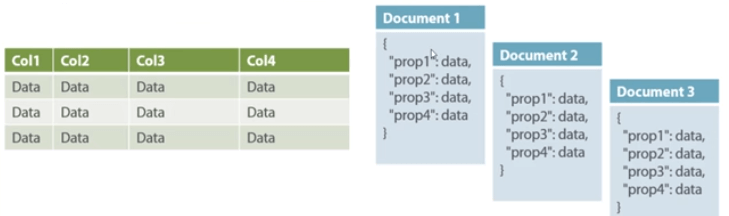
They deliver high performance on aggregation queries like SUM, COUNT, AVG, MIN etc. as the data is readily available in a column.

Column-based NoSQL databases are widely used to manage data warehouses, business intelligence, CRM, Library card catalogs,

HBase, Cassandra, HBase, Hypertable are 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

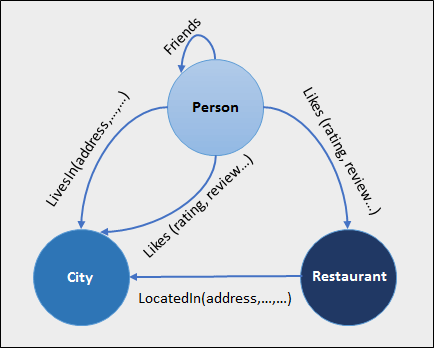
In this diagram on your left you can see we have rows and columns, and in the right, we have a document database which has a similar structure to JSON. Now for the relational database, you have to know what columns you have and so on. However, for a document database, you have data store like JSON object. 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

**What is the CAP Theorem?**

CAP theorem is also called brewer's theorem. It states that is impossible for a distributed data store to offer more than two out of three guarantees

1. Consistency
2. Availability
3. Partition Tolerance

**Consistency:**

The data should remain consistent even after the execution of an operation. This means once data is written, any future read request should contain that data. For example, after updating the order status, all the clients should be able to see the same data.

**Availability:**

The database should always be available and responsive. It should not have any downtime.

**Partition Tolerance:**

Partition Tolerance means that the system should continue to function even if the communication among the servers is not stable. For example, the servers can be partitioned into multiple groups which may not communicate with each other. Here, if part of the database is unavailable, other parts are always unaffected.

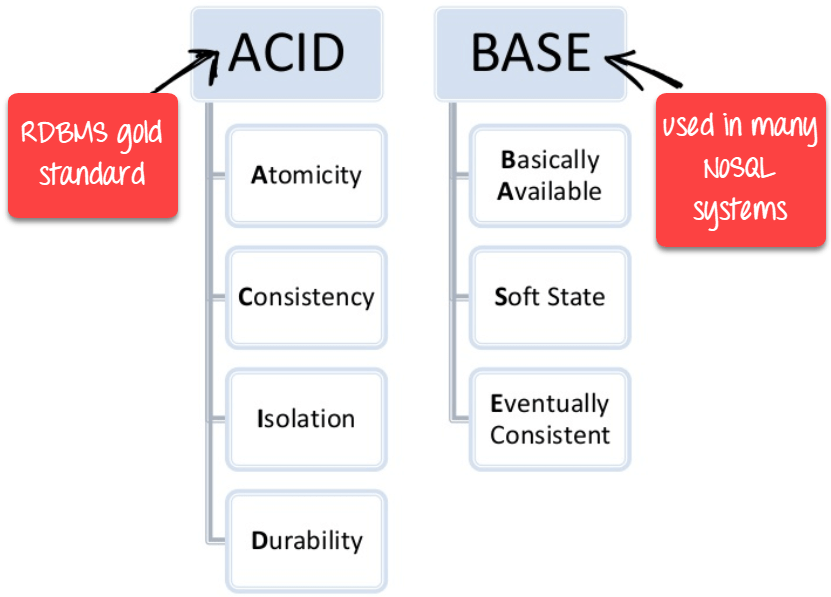
**Eventual Consistency**

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

[](https://www.guru99.com/images/1/101818_0537_NoSQLTutori10.png)

**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

Mongo DB:-

## Terminology and Concepts

Many concepts in MySQL have close analogs in MongoDB. The table below outlines the common concepts across MySQL and MongoDB.

| **MySQL** | **MongoDB** |
| --- | --- |
| ACID Transactions | ACID Transactions |
| Table | Collection |
| Row | Document |
| Column | Field |
| Secondary Index | Secondary Index |
| JOINs | Embedded documents, $lookup & $graphLookup |
| GROUP\_BY | Aggregation Pipeline |

## Query Language

Both MySQL and MongoDB have a rich query language. A comprehensive list of statements can be found in the [MongoDB documentation.](http://docs.mongodb.org/manual/reference/sql-comparison/)

|  |  |
| --- | --- |
| **MySQL** | **MongoDB** |
| INSERT INTO users (user\_id, age, status)  VALUES ('bcd001', 45, 'A') | db.users.insert({  user\_id: 'bcd001',  age: 45,ss  status: 'A'  }) |
| SELECT \* FROM users | db.users.find() |
| UPDATE users SET status = 'C'  WHERE age > 25 | db.users.update(  { age: { $gt: 25 } },  { $set: { status: 'C' } },  { multi: true }  ) |
| db.start\_transaction()  cursor.execute(orderInsert, orderData)  cursor.execute(stockUpdate, stockData)  db.commit() | s.start\_transaction()  orders.insert\_one(order, session=s)  stock.update\_one(item, stockUpdate, session=s)  s.commit\_transaction() |

## Find Documents by ID and other Attribute Values

When inserting a document into a collection it automatically gets a filed '\_id' that contains a unique identifier for this document. Retrieving a document when you know the \_id is quite easy. The following example shows how to find the document with the \_id=**568c28fffc4be30d44d0398e** from a collection called “**products**”:

Find by id

Find a JSON document by its internal object id.

**$** mongo localhost:27017/[database-name]  
**>** **db.products.find({"\_id": ObjectId("568c28fffc4be30d44d0398e")})**  
{  
  "\_id" : ObjectId("568c28fffc4be30d44d0398e"),  
  "product\_number" : 19,  
  "supplier\_id" : ObjectId("568c28fffc4be30d44d0397a"),  
  "title" : "Big Data: Principles and Best Practices of Scalable Realtime Data Systems",  
  "description" : "A book written by Nathan Marz and James Warren.",  
  "price" : 36.00  
}

If you are searching for a document using the \_id field you always get exactly one or no document back. It's also possible to search for other attributes, for example the title:

Find JSON document(s) by other attributes than the internal id.

**$** mongo localhost:27017/[database-name]  
**>** **db.products.find({"title": "Big Data: Principles and Best Practices of Scalable Realtime Data Systems"})**

Sometimes you have a collection of documents that contains a nested JSON structure. This means a value of a key- value tuple is another set of key- value pairs. The following document example shows this scenario:

{  
  "\_id" : ObjectId("568c28fffc4be30d44d039aa"),  
  "firstname" : "Max",  
  "lastname" : "Mustermann",  
  "email" : "m.mustermann@example.com",  
  "password" : "d9729feb74992cc3482b350163a1a010",  
  "last\_login" : "2015-01-07",  
  "note" : "Always pays in time, very good customer!",  
  "**address**" :  
  {  
    "country" : "Germany",  
    "street" : "Beispielstrasse 64",  
    "zip" : "62717"  
  }  
}

As you see the address attribute maps to another set of key- value pairs: country, street and zip. The following MongoDB finds all customers that are based in Germany:

Find JSON document(s) by attributes values in nested documents

**>** db.customers.find({"address.country": "**Germany**"})

## Find Documents using Logical Conditions

Sometimes you want to retrieve documents by the values of more than one attribute. For example: Let's find all customers from Germany that has been logged in today:

Find documents using the logical and operator.

**>** db.customers.find({  
  **$and**: [  
    {"address.country": "Germany"},  
    {"last\_login": "2019-09-08"}  
  ]  
})

In the next example we want to find all customers from the European Union. This means the customer must be from one country that is a member of that union:

Find documents using the logical or operator.  
Note: We will refactor this query in the next chapter.

**>** db.customers.find({  
  **$or**: [  
    {"address.country": "Austria"}, {"address.country": "Belgium"},  
    {"address.country": "Bulgaria"}, {"address.country": "Croatia"},  
    {"address.country": "Cyprus"}, {"address.country": "Czech Republic"},  
    {"address.country": "Denmark"}, {"address.country": "Estonia"},  
    {"address.country": "Finland"}, {"address.country": "France"},  
    {"address.country": "Germany"}, {"address.country": "Greece"},  
    {"address.country": "Hungary"}, {"address.country": "Ireland"},  
    {"address.country": "Italy"}, {"address.country": "Latvia"},  
    {"address.country": "Lithuania"}, {"address.country": "Luxembourg"},  
    {"address.country": "Malta"}, {"address.country": "The Netherlands"},  
    {"address.country": "Poland"}, {"address.country": "Portugal"},  
    {"address.country": "Romania"}, {"address.country": "Slovakia"},  
    {"address.country": "Slovenia"}, {"address.country": "Spain"},  
    {"address.country": "Sweden"}, {"address.country": "United Kingdom"}  
  ]  
})

The next example shows how to combine the “$and” and “$or” operators. Let's find all customers who have been logged in the last three days and who are based in Spain or France:

Find documents using the operators $or and $and in combination.  
Note: We will refactor this query in the next chapter.

**>** db.customers.find({  
  $and: [  
    {  
      $or: [{"last\_login": "2019-09-08"},  
            {"last\_login": "2019-09-07"},  
            {"last\_login": "2019-09-06"}]  
    },  
    {  
      $or: [{"address.country": "Spain"},  
            {"address.country": "France"}]  
    }  
  ]  
})

## Find Documents using Query Selectors

So far we only searched for documents by exact value comparison. Thankfully there are some more comparison operators than just an exact matching of a given value. For example we can search for all products that costs more than a certain amount or we can search for all customers that has the word “bad” in its note attribute.

But one thing at a time, first we'll have a look at all available operators and after that we will see some of them in action. The table below (copied form the official MonoDB documentation) shows an overview of those comparison operators:

| **Name** | **Description** |
| --- | --- |
| $eq | Matches values that are equal to a specified value. |
| $gt | Matches values that are greater than a specified value. |
| $gte | Matches values that are greater than or equal to a specified value. |
| $lt | Matches values that are less than a specified value. |
| $lte | Matches values that are less than or equal to a specified value. |
| $ne | Matches all values that are not equal to a specified value. |
| $in | Matches any of the values specified in an array. |
| $nin | Matches none of the values specified in an array. |

Source: [docs.mongodb.org](https://docs.mongodb.org/v3.0/reference/operator/query/)

As I have already noted, some queries from the last chapter can be expressed in a shorter way. With the “$in” operator we now have the tool to do that. Let's have a look at how this looks for the last example where we searched for all customers that have been logged in the last three days and are based in Spain or France:

Find customers from Spain or French who has been logged in the last three days using the $in operator.

**>** db.customers.find({  
  $and: [  
    { "last\_login": { $in: ["2019-09-08", "2019-09-07", "2019-09-06"] }},  
    { "address.country": { **$in**: ["Spain", "French"] }},  
  ]  
})

The $gte or $gt operators allows us to further simplify the query above. It will allow to find all customers whose last login time is greater than a specified value. The first intention is to use this operator with numbers but it also works with strings. In case two strings should be compared, it will decide which one is bigger according to the lexicographic order.

Find customers from Spain or French whose has been logged in the last three days using the $in and the $gte operator.

**>** db.customers.find({  
  $and: [  
    { "last\_login": { **$gte**: "2019-09-06" }},  
    { "address.country": { $in: ["Spain", "French"] }},  
  ]  
})

The opposite of the $gte/$gt operator is the $lte/$lt operator. We will use the $lt operator in the next example to show how to find the all customers that has not been logged in the last three days:

Find customers who has NOT been logged in the last three days using the $lt operator.

**>** db.customers.find({  
  "last\_login": { **$lt**: "2019-09-06" }  
})

The following query demonstrates how to use the “$nin” to find all customers that are NOT from Spain or French.

Find customers that are NOT from Spain or French unsing the $nin operator.

**>** db.customers.find({  
  "address.country": { **$nin**: ["Spain", "French"] }  
})