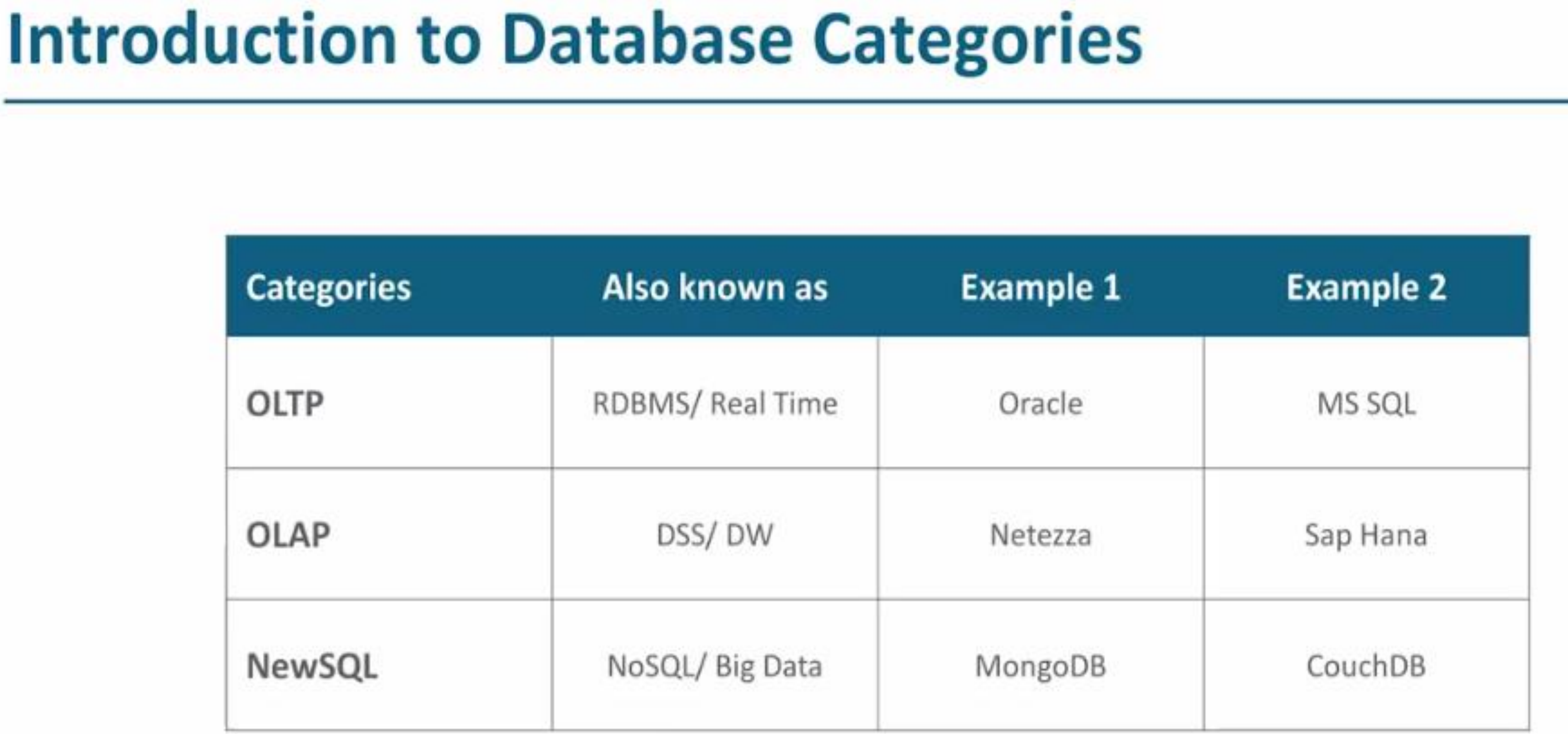
# NoSql & MongoDB

Kratika Sharma, Asst Professor, CBIT

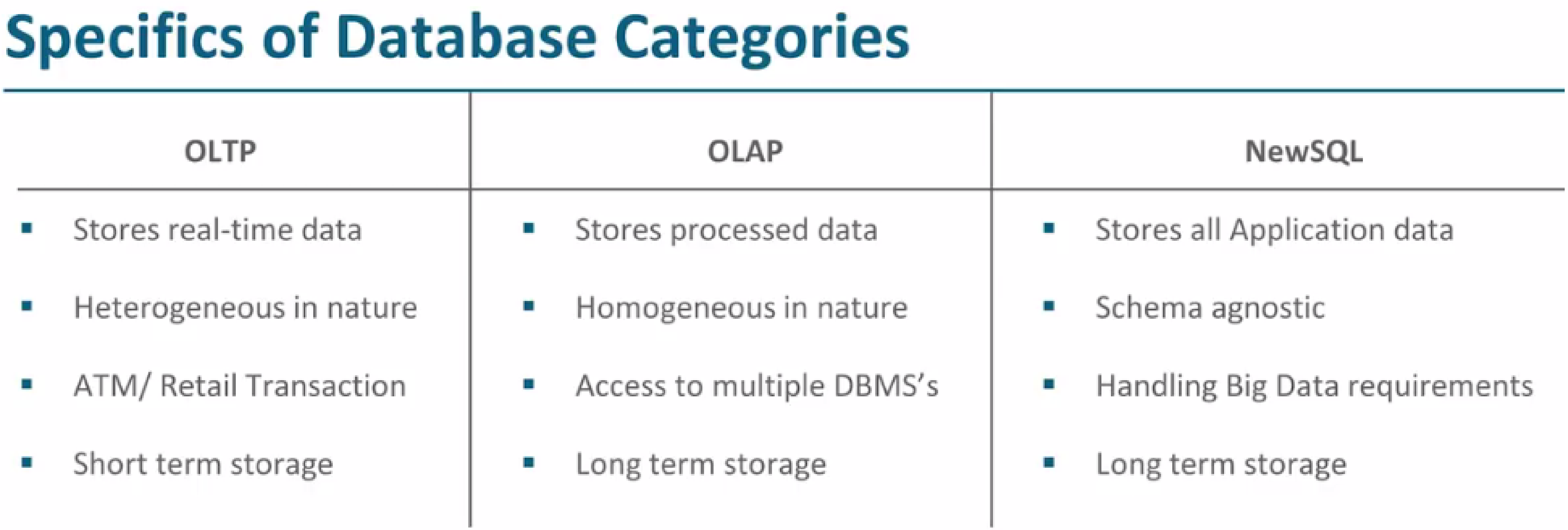
# Database – The Need

1. Store, organize, and manage large amounts of data on a single platform
2. Facilitate a well-planned platform for data analysis
3. Promote a disciplined approach to data management
4. One stop solution to manage security, multiuser access control, backup & recovery, etc.



# Introduction to Database Categories

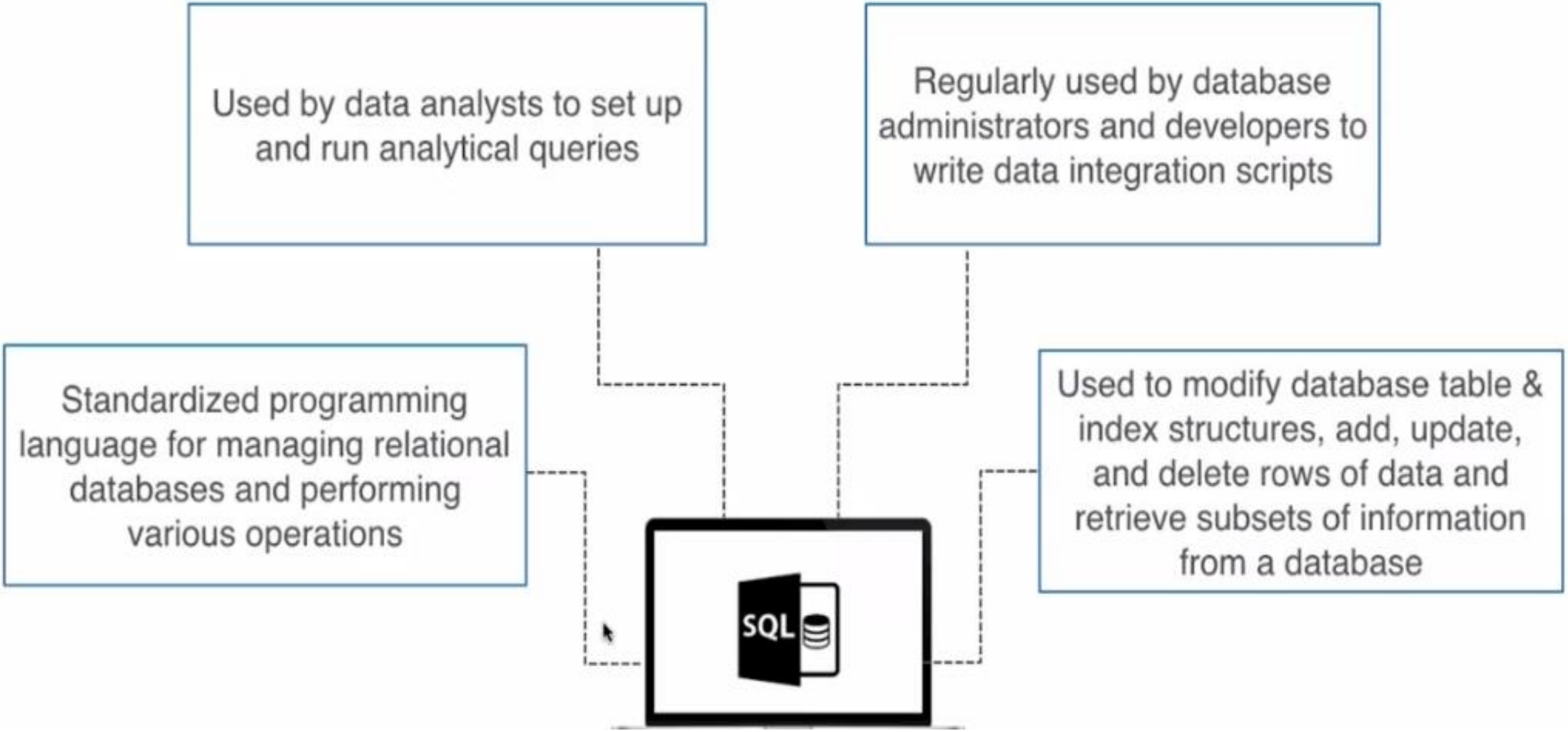
| Categories | Also known as | Example 1 | Example 2 |
|---|---|---|---|
| **OLTP** | RDBMS/ Real Time | Oracle | MS SQL |
| **OLAP** | DSS/ DW | Netezza | Sap Hana |
| **NewSQL** | NoSQL/ Big Data | MongoDB | CouchDB |



## Specifics of Database Categories

| OLTP | OLAP | NewSQL |
|---|---|---|
| Stores real-time data | Stores processed data | Stores all Application data |
| Heterogeneous in nature | Homogeneous in nature | Schema agnostic |
| ATM/ Retail Transaction | Access to multiple DBMS's | Handling Big Data requirements |
| Short term storage | Long term storage | Long term storage |

# SQL - Structured Query Language



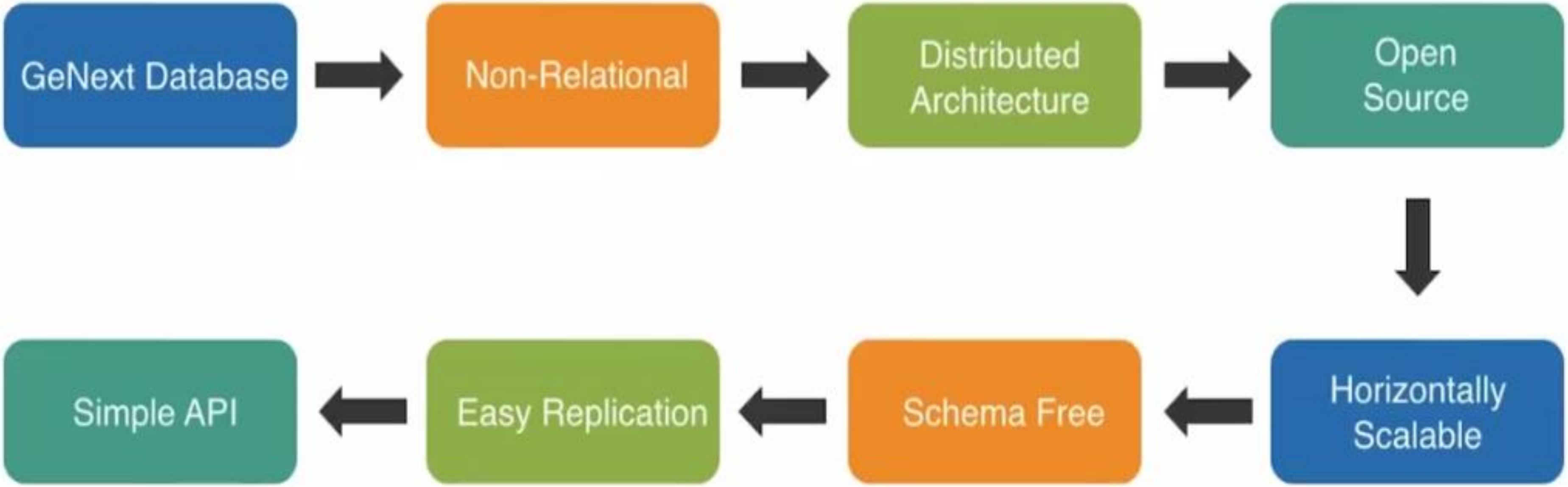
Used by data analysts to set up and run analytical queries

Regularly used by database administrators and developers to write data integration scripts

Standardized programming language for managing relational databases and performing various operations

Used to modify database table & index structures, add, update, and delete rows of data and retrieve subsets of information from a database

# NoSQL – You don't need SQL



GeNext Database

Non-Relational

Distributed Architecture

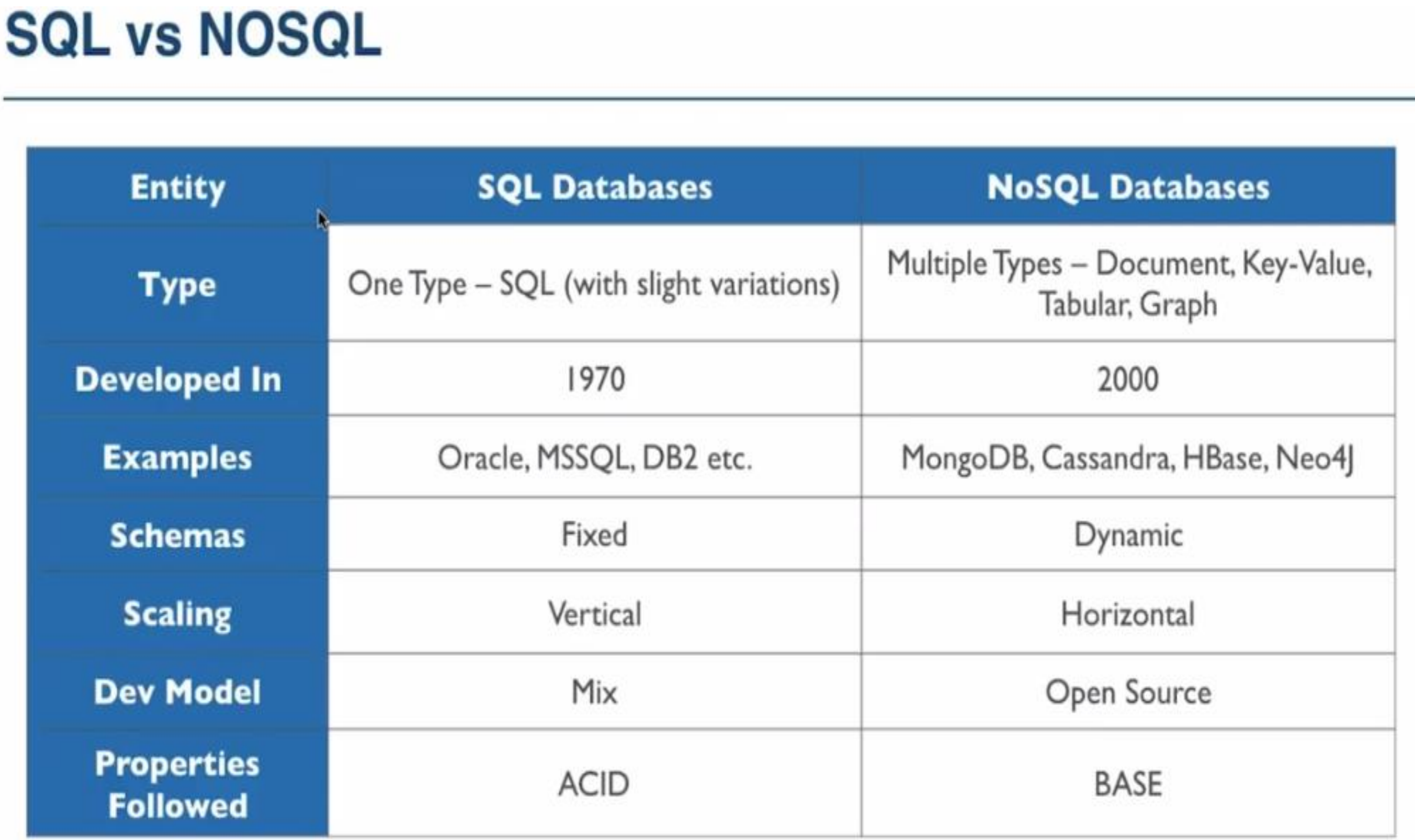
Open Source

Horizontally Scalable

Schema Free

Easy Replication

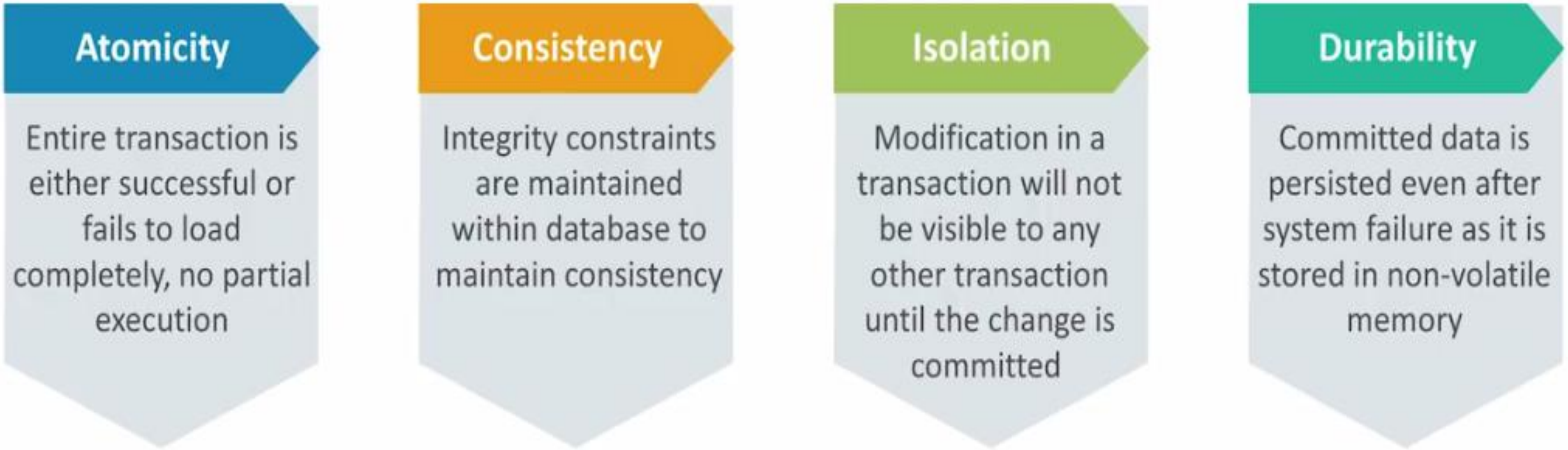
Simple API



# SQL vs NOSQL

| Entity | SQL Databases | NoSQL Databases |
|---|---|---|
| Type | One Type – SQL (with slight variations) | Multiple Types – Document, Key-Value, Tabular, Graph |
| Developed In | 1970 | 2000 |
| Examples | Oracle, MSSQL, DB2 etc. | MongoDB, Cassandra, HBase, Neo4J |
| Schemas | Fixed | Dynamic |
| Scaling | Vertical | Horizontal |
| Dev Model | Mix | Open Source |
| Properties Followed | ACID | BASE |

# SQL:- ACID Property



**Atomicity**

Entire transaction is either successful or fails to load completely, no partial execution

**Consistency**

Integrity constraints are maintained within database to maintain consistency

**Isolation**

Modification in a transaction will not be visible to any other transaction until the change is committed

**Durability**

Committed data is persisted even after system failure as it is stored in non-volatile memory

# NoSQL Base Property

**Basic Availability**

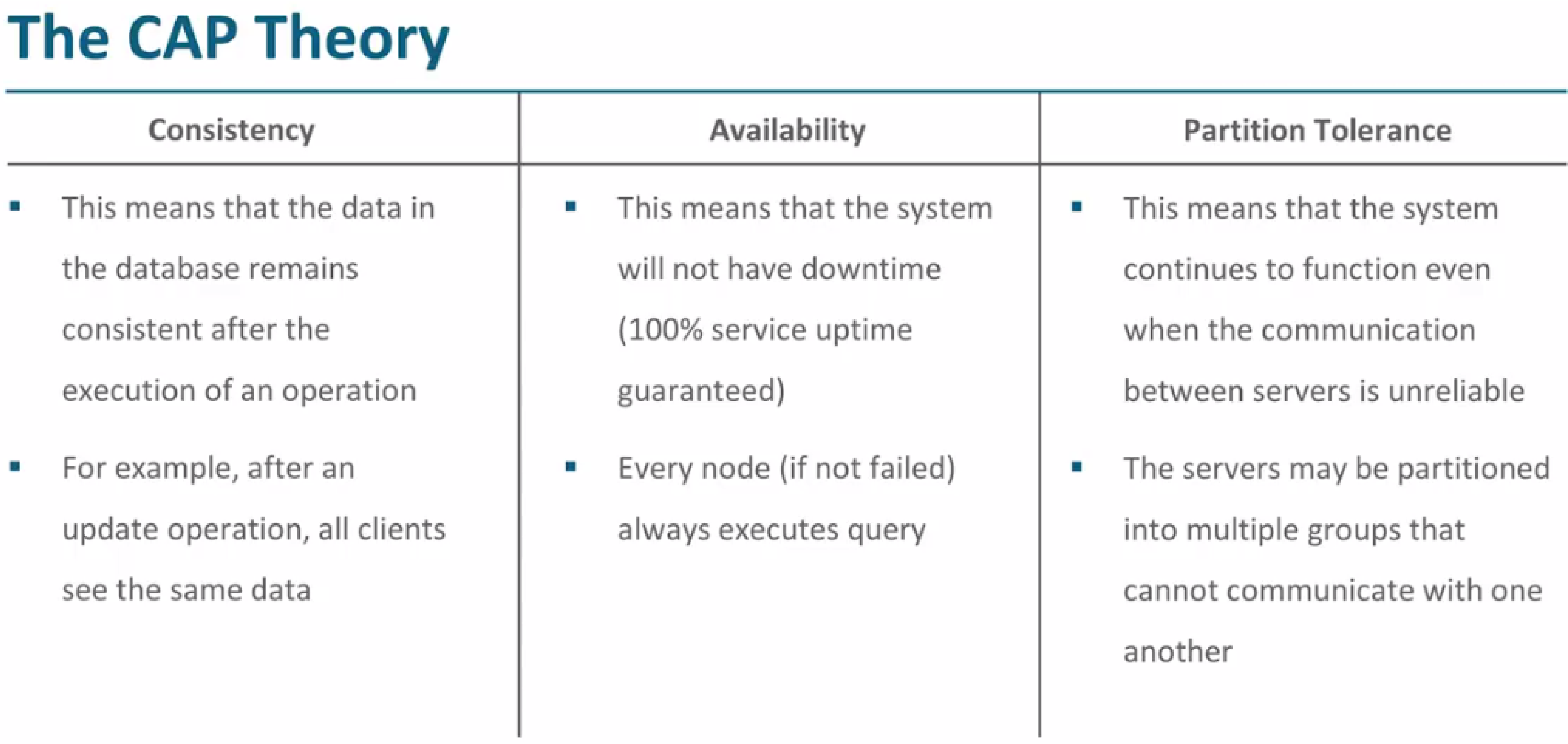
- The database appears to work most of the time

**Soft-state**

- Stores don’t have to be write-consistent or mutually consistent all the time

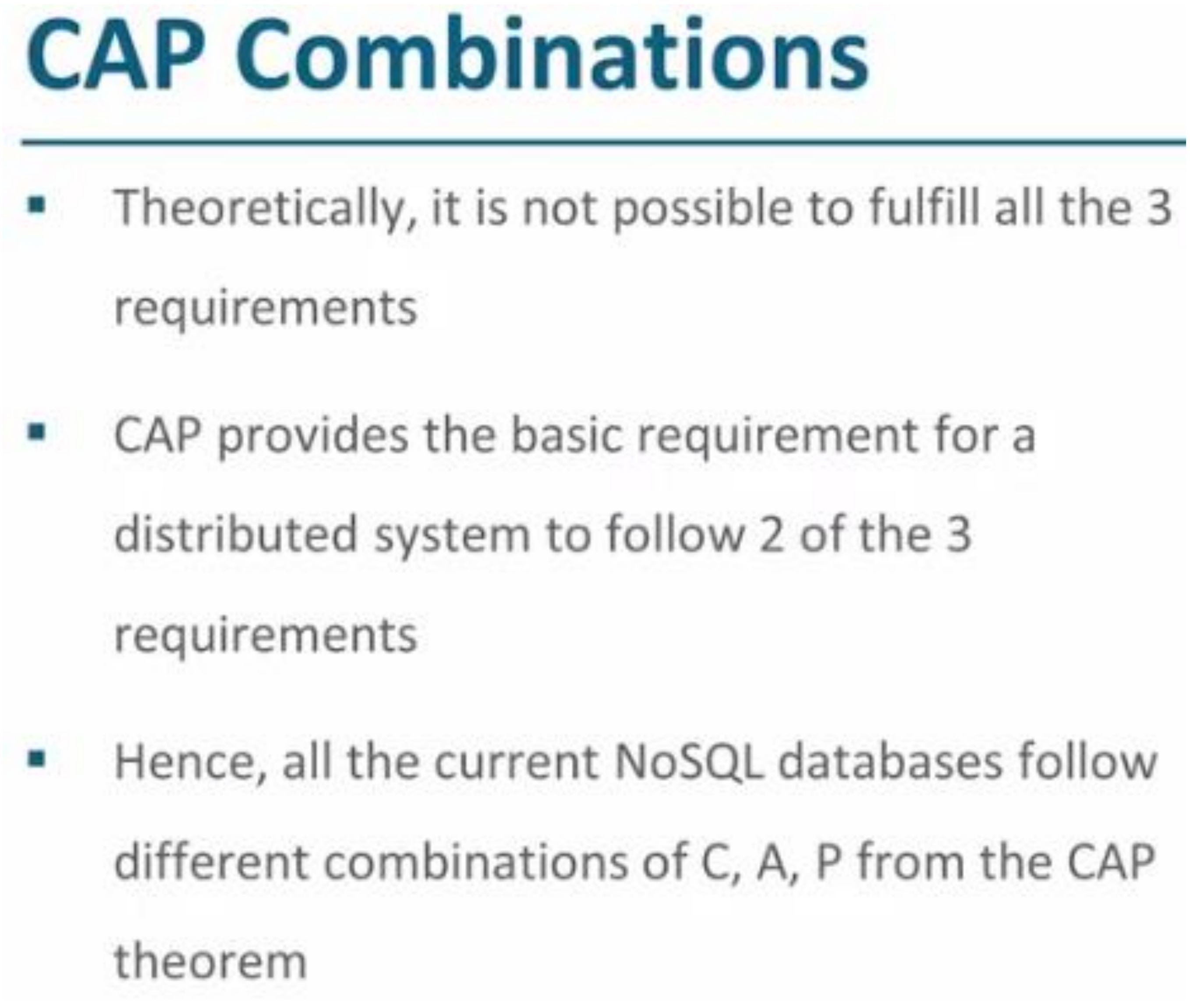
**Eventual consistency**

- Stores exhibit consistency at some later point



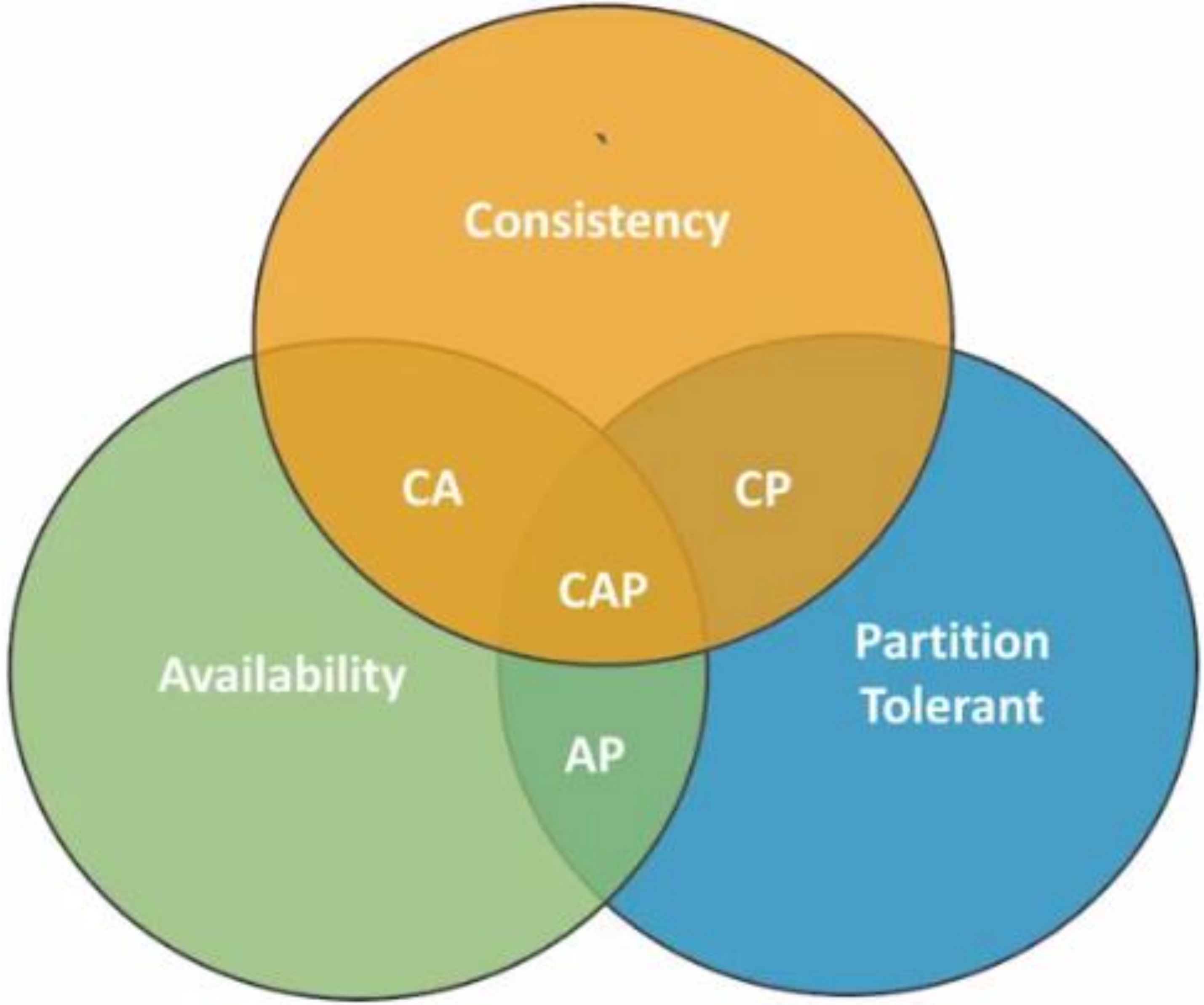
# The CAP Theory

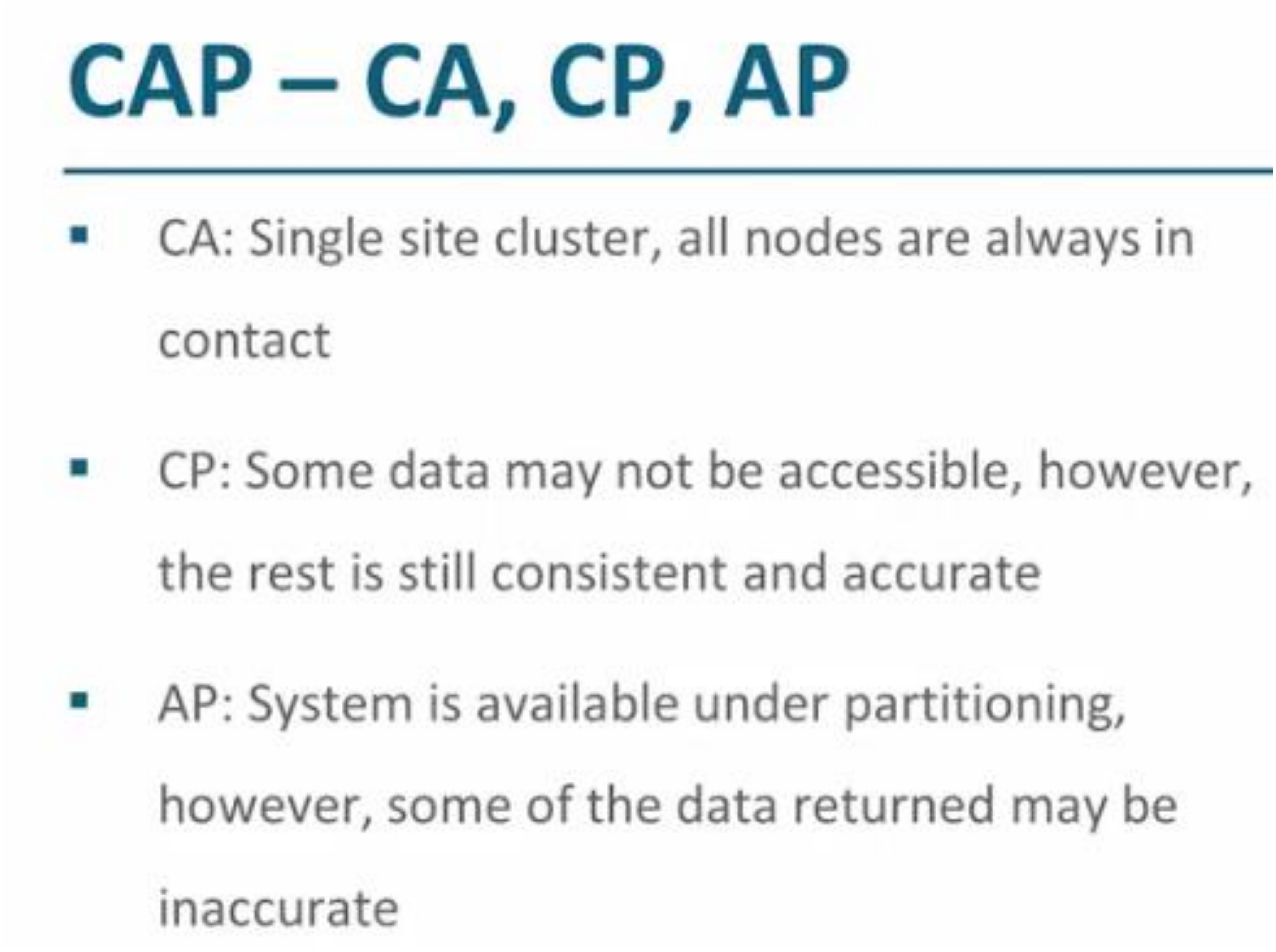
| Consistency | Availability | Partition Tolerance |
|---|---|---|
| ▪ This means that the data in the database remains consistent after the execution of an operation<br>▪ For example, after an update operation, all clients see the same data | ▪ This means that the system will not have downtime (100% service uptime guaranteed)<br>▪ Every node (if not failed) always executes query | ▪ This means that the system continues to function even when the communication between servers is unreliable<br>▪ The servers may be partitioned into multiple groups that cannot communicate with one another |



# CAP Combinations

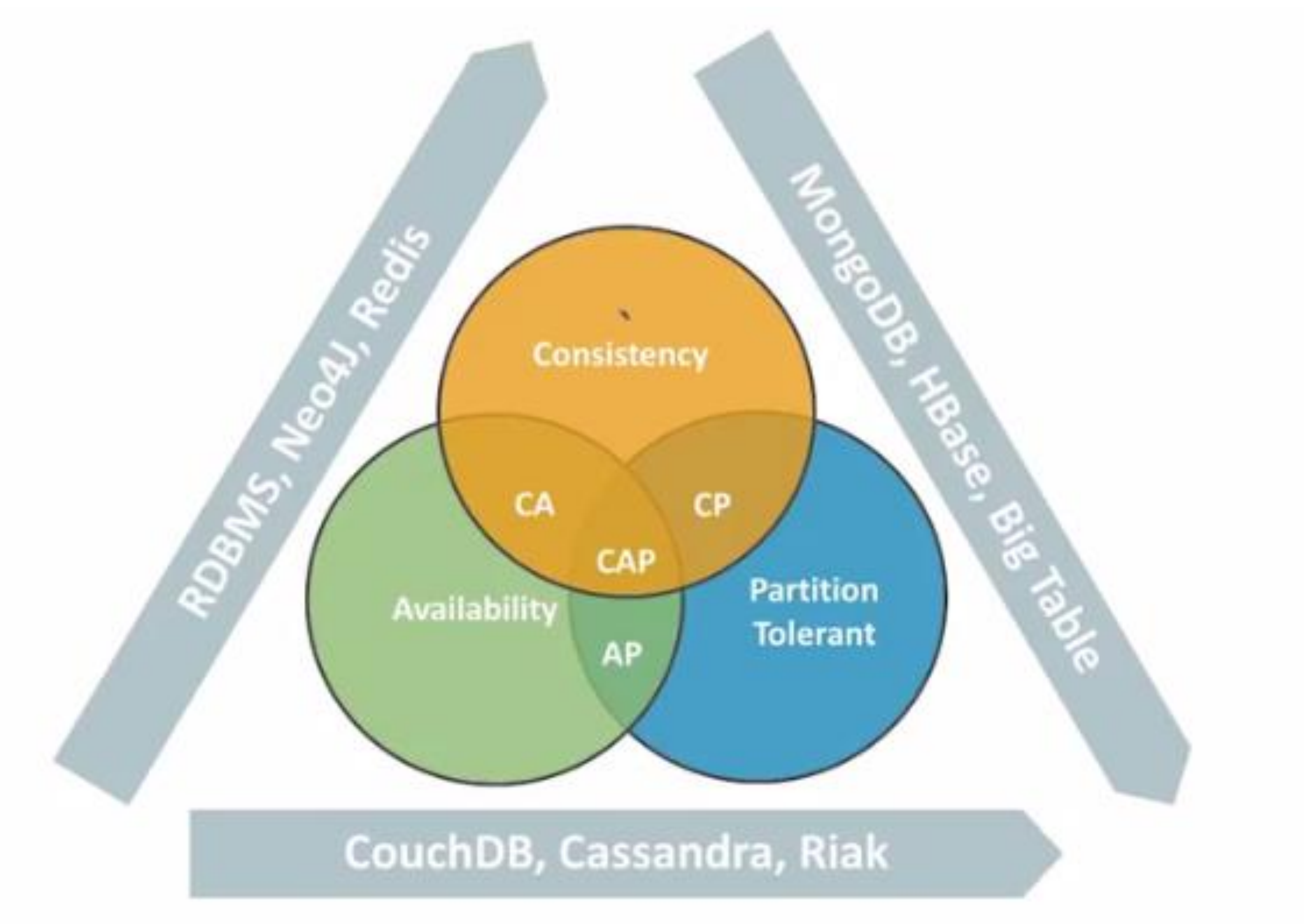
- Theoretically, it is not possible to fulfill all the 3 requirements
- CAP provides the basic requirement for a distributed system to follow 2 of the 3 requirements
- Hence, all the current NoSQL databases follow different combinations of C, A, P from the CAP theorem

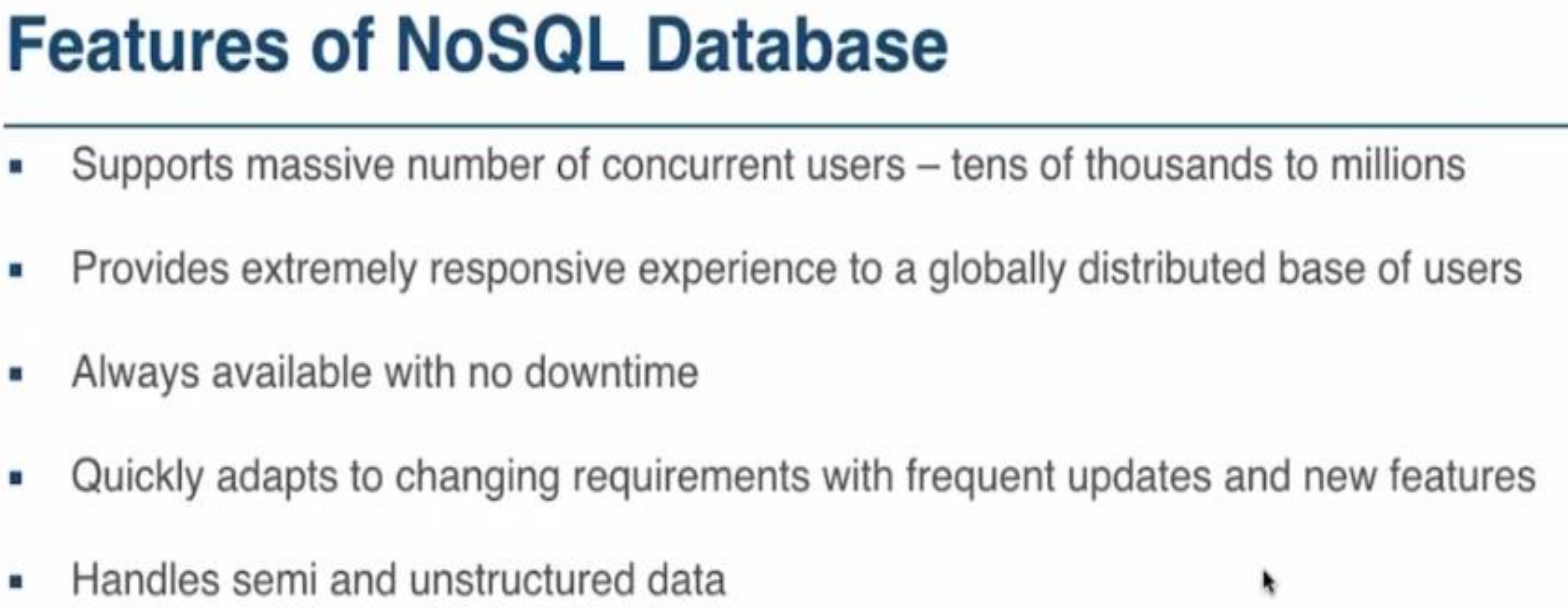

Consistency
CA
CP
CAP
Availability
Partition
Tolerant
AP



# CAP – CA, CP, AP

- CA: Single site cluster, all nodes are always in contact
- CP: Some data may not be accessible, however, the rest is still consistent and accurate
- AP: System is available under partitioning, however, some of the data returned may be inaccurate

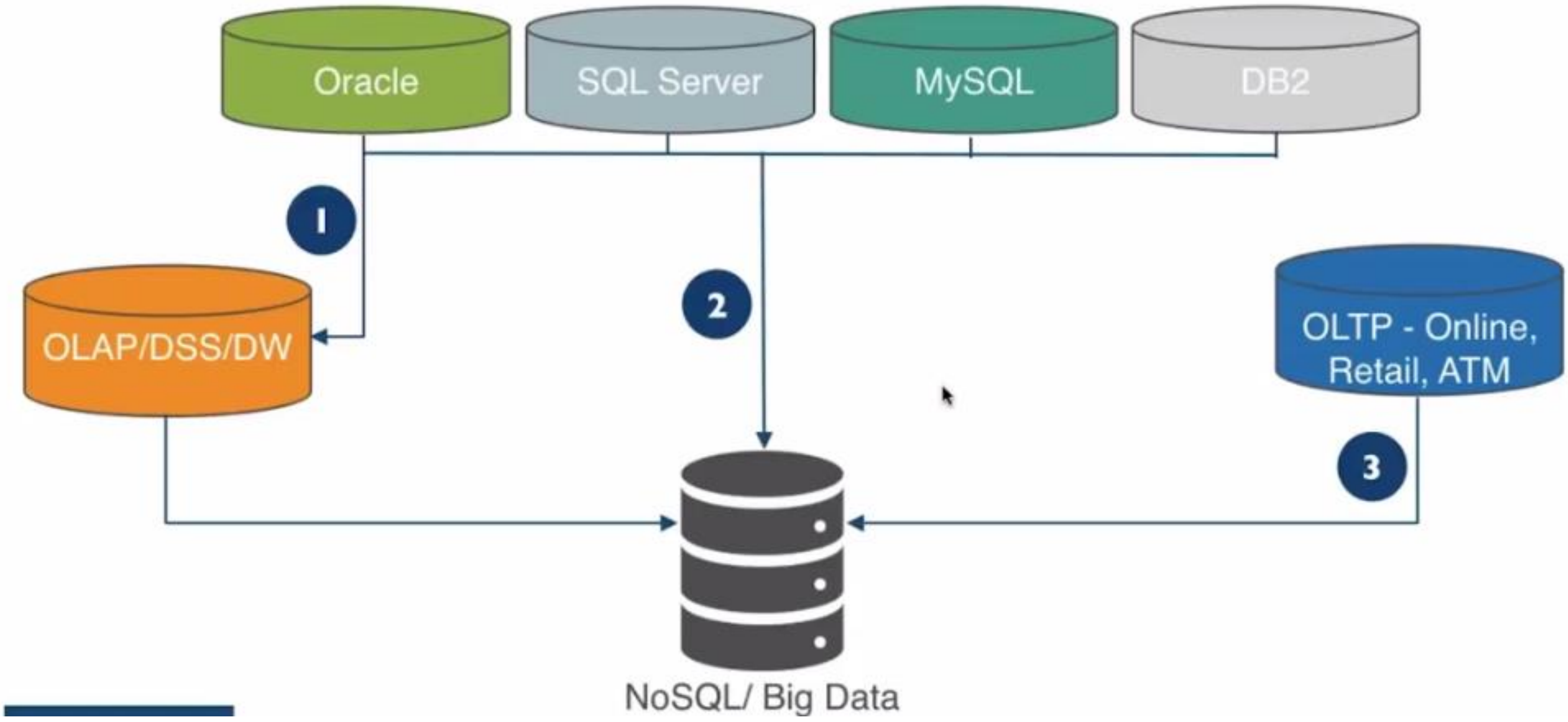

RDBMS, Neo4J, Redis
MongoDB, HBase, Big Table
Consistency
CA
CP
CAP
Availability
AP
Partition Tolerant
CouchDB, Cassandra, Riak



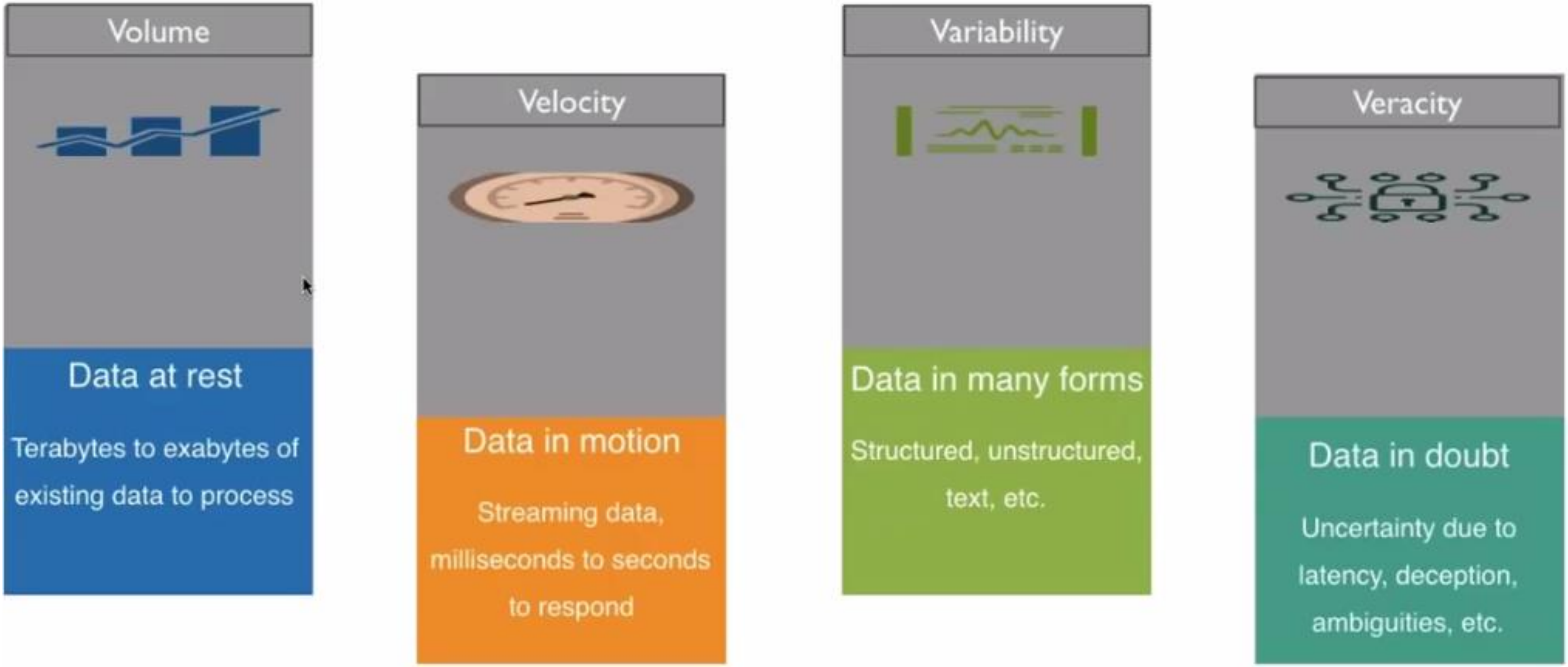
# Features of NoSQL Database

- Supports massive number of concurrent users – tens of thousands to millions
- Provides extremely responsive experience to a globally distributed base of users
- Always available with no downtime
- Quickly adapts to changing requirements with frequent updates and new features
- Handles semi and unstructured data

# NoSQL Database – Storage



# Advantages of NoSQL - 1



Volume

Data at rest

Terabytes to exabytes of existing data to process

Velocity

Data in motion

Streaming data, milliseconds to seconds to respond

Variability

Data in many forms

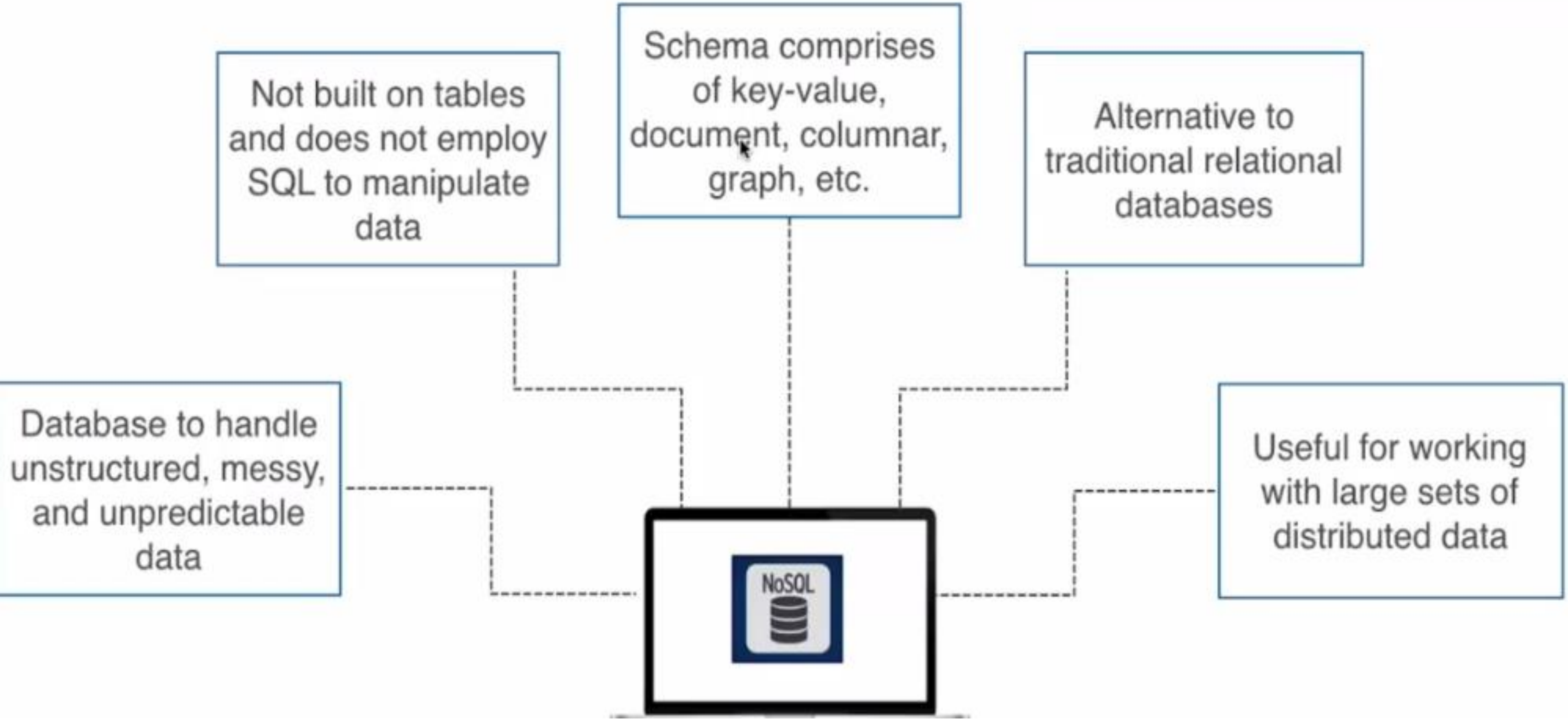
Structured, unstructured, text, etc.

Veracity

Data in doubt

Uncertainty due to latency, deception, ambiguities, etc.

# Advantages of NoSQL - 2



Not built on tables and does not employ SQL to manipulate data

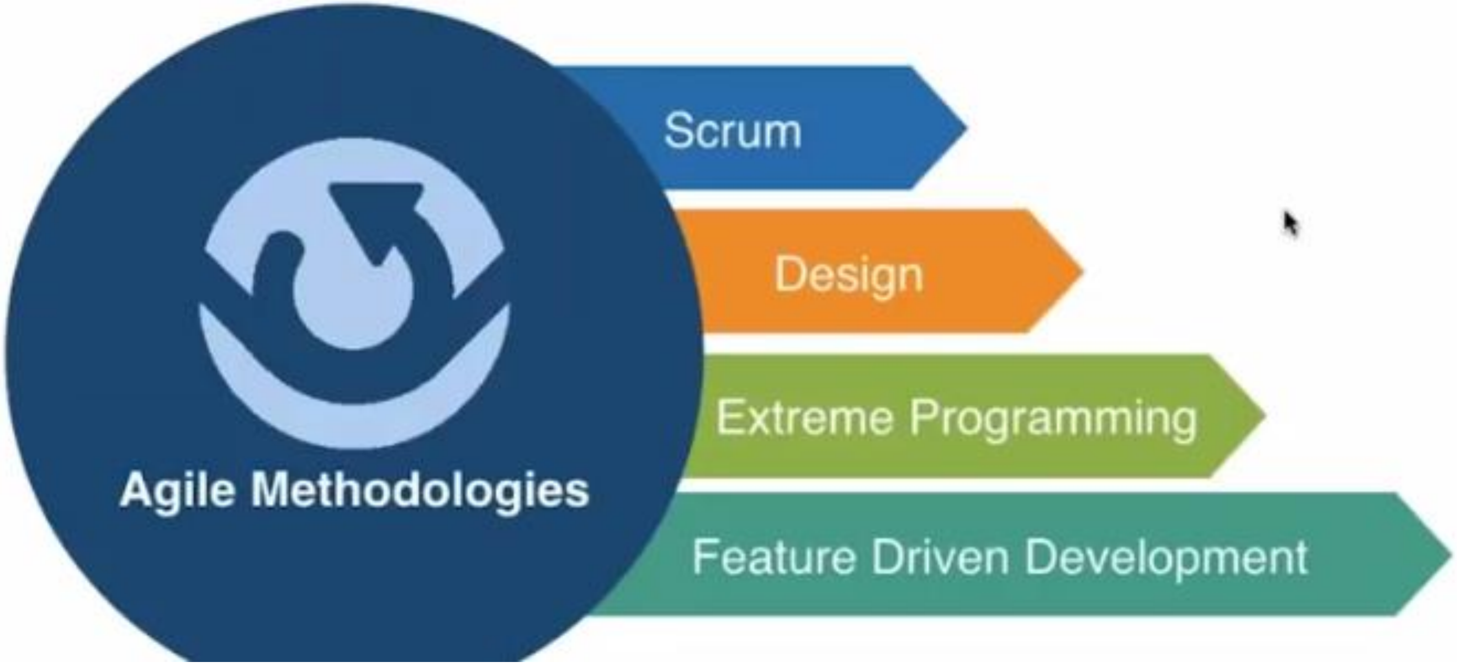
Schema comprises of key-value, document, columnar, graph, etc.

Alternative to traditional relational databases

Database to handle unstructured, messy, and unpredictable data

Useful for working with large sets of distributed data

# Advantages of NoSQL - 3



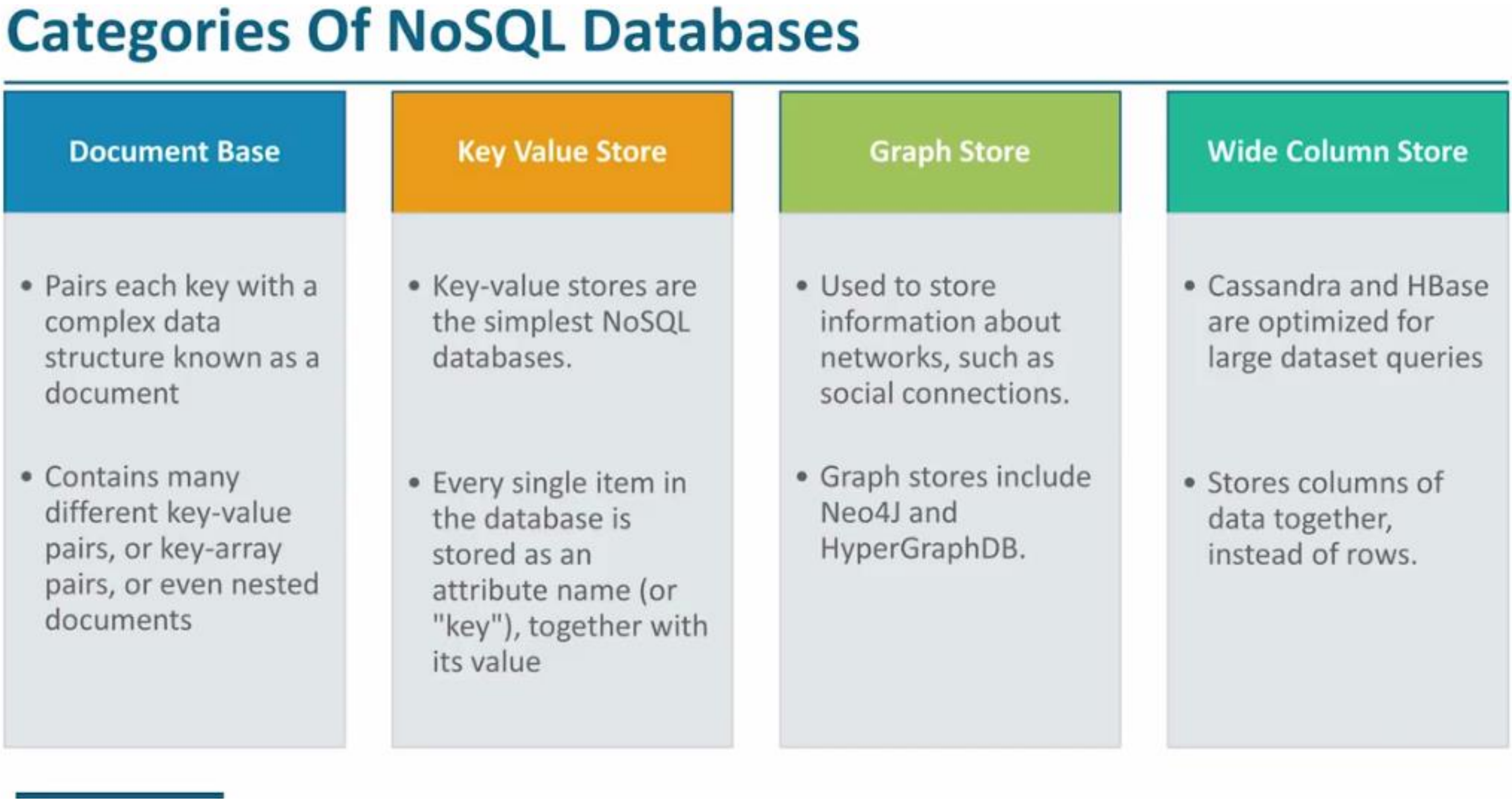
Agile Methodologies

Scrum

Design

Extreme Programming

Feature Driven Development



# Categories Of NoSQL Databases

### Document Base

- Pairs each key with a complex data structure known as a document
- Contains many different key-value pairs, or key-array pairs, or even nested documents

### Key Value Store

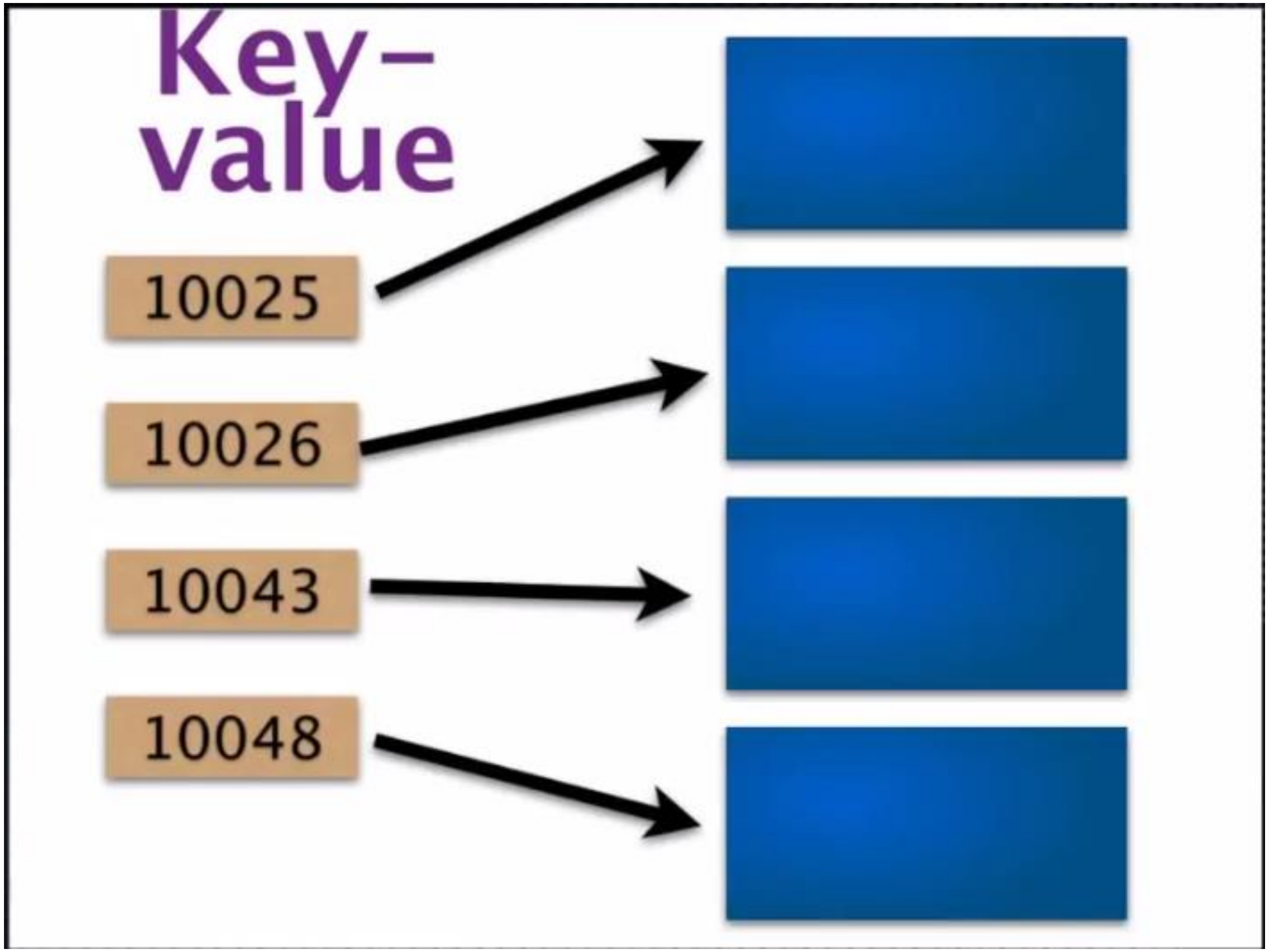
- Key-value stores are the simplest NoSQL databases.
- Every single item in the database is stored as an attribute name (or "key"), together with its value

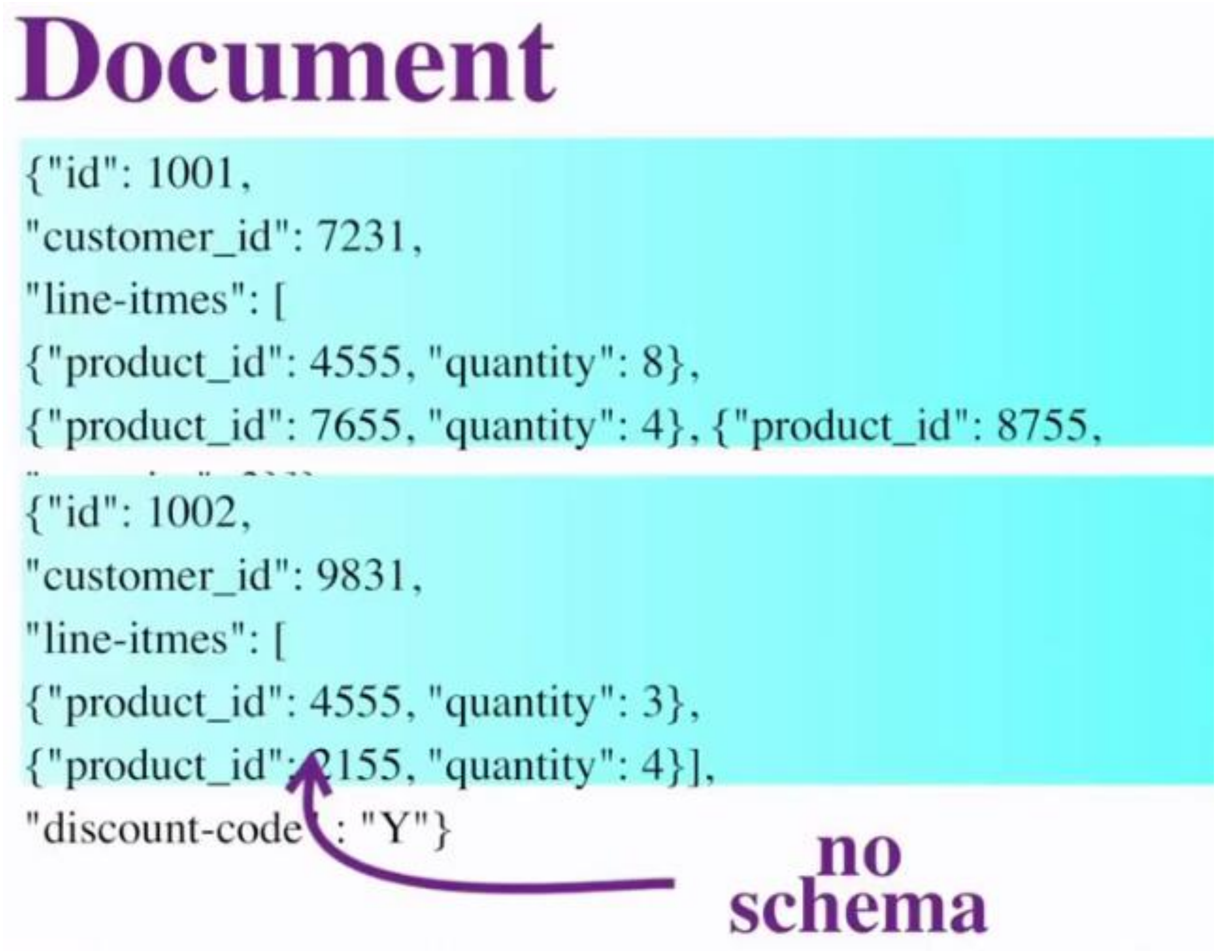
### Graph Store

- Used to store information about networks, such as social connections.
- Graph stores include Neo4J and HyperGraphDB.

### Wide Column Store

- Cassandra and HBase are optimized for large dataset queries
- Stores columns of data together, instead of rows.


Key-value
10025
10026
10043
10048



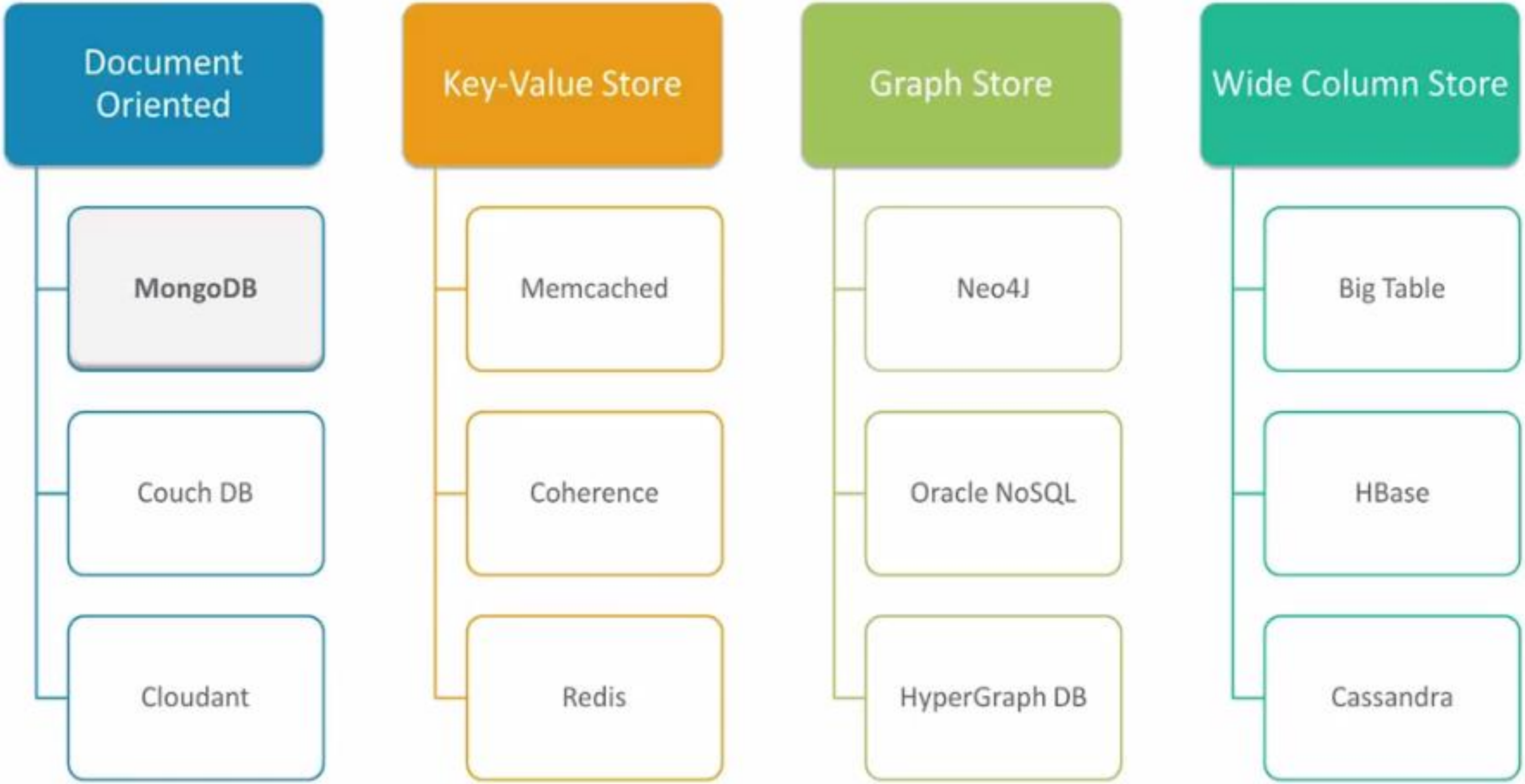
# Document

```
{"id": 1001,
"customer_id": 7231,
"line-itmes": [
{"product_id": 4555, "quantity": 8},
{"product_id": 7655, "quantity": 4}, {"product_id": 8755,
```

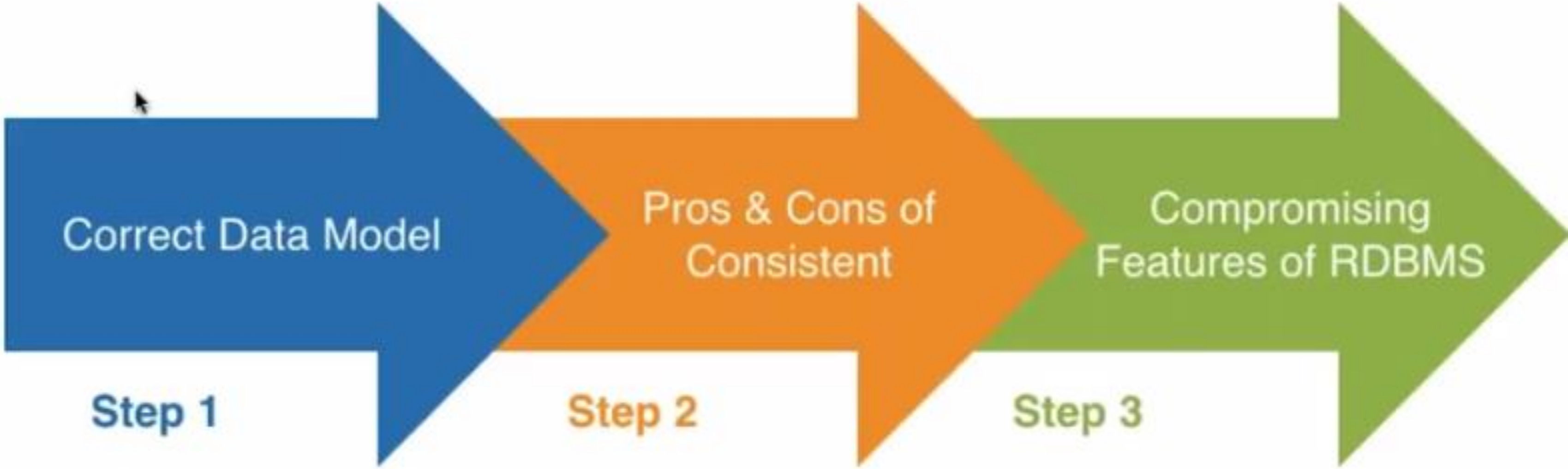
```
{"id": 1002,
"customer_id": 9831,
"line-itmes": [
{"product_id": 4555, "quantity": 3},
{"product_id": 2155, "quantity": 4}],
"discount-code": "Y"}
```

no schema

# Types Of NoSQL Databases



# NoSQL Database – Selection and Implementation



Correct Data Model

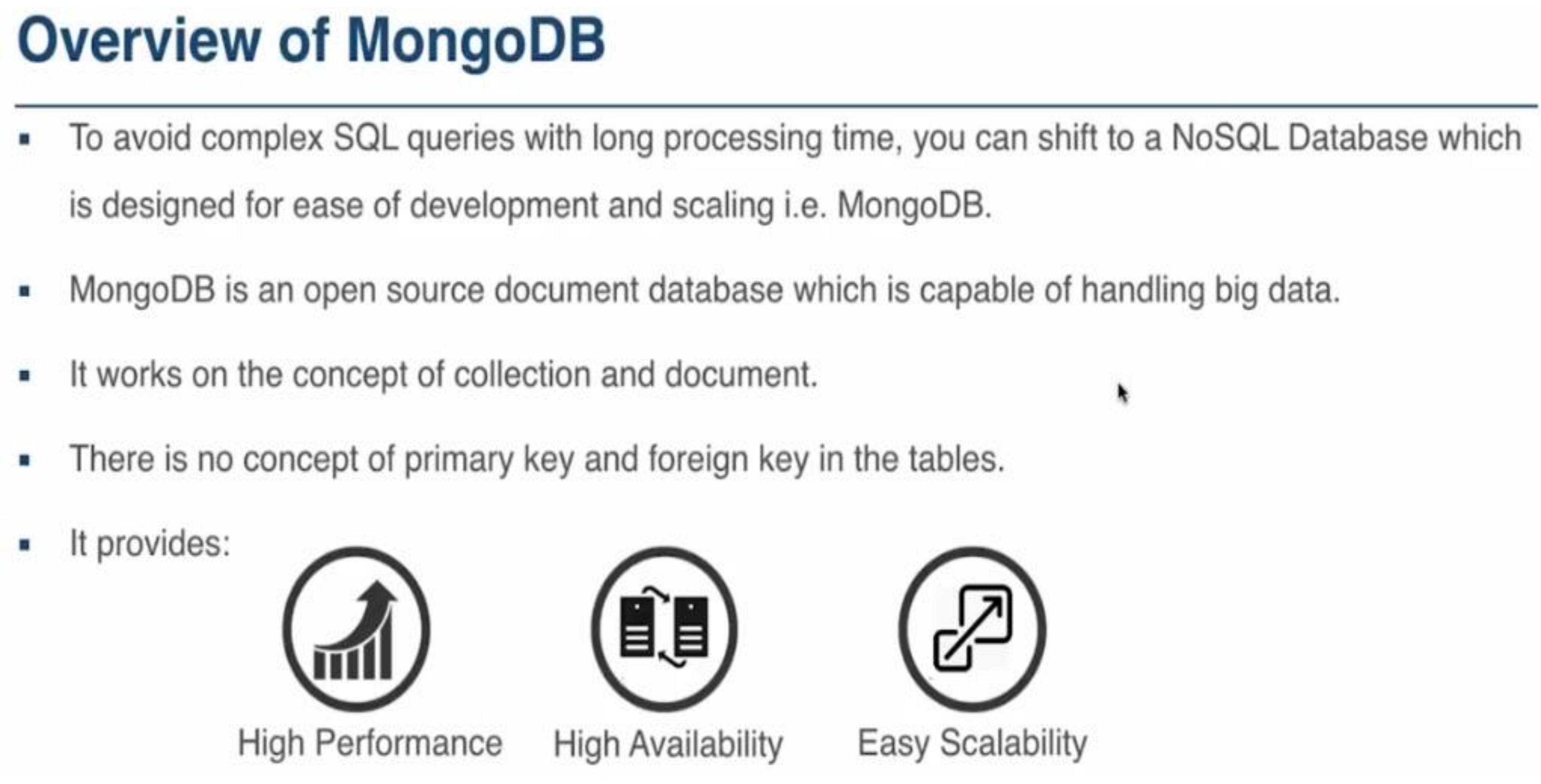
Pros & Cons of Consistent

Compromising Features of RDBMS

Step 1

Step 2

Step 3



# Overview of MongoDB

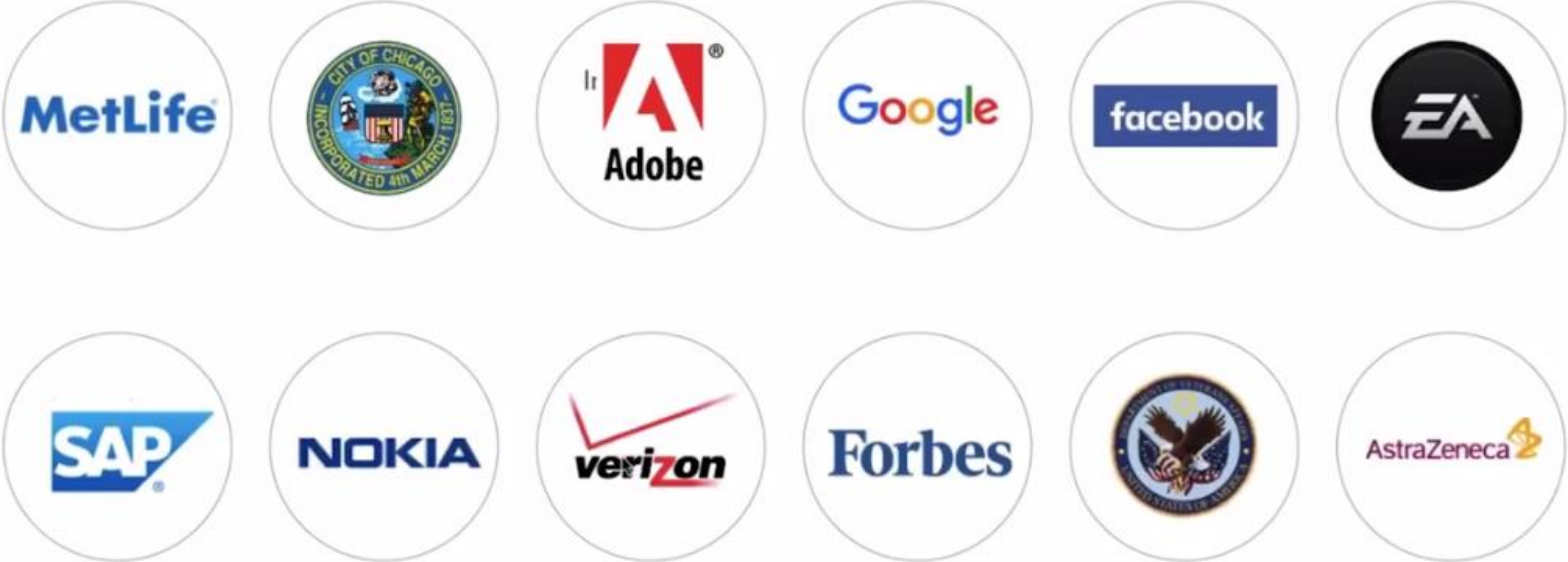
- To avoid complex SQL queries with long processing time, you can shift to a NoSQL Database which is designed for ease of development and scaling i.e. MongoDB.
- MongoDB is an open source document database which is capable of handling big data.
- It works on the concept of collection and document.
- There is no concept of primary key and foreign key in the tables.
- It provides:

High Performance

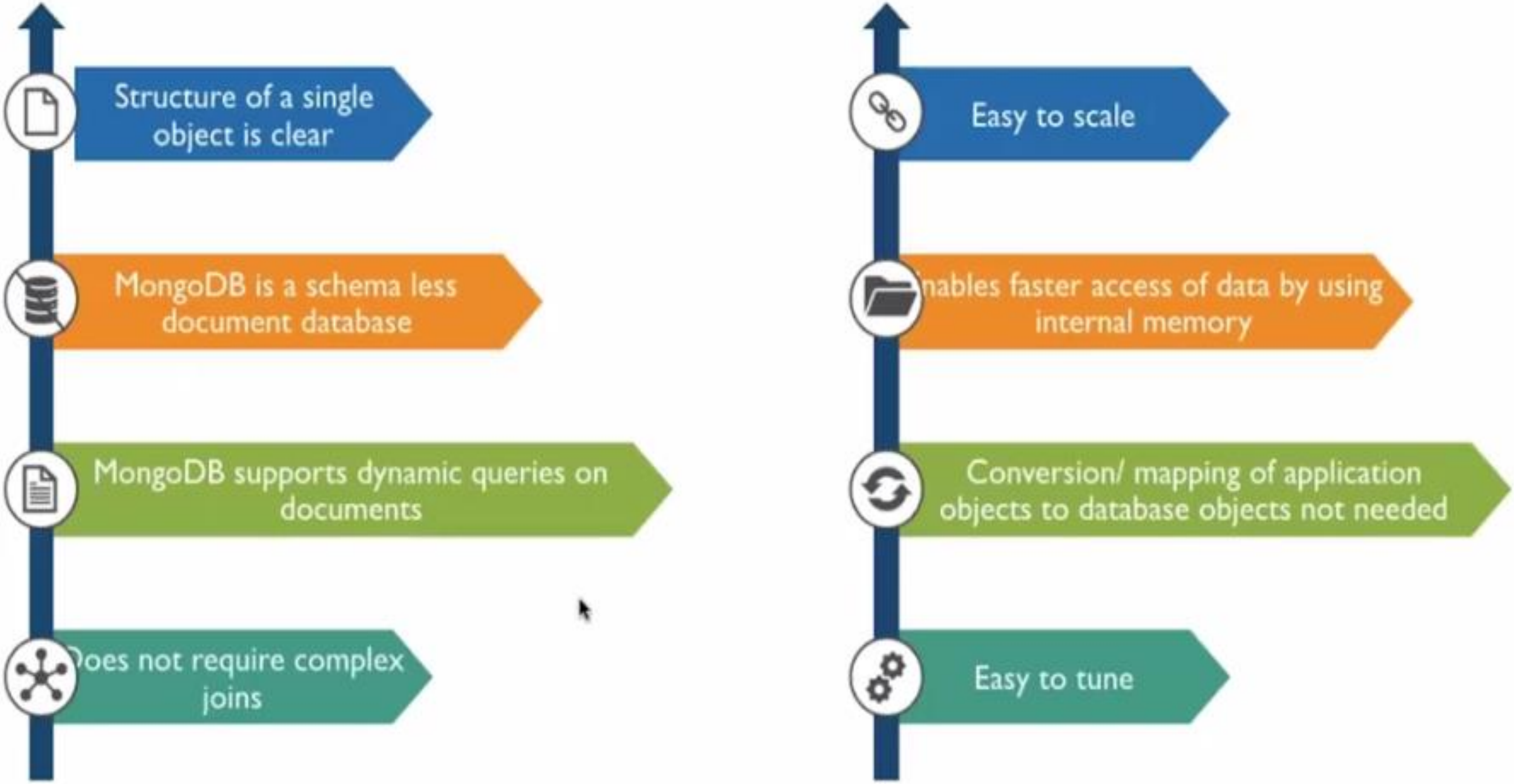
High Availability

Easy Scalability

# Few Customers of MongoDB



# Advantages of using MongoDB



Structure of a single object is clear

MongoDB is a schema less document database

MongoDB supports dynamic queries on documents

Does not require complex joins

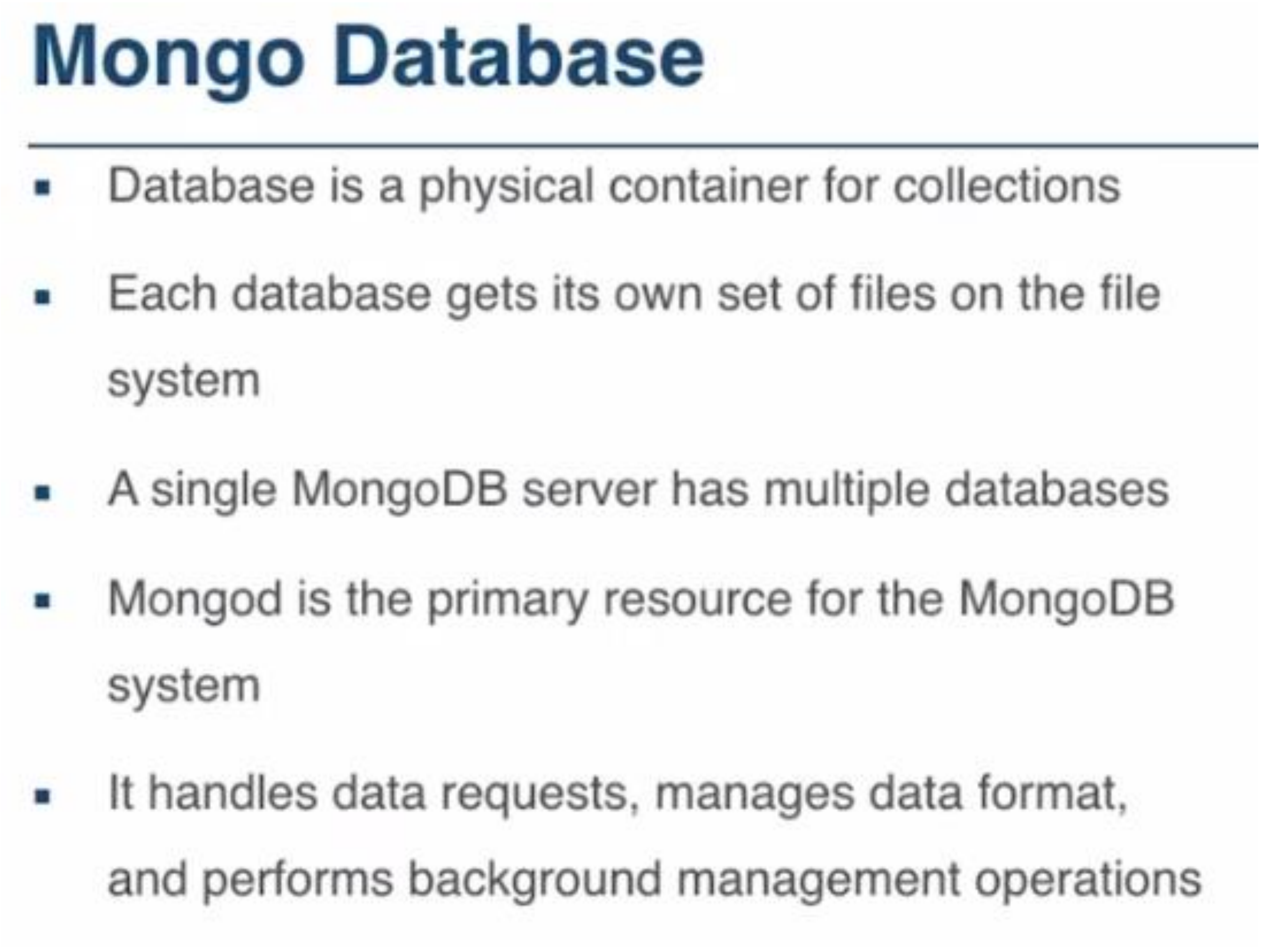
Easy to scale

Enables faster access of data by using internal memory

Conversion/ mapping of application objects to database objects not needed

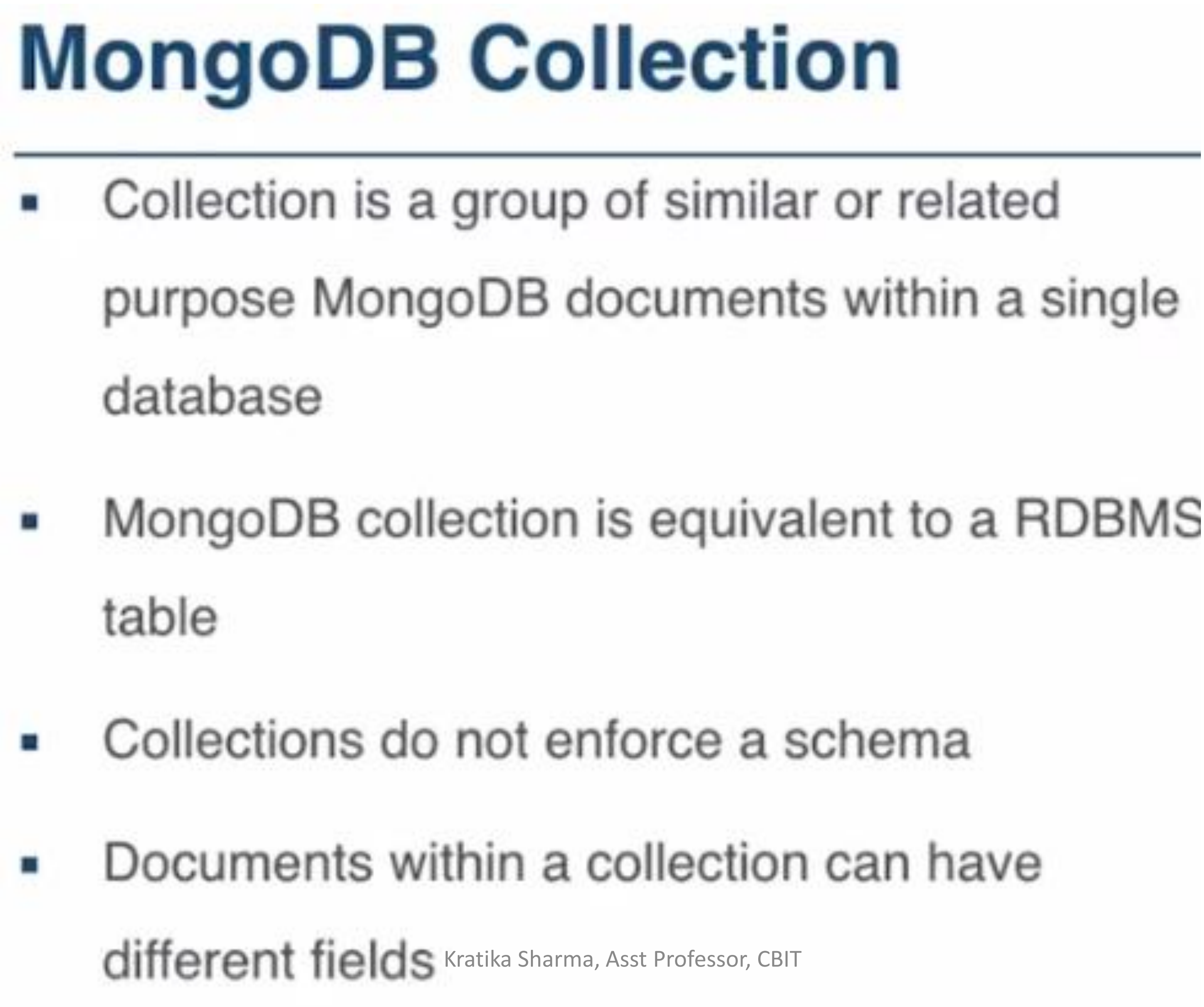
Easy to tune

# MongoDB Terminologies



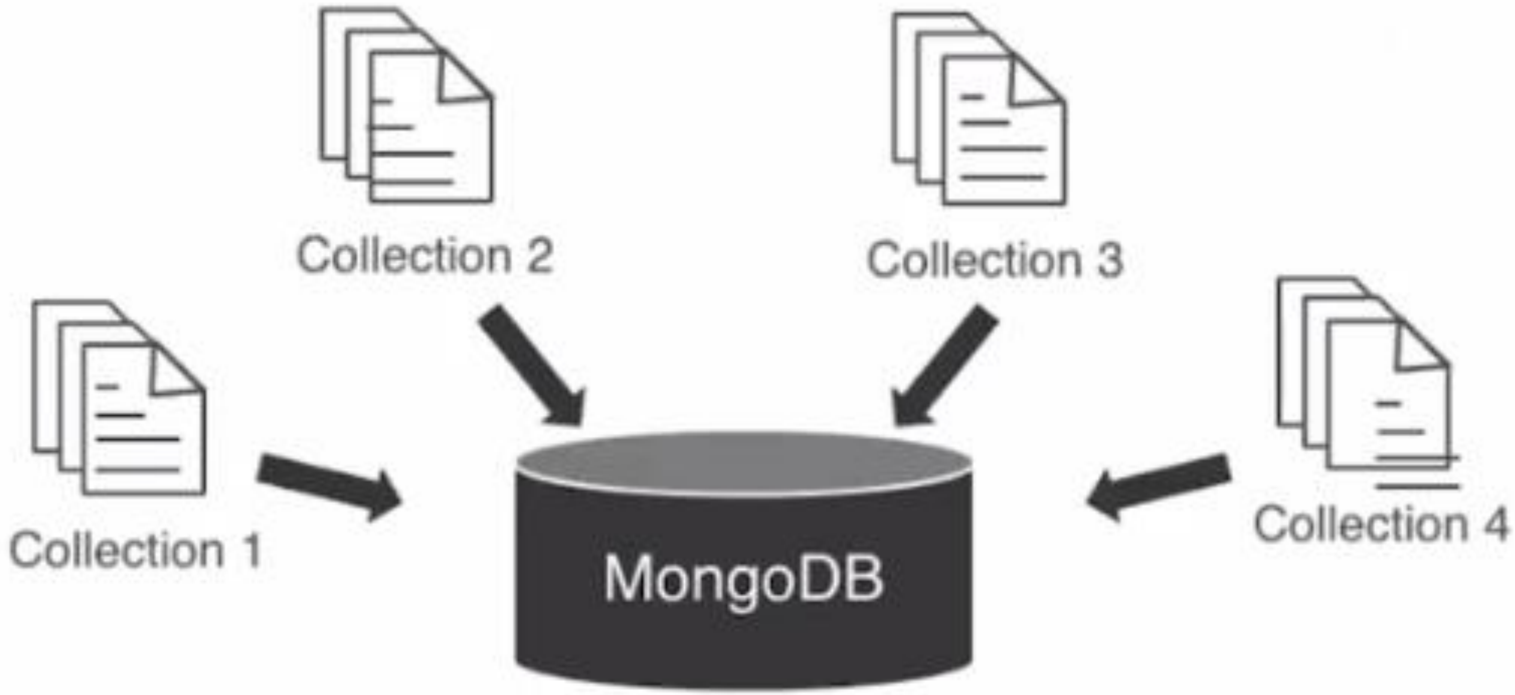
# Mongo Database

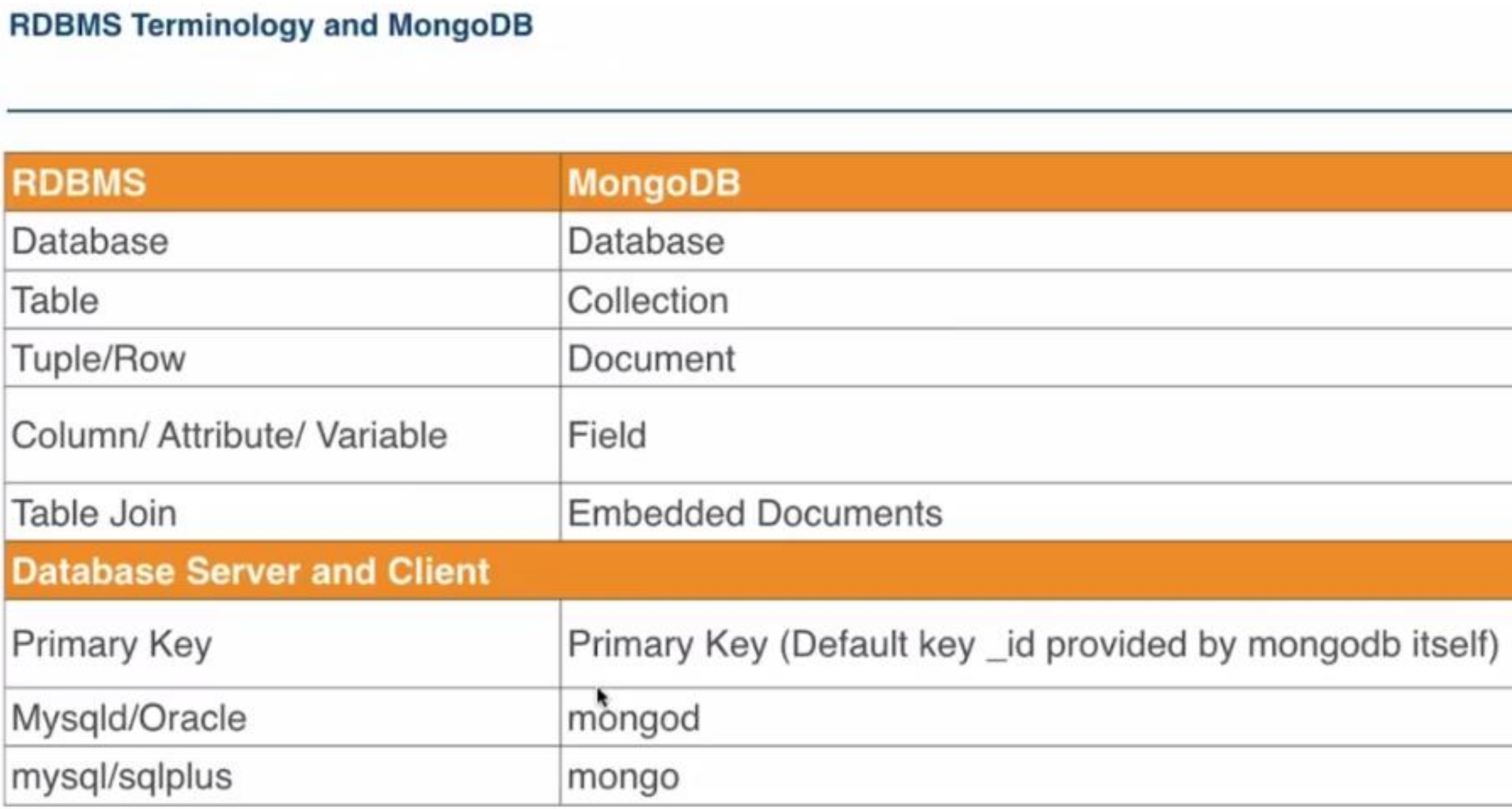
- Database is a physical container for collections
- Each database gets its own set of files on the file system
- A single MongoDB server has multiple databases
- Mongod is the primary resource for the MongoDB system
- It handles data requests, manages data format, and performs background management operations



# MongoDB Collection

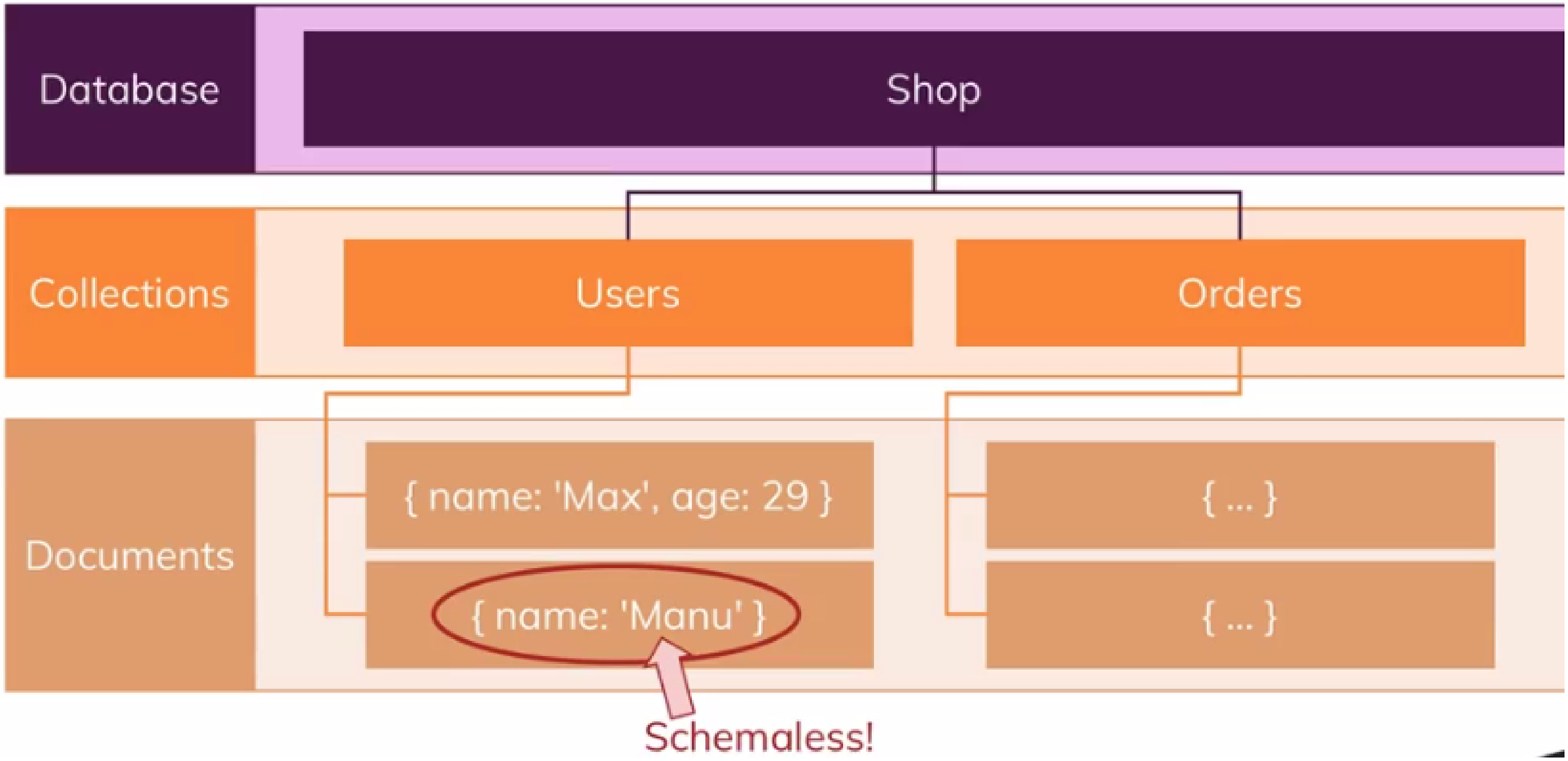
- Collection is a group of similar or related purpose MongoDB documents within a single database
- MongoDB collection is equivalent to a RDBMS table
- Collections do not enforce a schema
- Documents within a collection can have different fields


Collection 2
Collection 3
Collection 1
MongoDB
Collection 4



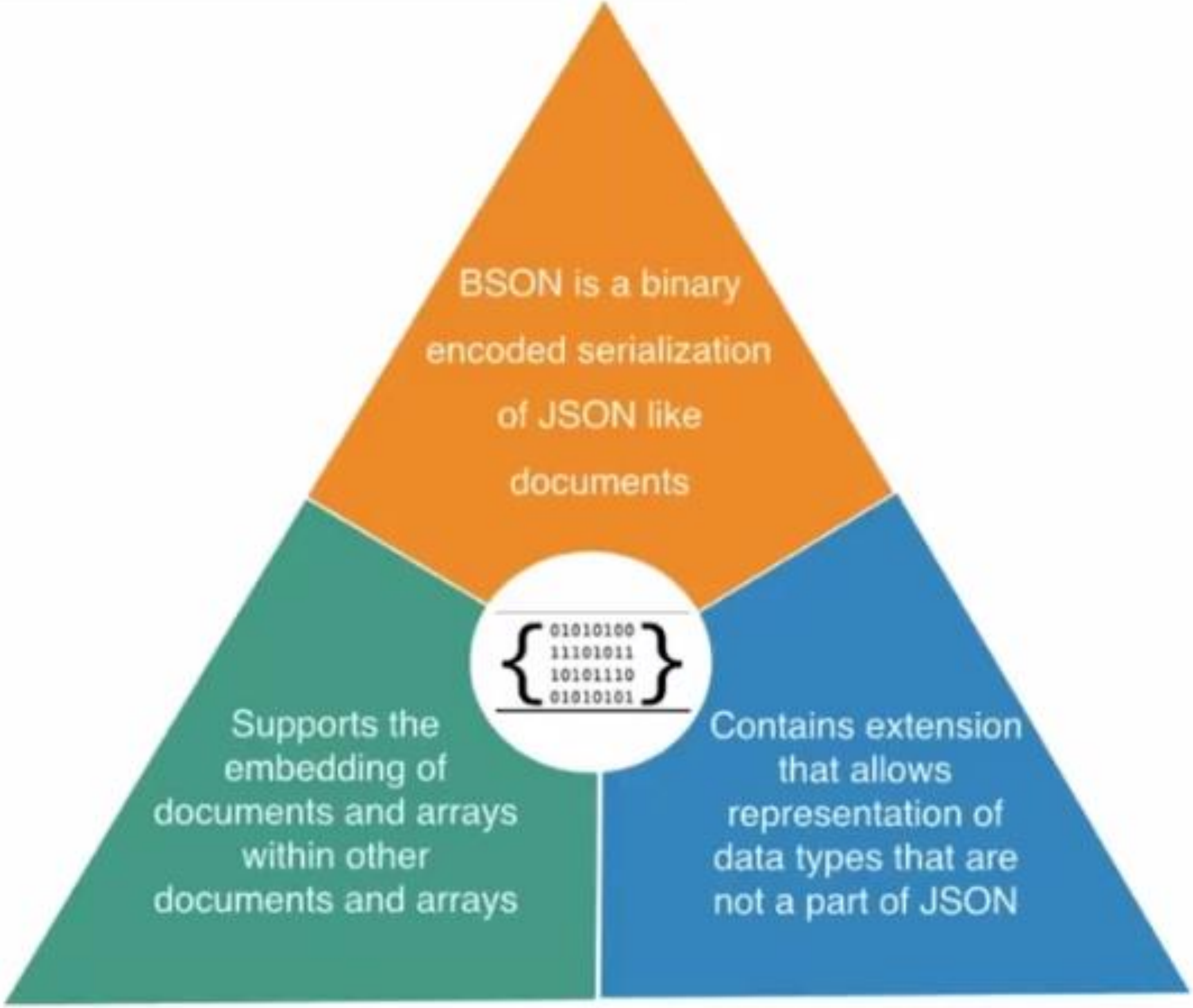
## RDBMS Terminology and MongoDB

| RDBMS | MongoDB |
|---|---|
| Database | Database |
| Table | Collection |
| Tuple/Row | Document |
| Column/ Attribute/ Variable | Field |
| Table Join | Embedded Documents |
| **Database Server and Client** | |
| Primary Key | Primary Key (Default key _id provided by mongodb itself) |
| Mysqld/Oracle | mongod |
| mysql/sqlplus | mongo |


Database
Shop
Collections
Users
Orders
Documents
{ name: 'Max', age: 29 }
{ ... }
{ name: 'Manu' }
{ ... }

Schemaless!

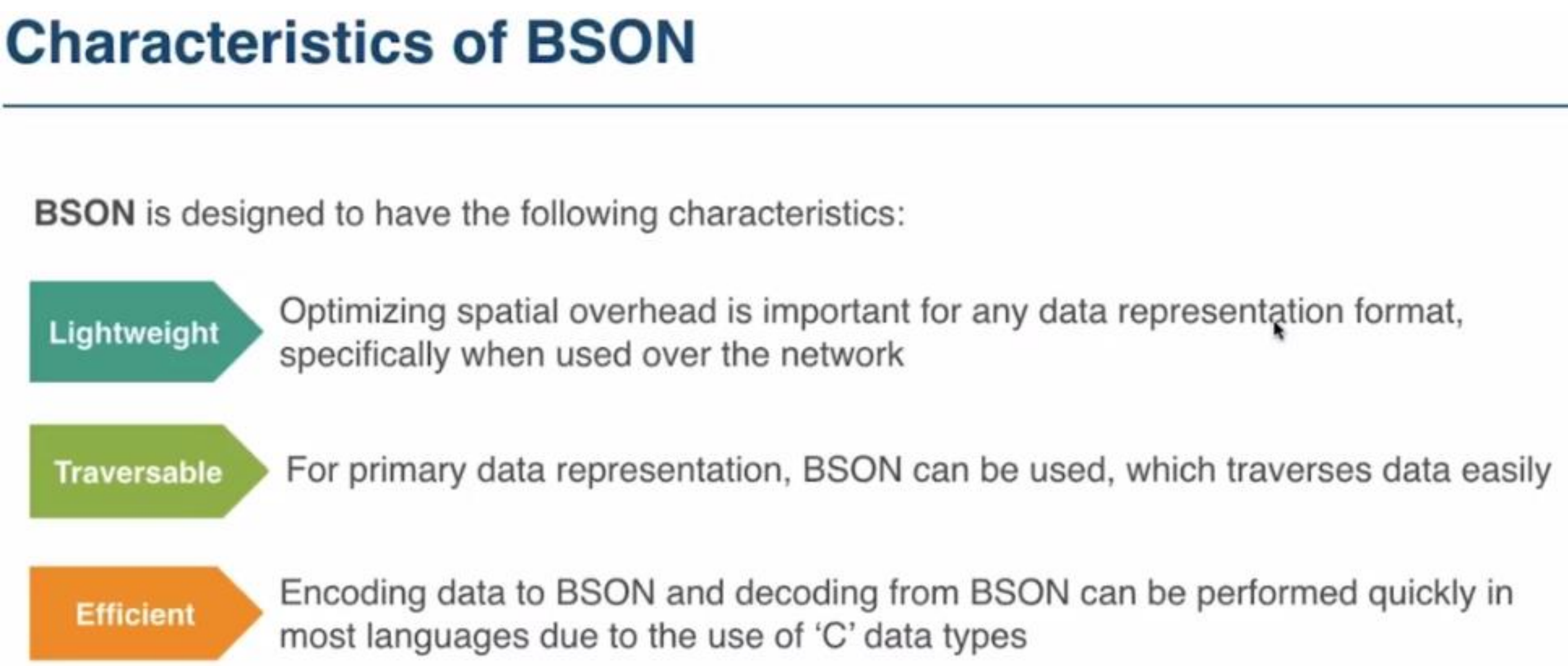
# Binary JSON (BSON)



BSON is a binary encoded serialization of JSON like documents

Supports the embedding of documents and arrays within other documents and arrays

Contains extension that allows representation of data types that are not a part of JSON



# Characteristics of BSON

**BSON** is designed to have the following characteristics:

- **Lightweight**: Optimizing spatial overhead is important for any data representation format, specifically when used over the network
- **Traversable**: For primary data representation, BSON can be used, which traverses data easily
- **Efficient**: Encoding data to BSON and decoding from BSON can be performed quickly in most languages due to the use of 'C' data types

# Sample Document

```
{ _id: ObjectId(7df78ad8902c)
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: ‘mongodb tutorials ',
url: 'http://www.mongodb.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100,
comments: [
                 { user:'user1',
                message: 'My first comment',
                dateCreated: new Date(2011,1,20,2,15),
                like: 0
              }
                  ]
}
```

# Explanation

**_id** is a 12 bytes hexadecimal number which assures the uniqueness of every document. You can provide _id while inserting the document. If you don’t provide then MongoDB provides a unique id for every document. These 12 bytes first 4 bytes for the current timestamp, next 3 bytes for machine id, next 2 bytes for process id of MongoDB server and remaining 3 bytes are simple incremental VALUE.

# Advantages of MongoDB

1. **Schema less** − MongoDB is a document database in which one collection holds different documents. Number of fields, content and size of the document can differ from one document to another.
2. Structure of a single object is clear.
3. No complex joins.
4. Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
5. Tuning.
6. **Ease of scale-out** − MongoDB is easy to scale.
7. Conversion/mapping of application objects to database objects not needed.
8. Uses internal memory for storing the (windowed) working set, enabling faster access of data.

# Disadvantages (Limitations) of MongoDB

1. **Joins not Supported:** MongoDB doesn’t support joins like a relational database. Yet one can use joins functionality by adding by coding it manually. But it may slow execution and affect performance.
2. **High Memory Usage:** MongoDB stores key names for each value pairs. Also, due to no functionality of joins, there is data redundancy. This results in increasing unnecessary usage of memory.
3. **Limited Data Size:** You can have document size, not more than 16MB.
4. **Limited Nesting:** You cannot perform nesting of documents for more than 100 levels.

# Data Modelling

When modeling data in Mongo, keep the following things in mind

- What are the needs of the application – Look at the business needs of the application and see what data and the type of data needed for the application. Based on this, ensure that the structure of the document is decided accordingly.
- What are data retrieval patterns – If you foresee a heavy query usage then consider the use of indexes in your data model to improve the efficiency of queries.
- Are frequent insert's, updates and removals happening in the database – Reconsider the use of indexes or incorporate sharding if required in your data modeling design to improve the efficiency of your overall MongoDB environment.
- Do joins while write, not on read.
- Optimize your schema for most frequent use cases.
- Do complex aggregation in the schema.

# MongoDB - Create Database

- MongoDB **use DATABASE_NAME** is used to create database. The command will create a new database if it doesn't exist, otherwise it will return the existing database.

**Syntax**

- Basic syntax of **use DATABASE** statement is as follows –

```
use DATABASE_NAME
```

**Example**

- If you want to use a database with name **<mydb>**, then **use DATABASE**statement would be as follows –
- >use mydb

```
switched to db mydb
```

- To check your currently selected database, use the command **db**

```
>db mydb
```

- If you want to check your databases list, use the command **show dbs**.

```
>show dbs local 0.78125GB test 0.23012GB
```

# MongoDB - Drop Database

- MongoDB **db.dropDatabase()** command is used to drop a existing database.

**Syntax**

- db.dropDatabase()
- This will delete the selected database. If you have not selected any database, then it will delete default 'test' database.

**Example**

- First, check the list of available databases by using the command, **show dbs**.
- >show dbs

```
local 0.78125GB
mydb 0.23012GB
 test 0.23012GB >
```

If you want to delete new database **<mydb>**, then **dropDatabase()**command would be as follows −

- >use mydb

```
switched to db mydb
>db.dropDatabase()
 >{ "dropped" : "mydb", "ok" : 1 } >
```

Now check list of databases.

```
>show dbs local 0.78125GB test 0.23012GB >
```

# MongoDB - Create Collection

MongoDB **db.createCollection(name, options)** is used to create collection.

**Syntax**

- Basic syntax of **createCollection()** command is as follows −

```
>db.createCollection(name, options)
```

- In the command, **name** is name of collection to be created. **Options** is a document and is used to specify configuration of collection.

**Examples**

- Basic syntax of **createCollection()** method without options is as follows −

```
>use test
switched to db test
>db.createCollection("mycollection")
{ "ok" : 1 } >
```

- In MongoDB, you don't need to create collection. MongoDB creates collection automatically, when you insert some document.

# MongoDB - Drop Collection

- MongoDB's **db.collection.drop()** is used to drop a collection from the database.

**Syntax**

- Basic syntax of **drop()** command is as follows –

```
>db.COLLECTION_NAME.drop()
```

# MongoDB - Datatypes

- **String** − This is the most commonly used datatype to store the data. String in MongoDB must be UTF-8 valid.
- **Integer** − This type is used to store a numerical value. Integer can be 32 bit or 64 bit depending upon your server.
- **Boolean** − This type is used to store a boolean (true/ false) value.
- **Double** − This type is used to store floating point values.
- **Min/ Max keys** − This type is used to compare a value against the lowest and highest BSON elements.
- **Arrays** − This type is used to store arrays or list or multiple values into one key.
- **Timestamp** − ctimestamp. This can be handy for recording when a document has been modified or added.
- **Object** − This datatype is used for embedded documents.
- **Null** − This type is used to store a Null value.
- **Symbol** − This datatype is used identically to a string; however, it's generally reserved for languages that use a specific symbol type.
- **Date** − This datatype is used to store the current date or time in UNIX time format. You can specify your own date time by creating object of Date and passing day, month, year into it.
- **Object ID** − This datatype is used to store the document’s ID.
- **Binary data** − This datatype is used to store binary data.
- **Code** − This datatype is used to store JavaScript code into the document.
- **Regular expression** − This datatype is used to store regular expression.

# MongoDB - Insert Document

- To insert data into MongoDB collection, you need to use MongoDB's **insert()**or **save()** method.

**Syntax**

- The basic syntax of **insert()** command is as follows

```
>db.COLLECTION_NAME.insert(document)
```

**Example**

```
>db.mycol.insert(
{ _id: ObjectId(7df78ad8902c),
title: 'MongoDB Overview',
description: 'MongoDB is no sql database',
by: ‘MongoDB',
url: 'http://www.mongodbpoint.com',
tags: ['mongodb', 'database', 'NoSQL'],
likes: 100 }
)
```

# MongoDB - Query Document

## 1. The find() Method

- To query data from MongoDB collection, you need to use MongoDB's **find()**method.

**Syntax**

```
>db.COLLECTION_NAME.find()
```

- **find()** method will display all the documents in a non-structured way.

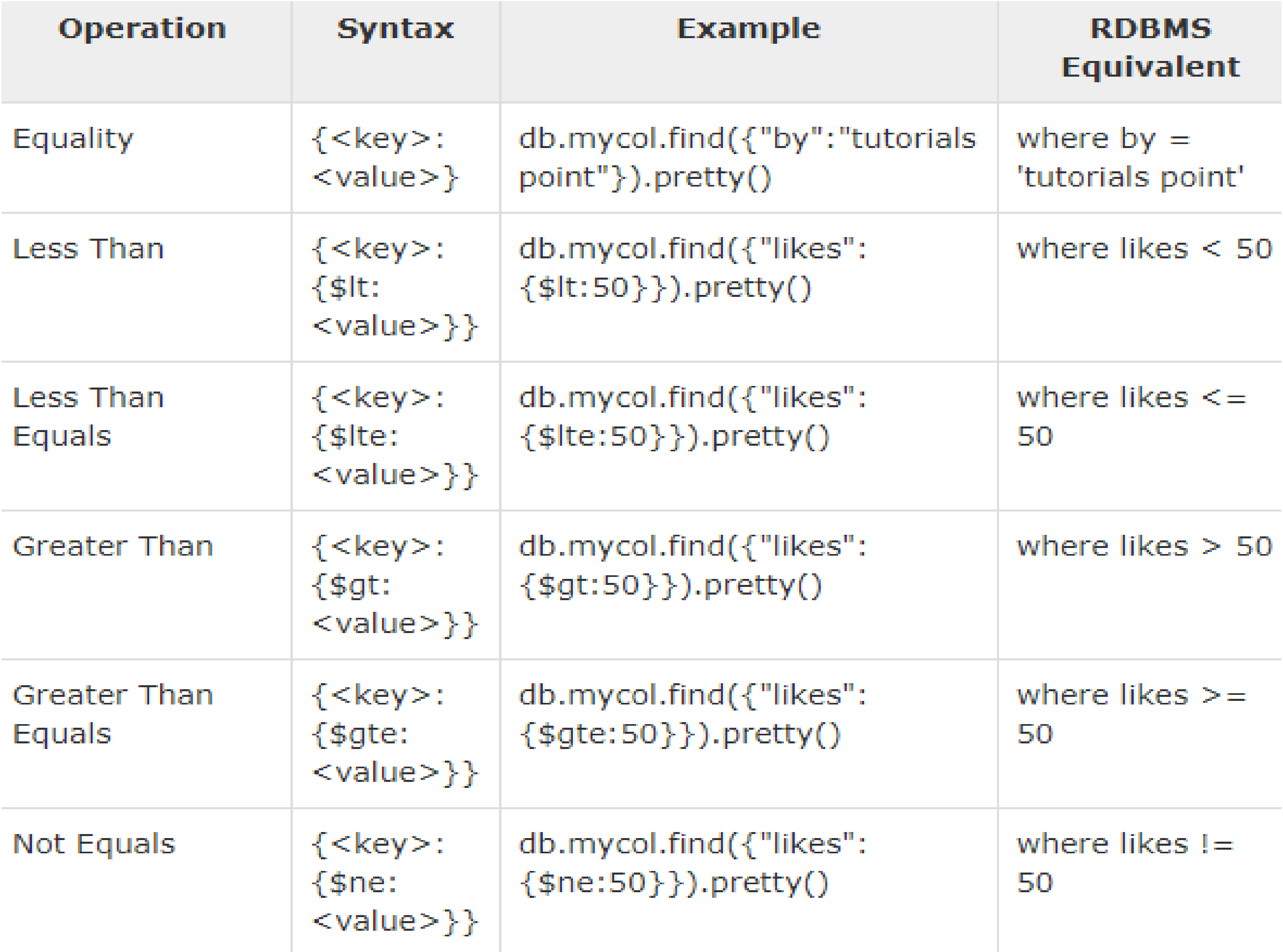
## 2. The pretty() Method

To display the results in a formatted way, you can use **pretty()** method.

**Syntax**

```
>db.mycol.find().pretty()
```

# RDBMS Where Clause Equivalents in MongoDB



| Operation | Syntax | Example | RDBMS Equivalent |
|---|---|---|---|
| Equality | {<key>:<value>} | db.mycol.find({"by":"tutorials point"}).pretty() | where by = 'tutorials point' |
| Less Than | {<key>:{$lt:<value>}} | db.mycol.find({"likes":{$lt:50}}).pretty() | where likes < 50 |
| Less Than Equals | {<key>:{$lte:<value>}} | db.mycol.find({"likes":{$lte:50}}).pretty() | where likes <= 50 |
| Greater Than | {<key>:{$gt:<value>}} | db.mycol.find({"likes":{$gt:50}}).pretty() | where likes > 50 |
| Greater Than Equals | {<key>:{$gte:<value>}} | db.mycol.find({"likes":{$gte:50}}).pretty() | where likes >= 50 |
| Not Equals | {<key>:{$ne:<value>}} | db.mycol.find({"likes":{$ne:50}}).pretty() | where likes != 50 |

# AND in MongoDB

## Syntax

- In the **find()** method, if you pass multiple keys by separating them by ',' then MongoDB treats it as **AND** condition

>**db.mycol.find( { $and: [ {key1: value1}, {key2:value2} ] } ).pretty()**

## Example

- Following example will show all the tutorials written by ‘mongoDB' and whose title is 'MongoDB Overview'.
- >db.mycol.find({$and:[{"by":“mongoDB"},{"title": "MongoDB Overview"}]}).pretty()
- { "_id": ObjectId(7df78ad8902c), "title": "MongoDB Overview", "description": "MongoDB is no sql database", "by": “mongoDB", "url": "http://www.mongoDB point.com", "tags": ["mongodb", "database", "NoSQL"], "likes": "100" }

# OR in MongoDB

Syntax

- To query documents based on the OR condition, you need to use **$or** keyword. Following is the basic syntax of **OR** −
- >db.mycol.find( { $or: [ {key1: value1}, {key2:value2} ] } ).pretty()

# Using AND and OR Together

## Example

- The following example will show the documents that have likes greater than 10 and whose title is either 'MongoDB Overview' or by is 'tutorials point'. Equivalent SQL where clause is **'where likes>10 AND (by = ‘mongoDB' OR title = 'MongoDB Overview')'**

```
>db.mycol.find({"likes": {$gt:10}, $or: [{"by": “mongoDB"}, {"title": "MongoDB Overview"}]}).pretty()
```

# MongoDB - Update Document

- MongoDB's **update()** and **save()** methods are used to update document into a collection.
- The update() method updates the values in the existing document while the save() method replaces the existing document with the document passed in save() method.

# MongoDB Update() Method

The update() method updates the values in the existing document.

**Syntax**

- The basic syntax of **update()** method is as follows –
- >db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

# MongoDB - Delete Document

The remove() Method

- MongoDB's **remove()** method is used to remove a document from the collection. remove() method accepts two parameters. One is deletion criteria and second is justOne flag.

1. **deletion criteria** – (Optional) deletion criteria according to documents will be removed.
2. **justOne** – (Optional) if set to true or 1, then remove only one document.

Syntax

>db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

# MongoDB - Indexing

- Indexes support the efficient resolution of queries. Without indexes, MongoDB must scan every document of a collection to select those documents that match the query statement. This scan is highly inefficient and require MongoDB to process a large volume of data.
- **Indexes are special data structures, that store a small portion of the data set in an easy-to-traverse form.**
- The index stores the value of a specific field or set of fields, ordered by the value of the field as specified in the index.

## The ensureIndex() Method

- To create an index you need to use ensureIndex() method of MongoDB.

## Syntax

- The basic syntax of **ensureIndex()** method is as follows().

```
>db.COLLECTION_NAME.ensureIndex({KEY:1})
```

Here key is the name of the field on which you want to create index and 1 is for ascending order. To create index in descending order you need to use -1.

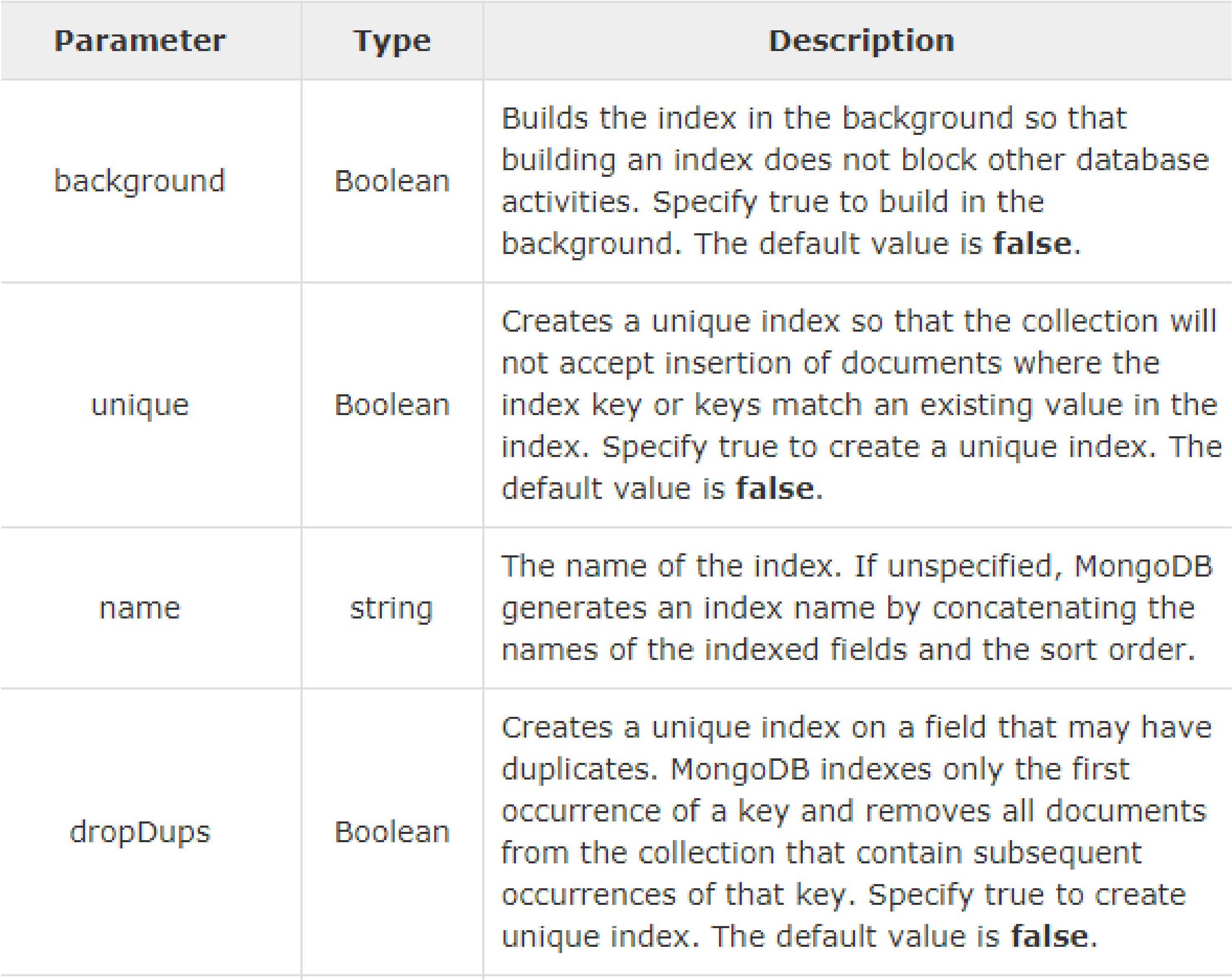
## Example

```
>db.mycol.ensureIndex({"title":1}) >
```

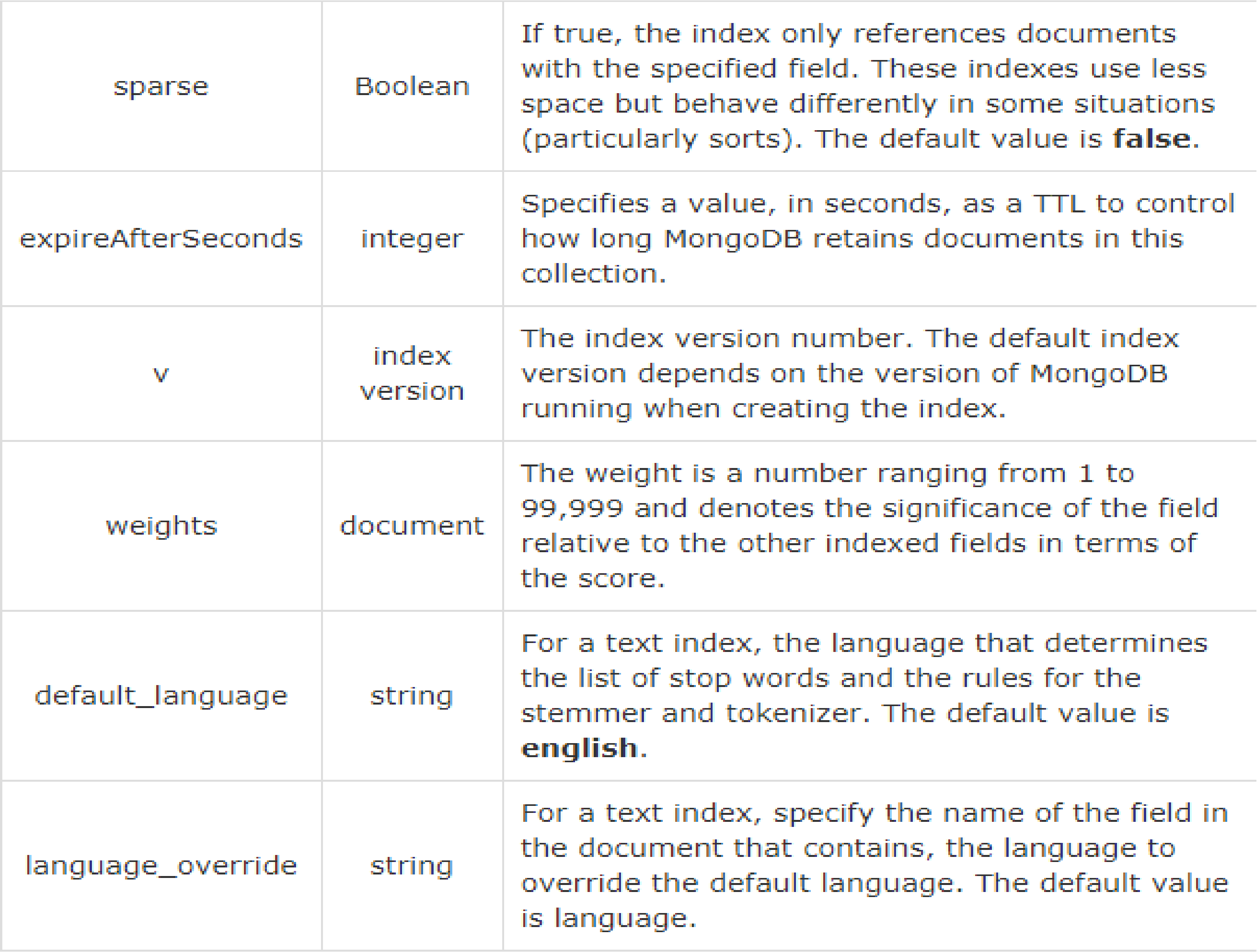
- In **ensureIndex()** method you can pass multiple fields, to create index on multiple fields.

```
>db.mycol.ensureIndex({"title":1,"description":-1}) >
```

**ensureIndex()** method also accepts list of options (which are optional). Following is the list



| Parameter | Type | Description |
|---|---|---|
| background | Boolean | Builds the index in the background so that building an index does not block other database activities. Specify true to build in the background. The default value is **false**. |
| unique | Boolean | Creates a unique index so that the collection will not accept insertion of documents where the index key or keys match an existing value in the index. Specify true to create a unique index. The default value is **false**. |
| name | string | The name of the index. If unspecified, MongoDB generates an index name by concatenating the names of the indexed fields and the sort order. |
| dropDups | Boolean | Creates a unique index on a field that may have duplicates. MongoDB indexes only the first occurrence of a key and removes all documents from the collection that contain subsequent occurrences of that key. Specify true to create unique index. The default value is **false**. |



| | | |
|---|---|---|
| sparse | Boolean | If true, the index only references documents with the specified field. These indexes use less space but behave differently in some situations (particularly sorts). The default value is **false**. |
| expireAfterSeconds | integer | Specifies a value, in seconds, as a TTL to control how long MongoDB retains documents in this collection. |
| v | index version | The index version number. The default index version depends on the version of MongoDB running when creating the index. |
| weights | document | The weight is a number ranging from 1 to 99,999 and denotes the significance of the field relative to the other indexed fields in terms of the score. |
| default_language | string | For a text index, the language that determines the list of stop words and the rules for the stemmer and tokenizer. The default value is **english**. |
| language_override | string | For a text index, specify the name of the field in the document that contains, the language to override the default language. The default value is language. |

# Types of Index

- **Default _id:** Each MongoDB collection contains an index on the default _id field. If no value is specified for _id, the language driver or the mongod creates a _id field and provides an ObjectId value.

- **Single Field:** For a single-field index and sort operation, the sort order of the index keys do not matter. MongoDB can traverse the indexes either in the ascending or descending order.

- **Compound Index:** For multiple fields, MongoDB supports user-defined indexes, such as compound indexes. The sequential order of fields in a compound index is significant in MongoDB.

- **Multikey Index:** To index array data, MongoDB uses multikey indexes. When indexing a field with an array value, MongoDB makes separate index entries for each array element.

- **Text Indexes**: These indexes in MongoDB searches data string in a collection.

- **Hashed Indexes:** MongoDB supports hash-based sharding and provides hashed indexes. These indexes the hashes of the field value.

# MongoDB Aggregation

- Aggregation operations process data records and return computed results.
- Aggregation operations group values from multiple documents together, and can perform a variety of operations on the grouped data to return a single result.
- In SQL count(*) and with group by is an equivalent of mongodb aggregation.

- Syntax

```
>db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION)
```

{ _id: ObjectId(7df78ad8902c) title: 'MongoDB Overview', description: 'MongoDB is no sql database', by_user: 'tutorials point', url: 'http://www.tutorialspoint.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 100 },

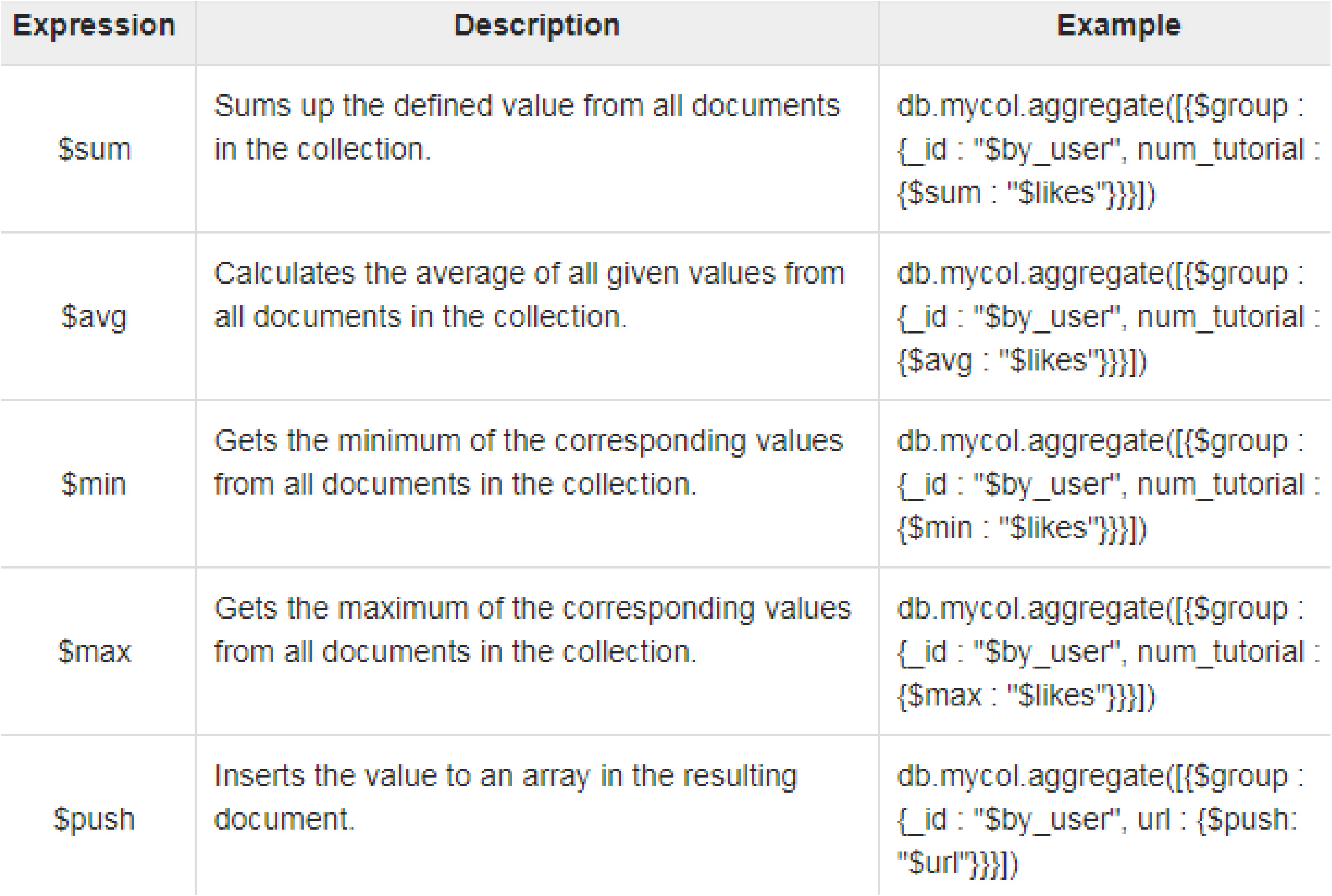
{ _id: ObjectId(7df78ad8902d) title: 'NoSQL Overview', description: 'No sql database is very fast', by_user: 'tutorials point', url: 'http://www.tutorialspoint.com', tags: ['mongodb', 'database', 'NoSQL'], likes: 10 },

{ _id: ObjectId(7df78ad8902e) title: 'Neo4j Overview', description: 'Neo4j is no sql database', by_user: 'Neo4j', url: 'http://www.neo4j.com', tags: ['neo4j', 'database', 'NoSQL'], likes: 750 }, if you want to display a list stating how many tutorials are written by each user??

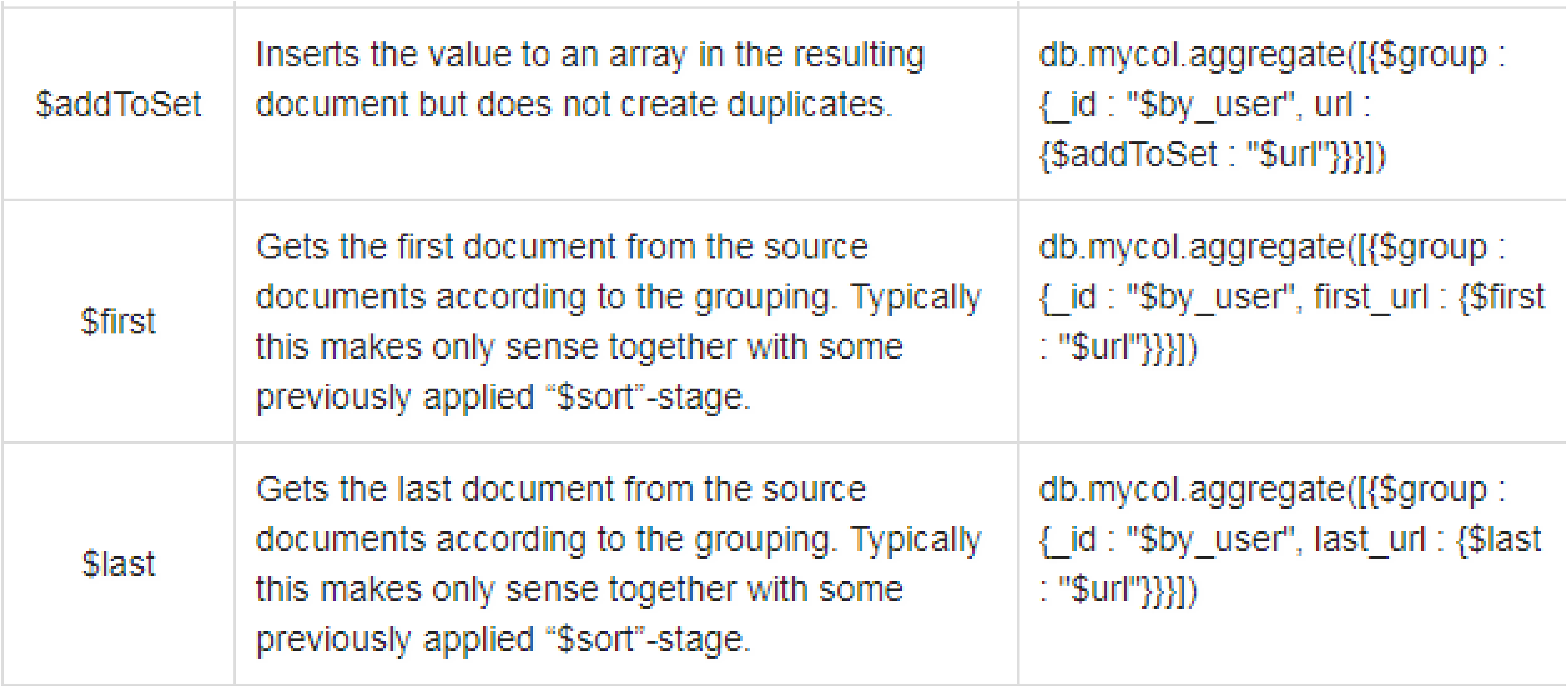
**select by_user, count(*) from mycol group by by_user**.

```
> db.mycol.aggregate([{$group : {_id : "
$by_user", num_tutorial : {$sum : 1}}}])
```

```
{ "result" : [
 { "_id" : "tutorials point", "num_tutorial" : 2 },
{ "_id" : "Neo4j", "num_tutorial" : 1 } ],
"ok" : 1 } >
```



| Expression | Description | Example |
|---|---|---|
| $sum | Sums up the defined value from all documents in the collection. | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$sum : "$likes"}}}]) |
| $avg | Calculates the average of all given values from all documents in the collection. | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$avg : "$likes"}}}]) |
| $min | Gets the minimum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$min : "$likes"}}}]) |
| $max | Gets the maximum of the corresponding values from all documents in the collection. | db.mycol.aggregate([{$group : {_id : "$by_user", num_tutorial : {$max : "$likes"}}}]) |
| $push | Inserts the value to an array in the resulting document. | db.mycol.aggregate([{$group : {_id : "$by_user", url : {$push: "$url"}}}]) |



| | | |
|---|---|---|
| $addToSet | Inserts the value to an array in the resulting document but does not create duplicates. | db.mycol.aggregate([{$group : {_id : "$by_user", url : {$addToSet : "$url"}}}]) |
| $first | Gets the first document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {_id : "$by_user", first_url : {$first : "$url"}}}]) |
| $last | Gets the last document from the source documents according to the grouping. Typically this makes only sense together with some previously applied “$sort”-stage. | db.mycol.aggregate([{$group : {_id : "$by_user", last_url : {$last : "$url"}}}]) |

# MongoDB order with Sort() & Limit()

- What is query modifications?

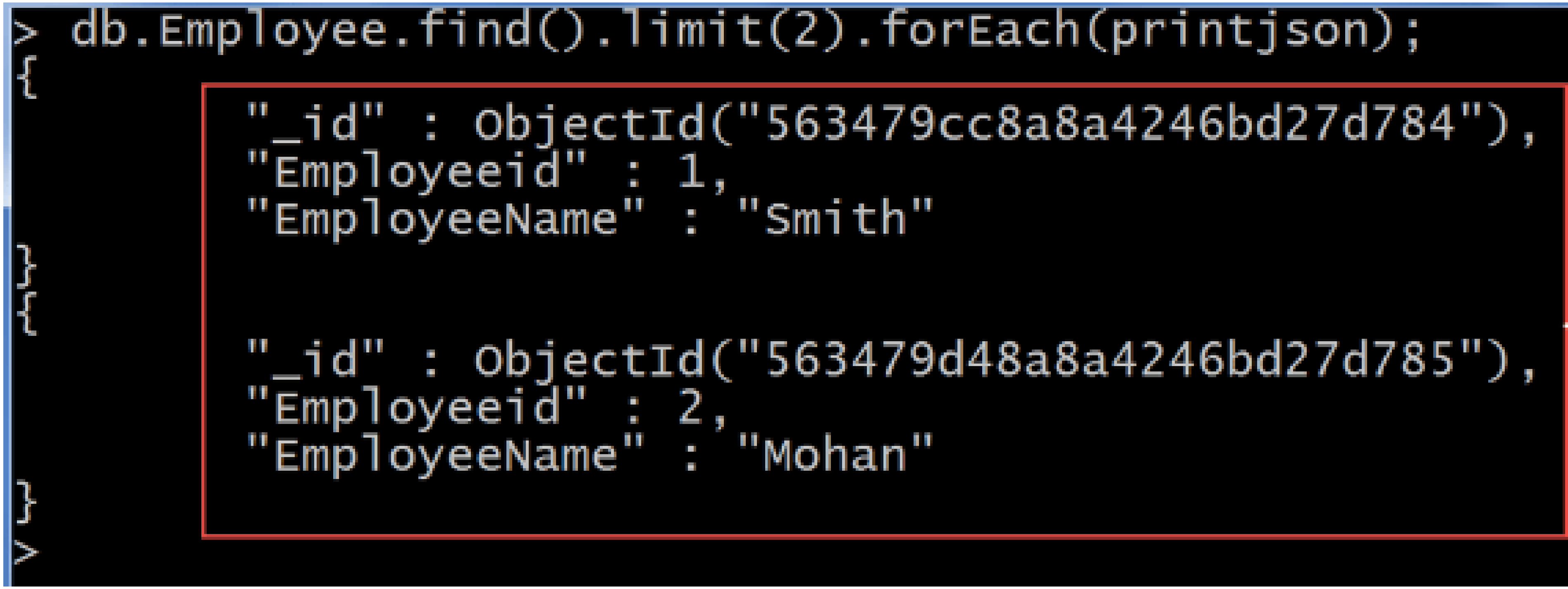
  Mongo DB provides query modifiers such as the 'limit' and 'Orders' clause to provide more flexibility when executing queries.

# MongoDB Limit

This modifier is used to limit the number of documents which are returned in the result set for a query.

```
>db.COLLECTION_NAME.find().limit(NUMBER)
```

```
db.Employee.find().limit(2).forEach(printjson);
```



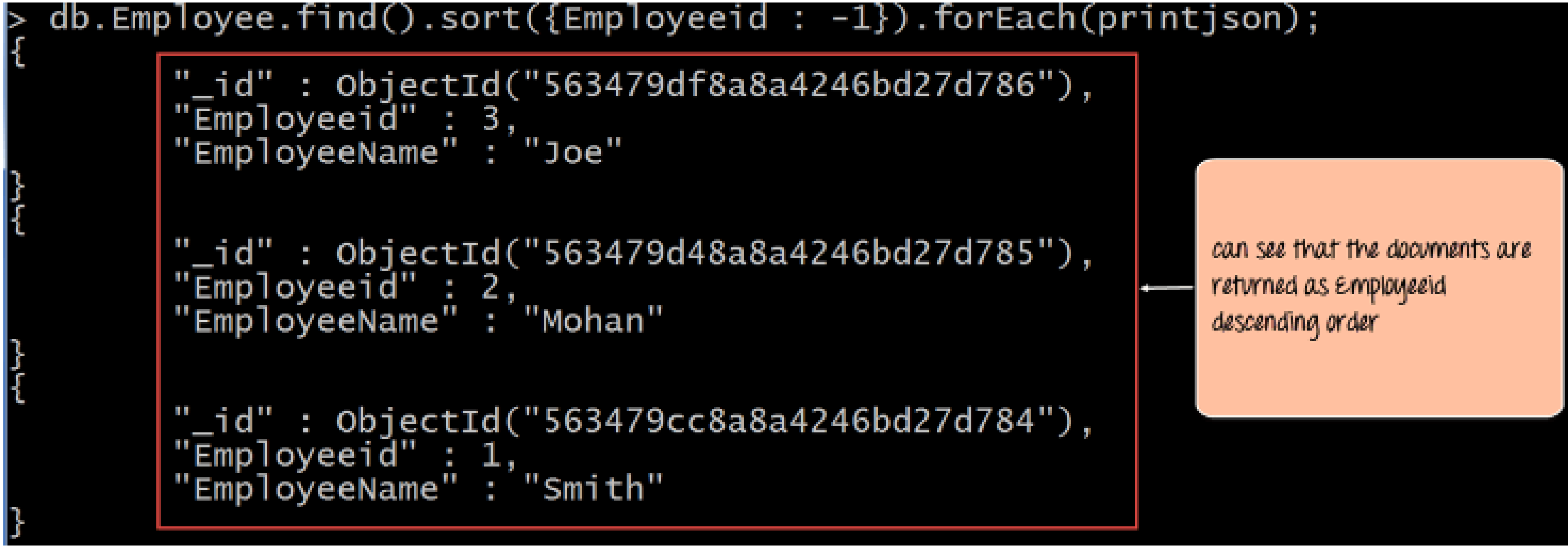
```
> db.Employee.find().limit(2).forEach(printjson);
{
        "_id" : ObjectId("563479cc8a8a4246bd27d784"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
{
        "_id" : ObjectId("563479d48a8a4246bd27d785"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
>
```

# MongoDB Sort

One can specify the order of documents to be returned based on ascending or descending order of any key in the collection.

```
>db.COLLECTION_NAME.find().sort({KEY:1})
```

```
db.Employee.find().sort({Employeeid:-1}). forEach
   (printjson)
```



```
> db.Employee.find().sort({Employeeid : -1}).forEach(printjson);
{
        "_id" : ObjectId("563479df8a8a4246bd27d786"),
        "Employeeid" : 3,
        "EmployeeName" : "Joe"
}
{

        "_id" : ObjectId("563479d48a8a4246bd27d785"),
        "Employeeid" : 2,
        "EmployeeName" : "Mohan"
}
{

        "_id" : ObjectId("563479cc8a8a4246bd27d784"),
        "Employeeid" : 1,
        "EmployeeName" : "Smith"
}
```

can see that the documents are returned as Employeeid descending order

# MongoDB Sharding

- Sharding is a method for distributing data across multiple machines.
- Database systems with large data sets or high throughput applications can challenge the capacity of a single server.
- For example, high query rates can exhaust the CPU capacity of the server. Working set sizes larger than the system’s RAM stress the I/O capacity of disk drives.

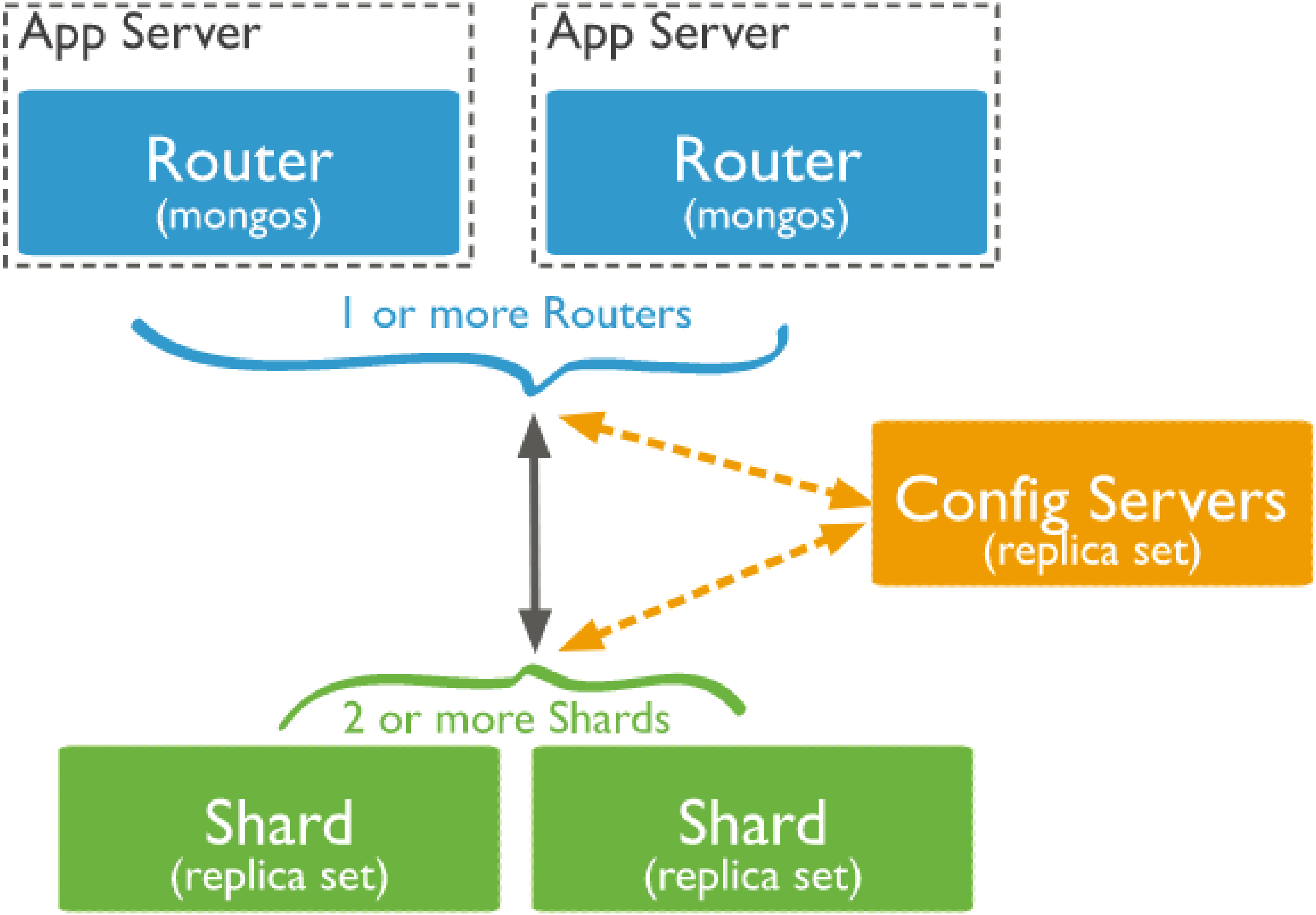
- There are two methods for addressing system growth:
- **vertcal and horizontal** scaling.
- ***Vertical Scaling*** involves increasing the capacity of a single server, such as using a more powerful CPU, adding more RAM, or increasing the amount of storage space.
- ***Horizontal Scaling*** involves dividing the system dataset and load over multiple servers, adding additional servers to increase capacity as required.

# MongoDB supports *horizontal scaling* through sharding.

## Sharded Cluster

A MongoDB sharded cluster consists of the following components:

- shard: Each shard contains a subset of the sharded data. Each shard can be deployed as a replica set.
- mongos: The mongos acts as a query router, providing an interface between client applications and the sharded cluster.
- config servers: Config servers store metadata and configuration settings for the cluster. As of MongoDB 3.4, config servers must be deployed as a replica set (CSRS).


App Server
Router
(mongos)
App Server
Router
(mongos)
1 or more Routers
Config Servers
(replica set)
2 or more Shards
Shard
(replica set)
Shard
(replica set)

# Advantages of Sharding

- **Reads / Writes:** MongoDB distributes the read and write workload across the shards in the sharded cluster, allowing each shard to process a subset of cluster operations. Both read and write workloads can be scaled horizontally across the cluster by adding more shards.
- **Storage Capacity:** Sharding distributes data across the shards in the cluster, allowing each shard to contain a subset of the total cluster data. As the data set grows, additional shards increase the storage capacity of the cluster.
- **High Availability:** A sharded cluster can continue to perform partial read / write operations even if one or more shards are unavailable. While the subset of data on the unavailable shards cannot be accessed during the downtime, reads or writes directed at the available shards can still succeed.

# MongoDB CURD

- **CRUD** operations refer to the basic **Create**(Insert), **Read**(find), **Update** and **Delete** operations.