NoSQL Database:

NoSQL databases come in a **variety** of **types** based on their **data model**. The main types are **document**, **key-value**, **wide-column**, and **graph**.

CAP Theorem:

- The **CAP** theorem is also called **Brewer's** Theorem, because it was **first** introduced by **Professor Eric A. Brewer** during a **talk** he gave on **distributed computing** in **2000**.
- Two years later, MIT professors Seth Gilbert and Nancy Lynch published a proof of "Brewer's Conjecture."

A **distributed system** is a **network** that **stores data** on **more** than **one node** (**physical** or **virtual** machines) at the **same time**. Because all **cloud applications** are **distributed systems**, it's **essential** to **understand** the **CAP theorem** when **designing** a **cloud app** so that you can **choose** a **data management system** that **delivers** the **characteristics** your **application needs most**.

The **CAP theorem applies** a **similar type** of **logic** to **distributed systems**—namely, that a **distributed system** can **deliver only two** of **three desired characteristics**: **consistency**, **availability**, and **partition tolerance** (the '**C,**' '**A**' and '**P**' in **CAP**).

Let's take a **detailed** look at the **three distributed system characteristics** to which the **CAP theorem** refers.

Consistency:

- **Consistency** means that **all clients** see the **same data** at the **same time**, no matter which **node** they **connect** to.
- For this to happen, whenever **data** is **written** to **one node**, it must be **instantly forwarded** or **replicated** to **all** the **other nodes** in the **system** before the **write** is deemed '**successful**.'

Availability:

- **Availability** means that **any client** making a **request** for **data gets** a **response**, even if **one** or **more nodes** are **down**.
- Another way to state this—**all working nodes** in the **distributed system return** a **valid response** for **any request**, **without exception**.

Partition tolerance:

- A **partition** is a **communications break** within a **distributed system**—a **lost or temporarily delayed** connection between **two nodes**.
- **Partition tolerance** means that the **cluster must continue** to **work** despite any **number** of **communication breakdowns** between **nodes** in the **system**.

NoSQL (non-relational) **databases** are **ideal** for **distributed network applications**. Unlike their **vertically scalable SQL** (relational) counterparts, **NoSQL** databases are **horizontally scalable** and **distributed** by design—they can **rapidly scale** across a **growing network** consisting of **multiple interconnected nodes**.

Today, NoSQL databases are classified based on the two **CAP** characteristics they support:

CP database:
A **CP** database **delivers consistency** and **partition tolerance** at the **expense** of **availability**. When a **partition occurs** between **any two nodes**, the **system** has to **shut down** the **non-consistent node** (i.e. make it unavailable) **until** the **partition** is **resolved**.
E.g. Mongo DB

AP database:
An **AP** database delivers **availability** and **partition tolerance** at the **expense** of **consistency**. When a **partition** occurs, **all nodes remain available** but those at the **wrong end** of a **partition** might **return** an **older version** of **data** than **others**. (When the **partition** is **resolved**, the **AP** databases typically **resync** the **nodes** to **repair all inconsistencies** in the **system**.)
E.g. Cassandra

CA database:
A **CA** database delivers **consistency** and **availability** across **all nodes**. It can't do this if there is a **partition** between **any two nodes** in the **system** and therefore **can't deliver fault tolerance**.
E.g. Postgres SQL DB

MongoDB:

MongoDB is a **cross-platform**, **document oriented database** that provides **high performance**, **high availability**, and **easy scalability**. MongoDB works on concept of **collection (Table)** and **document (Rows)**.

Database:
- **Database** is a **physical container** for **collections**.
- Each **database** gets its **own set** of **files** on the **file system**.
- A **single MongoDB server** typically has **multiple databases**.

Collection:
- **Collection** is a **group** of **documents**. It is the **equivalent** of an **RDBMS table**.
- A **collection exists** within a **single database**.
- **Collections do not enforce** a **schema**.
- **Documents** within a **collection** can have **different fields**.

Document:

- A **document** is a **set** of **key-value pairs**. **Documents** have **dynamic schema**.
- **Dynamic schema** means that **documents** in the **same collection do not need** to **have** the **same set** of **fields** or **structure**, and **common fields** in a **collection's documents** may **hold different types** of **data**.

The following table shows the **relationship** of **RDBMS terminology** with **MongoDB**.

RDBMS	MongoDB
Database	Database
Table	Collection
Tuple/Row	Document
column	Field
Table Join	Embedded Documents
Primary Key	Primary Key (Default key _id provided by MongoDB itself)
Database Server and Client	
mysqld/Oracle	mongod
mysql/sqlplus	mongo

```
{
  _id: ObjectId(7df78ad8902c)
  title: 'MongoDB Overview',
  description: 'MongoDB is no sql database',
  by: 'Saif Shaikh',
  url: 'http://www.smidsytechnologies.com',
  tags: ['mongodb', 'database', 'NoSQL'],
  likes: 100,
  comments: [
    {
      user:'user1',
      message: 'My first comment',
      dateCreated: new Date(2011,1,20,2,15),
      like: 0
    },
    {
      user:'user2',
      message: 'My second comments',
      dateCreated: new Date(2011,1,25,7,45),
      like: 5
```

```
    }
  ]
}
```

Note:
_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**.

MongoDB advantages over RDBMS:
- In recent days, **MongoDB** is a **new** and **popularly used database**. It is a **document** based, **non-relational** database provider.
- **Although** it is **100 times faster** than the **traditional database** but it is **early** to **say** that it will **broadly replace** the **traditional RDBMS**. But it may be **very useful** in **long term** to **gain performance** and **scalability**.
- A **Relational database** has a **typical schema design** that **shows number** of **tables** and the **relationship between** these **tables**, while in **MongoDB** there is **no concept** of **relationship**.

MongoDB Advantages:
- **MongoDB** is **schema less**. It is a **document database** in which **one collection holds different documents**.
- There may be **difference between number** of **fields**, **content** and **size** of the **document** from **one** to **other**.
- **Structure** of a **single object** is **clear** in **MongoDB**.
- There are **no complex joins** in MongoDB.
- MongoDB **provides** the **facility** of **deep query** because it **supports** a **powerful dynamic query** on **documents**.
- It is **very easy** to **scale**.
- It uses **internal memory** for **storing working sets** and this is the **reason** of its **fast access**.

→ Check Version:
mongod --version

→ Help for Commands:
mongo → to get mongodb prompt
db.help()

→ **Check Statistics:**
db.stats()

The use Command:
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**.

In MongoDB **default database** is **test**. If you **didn't** create any **database**, then **collections** will be **stored** in **test database**.

use saif_db

```
> use saif_db
switched to db saif_db
```

```
> db
saif_db
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

Your **created** database (**saif_db**) is **not present** in **list**. To **display database**, you **need** to **insert** at least **one document** into it.

```
> use saif_db1
switched to db saif_db1
```

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

```
> db.movie.insert({"name":"Saif Shaikh"})
WriteResult({ "nInserted" : 1 })
```

```
> show dbs
admin     0.000GB
config    0.000GB
local     0.000GB
saif_db1  0.000GB
```

The dropDatabase() Method:
MongoDB **db.dropDatabase()** command is used to **drop** a **existing database**.

This will **delete** the **selected database**. If you have **not selected** any database, then it will delete **default 'test'** database.

```
> db.dropDatabase()
{ "dropped" : "saif_db1", "ok" : 1 }
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
```

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

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

Parameter	Type	Description
Name	String	Name of the collection to be created
Options	Document	(Optional) Specify options about memory size and indexing

Following is the list of options you can use:

Field	Type	Description
capped	Boolean	(Optional) If true, enables a capped collection. Capped collection is a fixed size collection that automatically overwrites its oldest entries when it reaches its maximum size. **If you specify true, you need to specify size parameter also.**
autoIndexId	Boolean	(Optional) If true, automatically create index on _id field.s Default value is false.
size	number	(Optional) Specifies a maximum size in bytes for a capped collection. **If capped is true, then you need to specify this field also.**
max	number	(Optional) Specifies the maximum number of documents allowed in the capped collection.

Note: While inserting the document, MongoDB first checks size field of capped collection, then it checks max field.

1) Basic syntax of **createCollection()** method **without options** is as **follows**:

```
> db.createCollection("mycollection")
{ "ok" : 1 }
```

2) You can **check** the **created collection** by **using** the command **show collections**.

```
> show collections
mycollection
```

The drop() Method:
MongoDB **db.collection.drop()** is used to **drop** a **collection** from the **database**.

```
> show collections
mycollection
```

Now **drop** the **collection** with the name **mycollection**.

```
> db.mycollection.drop()
true
```

```
> show collections
```

Note: drop() method will **return true**, if the **selected collection** is **dropped successfully**, otherwise it will **return false**.

The insert() Method:
To **insert data** into **MongoDB collection**, you need to use MongoDB **insert()** or **save()** method.

Syntax: db.COLLECTION_NAME.insert(document)

```
> db.users.insert({
... title: "MongoDB Practice",
... description: "MongoDB is NoSql database",
... by: "Saif Shaikh",
... url: "http://www.smidsytechnologies.com",
... tags: ['mongodb', 'database', 'NoSQL'],
... likes: 100
... })
WriteResult({ "nInserted" : 1 })
```

If the collection doesn't exist in the database, then MongoDB will create this collection and then insert a document into it.

In the inserted document, if we don't specify the _id parameter, then MongoDB assigns a unique ObjectId for this document.

_id is 12 bytes hexadecimal number unique for every document in a collection. 12 bytes are divided as follows:

_id: ObjectId(4 bytes timestamp, 3 bytes machine id, 2 bytes process id, 3 bytes incrementer)

db.users.find()

```
> db.users.find()
{ "_id" : ObjectId("61befc33649188e4c42e27b0"), "title" : "MongoDB Practice", "description" : "Mon
goDB is NoSql database", "by" : "Saif Shaikh", "url" : "http://www.smidsytechnologies.com", "tags"
 : [ "mongodb", "database", "NoSQL" ], "likes" : 100 }
```

You can also pass an array of documents into the insert() method as shown below:

```
> db.createCollection('post')
{ "ok" : 1 }
```

```
> db.post.insert([
... {
... title: "MongoDB Overview",
... description: "MongoDB is NoSQL database",
... by: "Saif Shaikh",
... url: "http://www.smidsytechnologies.com",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 100
... },
... {
... title: "NoSQL Database",
... description: "NoSQL database doesn't have tables",
... by: "Saif Shaikh",
... url: "http://www.smidsytechnologies.com",
... tags: ["mongodb", "database", "NoSQL"],
... likes: 20,
... comments: [
... {
... user:"user1",
... message: "My first comment",
... dateCreated: new Date(2013,11,10,2,35),
... like: 0
... }
... ]
... }
... ])
```

```
BulkWriteResult({
        "writeErrors" : [ ],
        "writeConcernErrors" : [ ],
        "nInserted" : 2,
        "nUpserted" : 0,
        "nMatched" : 0,
        "nModified" : 0,
        "nRemoved" : 0,
        "upserted" : [ ]
})
```

The insertOne() method:
If you need to insert only one document into a collection you can use this method.

Syntax: db.COLLECTION_NAME.insertOne(document)

Following example creates a new collection named empDetails and inserts a document using the insertOne() method.

```
> db.createCollection("empDetails")
{ "ok" : 1 }
```

```
> db.empDetails.insertOne(
... {
... First_Name: "Saif",
... Last_Name: "Shaikh",
... Date_Of_Birth: "1991-04-14",
... e_mail: "saifshk85@gmail.com",
... phone: "9848022338"
... })
{
        "acknowledged" : true,
        "insertedId" : ObjectId("61bf0167649188e4c42e27b3")
}
```

The insertMany() method:

You can insert multiple documents using the insertMany() method. To this method you need to pass an array of documents.

```
> db.empDetails.insertMany(
... [
... {
... First_Name: "Radhika",
... Last_Name: "Sharma",
... Date_Of_Birth: "1995-09-26",
... e_mail: "radhika_sharma.123@gmail.com",
... phone: "9000012345"
... },
... {
... First_Name: "Rachel",
... Last_Name: "Christopher",
... Date_Of_Birth: "1990-02-16",
... e_mail: "Rachel_Christopher.123@gmail.com",
... phone: "9000054321"
... },
... {
... First_Name: "Fathima",
... Last_Name: "Sheik",
... Date_Of_Birth: "1990-02-16",
... e_mail: "Fathima_Sheik.123@gmail.com",
... phone: "9000054321"
... }
... ]
... )
```

```
{
        "acknowledged" : true,
        "insertedIds" : [
                ObjectId("61bf0228649188e4c42e27b4"),
                ObjectId("61bf0228649188e4c42e27b5"),
                ObjectId("61bf0228649188e4c42e27b6")
        ]
}
```

MongoDB - Query Document:

The find() Method:

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

db.empDetails.find()

```
> db.empDetails.find()
{ "_id" : ObjectId("61bf0167649188e4c42e27b3"), "First_Name" : "Saif", "Last_Name" : "Shaikh", "Date_Of_Birth" : "1
991-04-14", "e_mail" : "saifshk85@gmail.com", "phone" : "9848022338" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b4"), "First_Name" : "Radhika", "Last_Name" : "Sharma", "Date_Of_Birth" :
 "1995-09-26", "e_mail" : "radhika_sharma.123@gmail.com", "phone" : "9000012345" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b5"), "First_Name" : "Rachel", "Last_Name" : "Christopher", "Date_Of_Birt
h" : "1990-02-16", "e_mail" : "Rachel_Christopher.123@gmail.com", "phone" : "9000054321" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b6"), "First_Name" : "Fathima", "Last_Name" : "Sheik", "Date_Of_Birth" :
"1990-02-16", "e_mail" : "Fathima_Sheik.123@gmail.com", "phone" : "9000054321" }
```

To display the results in a formatted way, you can use pretty() method:
db.empDetails.find().pretty()

```
> db.empDetails.find().pretty()
{
        "_id" : ObjectId("61bf0167649188e4c42e27b3"),
        "First_Name" : "Saif",
        "Last_Name" : "Shaikh",
        "Date_Of_Birth" : "1991-04-14",
        "e_mail" : "saifshk85@gmail.com",
        "phone" : "9848022338"
}
{
        "_id" : ObjectId("61bf0228649188e4c42e27b4"),
        "First_Name" : "Radhika",
        "Last_Name" : "Sharma",
        "Date_Of_Birth" : "1995-09-26",
        "e_mail" : "radhika_sharma.123@gmail.com",
        "phone" : "9000012345"
}
{
        "_id" : ObjectId("61bf0228649188e4c42e27b5"),
        "First_Name" : "Rachel",
        "Last_Name" : "Christopher",
        "Date_Of_Birth" : "1990-02-16",
        "e_mail" : "Rachel_Christopher.123@gmail.com",
        "phone" : "9000054321"
}
```

The findOne() method:

Apart from the find() method, there is findOne() method, that returns only one document.

```
> db.empDetails.findOne({'First_Name':'Saif'})
{
        "_id" : ObjectId("61bf0167649188e4c42e27b3"),
        "First_Name" : "Saif",
        "Last_Name" : "Shaikh",
        "Date_Of_Birth" : "1991-04-14",
        "e_mail" : "saifshk85@gmail.com",
        "phone" : "9848022338"
}
```

RDBMS Where Clause Equivalents in MongoDB:

To query the document on the basis of some condition, you can use following operations.

Operation	Syntax	Example	RDBMS Equivalent
Equality	{<key>:{$eg; <value>}}	db.mycol.find({"by":"tutorials point"}).pretty()	where by = 'tutorials point'
Less Than	{<key>:{$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": {$ne:50}}).pretty()	where likes != 50
Values in an array	{<key>:{$in: [<value1>, <value2>,..... <valueN>]}}	db.mycol.find({"name":{$in: ["Raj", "Ram", "Raghu"]}}).pretty()	Where name matches any of the value in :["Raj", "Ram", "Raghu"]
Values not in an array	{<key>:{$nin: <value>}}	db.mycol.find({"name": {$nin:["Ramu", "Raghav"]}}).pretty()	Where name values is not in the array : ["Ramu", "Raghav"] or, doesn't exist at all

AND in MongoDB:
To query documents based on the AND condition, you need to use $and keyword.

Syntax: db.collection.find({ $and: [{<key1>:<value1>}, { <key2>:<value2>}] })

```
> db.empDetails.find({$and:[{"First_Name":"Saif"},{"Last_Name": "Shaikh"}]}).pretty()
{
        "_id" : ObjectId("61bf0167649188e4c42e27b3"),
        "First_Name" : "Saif",
        "Last_Name" : "Shaikh",
        "Date_Of_Birth" : "1991-04-14",
        "e_mail" : "saifshk85@gmail.com",
        "phone" : "9848022338"
}
```

Note: You can pass any number of key, value pairs in find clause.

OR in MongoDB: To query documents based on the OR condition, you need to use $or keyword.

Syntax: db.collection.find({ $or: [{key1: value1}, {key2:value2}] }).pretty()

```
> db.empDetails.find({$or:[{"First_Name":"Saif"},{"Last_Name": "Sharma"}]}).pretty()
{
        "_id" : ObjectId("61bf0167649188e4c42e27b3"),
        "First_Name" : "Saif",
        "Last_Name" : "Shaikh",
        "Date_Of_Birth" : "1991-04-14",
        "e_mail" : "saifshk85@gmail.com",
        "phone" : "9848022338"
}
{
        "_id" : ObjectId("61bf0228649188e4c42e27b4"),
        "First_Name" : "Radhika",
        "Last_Name" : "Sharma",
        "Date_Of_Birth" : "1995-09-26",
        "e_mail" : "radhika_sharma.123@gmail.com",
        "phone" : "9000012345"
}
```

NOR in MongoDB:
To query documents based on the NOT condition, you need to use $not keyword.

Syntax: db.collection.find({ $or: [{key1: value1}, {key2:value2}] }).pretty()

```
> db.empDetails.find({$nor:[{"First_Name":"Saif"},{"Last_Name": "Sharma"}]}).pretty()
{
        "_id" : ObjectId("61bf0228649188e4c42e27b5"),
        "First_Name" : "Rachel",
        "Last_Name" : "Christopher",
        "Date_Of_Birth" : "1990-02-16",
        "e_mail" : "Rachel_Christopher.123@gmail.com",
        "phone" : "9000054321"
}
{
        "_id" : ObjectId("61bf0228649188e4c42e27b6"),
        "First_Name" : "Fathima",
        "Last_Name" : "Sheik",
        "Date_Of_Birth" : "1990-02-16",
        "e_mail" : "Fathima_Sheik.123@gmail.com",
        "phone" : "9000054321"
}
```

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: db.COLLECTION_NAME.update(SELECTION_CRITERIA, UPDATED_DATA)

```
> db.empDetails.update({'phone':'9848022338'},{$set:{'phone':'8657250250'}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
>
> db.empDetails.find()
{ "_id" : ObjectId("61bf0167649188e4c42e27b3"), "First_Name" : "Saif", "Last_Name" : "Shaikh", "Date_Of_Birth" : "1
991-04-14", "e_mail" : "saifshk85@gmail.com", "phone" : "8657250250" }
```

By default, MongoDB will update only a single document. To update multiple documents, you need to set a parameter 'multi' to true.

```
> db.empDetails.update({'phone':'9000054321'},{$set:{'phone':'8657250250'}},{multi:true})
WriteResult({ "nMatched" : 2, "nUpserted" : 0, "nModified" : 2 })
```

```
> db.empDetails.find()
{ "_id" : ObjectId("61bf0167649188e4c42e27b3"), "First_Name" : "Saif", "Last_Name" : "Shaikh", "Date_Of_Birth" : "1
991-04-14", "e_mail" : "saifshk85@gmail.com", "phone" : "8657250250" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b4"), "First_Name" : "Radhika", "Last_Name" : "Sharma", "Date_Of_Birth" :
 "1995-09-26", "e_mail" : "radhika_sharma.123@gmail.com", "phone" : "9000012345" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b5"), "First_Name" : "Rachel", "Last_Name" : "Christopher", "Date_Of_Birt
h" : "1990-02-16", "e_mail" : "Rachel_Christopher.123@gmail.com", "phone" : "8657250250" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b6"), "First_Name" : "Fathima", "Last_Name" : "Sheik", "Date_Of_Birth" :
"1990-02-16", "e_mail" : "Fathima_Sheik.123@gmail.com", "phone" : "8657250250" }
```

MongoDB findOneAndUpdate() method:
The findOneAndUpdate() method updates the values in the existing document:

Syntax: db.COLLECTION_NAME.findOneAndUpdate(SELECTIOIN_CRITERIA, UPDATED_DATA)

```
> db.empDetails.findOneAndUpdate(
... {First_Name: 'Radhika'},
... { $set: { Age: '30',e_mail: 'radhika_newemail@gmail.com'}}
... )
{
        "_id" : ObjectId("61bf0228649188e4c42e27b4"),
        "First_Name" : "Radhika",
        "Last_Name" : "Sharma",
        "Date_Of_Birth" : "1995-09-26",
        "e_mail" : "radhika_sharma.123@gmail.com",
        "phone" : "9000012345"
}
```

MongoDB updateOne() method:
This methods updates a single document which matches the given filter.

Syntax: db.COLLECTION_NAME.updateOne(<filter>, <update>)

```
> db.empDetails.find()
{ "_id" : ObjectId("61bf0167649188e4c42e27b3"), "First_Name" : "Saif", "Last_Name" : "Shaikh", "Date_Of_Birth" : "1
991-04-14", "e_mail" : "saifshk85@gmail.com", "phone" : "8657250250" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b4"), "First_Name" : "Radhika", "Last_Name" : "Sharma", "Date_Of_Birth" :
 "1995-09-26", "e_mail" : "radhika_newemail@gmail.com", "phone" : "9000012345", "Age" : "31" }
```

MongoDB updateMany() method:
The updateMany() method updates all the documents that matches the given filter.

Syntax: db.COLLECTION_NAME.update(<filter>, <update>)

```
> db.empDetails.updateMany(
... {Age:{ $gt: "25" }},
... { $set: { Age: '00'}}
... )
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

```
> db.empDetails.find()
{ "_id" : ObjectId("61bf0167649188e4c42e27b3"), "First_Name" : "Saif", "Last_Name" : "Shaikh", "Date_Of_Birth" : "1
991-04-14", "e_mail" : "saifshk85@gmail.com", "phone" : "8657250250" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b4"), "First_Name" : "Radhika", "Last_Name" : "Sharma", "Date_Of_Birth" :
 "1995-09-26", "e_mail" : "radhika_newemail@gmail.com", "phone" : "9000012345", "Age" : "00" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b5"), "First_Name" : "Rachel", "Last_Name" : "Christopher", "Date_Of_Birt
h" : "1990-02-16", "e_mail" : "radhika_newemail@gmail.com", "phone" : "8657250250", "Age" : "00" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b6"), "First_Name" : "Fathima", "Last_Name" : "Sheik", "Date_Of_Birth" :
"1990-02-16", "e_mail" : "Fathima_Sheik.123@gmail.com", "phone" : "8657250250" }
```

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.

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

Syntax: db.COLLECTION_NAME.remove(DELLETION_CRITTERIA)

```
> db.empDetails.remove({'Age':'00'})
WriteResult({ "nRemoved" : 2 })
```

```
> db.empDetails.find()
{ "_id" : ObjectId("61bf0167649188e4c42e27b3"), "First_Name" : "Saif", "Last_Name" : "Shaikh", "Date_Of_Birth" : "1
991-04-14", "e_mail" : "saifshk85@gmail.com", "phone" : "8657250250" }
{ "_id" : ObjectId("61bf0228649188e4c42e27b6"), "First_Name" : "Fathima", "Last_Name" : "Sheik", "Date_Of_Birth" :
"1990-02-16", "e_mail" : "Fathima_Sheik.123@gmail.com", "phone" : "8657250250" }
```

Note:
Above example will remove all the documents whose Age is '00'.

Remove Only One:
If there are multiple records and you want to delete only the first record, then set justOne parameter in remove() method.

Syntax: db.COLLECTION_NAME.remove(DELETION_CRITERIA,1)

Remove All Documents:
If you don't specify deletion criteria, then MongoDB will delete whole documents from the collection. This is equivalent of SQL's truncate command.

Syntax: db.mycol.remove({})