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

**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 MongoDB 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. Typically, all documents in a collection are of similar or related purpose.

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

**Sample Document**

Following example shows the document structure of a blog site, which is simply a comma separated key value pair.

**{**

**\_id: ObjectId(7df78ad8902c)**

**title: 'MongoDB Overview',**

**description: 'MongoDB is no sql database',**

**by: 'tutorials point',**

**url: 'http://www.tutorialspoint.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**

**}**

**]**

**}**

**\_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.

**What is MongoDB?**

MongoDB is an open-source, cross-platform, and distributed document-based database designed for ease of application development and scaling. It is a NoSQL database developed by [MongoDB Inc](https://www.mongodb.com/company).

MongoDB name is derived from the word "Humongous" which means huge, enormous. MongoDB database is built to store a huge amount of data and also perform fast.

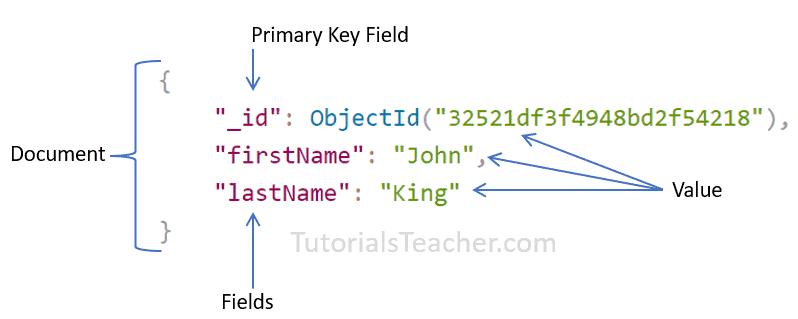
MongoDB is not a Relational Database Management System (RDBMS). It's called a "NoSQL" database. It is opposite to SQL based databases where it does not normalize data under schemas and tables where every table has a fixed structure. Instead, it stores data in the collections as JSON based documents and does not enforce schemas. It does not have tables, rows, and columns as other SQL (RDBMS) databases.

The following table lists the relation between MongoDB and RDBMS terminologies.

| MongoDB (NoSQL Database) | RDBMS (SQL Server, Oracle, etc.) |
| --- | --- |
| Database | Database |
| Collection | Table |
| Document | Row (Record) |
| Field | Column |

In the RDBMS database, a table can have multiple rows and columns. Similarly in MongoDB, a collection can have multiple documents which are equivalent to the rows. Each document has multiple "fields" which are equivalent to the columns. Documents in a single collection can have different fields.

The following is an example of JSON based document.



Advantages of MongoDB

1. MongoDB stores data as JSON based document that does not enforce the schema. It allows us to store hierarchical data in a document. This makes it easy to store and retrieve data in an efficient manner.
2. It is easy to scale up or down as per the requirement since it is a document based database. MongoDB also allows us to split data across multiple servers.
3. MongoDB provides rich features like indexing, aggregation, file store, etc.
4. MongoDB performs fast with huge data.
5. MongoDB provides drivers to store and fetch data from different applications developed in different technologies such as C#, Java, Python, Node.js, etc.
6. MongoDB provides tools to manage MongoDB databases.

**MongoDB Basics**

**Summary**: in this tutorial, you’ll learn some basic concepts of MongoDB such as documents, collections, databases, and namespaces.

Data formats

In MongoDB, you will often deal with JSON and BSON formats. Therefore, it’s important to fully understand them.

**JSON**

JSON stands for JavaScript Object Notation. JSON syntax is based on a subset of JavaScript ECMA-262 3rd edition.

A JSON document is a collection of fields and values in a structured format. For example:

**{**

**"first\_name": "John",**

**"last\_name": "Doe",**

**"age": 22,**

**"skills": ["Programming","Databases", "API"]**

**}**

Code language: JSON / JSON with Comments (json)

**BSON**

BSON stands for Binary JSON, which is a binary-coded serialization of JSON-like documents.

**Documents**

MongoDB stores data records as BSON documents, which are simply called documents.

A computer screen shot of a computer code

Description automatically generated

A document is a set of field-and-value pairs with the following structure:

**{**

**field\_name1: value1,**

**field\_name2: value2,**

**field\_name3: value3,**

**...**

**}**

In this syntax, the field names are strings and values can be numbers, strings, objects, arrays, etc. For example:

**{**

**\_id: ObjectId("5f339953491024badf1138ec"),**

**title: "MongoDB Tutorial",**

**isbn: "978-4-7766-7944-8",**

**published\_date: new Date('June 01, 2020'),**

**author: { first\_name: "John"**

**, last\_name: "Doe"}**

**}**

Code language: CSS (css)

This document has the following field-and-value pairs:

* \_id holds an ObjectId
* title holds a string.
* isbn holds a string.
* published\_date holds a value of the Date type.
* author holds an embedded document that contains two fields first\_name and last\_name.

If you are familiar with a relational database management system (RDBMS), you will find that a document is similar to a row in a table, but it is much more expressive.

Field names have the following restrictions:

* MongoDB reserves the field \_id and uses it to uniquely identify the document.
* Field names cannot contain null characters.
* Top-level field names cannot start with the dollar sign ($) character.

**Collections**

MongoDB stores documents in a collection. A collection is a group of documents.

A screenshot of a computer program

Description automatically generated

A collection is analogous to a table in an RDBMS.

|  |  |
| --- | --- |
| **MongoDB** | **RDBMS** |
| Documents | Rows |
| Collections | Tables |

Unlike a table that has a fixed schema, a collection has a dynamic schema.

It means that a collection may contain documents that have any number of different “shapes”. For example, you can store the following documents in the same collection:

**{**

**title: "MongoDB Tutorial",**

**published\_date: new Date('June 01, 2020')**

**}**

**{**

**title: "MongoDB Basics",**

**published\_date: new Date('Jan 01, 2021'),**

**isbn": "978-4-7766-7944-8"**

**}**

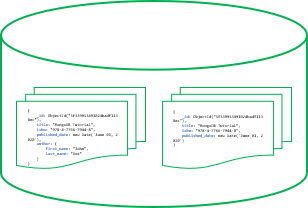
Note that the second document has one more field than the first one. In theory, you can have completely different fields for every document.

A collection has a name e.g., books. The collection name cannot:

* contain the dollar sign ($)
* contain the null character (\0).
* be an empty string.
* begin with the system because MongoDB reserves system\* for internal collection names.

**Databases**

MongoDB stores collections into a database. A single instance of MongoDB can host multiple databases.



A database can be referenced by a name for example bookdb. The database names cannot:

* Be an empty string ("").
* Contain any of these characters: */*, *\*, *.*, *“*, *\**, *<*, *>*, *:*, *|*, *?*, *$*, (a single space), or *\0* (the null character).
* Exceed the maximum size which is 64 bytes.

MongoDB also has some reserved database names such as admin, local, and config that you cannot use to create new databases.

**Namespace**

A namespace is a concatenation of the database name with a collection in that database. Namespaces allow you to fully qualify collections.

For example, if the collection name is books and database name is bookdb, the namespace of the books collection would be bookdb.books.

**Summary**

* MongoDB stores data records as BSON documents. A document is a set of field-and-value pairs.
* MongoDB stores documents in a collection and collections in a database.
* A namespace is a concatenation of the database name and the collection name (database\_name.collection\_name) to fully qualify the collection.

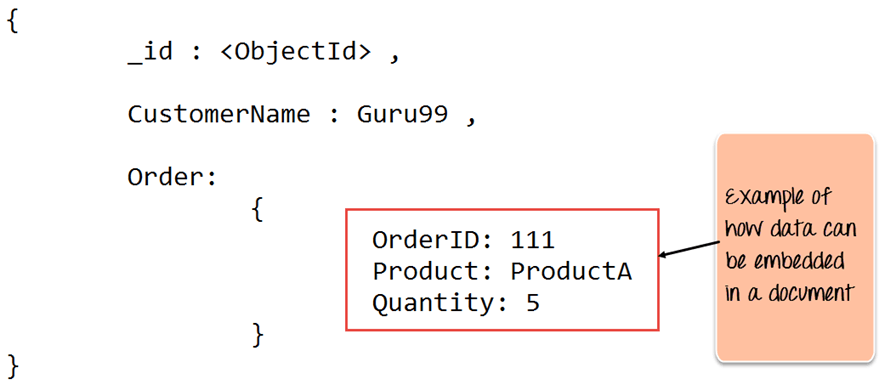
**MongoDB Features**

1. Each database contains collections which in turn contains documents. Each document can be different with a varying number of fields. The size and content of each document can be different from each other.
2. The document structure is more in line with how developers construct their classes and objects in their respective programming languages. Developers will often say that their classes are not rows and columns but have a clear structure with key-value pairs.
3. The rows (or documents as called in MongoDB) doesn’t need to have a schema defined beforehand. Instead, the fields can be created on the fly.
4. The data model available within MongoDB allows you to represent hierarchical relationships, to store arrays, and other more complex structures more easily.
5. Scalability – The MongoDB environments are very scalable. Companies across the world have defined clusters with some of them running 100+ nodes with around millions of documents within the database

**MongoDB Example**

The below example shows how a document can be modeled in MongoDB.

1. The \_id field is added by MongoDB to uniquely identify the document in the collection.
2. What you can note is that the Order Data (OrderID, Product, and Quantity ) which in RDBMS will normally be stored in a separate table, while in MongoDB it is actually stored as an embedded document in the collection itself. This is one of the key differences in how data is modeled in MongoDB.

[](https://www.guru99.com/images/MongoDB/112015_1051_Introductio1.png)

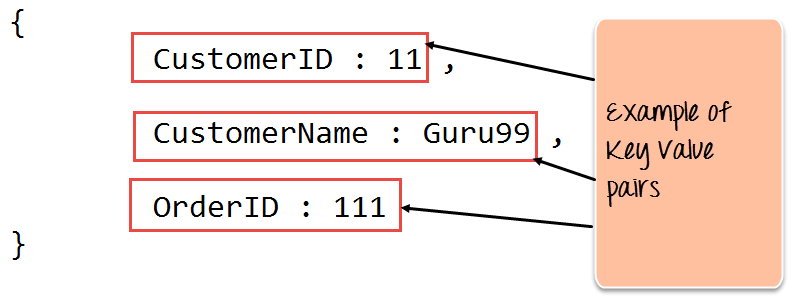
**Key Components of MongoDB Architecture**

Below are a few of the common terms used in MongoDB

1. **\_id** – This is a field required in every MongoDB document. The \_id field represents a unique value in the MongoDB document. The \_id field is like the document’s primary key. If you create a new document without an \_id field, MongoDB will automatically create the field. So for example, if we see the example of the above customer table, Mongo DB will add a 24 digit unique identifier to each document in the collection.

| **\_Id** | **CustomerID** | **CustomerName** | **OrderID** |
| --- | --- | --- | --- |
| 563479cc8a8a4246bd27d784 | 11 | Guru99 | 111 |
| 563479cc7a8a4246bd47d784 | 22 | Trevor Smith | 222 |
| 563479cc9a8a4246bd57d784 | 33 | Nicole | 333 |

1. **Collection** – This is a grouping of MongoDB documents. A collection is the equivalent of a table which is created in any other RDMS such as Oracle or MS SQL. A collection exists within a single database. As seen from the introduction collections don’t enforce any sort of structure.
2. **Cursor** – This is a pointer to the result set of a query. Clients can iterate through a cursor to retrieve results.
3. **Database** – This is a container for collections like in RDMS wherein it is a container for tables. Each database gets its own set of files on the file system. A MongoDB server can store multiple databases.
4. **Document** – A record in a MongoDB collection is basically called a document. The document, in turn, will consist of field name and values.
5. **Field** – A name-value pair in a document. A document has zero or more fields. Fields are analogous to columns in relational databases.The following diagram shows an example of Fields with Key value pairs. So in the example below CustomerID and 11 is one of the key value pair’s defined in the document.

[](https://www.guru99.com/images/MongoDB/112015_1051_Introductio2.png)

1. **JSON** – This is known as[JavaScript](https://www.guru99.com/interactive-javascript-tutorials.html)Object Notation. This is a human-readable, plain text format for expressing structured data. JSON is currently supported in many programming languages.

Just a quick note on the key difference between the \_id field and a normal collection field. The \_id field is used to uniquely identify the documents in a  collection and is automatically added by MongoDB when the collection is created.

**Why Use MongoDB?**

Below are the few of the reasons as to why one should start using MongoDB

1. Document-oriented – Since MongoDB is a [NoSQL](https://www.guru99.com/nosql-tutorial.html) type database, instead of having data in a relational type format, it stores the data in documents. This makes MongoDB very flexible and adaptable to real business world situation and requirements.
2. Ad hoc queries – MongoDB supports searching by field, range queries, and regular expression searches. Queries can be made to return specific fields within documents.
3. Indexing – Indexes can be created to improve the performance of searches within MongoDB. Any field in a MongoDB document can be indexed.
4. Replication – MongoDB can provide high availability with replica sets. A replica set consists of two or more mongo DB instances. Each replica set member may act in the role of the primary or secondary replica at any time. The primary replica is the main server which interacts with the client and performs all the read/write operations. The Secondary replicas maintain a copy of the data of the primary using built-in replication. When a primary replica fails, the replica set automatically switches over to the secondary and then it becomes the primary server.
5. Load balancing – MongoDB uses the concept of sharding to scale horizontally by splitting data across multiple MongoDB instances. MongoDB can run over multiple servers, balancing the load and/or duplicating data to keep the system up and running in case of hardware failure.

**Data Modelling in MongoDB**

As we have seen from the Introduction section, the data in MongoDB has a flexible schema. Unlike in[SQL](https://www.guru99.com/sql.html)databases, where you must have a table’s schema declared before inserting data, MongoDB’s collections do not enforce document structure. This sort of flexibility is what makes MongoDB so powerful.

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

1. 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.
2. 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.
3. Are frequent inserts, 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.

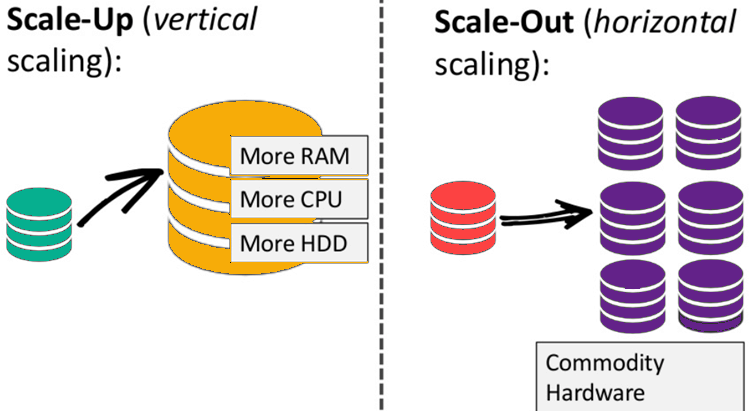
**Difference between MongoDB & RDBMS**

Below are some of the key term differences between MongoDB and RDBMS

| **RDBMS** | **MongoDB** | **Difference** |
| --- | --- | --- |
| Table | Collection | In [RDBMS](https://www.guru99.com/relational-data-model-dbms.html), the table contains the columns and rows which are used to store the data whereas, in MongoDB, this same structure is known as a collection. The collection contains documents which in turn contains Fields, which in turn are key-value pairs. |
| Row | Document | In RDBMS, the row represents a single, implicitly structured data item in a table. In MongoDB, the data is stored in documents. |
| Column | Field | In RDBMS, the column denotes a set of data values. These in MongoDB are known as Fields. |
| Joins | Embedded documents | In RDBMS, data is sometimes spread across various tables and in order to show a complete view of all data, a join is sometimes formed across tables to get the data. In MongoDB, the data is normally stored in a single collection, but separated by using Embedded documents. So there is no concept of joins in MongoDB. |

Apart from the terms differences, a few other differences are shown below

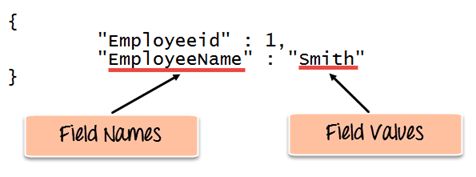
1. Relational databases are known for enforcing data integrity. This is not an explicit requirement in MongoDB.
2. RDBMS requires that data be [normalized](https://www.guru99.com/database-normalization.html) first so that it can prevent orphan records and duplicates Normalizing data then has the requirement of more tables, which will then result in more table joins, thus requiring more keys and indexes.As databases start to grow, performance can start becoming an issue. Again this is not an explicit requirement in MongoDB. MongoDB is flexible and does not need the data to be normalized first.

[](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.

In MongoDB, the first basic step is to have a database and collection in place. The database is used to store all of the collections, and the collection in turn is used to store all of the documents. The documents in turn will contain the relevant Field Name and Field values.

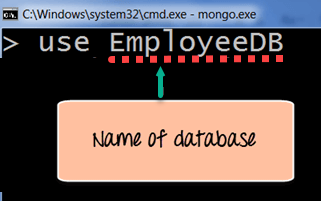
The snapshot below shows a basic example of how a document would look like.

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio1.png)

The Field Names of the document are “Employeeid” and “EmployeeName” and the Field values are “1” and “Smith’ respectively. A bunch of documents would then make up a collection in MongoDB.

**Creating a database using “use” command**

**Creating a database in MongoDB is as simple as issuing the “using” command. The following example shows how this can be done.**

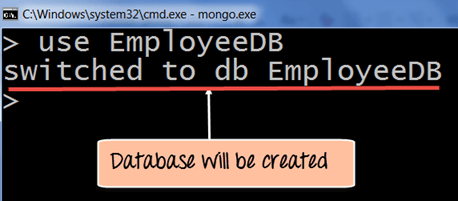
**[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio2.png)**

**Code Explanation:**

* **The “use” command is used to create a database in**[**MongoDB**](https://www.guru99.com/what-is-mongodb.html)**. If the database does not exist a new one will be created.**

**If the command is executed successfully, the following Output will be shown:**

**Output:**

**[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio3.png)**

**MongoDB will automatically switch to the database once created.**

**Creating a Collection/Table using insert()**

**The easiest way to create a collection is to insert a record (which is nothing but a document consisting of Field names and Values) into a collection. If the collection does not exist a new one will be created.**

**The following example shows how this can be done.**

**db.Employee.insert**

**(**

**{**

**"Employeeid" : 1,**

**"EmployeeName" : "Martin"**

**}**

**)**

**Code Explanation:**

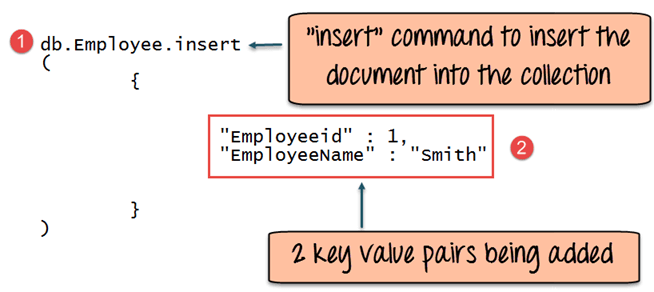
* **As seen above, by using the “insert” command the collection will be created.**

**Adding documents using insert() command**

**MongoDB provides the insert () command to insert documents into a collection. The following example shows how this can be done.**

**Step 1) Write the “insert” command**

**Step 2) Within the “insert” command, add the required Field Name and Field Value for the document which needs to be created.**

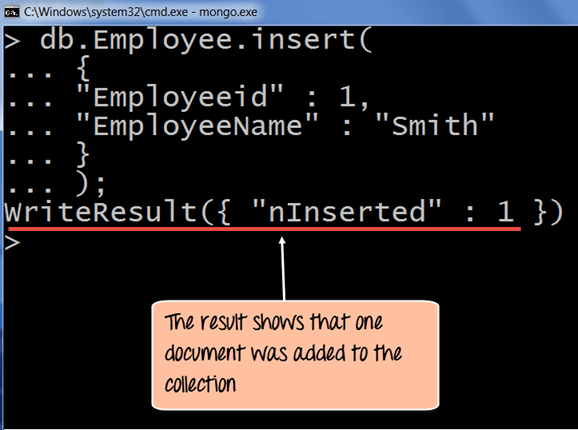
**[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio5.png)**

**Code Explanation:**

1. **The first part of the command is the “insert statement” which is the statement used to insert a document into the collection.**
2. **The second part of the statement is to add the Field name and the Field value, in other words, what is the document in the collection going to contain.**

**If the command is executed successfully, the following Output will be shown**

**Output:**

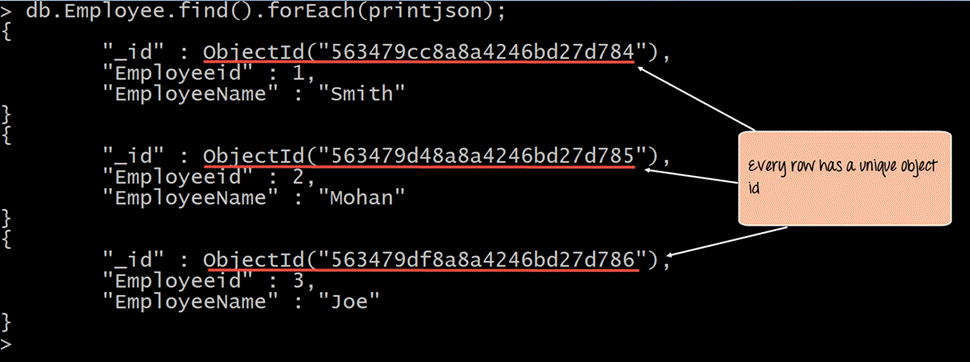
**[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio6.png)**

**The output shows that the operation performed was an insert operation and that one record was inserted into the collection.**

**What is Primary Key in MongoDB?**

In MongoDB, \_id field as the primary key for the collection so that each document can be uniquely identified in the collection. The \_id field contains a unique ObjectID value.

By default when inserting documents in the collection, if you don’t add a field name with the \_id in the field name, then MongoDB will automatically add an Object id field as shown below

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio11.png)

When you query the documents in a collection, you can see the ObjectId for each document in the collection.

If you want to ensure that [MongoDB](https://www.guru99.com/what-is-mongodb.html) does not create the \_id Field when the collection is created and if you want to specify your own id as the \_id of the collection, then you need to explicitly define this while creating the collection.

When explicitly creating an id field, it needs to be created with \_id in its name.

Let’s look at an example on how we can achieve this.

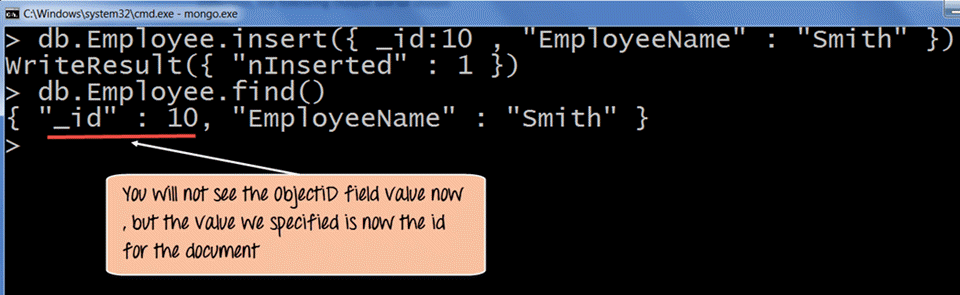
db.Employee.insert({\_id:10, "EmployeeName" : "Smith"})

**Code Explanation:**

1. We are assuming that we are creating the first document in the collection and hence in the above statement while creating the collection, we explicitly define the field \_id and define a value for it.

If the command is executed successfully and now use the find command to display the documents in the collection, the following Output will be shown

**Output:**

[](https://www.guru99.com/images/MongoDB/112115_0607_Introductio13.png)

The output clearly shows that the \_id field we defined while creating the collection is now used as the primary key for the collection.

Purpose of Building MongoDB

It may be a very genuine question that - "what was the need of MongoDB although there were many databases in action?"

**There is a simple answer:**

All the modern applications require big data, fast features development, flexible deployment, and the older database systems not competent enough, so the MongoDB was needed.

**The primary purpose of building MongoDB is:**

* Scalability
* Performance
* High Availability
* Scaling from single server deployments to large, complex multi-site architectures.
* Key points of MongoDB
* Develop Faster
* Deploy Easier
* Scale Bigger

First of all, we should know what is document oriented database?

Example of Document-Oriented Database

MongoDB is a document-oriented database. It is a key feature of MongoDB. It offers a document-oriented storage. It is very simple you can program it easily.

MongoDB stores data as documents, so it is known as document-oriented database.

1. FirstName = "John",
2. Address = "Detroit",
3. Spouse = [{**Name**: "Angela"}].
4. FirstName ="John",
5. Address = "Wick"

**There are two different documents (separated by ".").**

Storing data in this manner is called as document-oriented database.

Mongo DB falls into a class of databases that calls Document Oriented Databases. There is also a broad category of database known as [No SQL Databases](https://www.javatpoint.com/nosql-databases).

Features of MongoDB

These are some important features of MongoDB:

**1. Support ad hoc queries**

In MongoDB, you can search by field, range query and it also supports regular expression searches.

**2. Indexing**

You can index any field in a document.

**3. Replication**

MongoDB supports Master Slave replication.

A master can perform Reads and Writes and a Slave copies data from the master and can only be used for reads or back up (not writes)

**4. Duplication of data**

MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.

**5. Load balancing**

It has an automatic load balancing configuration because of data placed in shards.

**6. Supports map reduce and aggregation tools**.

**7. Uses**[**JavaScript**](https://www.javatpoint.com/javascript-tutorial)**instead of Procedures**.

**8. It is a schema-less database written in**[**C++**](https://www.javatpoint.com/cpp-tutorial).

**9. Provides high performance**.

**10. Stores files of any size easily without complicating your stack**.

**11. Easy to administer in the case of failures**.

**12. It also supports:**

* JSON data model with dynamic schemas
* Auto-sharding for horizontal scalability
* Built in replication for high availability
* Now a day many companies using MongoDB to create new types of applications, improve performance and availability.

# **MongoDB Data Types**

**Summary**: in this tutorial, you will learn about the most commonly used MongoDB data types.

Null

The null type is used to represent a null and a field that does not exist. For example:

{

"isbn": null

}

Code language: JSON / JSON with Comments (json)

Boolean

The boolean type has two values true and false. For example:

{

"best\_seller": true

}

Code language: JSON / JSON with Comments (json)

Number

By default, the mongo shell uses the 64-bit floating-point numbers. For example:

{

"price": 9.95,

"pages": 851

}

Code language: JSON / JSON with Comments (json)

The NumberInt and NumberLong classes represent 4-byte and 8-byte integers respectively. For example:

{

"year": NumberInt("2020"),

"words": NumberLong("95403")

}

Code language: JSON / JSON with Comments (json)

String

The string type represents any string of UTF-8 characters. For example:

{

"title": "MongDB Data Types"

}

Code language: JSON / JSON with Comments (json)

Date

The date type stores dates as 64-bit integers that represents milliseconds since the Unix epoch (January 1, 1970). It does not store the time zone. For example:

{

"updated\_at": new Date()

}

Code language: JSON / JSON with Comments (json)

In JavaScript, the Date class is used to represent the date type in MongoDB.

Note that you should always call the new Date(), not just Date() when you create a new Date object because the Date() returns a string representation of the date, not the Date object.

The mongo shell displays dates using local time zone settings. However, MongoDB does not store date with the time zone. To store the time zone, you can use another key e.g., timezone.

Regular Expression

MongoDB allows you to store [JavaScript regular expressions](https://www.javascripttutorial.net/javascript-regex/). For example:

{

"pattern": /\d+/

}

Code language: JSON / JSON with Comments (json)

In this example, /\d+/ is a regular expression that matches one or more digits.

Array

The array type allows you to store a list of values of any type. The values do not have to be in the same type, for example:

{

"title": "MongoDB Array",

"reviews": ["John", 3.5, "Jane", 5]

}

Code language: JSON / JSON with Comments (json)

The good thing about arrays in the document is that MongoDB understands their structures and allows you to carry operations on their elements.

For example, you can query all documents where 5 is an element of the reviews array. Also, you can create an index on the reviews array to improve the query performance.

Embeded Document

A value of a document can be another document that is often referred to as an embedded document.

The following example shows a book document that contains the author document as an embedded document:

{

"title": "MongoDB Tutorial",

"pages": 945,

"author": {

"first\_name": "John",

"last\_name": "Doe"

}

}

Code language: JSON / JSON with Comments (json)

In this example, the author document has its own key/value pairs including first\_name and last\_name.

Object ID

In MongoDB, every document has an "\_id" key. The value of the "\_id" key can be any type. However, it defaults to an ObjectId.

The value of the "\_id" key must be unique within a collection so that MongoDB can identify every document in the collection.

The ObjectId class is the default type for "\_id". It is used to generate unique values globally across servers. Since MongoDB is designed to be distributed, it is important to ensure the identifiers are unique in the shared environment.

The ObjectId uses 12 bytes for storage, where each byte represents 2 hexadecimal digits. In total, an ObjectId is 24 hexadecimal digits.

The 12-byte ObjectId value consists of:

* A 4-byte timestamp value that represents the ObjectId‘s generated time measured in seconds since the Unix epoch.
* A 5-byte random value.
* A 3-byte increment counter, initialized to a random value.

These first 9 bytes of an ObjectId guarantee its uniqueness across servers and processes for a single second. The last 3 bytes guarantee uniqueness within a second in a single process.

As a result, these 12-bytes allow for up to 2563 (16,777,216) unique ObjectIds values to be generated per process in a single second.

When you insert a document without specifying a value for the "\_id" key, MongoDB automatically generates a unique id for the document. For example:

db.books.insertOne({

"title": "MongoDB Basics"

});

Code language: JavaScript (javascript)

Output:

{

"acknowledged" : true,

"insertedId" : ObjectId("5f2fcae09b58c38603442a4f")

}

Code language: JSON / JSON with Comments (json)

MongoDB generated the id with the value ObjectId("5f2fcae09b58c38603442a4f"). You can view the inserted document like this:

db.books.find().pretty()

Code language: CSS (css)

Output:

{

"\_id" : ObjectId("5f2fcae09b58c38603442a4f"),

"title" : "MongoDB Basics"

}

Code language: JSON / JSON with Comments (json)

In this tutorial, you have learned the most commonly used MongoDB data types including null, number, string, array, regular expression, date, and ObjectId.

**Naming Restrictions for Database in MongoDB:**

Now we understand the naming restriction for the database:

1. The names of the database are case insensitive but remember the names cannot differ only by the case of the character.
2. For Windows, the database name should not contain any of the below characters in it: / \. “$\*:|?
3. For Unix and Linux, the database name should not contain any of the below characters in it: / \. “$
4. For both windows and Unix and Linux, the database name should not contain null characters.
5. The name of the database would not be empty and the names should be less than 64 characters.

**Naming Restriction on Creating a Collection in MongoDB:**

Following are the naming restrictions for collections:

1. The collection name should start with a character or an underscore.
2. The name should not contain $
3. The name should not be an empty string (””)
4. The name should not contain null characters.
5. The name should not begin with system.prefix
6. The maximum length of the name is 120 bytes including the database name, dot separator, and collection name.

Any 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.

Advantages of MongoDB over RDBMS

* **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.
* Structure of a single object is clear.
* No complex joins.
* Deep query-ability. MongoDB supports dynamic queries on documents using a document-based query language that's nearly as powerful as SQL.
* Tuning.
* **Ease of scale-out** − MongoDB is easy to scale.
* Conversion/mapping of application objects to database objects not needed.
* Uses internal memory for storing the (windowed) working set, enabling faster access of data.

**Why Use MongoDB?**

* **Document Oriented Storage** − Data is stored in the form of JSON style documents.
* Index on any attribute
* Replication and high availability
* Auto-Sharding
* Rich queries
* Fast in-place updates
* Professional support by MongoDB

**Where to Use MongoDB?**

* Big Data
* Content Management and Delivery
* Mobile and Social Infrastructure
* User Data Management
* Data Hub

# MongoDB - Data Modelling

Data in MongoDB has a flexible schema.documents in the same collection. They do not need to have the same set of fields or structure Common fields in a collection’s documents may hold different types of data.

Data Model Design

MongoDB provides two types of data models: — Embedded data model and Normalized data model. Based on the requirement, you can use either of the models while preparing your document.

Embedded Data Model

In this model, you can have (embed) all the related data in a single document, it is also known as de-normalized data model.

For example, assume we are getting the details of employees in three different documents namely, Personal\_details, Contact and, Address, you can embed all the three documents in a single one as shown below −

{

\_id: ,

Emp\_ID: "10025AE336"

Personal\_details:{

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26"

},

Contact: {

e-mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

},

Address: {

city: "Hyderabad",

Area: "Madapur",

State: "Telangana"

}

}

Normalized Data Model

In this model, you can refer the sub documents in the original document, using references. For example, you can re-write the above document in the normalized model as:

**Employee:**

{

\_id: <ObjectId101>,

Emp\_ID: "10025AE336"

}

**Personal\_details:**

{

\_id: <ObjectId102>,

empDocID: " ObjectId101",

First\_Name: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26"

}

**Contact:**

{

\_id: <ObjectId103>,

empDocID: " ObjectId101",

e-mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

}

**Address:**

{

\_id: <ObjectId104>,

empDocID: " ObjectId101",

city: "Hyderabad",

Area: "Madapur",

State: "Telangana"

}

Considerations while designing Schema in MongoDB

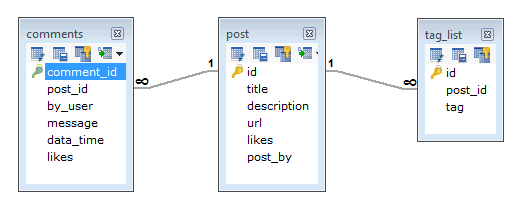
* Design your schema according to user requirements.
* Combine objects into one document if you will use them together. Otherwise separate them (but make sure there should not be need of joins).
* Duplicate the data (but limited) because disk space is cheap as compare to compute time.
* Do joins while write, not on read.
* Optimize your schema for most frequent use cases.
* Do complex aggregation in the schema.

Example

Suppose a client needs a database design for his blog/website and see the differences between RDBMS and MongoDB schema design. Website has the following requirements.

* Every post has the unique title, description and url.
* Every post can have one or more tags.
* Every post has the name of its publisher and total number of likes.
* Every post has comments given by users along with their name, message, data-time and likes.
* On each post, there can be zero or more comments.

In RDBMS schema, design for above requirements will have minimum three tables.



While in MongoDB schema, design will have one collection post and the following structure −

{

\_id: POST\_ID

title: TITLE\_OF\_POST,

description: POST\_DESCRIPTION,

by: POST\_BY,

url: URL\_OF\_POST,

tags: [TAG1, TAG2, TAG3],

likes: TOTAL\_LIKES,

comments: [

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

},

{

user:'COMMENT\_BY',

message: TEXT,

dateCreated: DATE\_TIME,

like: LIKES

}

]

}

So while showing the data, in RDBMS you need to

The createCollection() Method

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.

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

Options parameter is optional, so you need to specify only the name of the collection. 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. |

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

Examples

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

>use test

switched to db test

>db.createCollection("mycollection")

{ "ok" : 1 }

>

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

>show collections

mycollection

system.indexes

The following example shows the syntax of **createCollection()** method with few important options −

> db.createCollection("mycol", { capped : true, autoIndexID : true, size : 6142800, max : 10000 } ){

"ok" : 0,

"errmsg" : "BSON field 'create.autoIndexID' is an unknown field.",

"code" : 40415,

"codeName" : "Location40415"

}

>

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

>db.tutorialspoint.insert({"name" : "tutorialspoint"}),

WriteResult({ "nInserted" : 1 })

>show collections

mycol

mycollection

system.indexes

tutorialspoint

db.mycollection.drop()

show collections

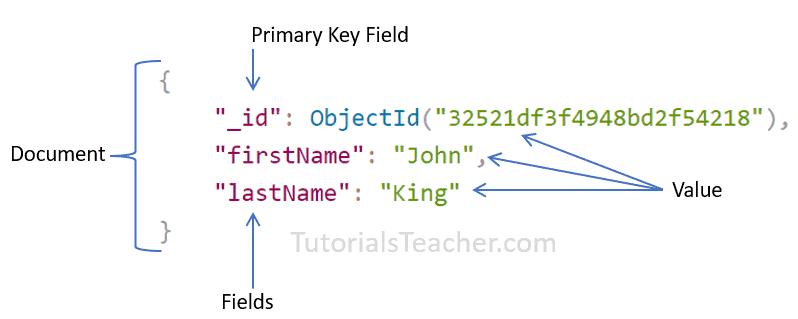
MongoDB supports many datatypes. Some of them are −

* **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 Documents: Document, Array, Embedded Document

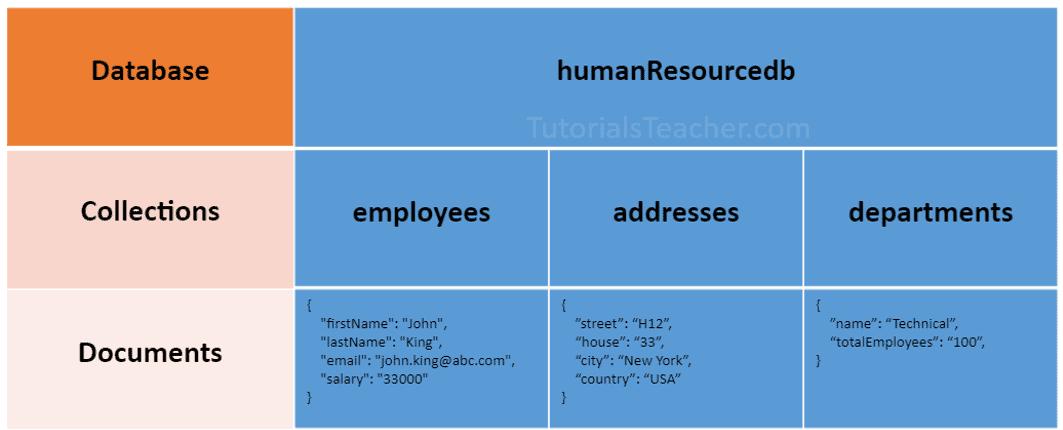
In the RDBMS database, a table can have multiple rows and columns. Similarly in MongoDB, a collection can have multiple documents which are equivalent to the rows. Each document has multiple "fields" which are equivalent to the columns. So in simple terms, each MongoDB document is a record and a collection is a table that can store multiple documents.

The following is an example of JSON based document.



In the above example, a document is contained within the curly braces. It contains multiple fields in "field":"value" format. Above, "\_id", "firstName", and "lastName" are field names with their respective values after a colon :. Fields are separated by a comma. A single collection can have multiple such documents separated by a comma.

The following chart to understand the relation between database, collections, and documents.



The following is an example of a document that contains an array and an embedded document.

Example: MongoDB Document

Copy

{

"\_id": ObjectId("32521df3f4948bd2f54218"),

"firstName": "John",

"lastName": "King",

"email": "john.king@abc.com",

"salary": "33000",

"DoB": new Date('Mar 24, 2011'),

"skills": [ "Angular", "React", "MongoDB" ],

"address": {

"street":"Upper Street",

"house":"No 1",

"city":"New York",

"country":"USA"

}

}

MongoDB document stores data in JSON format. In the above document, "firstName", "lastName", "email", and "salary" are the fields (like columns of a table in RDBMS) with their corresponding values (e.g value of a column in a row). Consider "\_id" field as a primary key field that stores a unique [ObjectId](https://docs.mongodb.com/manual/reference/bson-types/" \l "std-label-objectid" \t "_blank)."skills" is an array and "address" holds another JSON document.

The field names can be specified without surrounding quotation marks, as shown below.

Example: MongoDB Document

Copy

{

\_id: ObjectId("32521df3f4948bd2f54218"),

firstName: "John",

lastName: "King",

email: "john.king@abc.com",

salary: "33000",

DoB: new Date('Mar 24, 2011'),

skills: [ "Angular", "React", "MongoDB" ],

address: {

street:"Upper Street",

house:"No 1",

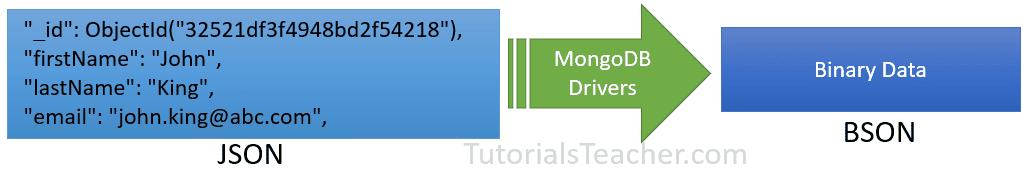
city:"New York",

country:"USA"

}

}

MongoDB stores data in key-value pairs as a BSON document. BSON is a binary representation of a JSON document that supports more data types than JSON. MongoDB drivers convert JSON document to BSON data.



Important Points:

* MongoDB reserves \_id name for use as a unique primary key field that holds ObjectId type. However, you are free to give any name you like with any data type other than the array.
* A document field name cannot be null but the value can be.
* Most MongoDB documents cannot have duplicate field names. However, it depends on the driver you use to store a document in your application.
* A document fields can be without quotation marks " " if it does not contain spaces, e.g. { name: "Steve"}, { "first name": "Steve"} are valid fields.
* Use the dot notation to access array elements or embedded documents.
* MongoDB supports maximum document size of 16mb. Use [GridFS](https://docs.mongodb.com/manual/core/gridfs/" \t "_blank) to store more than 16 MB document.
* Fields in a BSON document are ordered. It means fields order is important while comparing two documents, e.g. {x: 1, y: 2} is not equal to {y: 2, x: 1}
* MogoDB keeps the order of the fields except \_id field which is always the first field.
* MongoDB collection can store documents with different fields. It does not enforce any schema.

Embedded Documents:

A document in MongoDB can have fields that hold another document. It is also called nested documents.

The following is an embedded document where the department and address field contains another document.

Example: Embedded Document

Copy

{

\_id: ObjectId("32521df3f4948bd2f54218"),

firstName: "John",

lastName: "King",

department: {

\_id: ObjectId("55214df3f4948bd2f8753"),

name:"Finance"

},

address: {

phone: { type: "Home", number: "111-000-000" }

}

}

In the above embedded document, notice that the address field contains the phone field which holds a second level document.

* An embedded document can contain upto 100 levels of nesting.
* Supports a maximum size of 16 mb.
* Embedded documents can be accessed using dot notation embedded-document.fieldname, e.g. access phone number using address.phone.number.

Array

A field in a document can hold array. Arrays can hold any type of data or embedded documents.

Array elements in a document can be accessed using dot notation with the zero-based index position and enclose in quotes.

Example: MongoDB Document with an Array

Copy

{

\_id: ObjectId("32521df3f4948bd2f54218"),

firstName: "John",

lastName: "King",

email: "john.king@abc.com",

skills: [ "Angular", "React", "MongoDB" ],

}

The above document contains the skills field that holds an array of strings. To specify or access the second element in the skills array, use skills.1.

**MongoDB - Insert Document**

The insert() Method

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.users.insert({

... \_id : ObjectId("507f191e810c19729de860ea"),

... title: "MongoDB Overview",

... description: "MongoDB is no sql database",

... by: "tutorials point",

... url: "http://www.tutorialspoint.com",

... tags: ['mongodb', 'database', 'NoSQL'],

... likes: 100

... })

WriteResult({ "nInserted" : 1 })

>

Here **mycol** is our collection name, as created in the previous chapter. 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)

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

> db.createCollection("post")

> db.post.insert([

{

title: "MongoDB Overview",

description: "MongoDB is no SQL database",

by: "tutorials point",

url: "http://www.tutorialspoint.com",

tags: ["mongodb", "database", "NoSQL"],

likes: 100

},

{

title: "NoSQL Database",

description: "NoSQL database doesn't have tables",

by: "tutorials point",

url: "http://www.tutorialspoint.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" : [ ]

})

>

To insert the document you can use **db.post.save(document)** also. If you don't specify **\_id** in the document then **save()** method will work same as **insert()** method. If you specify \_id then it will replace whole data of document containing \_id as specified in save() method.

The insertOne() method

If you need to insert only one document into a collection you can use this method.

Syntax

The basic syntax of insert() command is as follows −

>db.COLLECTION\_NAME.insertOne(document)

Example

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: "Radhika",

Last\_Name: "Sharma",

Date\_Of\_Birth: "1995-09-26",

e\_mail: "radhika\_sharma.123@gmail.com",

phone: "9848022338"

})

{

"acknowledged" : true,

"insertedId" : ObjectId("5dd62b4070fb13eec3963bea")

}

>

[Explore our](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)**[latest online courses](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)**[and learn new skills at your own pace. Enroll and become a certified expert to boost your career.](https://www.tutorialspoint.com/latest/courses?utm_source=tutorialspoint&utm_medium=tutorials_3p&utm_campaign=internal" \t "_blank)

The insertMany() method

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

Example

Following example inserts three different documents into the empDetails collection using the insertMany() method.

> 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("5dd631f270fb13eec3963bed"),

ObjectId("5dd631f270fb13eec3963bee"),

ObjectId("5dd631f270fb13eec3963bef")

]

}

The pretty() Method

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

Syntax

>db.COLLECTION\_NAME.find().pretty()