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.

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.

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.

## 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** | **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 |

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 Collections

A collection in MongoDB is similar to a table in RDBMS. MongoDB collections do not enforce schemas. Each MongoDB collection can have multiple documents. A document is equilant to row in a table in RDBMS.

To create a collection, use the db.createCollection() command.

Mydb>db.createCollection(“student”);

Above, the student collection is created using the creatCollection() method. It returns an object { ok: 1 }, which indicates the collection was created successfully.

Use the show collections commands to list all the collections in a database.

Mydb>show collections

To delete a collection, use the db.<collection-name>.drop() method.

db.student.drop();

# MongoDB Documents: 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.

MongoDB 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.

{

\_id

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

Example: MongoDB Document

{

"\_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/#std-label-objectid). "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/) 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}
* MongoDB 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.

# MongoDB - Insert Single Document in a Collection using insertOne()

In MongoDB, a collection represents a table in RDBMS and a document is like a record in a table. Learn how to insert a single document into a collection.

MongoDB provides the following methods to insert documents into a collection:

1. insertOne() - Inserts a single document into a collection.
2. insert() - Inserts one or more documents into a collection.
3. insertMany() - Insert multiple documents into a collection.

## insertOne()

Use the db.<collection>.insertOne() method to insert a single document in a collection. db points to the current database, <collection> is existing or a new collection name.

#### **Syntax:**

db.collection.insertOne(document, [writeConcern])

#### **Parameters:**

1. document: A document to insert into the collection.
2. writeConcern: Optional. A document expressing the write concern to override the default write concern.

The following inserts a document into employees collection.

Example: insertOne()

db.employees.insertOne({

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

})

Output

{

acknowledged: true,

insertedId: ObjectId("616d44bea861820797edd9b0")

}

In the above example, we passed a document to the insertOne() method. Notice that we haven't specified \_id field. So, MongoDB inserts a document to a collection with the auto-generated unique \_id field. It returns an object with a boolean field acknowledged that indicates whether the insert operation is successful or not, and insertedId field with the newly inserted \_id value.

The following shows the insert operation in the mongosh shell:

InsertOne() in mongosh Shell

Use the find() to list all data of a collection, and the pretty() method to format resulted data.

db.employees.find().pretty()

Output

{

\_id: ObjectId("616d44bea861820797edd9b0"),

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

}

MongoDB is NoSQL database. So, it does not enforce schema to any collection. It means you can insert a document with any fields to a collection. For example, the following inserts a document with different fields to the employees collection.

Example: Insert a Document

db.employees.insertOne({

fName: "John",

lName: "King",

emailid: "john.king@abc.com"

})

Output

{

acknowledged: true,

insertedId: ObjectId("546d44bea861820797ed214")

}

It is recommended to keep the field names same in all the documents of a collection to manage them easily.

## Insert \_id Manually

It is not necessary to insert auto-generated \_id value. You can manually specify a unique value for the \_id field, as shown below.

db.employees.insertOne({

\_id:"1",

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

})

Output

{

acknowledged: true,

insertedId: 1

}

Note that while adding your custom value to \_id field, a value must be unique; otherwise, it will throw an error. The following tries to add the same \_id value.

lName: "King",

emailid: "john.king@abc.com"

})

Output

{

acknowledged: true,

insertedId: ObjectId("546d44bea861820797ed214")

}

It is recommended to keep the field names same in all the documents of a collection to manage them easily.

# MongoDB - Insert Multiple Documents in a Collection

In MongoDB, a collection represents a table in RDBMS and a document is like a record in a table. Learn how to insert one or more documents into a collection.

MongoDB provides the following methods to insert documents into a collection:

1. [insertOne()](https://www.tutorialsteacher.com/mongodb/insert-single-document) - Inserts a single document into a collection.
2. [insert()](https://www.tutorialsteacher.com/mongodb/insert-documents#insert) - Inserts one or more documents into a collection.
3. [insertMany()](https://www.tutorialsteacher.com/mongodb/insert-documents#insertmany) - Insert multiple documents into a collection.

## insert()

The db.<collection>.insert() method inserts one document or an array of documents into a collection.

#### **Syntax:**

db.collection.insert(

document or array of documents,

[writeConcern],

[ordered])

#### **Parameters:**

1. document or array of documents: A single document or array of documents to insert into the collection.
2. writeConcern: Optional. A document expressing the [write concern](https://docs.mongodb.com/manual/reference/write-concern/) to override the default write concern.
3. ordered: Optional. A boolean value indicating whether it's ordered or unordered insert operation. If true then performs ordered insert where if an error occurs with one of the documents, then MongoDB will return without processing the remaining documents in the array. If false, then process all the documents even if an error occurred. Defaults to true.

599.7K

Go Lang Quiz

The following inserts a single document. It's the same as the insertOne() method.

Example: insert()

db.employees.insert({

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

})

Output

{

acknowledged: true,

insertedIds: { '0': ObjectId("616d62d9a861820797edd9b2") }

}

The following inserts multiple documents into a collection by passing an array of documents.

Example: Insert a Document

 Copy

db.employees.insert(

[

{

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

},

{

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com"

},

{

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com"

}

])

Output

{

acknowledged: true,

insertedIds: {

'0': ObjectId("616d63eda861820797edd9b3"),

'1': ObjectId("616d63eda861820797edd9b4"),

'2': ObjectId("616d63eda861820797edd9b5")

}

}

You can specify a different \_id field value into one or more documents, as shown below.

Example: Insert a Document

 Copy

db.employees.insert(

[

{

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

},

{

\_id:1,

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com"

},

{

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com"

},

])

Output

{

acknowledged: true,

insertedIds: {

'0': ObjectId("616d63eda861820797edd9b3"),

'1': 1,

'2': ObjectId("616d63eda861820797edd9b5")

}

}

By default, the insert() method performs ordered inserts. So, if an error occurred in any of the documents, then it won't process the remaining documents. For example, the following tries to insert a second document with an existing \_id value.

Example: Insert a Document

db.employees.insert(

[

{

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com"

},

{

\_id:1,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com"

},

{

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com"

},

])

Output

Uncaught:

MongoBulkWriteError: E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }

Result: BulkWriteResult {

result: {

ok: 1,

writeErrors: [

WriteError {

err: {

index: 1,

code: 11000,

errmsg: 'E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }',

errInfo: undefined,

op: {

\_id: 1,

firstName: 'Kapil',

lastName: 'D',

email: 'kapil.d@abc.com'

}

}

}

],

writeConcernErrors: [],

insertedIds: [

{ index: 0, \_id: ObjectId("616e6b7e3fa8bd4420d49371") },

{ index: 1, \_id: 1 },

{ index: 2, \_id: ObjectId("616e6b7e3fa8bd4420d49372") }

],

**nInserted: 1,**

nUpserted: 0,

nMatched: 0,

nModified: 0,

nRemoved: 0,

upserted: []

}

}

In the above output, it encounters an error while inserting the second document. So, it only inserts the first document. nInserted indicates the number of documents inserted successfully.

Specify the {ordered:false} to perform unorder insert which will insert all the valid documents even if an error occurs.

Example: Insert a Document

 Copy

db.employees.insert(

[

{

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com"

},

{

\_id:1,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com"

},

{

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com"

},

],

{ ordered: false}

)

Output

Uncaught:

MongoBulkWriteError: E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }

Result: BulkWriteResult {

result: {

ok: 1,

writeErrors: [

WriteError {

err: {

index: 1,

code: 11000,

errmsg: 'E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }',

errInfo: undefined,

op: {

\_id: 1,

firstName: 'Kapil',

lastName: 'D',

email: 'kapil.d@abc.com'

}

}

}

],

writeConcernErrors: [],

insertedIds: [

{ index: 0, \_id: ObjectId("616e6be33fa8bd4420d49373") },

{ index: 1, \_id: 1 },

{ index: 2, \_id: ObjectId("616e6be33fa8bd4420d49374") }

],

**nInserted: 2,**

nUpserted: 0,

nMatched: 0,

nModified: 0,

nRemoved: 0,

upserted: []

}

}

## insertMany()

The db.<collection>.insertMany() inserts multiple documents into a collection. It cannot insert a single document.

#### **Syntax:**

db.collection.insertMany(

[document1, document2, ....],

{

writeConcern: <document>,

ordered: <boolean>

}

)

The following adds multiple documents using the insertMany() method.

Example: Insert a Document

 Copy

db.employees.insertMany(

[

{

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

},

{

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com"

},

{

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com"

},

])

Output

{

acknowledged: true,

insertedIds: {

'0': ObjectId("616d63eda861820797edd9b3"),

'1': ObjectId("616d63eda861820797edd9b4"),

'2': ObjectId("616d63eda861820797edd9b5")

}

}

The following inserts custom \_id values.

db.employees.insertMany([

{

\_id:1,

firstName: "John",

lastName: "King",

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

salary: 5000

},

{

\_id:2,

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com",

salary: 8000

},

{

\_id:3,

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com",

salary: 7500

},

{

\_id:4,

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com",

salary: 9000

},

{

\_id:5,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com",

salary: 4500

},

{

\_id:6,

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com",

salary: 11000

}

])

Output

{

acknowledged: true,

insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6 }

}

Use the ordered parameter the same way as the insert() method to process all the documents even if an error occurred.

Note: If the collection used with these methods does not exist, then they create the specified collection. MongoDB is case-sensitive, so employees and Employees are considered two different collections.

# MongoDB - Insert Multiple Documents in a Collection

In MongoDB, a collection represents a table in RDBMS and a document is like a record in a table. Learn how to insert one or more documents into a collection.

MongoDB provides the following methods to insert documents into a collection:

1. insertOne() - Inserts a single document into a collection.
2. insert() - Inserts one or more documents into a collection.
3. insertMany() - Insert multiple documents into a collection.

## insert()

The db.<collection>.insert() method inserts one document or an array of documents into a collection.

#### **Syntax:**

db.collection.insert(

document or array of documents,

[writeConcern],

[ordered])

#### **Parameters:**

1. document or array of documents: A single document or array of documents to insert into the collection.
2. writeConcern: Optional. A document expressing the write concern to override the default write concern.
3. ordered: Optional. A boolean value indicating whether it's ordered or unordered insert operation. If true then performs ordered insert where if an error occurs with one of the documents, then MongoDB will return without processing the remaining documents in the array. If false, then process all the documents even if an error occurred. Defaults to true.

599.7K

Go Lang Quiz

The following inserts a single document. It's the same as the insertOne() method.

Example: insert()

db.employees.insert({

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

})

Output

{

acknowledged: true,

insertedIds: { '0': ObjectId("616d62d9a861820797edd9b2") }

}

The following inserts multiple documents into a collection by passing an array of documents.

Example: Insert a Document

 Copy

db.employees.insert(

[

{

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

},

{

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com"

},

{

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com"

}

])

Output

{

acknowledged: true,

insertedIds: {

'0': ObjectId("616d63eda861820797edd9b3"),

'1': ObjectId("616d63eda861820797edd9b4"),

'2': ObjectId("616d63eda861820797edd9b5")

}

}

You can specify a different \_id field value into one or more documents, as shown below.

Example: Insert a Document

db.employees.insert(

[

{

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

},

{

\_id:1,

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com"

},

{

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com"

},

])

Output

{

acknowledged: true,

insertedIds: {

'0': ObjectId("616d63eda861820797edd9b3"),

'1': 1,

'2': ObjectId("616d63eda861820797edd9b5")

}

}

By default, the insert() method performs ordered inserts. So, if an error occurred in any of the documents, then it won't process the remaining documents. For example, the following tries to insert a second document with an existing \_id value.

Example: Insert a Document

db.employees.insert(

[

{

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com"

},

{

\_id:1,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com"

},

{

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com"

},

])

Output

Uncaught:

MongoBulkWriteError: E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }

Result: BulkWriteResult {

result: {

ok: 1,

writeErrors: [

WriteError {

err: {

index: 1,

code: 11000,

errmsg: 'E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }',

errInfo: undefined,

op: {

\_id: 1,

firstName: 'Kapil',

lastName: 'D',

email: 'kapil.d@abc.com'

}

}

}

],

writeConcernErrors: [],

insertedIds: [

{ index: 0, \_id: ObjectId("616e6b7e3fa8bd4420d49371") },

{ index: 1, \_id: 1 },

{ index: 2, \_id: ObjectId("616e6b7e3fa8bd4420d49372") }

],

**nInserted: 1,**

nUpserted: 0,

nMatched: 0,

nModified: 0,

nRemoved: 0,

upserted: []

}

}

In the above output, it encounters an error while inserting the second document. So, it only inserts the first document. nInserted indicates the number of documents inserted successfully.

Specify the {ordered:false} to perform unorder insert which will insert all the valid documents even if an error occurs.

Example: Insert a Document

 Copy

db.employees.insert(

[

{

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com"

},

{

\_id:1,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com"

},

{

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com"

},

],

{ ordered: false}

)

Output

Uncaught:

MongoBulkWriteError: E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }

Result: BulkWriteResult {

result: {

ok: 1,

writeErrors: [

WriteError {

err: {

index: 1,

code: 11000,

errmsg: 'E11000 duplicate key error collection: humanResourceDB.employees index: \_id\_ dup key: { \_id: 1 }',

errInfo: undefined,

op: {

\_id: 1,

firstName: 'Kapil',

lastName: 'D',

email: 'kapil.d@abc.com'

}

}

}

],

writeConcernErrors: [],

insertedIds: [

{ index: 0, \_id: ObjectId("616e6be33fa8bd4420d49373") },

{ index: 1, \_id: 1 },

{ index: 2, \_id: ObjectId("616e6be33fa8bd4420d49374") }

],

**nInserted: 2,**

nUpserted: 0,

nMatched: 0,

nModified: 0,

nRemoved: 0,

upserted: []

}

}

## insertMany()

The db.<collection>.insertMany() inserts multiple documents into a collection. It cannot insert a single document.

#### **Syntax:**

db.collection.insertMany(

[document1, document2, ....],

{

writeConcern: <document>,

ordered: <boolean>

}

)

The following adds multiple documents using the insertMany() method.

Example: Insert a Document

db.employees.insertMany(

[

{

firstName: "John",

lastName: "King",

email: "john.king@abc.com"

},

{

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com"

},

{

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com"

},

])

Output

{

acknowledged: true,

insertedIds: {

'0': ObjectId("616d63eda861820797edd9b3"),

'1': ObjectId("616d63eda861820797edd9b4"),

'2': ObjectId("616d63eda861820797edd9b5")

}

}

The following inserts custom \_id values.

Example: insertMany() with Custom \_iddb.employees.insertMany([

{

\_id:1,

firstName: "John",

lastName: "King",

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

salary: 5000

},

{

\_id:2,

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com",

salary: 8000

},

{

\_id:3,

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com",

salary: 7500

},

{

\_id:4,

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com",

salary: 9000

},

{

\_id:5,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com",

salary: 4500

},

{

\_id:6,

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com",

salary: 11000

}

])

Output

{

acknowledged: true,

insertedIds: { '0': 1, '1': 2, '2': 3, '3': 4, '4': 5, '5': 6 }

}

Use the ordered parameter the same way as the insert() method to process all the documents even if an error occurred.

Note: If the collection used with these methods does not exist, then they create the specified collection. MongoDB is case-sensitive, so employees and Employees are considered two different collections.

# MongoDB – Comparison Query Operators

MongoDB uses various comparison query operators to compare the values of the documents. The following table contains the comparison query operators:

|  |  |
| --- | --- |
| **Operators** | **Description** |
| **$eq** | It is used to match the values of the fields that are equal to a specified value. |
| **$ne** | It is used to match all values of the field that are not equal to a specified value. |
| **$gt** | It is used to match values of the fields that are greater than a specified value. |
| **$gte** | It is used to match values of the fields that are greater than equal to the specified value. |
| **$lt** | It is used to match values of the fields that are less than a specified valueo |
| **$lte** | It is used to match values of the fields that are less than equals to the specified value |
| **$in** | It is used to match any of the values specified in an array. |
| **$nin** | It is used to match none of the values specified in an array. |

## Matching values using $nin operator:

In this example, we are retrieving only those employee’s documents whose name is not Amit or Suman.

db.contributor.find({name: {$nin: ["Amit", "Suman"]}}).pretty()

## Matching values using $in operator:

In this example, we are retrieving only those employee’s documents whose name is either Amit or Suman.

db.contributor.find({name: {$in: ["Amit", "Suman"]}}).pretty()

## Matching values using $lt operator:

In this example, we are selecting those documents where the value of the salary field is less than 2000.

db.contributor.find({salary: {$lt: 2000}}).pretty()

## Matching values using $eq operator:

In this example, we are selecting those documents where the value of the branch field is equal to CSE.

db.contributor.find({branch: {$eq: "CSE"}}).pretty()

## Matching values using $ne operator:

In this example, we are selecting those documents where the value of the branch field is not equal to CSE.

db.contributor.find({branch: {$ne: "CSE"}}).pretty()

## Matching values using $gt operator:

In this example, we are selecting those documents where the value of the salary field is greater than 1000.

d.contributor.find({salary: {$gt: 1000}}).pretty()

## Matching values using $gte operator:

In this example, we are selecting those documents where the value of the joiningYear field is greater than equals to 2017.

db.contributor.find({joiningYear: {$gte: 2017}})

## Matching values using $lte operator:

In this example, we are selecting those documents where the value of the salary field is less than equals to 1000.

db.contributor.find({salary: {$lte: 1000}}).pretty()

# Update Single Document using updateOne() in MongoDB

Learn how to update a single document using the updateOne() method in MongoDB.

MongoDB provides the following methods to update existing documents in a collection:

* db.collection.updateOne() - Modifies a single document in a collection.
* db.collection.updateMany() - Modifies one or more documents in a collection.

## db.collection.updateOne()

Use the db.<collection>.updateOne() method to update a single document in a collection that matches with the specified filter criteria. It updates the first matching document even if multiple documents match with the criteria.

#### **Syntax:**

db.collection.updateOne(filter, document, options)

#### **Parameters:**

1. filter: The selection criteria for the update, same as find() method.
2. document: A document or pipeline that contains modifications to apply.
3. options: Optional. May contains options for update behaviour. It includes upsert, writeConcern, collation, etc.

## Update a Single Field

The following updates a single field in a single document in employees collection.

db.employees.updateOne({\_id:1}, { $set: {firstName:'Morgan'}})

Use the $inc update operator to increase the value of the field by the specified amount.

db.employees.updateOne({firstName:"Steve"}, { $inc: {salary: 500}})

## Update Multiple Fields

You can also specify multiple fields to update. The following updates email and lastName fields.

db.employees.updateOne({\_id:2}, { $set: {lastName:"Tendulkar", email:"sachin.tendulkar@abc.com"}})

## Upsert - Add if not Exist

Specify {upsert:true} as a third parameter in the UpdateOne() method. The upsert:true adds a new document if the matching document does not found.

db.employees.updateOne({firstName:"Heer"}, { $set: {lastName:"Patel", salary:2000}},{upsert:true})

## Unset – remove field from collection.

db.employees.updateOne({

\_id: 1

}, {

$unset: {

salary: ""

}

})

# MongoDB: Delete Documents in a Collection

MongoDB provides the following methods to delete one or more documents in a collection.

* db.collection.deleteOne() - Deletes the first matching document in a collection.
* db.collection.deleteMany() - Deletes all the matching documents in a collection.
* db.collection.remove() - Deletes the first matching document in a collection.

## db.collection.deleteOne()

Use the db.<collection>.deleteOne() method to delete the first documents that match with the specified filter criteria in a collection.

#### **Syntax:**

db.collection.deleteOne(filter, options)

#### **Parameters:**

1. filter: The selection criteria for the update, same as find() method.
2. options: Optional. May contains options for update behavior. It includes writeConcern, collation, and hint parameters.

In the above syntax, db points to the current database, <collection> points is an existing collection name.

To demonstrate the delete operation, insert the following sample documents in the employees collection.

Sample Data

db.employees.insertMany([

{

\_id:1,

firstName: "John",

lastName: "King",

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

salary: 5000

},

{

\_id:2,

firstName: "Sachin",

lastName: "T",

email: "sachin.t@abc.com",

salary: 8000

},

{

\_id:3,

firstName: "James",

lastName: "Bond",

email: "jamesb@abc.com",

salary: 7500

},

{

\_id:4,

firstName: "Steve",

lastName: "J",

email: "steve.j@abc.com",

salary: 7000

},

{

\_id:5,

firstName: "Kapil",

lastName: "D",

email: "kapil.d@abc.com",

salary: 4500

},

{

\_id:6,

firstName: "Amitabh",

lastName: "B",

email: "amitabh.b@abc.com",

salary: 7000

}

])

The following deletes a document from the employees collection.

Example: Delete Single Document using deleteOne()

 Copy

db.employees.deleteOne({ salary:7000 })