



mongoDB®

INTRODUCTION

What is MongoDB?

MongoDB is a source-available, cross-platform, document-oriented database program.

Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas.

What is database?

- A database is an organized collection of data stored in a computer system and usually controlled by a database management system (DBMS).
- The data in common databases is modeled in tables, making querying and processing efficient.

Structured Data:

- Structured data refers to data that is organized and formatted in a specific way to make it easily readable and understandable by both humans and machines.

- This is typically achieved through the use of a well-defined schema or data model, which provides a structure for the data.

Database Management System:

- A database management system (DBMS) is system software for creating and managing databases.
- A DBMS makes it possible for end users to create, protect, read, update and delete data in a database.

SET UP:

https://www.geeksforgeeks.org/how-to-install-mongodb-on-windows/?ref=ml_lbp

FEW COMMANDS TO TEST AFTER CONNECTIONS

show dbs:

It will display a list of all the available databases.

use db:

To create a database.

show collections:

It will show all the collections in the currently selected database.

db.collection.insert():

Inserts a document or documents into a collection.

db.collection.find():

Selects documents in a collection or view and returns a cursor to the selected documents.

Document:

A **document** is a fundamental unit of data storage. It's a record that contains field-and-value pairs, similar to a JSON object.

The representation of a document varies by programming language, but most languages have a data structure that is a natural fit, such as a map, hash, or dictionary.

{"greeting" : "DATA SCIENCE"}

Collections:

A collection is a grouping of MongoDB documents. Each document within a collection can have different fields. They are analogous to tables in relational databases.

Database:

MongoDB groups collections into databases.

A single instance of MongoDB can host several databases, each grouping together zero or more collections.

A database has its own permissions, and each database is stored in separate files on disk.

A good rule of thumb is to store all data for a single application in the same database.

Datatype:

Basically each document will be in JSON format which will be as follows. Where each attributes inside can be of multiple data types

```
{  
  "name: Nanditha Naveen",  
  "address": {  
    "street": "Jai Maruthi Nagar",  
    "city": "Chikmagalur",  
    "state": "Karnataka"  
  }  
}
```

WHERE, AND, OR & CRUD

WHERE

Given a Collection you want to FILTER a subset based on a condition. That is the place WHERE is used.

```
db.students.find({ gpa: {$lt: 2.5}});
```

OUTPUT

```
db> db.students.find({gpa:{$lt:4.5}});
[
  {
    _id: ObjectId('6645f14f8419dc976a376b76'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b77'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.77,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b78'),
    name: 'Student 316',
    age: 20,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 2.82,
    blood_group: 'B+',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b79'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b7a'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b7b'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: false
  }
]
```

OR

The \$or operator is used to specify a compound query with multiple conditions, where at least one condition must be satisfied for a document to match.

```
db.students.find({$or:[{home_city:"City 4"},{gpa:{$gt:3.0}}]});
```

OUTPUT

```
db> db.students.find({$or:[{home_city:"City 4"},{gpa:{$gt:3.0}}]});
[
  {
    _id: ObjectId('6645f14f8419dc976a376b76'),
    name: 'Student 948',
    age: 19,
    courses: "['English', 'Computer Science', 'Physics', 'Mathematics']",
    gpa: 3.44,
    home_city: 'City 2',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b77'),
    name: 'Student 157',
    age: 20,
    courses: "['Physics', 'English']",
    gpa: 2.77,
    home_city: 'City 4',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b79'),
    name: 'Student 346',
    age: 25,
    courses: "['Mathematics', 'History', 'English']",
    gpa: 3.31,
    home_city: 'City 8',
    blood_group: 'O-',
    _id: ObjectId('6645f14f8419dc976a376b7a'),
    name: 'Student 930',
    age: 25,
    courses: "['English', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.63,
    home_city: 'City 3',
    blood_group: 'A-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b7b'),
    name: 'Student 305',
    age: 24,
    courses: "['History', 'Physics', 'Computer Science', 'Mathematics']",
    gpa: 3.4,
    home_city: 'City 6',
    blood_group: 'O+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b80'),
    name: 'Student 256',
    age: 19,
    courses: "['Computer Science', 'Mathematics', 'History', 'English']",
    gpa: 3.44,
    home_city: 'City 1',
    blood_group: 'B+',
  }
]
```


AND

The `$and` operator allows you to specify multiple conditions that documents must satisfy to match the query.

```
db.students.find({ $and: [{ is_hotel_resident:true}, { home_city:"City 5"}]});
```

OUTPUT

```
db> db.students.find({$and:[{is_hotel_resident:true},{home_city:"City 5"}]});
[
  {
    _id: ObjectId('6645f14f8419dc976a376bb4'),
    name: 'Student 219',
    age: 18,
    courses: "['Physics', 'Computer Science', 'Mathematics', 'History']",
    gpa: 3.4,
    home_city: 'City 5',
    blood_group: 'B-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376bb9'),
    name: 'Student 632',
    age: 23,
    courses: "['Physics', 'English', 'Computer Science']",
    gpa: 3.76,
    home_city: 'City 5',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376bcf'),
    name: 'Student 989',
    age: 24,
    courses: "['Computer Science', 'Mathematics']",
    gpa: 3.32,
```

```

  {
    _id: ObjectId('6645f14f8419dc976a376bda'),
    name: 'Student 880',
    age: 24,
    courses: "['History', 'Mathematics', 'English', 'Computer Science']",
    gpa: 3.12,
    home_city: 'City 5',
    blood_group: 'O-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376bf2'),
    name: 'Student 928',
    age: 21,
    courses: "['Computer Science', 'Physics', 'English', 'History']",
    gpa: 2.66,
    home_city: 'City 5',
    blood_group: 'AB-',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('6645f14f8419dc976a376c4f'),
    name: 'Student 578',
    age: 20,
    courses: "['History', 'English', 'Physics', 'Mathematics']",
    gpa: 3.48,
    home_city: 'City 5',
    blood_group: 'AB+',
    is_hotel_resident: true
  },
]
```


CRUD

- C - Create / Insert
- R - Remove
- U - update
- D - Delete

This is applicable for a Collection (Table) or a Document (Row)

INSERT

- The **insert()** method in MongoDB inserts documents in the MongoDB collection. This method is also used to create a new **collection** by inserting documents.
- You can insert documents with or without the **_id field**. If you insert a document in the collection without the **_id** field, then MongoDB will automatically add an **_id** field and assign it with a unique ObjectId.

And if you insert a document with the **_id** field, then the value of the **_id** field must be unique to avoid the duplicate key error.

insertOne()

- **insertOne()** method inserts a document into the collection. This method inserts only one document at a time. This method can also be used inside multi-document transactions.

```
const studentData = {  
  "name": "John Doe",  
  "age": 24,  
  "courses": ['Physics', 'Computer Science', 'Mathematics', 'History'],  
  "gpa": 4.5,  
  "home_city": "New York",  
}
```

```
    "blood_group": "O+",  
    "is_hotel_resident": true  
  }:  

```

UPDATE

- The **update()** method in MongoDB updates a document or multiple documents in the collection. When the document is updated the **_id field** remains unchanged.
- This method can be used for a single updating of documents as well as multiple documents. By default, the **db.collection.update() method** updates a single document.

UpdateOne()

- The **updateOne()** method in MongoDB updates the first matched document within the collection based on the given query.
- The value of the **_id field** remains unchanged after updating the value. This method updates one document at a time and can also add new fields to the given document.

```
db.students.updateOne({ name: 'John Doe'},  
{ $set: { gpa: 4.2 }});
```

OUTPUT:

```
db> db.students.updateOne({ name: 'John Doe' }, { $set: { gpa: 4.2 }});  
{  
  acknowledged: true,  
  insertedId: null,  
  matchedCount: 0,  
  modifiedCount: 0,  
  upsertedCount: 0  
}
```

UpdateMany()

- The **updateMany()** method updates all the documents in MongoDB collections that match the given query. When you update your document, the value of the `_id` field remains unchanged.
- You can also use this method inside multi-document transactions.

```
db.students.updateMany({ gpa: { $lt: 4.0 }},{ $inc:  
{ gpa: 0.5 }});
```

OUTPUT:

```
db> db.students.updateMany({ gpa: { $lt: 4.0 } }, { $inc: { gpa: 0.5 } });
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 246,
  modifiedCount: 246,
  upsertedCount: 0
}
```

DeleteMany()

- The deleteMany() method is a part of MongoDB's CRUD (Create, Read, Update, and Delete) operations. As the name suggests, it is used to delete more than one document that satisfies the specified criteria.
- deleteMany() returns acknowledged as true if the function runs with the writeConcern parameter; otherwise, false.

```
db.students.deleteMany({ is_hotel_resident: true });
```

OUTPUT:

```
db> db.students.deleteMany({ is_hotel_resident: true});  
{ acknowledged: true, deletedCount: 246 }  
db> |
```


PROJECTION

- MongoDB Projection is a special feature allowing you to select only the necessary data rather than selecting the whole set of data from the document.
- For Example, If a Document contains 10 fields and only 5 fields are to be shown the same can be achieved using the Projections. This enable us to:
- Project concise yet transparent data
- Filter data without impacting the overall database performance.

How Does MongoDB Projection Works?

- MongoDB projections are constructed on top of the existing find() query and therefore making it easier to use without significant modifications to the existing functions/queries.
- Moreover, projection plays a key factor when looking for user-centric data from a given large data set.

EXAMPLE 1:

```
db.students.find({}, {_id: 1, gpa: 1});
```

OUTPUT

```
db> db.students.find({}, {_id:1,gpa:1});
[
  { _id: ObjectId('6645f14f8419dc976a376b76'), gpa: 3.44 },
  { _id: ObjectId('6645f14f8419dc976a376b77'), gpa: 2.77 },
  { _id: ObjectId('6645f14f8419dc976a376b78'), gpa: 2.82 },
  { _id: ObjectId('6645f14f8419dc976a376b79'), gpa: 3.31 },
  { _id: ObjectId('6645f14f8419dc976a376b7a'), gpa: 3.63 },
  { _id: ObjectId('6645f14f8419dc976a376b7b'), gpa: 3.4 },
  { _id: ObjectId('6645f14f8419dc976a376b7e'), gpa: 2.56 },
  { _id: ObjectId('6645f14f8419dc976a376b80'), gpa: 3.44 },
  { _id: ObjectId('6645f14f8419dc976a376b81'), gpa: 3.02 },
  { _id: ObjectId('6645f14f8419dc976a376b83'), gpa: 2.6 },
  { _id: ObjectId('6645f14f8419dc976a376b84'), gpa: 2.89 },
  { _id: ObjectId('6645f14f8419dc976a376b85'), gpa: 2.75 },
  { _id: ObjectId('6645f14f8419dc976a376b88'), gpa: 3.43 },
  { _id: ObjectId('6645f14f8419dc976a376b89'), gpa: 3.04 },
  { _id: ObjectId('6645f14f8419dc976a376b8a'), gpa: 3.42 },
  { _id: ObjectId('6645f14f8419dc976a376b8d'), gpa: 3.97 },
  { _id: ObjectId('6645f14f8419dc976a376b8e'), gpa: 2.92 },
  { _id: ObjectId('6645f14f8419dc976a376b90'), gpa: 3.37 },
  { _id: ObjectId('6645f14f8419dc976a376b94'), gpa: 3.11 },
  { _id: ObjectId('6645f14f8419dc976a376b99'), gpa: 3.71 }
]
```

EXAMPLE 2 :

```
db.students.find({}, {name:1, home_city:1, _id:0});
```

OUTPUT

```
db> db.students.find({}, {name:1, home_city:1, _id:0});
[
  { name: 'Student 948', home_city: 'City 2' },
  { name: 'Student 157', home_city: 'City 4' },
  { name: 'Student 316' },
  { name: 'Student 346', home_city: 'City 8' },
  { name: 'Student 930', home_city: 'City 3' },
  { name: 'Student 305', home_city: 'City 6' },
  { name: 'Student 440', home_city: 'City 10' },
  { name: 'Student 256', home_city: 'City 1' },
  { name: 'Student 177', home_city: 'City 10' },
  { name: 'Student 487', home_city: 'City 3' },
  { name: 'Student 213' },
  { name: 'Student 690', home_city: 'City 7' },
  { name: 'Student 647', home_city: 'City 6' },
  { name: 'Student 232', home_city: 'City 1' },
  { name: 'Student 328', home_city: 'City 2' },
  { name: 'Student 468' },
  { name: 'Student 504', home_city: 'City 2' },
  { name: 'Student 915' },
  { name: 'Student 367', home_city: 'City 1' },
  { name: 'Student 969' }
]
```

MongoDB Projection Operators

MongoDB projection method positively impacts database performance as it reduces the workload of the find query when trying to retrieve specific data from a document, minimizing resource usage.

To enhance the querying and reduce the workload, multiple operators can be used within a projection query like the ones below:

- \$slice
- \$elemMatch
- \$meta

1. \$slice operator :

The \$slice operator bounds the number of elements that should be returned as the output of a MongoDB projection query.

Limitations in \$slice operator:

- In a nested array the **\$slice** operator will only return the sliced element and will not return any other item, especially from MongoDB 4.4.
- The find() action is not supported by the \$slice operator on MongoDB views.
- Due to a constraint imposed by MongoDB, the \$slice operator cannot be combined with the \$ projection operator. This is because top-level fields are not permitted to contain the \$ symbol as part of the field name.

EXAMPLE:

```
db.students.find({}, {name:1, age: {$slice:4}});
```

OUTPUT:

```
db> db.students.find({}, {name:1, age: {$slice:4}});
[
  {
    _id: ObjectId('6645f14f8419dc976a376b76'),
    name: 'Student 948',
    age: 19
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b77'),
    name: 'Student 157',
    age: 20
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b78'),
    name: 'Student 316',
    age: 20
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b79'),
    name: 'Student 346',
    age: 25
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b7a'),
    name: 'Student 930',
    age: 25
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b7b'),
    name: 'Student 305',
    age: 21
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b7e'),
    name: 'Student 440',
    age: 21
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b80'),
    name: 'Student 256',
    age: 19
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b81'),
    name: 'Student 177',
    age: 23
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b83'),
    name: 'Student 487',
    age: 21
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b84'),
    name: 'Student 213',
    age: 18
  },
  {
    _id: ObjectId('6645f14f8419dc976a376b85'),
    name: 'Student 690',
    age: 22
  }
]
```

2. \$elemMatch :

The \$elemMatch operator also limits the contents of an array to the first element that fits the given constraint. Though, there is a minor difference from the \$ operator because the \$elemMatch projection operator needs an explicit condition argument.

Limitations of \$elemMatch operator are as follows:

- The field to which the \$elemMatch projection is applied is returned as the last field of the document, regardless of the order in which the fields are ordered.
- \$elemMatch projection operator is not supported by find() operations on MongoDB views.
- The \$elemMatch operator does not handle \$text query expressions.

EXAMPLE :

```
db.candidates.find({courses:  
{$elemMatch:{$eq:"English"}}},{age:1,"courses.$":1})  
;
```

OUTPUT:

```
db> db.candidates.find({courses: {$elemMatch:{$eq:"English"}}},{age:1,"courses.$":1});
[
  {
    _id: ObjectId('66685877770ac6da0c342116'),
    age: 20,
    courses: [ 'English' ]
  },
  {
    _id: ObjectId('66685877770ac6da0c342118'),
    age: 19,
    courses: [ 'English' ]
  },
  {
    _id: ObjectId('66685877770ac6da0c34211a'),
    age: 23,
    courses: [ 'English' ]
  },
  {
    _id: ObjectId('66685877770ac6da0c34211e'),
    age: 22,
    courses: [ 'English' ]
  }
]
```

BENEFITS OF PROJECTION:

1. **Data Reduction:** By specifying which fields to include or exclude, you can reduce the amount of data sent to applications. This optimization is crucial for bandwidth efficiency.
2. **Performance Optimization:** Limiting the fields returned can improve query performance. Instead of retrieving all data, you only fetch what's necessary.
3. **User-Centric Data:** Projection helps extract specific fields relevant to user needs from large datasets¹². For example, you can select only the name and age of employees rather than displaying all details.

LIMIT

- The **limit()** method limits the number of records or documents that you want. It basically defines the max limit of records/documents that you want.
- Or in other words, this method uses on cursor to specify the maximum number of documents/ records the cursor will return.
- We can use this method after the find() method and find() will give you all the records or documents in the collection. You can also use some conditions inside the find to give you the result that you want.

Syntax:

```
db.collectionName.find(<query>).limit(<number>)
```

EXAMPLE 1:

```
db.candidates.find({}, { _id:0 }).limit(3);
```

OUTPUT:

```
db> db.candidates.find({}, {_id:0}).limit(3);
[
  {
    name: 'Alice Smith',
    age: 20,
    courses: [ 'English', 'Biology', 'Chemistry' ],
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    name: 'Bob Johnson',
    age: 22,
    courses: [ 'Computer Science', 'Mathematics', 'Physics' ],
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Charlie Lee',
    age: 19,
    courses: [ 'History', 'English', 'Psychology' ],
    gpa: 3.2,
    home_city: 'Chicago',
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
```

EXAMPLE 2:

```
db.candidates.find({gpa: {$gt:2.5}}, {_id:1}).limit(3);
```

OUTPUT

```
db> db.candidates.find({gpa:{$gt:2.5}},{_id:1}).limit(3);
[
  { _id: ObjectId('66685877770ac6da0c342116') },
  { _id: ObjectId('66685877770ac6da0c342117') },
  { _id: ObjectId('66685877770ac6da0c342118') }
]
```

Use of sort():

- The sort() method in MongoDB is used to specify the order in which the query returns matching documents from a given collection. It must be applied to the cursor before retrieving any documents from the database.
- It takes a document as a parameter that contains a **field: value** pair that defines the sort order of the result set. The value is 1 or -1 specifying an **ascending** or **descending** sort respectively.
- We can use the limit() method with the sort() method, it will return the first **m** documents, where **m** is the given limit.

EXAMPLE 3:

```
db.students.find({}, {_id:1}).sort({_id:-1}).limit(4);
```

OUTPUT

```
db> db.candidates.find({}, {_id:0}).sort({_id:1}).limit(3);
[
  {
    name: 'Alice Smith',
    age: 20,
    courses: [ 'English', 'Biology', 'Chemistry' ],
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    name: 'Bob Johnson',
    age: 22,
    courses: [ 'Computer Science', 'Mathematics', 'Physics' ],
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    name: 'Charlie Lee',
    age: 19,
    courses: [ 'History', 'English', 'Psychology' ],
    gpa: 3.2,
    home_city: 'Chicago',
    blood_group: 'B+',
    is_hotel_resident: true
  }
]
```

SELECTORS

Comparison gt lt:

```
db.candidates.find({ age: { $gt: 10}});
```

OUTPUT

```
db> db.candidates.find({ age: { $gt: 10}});
[
  {
    _id: ObjectId('66685877770ac6da0c342116'),
    name: 'Alice Smith',
    age: 20,
    courses: [ 'English', 'Biology', 'Chemistry' ],
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66685877770ac6da0c342117'),
    name: 'Bob Johnson',
    age: 22,
    courses: [ 'Computer Science', 'Mathematics', 'Physics' ],
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66685877770ac6da0c342118'),
    name: 'Charlie Lee',
    age: 19,
    courses: [ 'History', 'English', 'Psychology' ],
    gpa: 3.2,
    home_city: 'Chicago',
```

```
db.candidates.find({age: {$lt: 25}});
```

OUTPUT:

```
db> db.candidates.find({age: {$lt: 25}});
[
  {
    _id: ObjectId('66685877770ac6da0c342116'),
    name: 'Alice Smith',
    age: 20,
    courses: [ 'English', 'Biology', 'Chemistry' ],
    gpa: 3.4,
    home_city: 'New York City',
    blood_group: 'A+',
    is_hotel_resident: true
  },
  {
    _id: ObjectId('66685877770ac6da0c342117'),
    name: 'Bob Johnson',
    age: 22,
    courses: [ 'Computer Science', 'Mathematics', 'Physics' ],
    gpa: 3.8,
    home_city: 'Los Angeles',
    blood_group: 'O-',
    is_hotel_resident: false
  },
  {
    _id: ObjectId('66685877770ac6da0c342118'),
    name: 'Charlie Lee',
    age: 19,
    courses: [ 'History', 'English', 'Psychology' ],
    gpa: 3.2,
    home_city: 'Chicago',
```