
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. MongoDB is developed by MongoDB Inc. and current versions are licensed under the Server Side Public License (SSPL). MongoDB is a member of the MACH Alliance.

What is Database?

In computing, a database is an organized collection of data or a type of data store based on the use of a database management system (DBMS), the software that interacts with end users, applications, and the database itself to capture and analyze the data. The DBMS additionally encompasses the core facilities provided to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a database systems.

SET UP:

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

DATABASE MANAGEMENT SYSTEMS

Database Management Systems are software systems used to store, retrieve, and run queries on data. A DBMS serves as an interface between an end-user and a database, allowing users to create, read, update, and delete data in the database.

Few Commands to test after connections

Command	Expected Output	Notes
show dbs	admin 40.00 KiB config 72.00 KiB db 128.00 KiB local 40.00 KiB	All Databases are shown
use db	switched to db db	Connect and use db
show collections	Students	Show all tables
db.foo.insert({"bar" : "baz"})		Insert a record to collection. Create Collection if not exists

DOCUMENTS, COLLECTIONS AND DATATYPES

DOCUMENTS

A record in MongoDB is a document, which is a data structure composed of field and value pairs. MongoDB documents are similar to JSON objects. The values of fields may include other documents, arrays, and arrays of documents.

{"greetings":" Hello World!"}

COLLECTIONS

A Collection is a group of documents . MongoDB stores documents in collections. Collections are analogous to tables in relational databases.

DATATYPES

Basically each document will be stored in JSON format , where each attributes is of multiple data types

{
"name": "Edward Bill",
"skills": "Software Development",
"salary": 7500,
"status": true,
}

Experiment 1

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.

```
// Find all students with GPA greater than 3.5
db.students.find({ gpa: { $gt: 3.5 } });

// Find all students from "City 3"
db.students.find({ home_city: "City 3" });
```

AND

● Given a Collection you want to FILTER a subset based on multiple Conditions.

```
// Find all students who live in "City 5" AND have a blood group of "A+"
db.students.find({
  $and: [
    { home_city: "City 5" },
    { blood_group: "A+" }
  ]
});
```

OR

● Given a Collection you want to FILTER a subset based on multiple conditions but Any One is Sufficient

```
// Find all students who are hotel residents OR have a GPA less than 3.
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 3.0 } }
  ]
});
```

CRUD

● C - Create / Insert

● R - Remove

● U - update

● D - Delete

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

Insert

```
// Define the student data as a JSON document
const studentData = {
  "name": "Alice Smith",
  "age": 22,
  "courses": ["Mathematics", "Computer Science", "English"],
  "gpa": 3.8,
  "home_city": "New York",
  "blood_group": "A+",
  "is_hotel_resident": false
};

// Insert the student document into the "students" collection
db.students.insertOne(studentData);
```

Update

```
// Find a student by name and update their GPA
db.students.updateOne({ name: "Alice Smith" }, { $set: { gpa: 3.8 } });
```

Delete

```
// Delete a student by name
db.students.deleteOne({ name: "John Doe" });
```

Update many

```
// Update all students with a GPA less than 3.0 by increasing it by 0.5
db.students.updateMany({ gpa: { $lt: 3.0 } }, { $inc: { gpa: 0.5 } });
```

Delete many

```
// Delete all students who are not hotel residents
db.students.deleteMany({ is_hotel_resident: false });
```

Projection

This is used when we don’t need all columns/attributes.

```
// Get only the name and gpa for all students
db.students.find({}, { name: 1, gpa: 1 });

// Exclude the "_id" field from all queries by default
db.students.find({}, { _id: 0 });
```

Class 4: Projection, Limit & Selectors

Experiment 2 & 3

Agenda

- Understand Projection
    - Develop a MongoDB query to select certain fields and ignore some fields of the documents from
- Understand LIMIT
    - Develop a MongoDB query to display the first 5 documents from the results obtained.

PROJECTION

Projection

● Use the projection document as the second argument to the find method.

● Include field names with a value of 1 to specify fields to be returned.

● Omit fields or set them to 0 to exclude them from the results.

Get Selected Attributes

● Given a Collection you want to FILTER a subset of attributes. That is the place Projection is used.

```
// Get only the name and age for all students
db.students.find({}, { name: 1, age: 1 });
```

Ignore Attributes

```
// Get all student data but exclude the _id field
db.students.find({}, { _id: 0 });
```

Benefits of Projection

● Reduces data transferred between the database and your application.

● Improves query performance by retrieving only necessary data.

● Simplifies your code by focusing on the specific information you need.

LIMIT

Limit

- The limit operator is used with the find method.
- It's chained after the filter criteria or any sorting operations.
- Syntax: db.collection.find({filter}, {projection}).limit(number)

Get First 5 document

```
// Assuming you have already executed a query on the student collection
// Limit the results to the first 5 documents
db.students.find({}, { _id: 0 }).limit(5);
```

Limiting Results

```
// Find all students with GPA greater than 3.5 and limit to 2 documents
db.students.find({ gpa: { $gt: 3.5 } }, { _id: 0 }).limit(2);
```

Experiment 3 - SELECTORS

Comparison gt lt

```
// Find all students with age greater than 20
db.students.find({ age: { $gt: 20 } });
```

AND operator

```
// Find students from "City 2" with blood group "B+"
db.students.find({
  $and: [
    { home_city: "City 2" },
    { blood_group: "B+" }
  ]
});
```

OR operator

```
// Find students who are hotel residents OR have a GPA less than 3.0
db.students.find({
  $or: [
    { is_hotel_resident: true },
    { gpa: { $lt: 3.0 } }
  ]
});
```

Geospatial Query

```
db.locations.find({
  location: {
    $geoWithin: {
      $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in
    }
  }
});
```

Output

```
db> db.locations.find({
...   location: {
...     $geoWithin: {
...       $centerSphere: [[-74.005, 40.712], 0.00621376] // 1 kilometer in radians
...     }
...   }
... });
[
  {
    _id: 1,
    name: 'Coffee Shop A',
    location: { type: 'Point', coordinates: [ -73.985, 40.748 ] }
  },
  {
    _id: 2,
    name: 'Restaurant B',
    location: { type: 'Point', coordinates: [ -74.009, 40.712 ] }
  },
  {
    _id: 5,
    name: 'Park E',
    location: { type: 'Point', coordinates: [ -74.006, 40.705 ] }
  }
]
db>
```

Data types and Operations

- DataType
  - Point
  - Line String
  - Polygon

Data types and Operations

Name	Description
`$geoIntersects`	Selects geometries that intersect with a GeoJSON geometry. The `2dsphere` index supports `$geoIntersects`.
`$geoWithin`	Selects geometries within a bounding GeoJSON geometry. The `2dsphere` and `2d` indexes support `$geoWithin`.
`$near`	Returns geospatial objects in proximity to a point. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$near`.
`$nearSphere`	Returns geospatial objects in proximity to a point on a sphere. Requires a geospatial index. The `2dsphere` and `2d` indexes support `$nearSphere`.