

MongoDB Exercises

Topics : [MongoDB](#)

Written on [December 30, 2023](#)

Exercise 1: Insert Data

1. Create a new database named mydatabase.
2. Create a collection named students.
3. Insert two documents into the students collection with fields such as name, age, and subject.

Answers :

```
// 1. Create a new database named `mydatabase`.  
use mydatabase
```

```
// 2. Create a collection named `students`.  
db.createCollection("students")
```

```
// 3. Insert two documents into the `students` collection.  
db.students.insertMany([  
  { name: "John", age: 22, subject: "Math" },  
  { name: "Alice", age: 25, subject: "History" }  
])
```

Exercise 2: Query Data

1. Retrieve all documents from the students collection.
2. Retrieve only the names of students from the students collection.
3. Find all students who are 25 years old.

Answers :

```
// 1. Retrieve all documents from the `students` collection.  
db.students.find()
```

```
// 2. Retrieve only the names of students from the `students` collection.  
db.students.find({}, { name: 1, _id: 0 })
```

```
// 3. Find all students who are 25 years old.  
db.students.find({ age: 25 })
```

Exercise 3: Update Data

1. Update the age of a specific student in the `students` collection.
2. Add a new field, `grade`, with the value "A" to all documents in the `students` collection.

Answers :

```
// 1. Update the age of a specific student in the `students` collection.  
db.students.updateOne({ name: "John" }, { $set: { age: 23 } })
```

```
// 2. Add a new field, `grade`, with the value "A" to all documents in the `students` collection.  
db.students.updateMany({}, { $set: { grade: "A" } })
```

Exercise 4: Delete Data

1. Delete a specific student from the `students` collection.
2. Remove the `grade` field from all documents in the `students` collection.

Answers :

```
// 1. Delete a specific student from the `students` collection.  
db.students.deleteOne({ name: "Alice" })
```

```
// 2. Remove the `grade` field from all documents in the `students` collection.  
db.students.updateMany({}, { $unset: { grade: 1 } })
```

Exercise 5: Aggregation

1. Calculate the average age of students in the `students` collection.
2. Group students by their subjects and calculate the count of students in each subject.

Answers :

```
// 1. Calculate the average age of students in the `students` collection.  
db.students.aggregate([  
  { $group: { _id: null, avgAge: { $avg: "$age" } } }  
])
```

```
// 2. Group students by their subjects and calculate the count of students in each subject.  
db.students.aggregate([  
  { $group: { _id: "$subject", count: { $sum: 1 } } }  
])
```

Exercise 6: Indexing

1. Create an index on the `name` field in the `students` collection.

2. Check the execution plan of a query to see if the created index is being used.

Answers :

```
// 1. Create an index on the `name` field in the `students` collection.
db.students.createIndex({ name: 1 })

// 2. Check the execution plan of a query to see if the created index is being used.
db.students.find({ name: "John" }).explain("executionStats")
```

Exercise 7: Text Search

1. Create a text index on the name and subject fields in the students collection.
2. Perform a text search for students with a specific keyword.

Answers :

```
// 1. Create a text index on the `name` and `subject` fields in the `students` collection.
db.students.createIndex({ name: "text", subject: "text" })

// 2. Perform a text search for students with a specific keyword.
db.students.find({ $text: { $search: "Math" } })
```

Exercise 8: Working with Dates

1. Insert a document with a birthDate field representing a date of birth.
2. Find students born after a certain date.

Answers :

```
// 1. Insert a document with a `birthDate` field representing a date of birth.
db.students.insertOne({ name: "Bob", birthDate: ISODate("1990-01-01") })

// 2. Find students born after a certain date.
db.students.find({ birthDate: { $gt: ISODate("1990-01-01") } })
```

Exercise 9: Geospatial Query

1. Create a collection named locations.
2. Insert documents representing locations with latitude and longitude fields.
3. Find locations near a specific point using geospatial queries.

Answers :

```
// 1. Create a collection named `locations`.
db.createCollection("locations")

// 2. Insert documents representing locations with `latitude` and `longitude` fields.
```

```
db.locations.insertMany([
  { name: "Location1", location: { type: "Point", coordinates: [1, 1] } },
  { name: "Location2", location: { type: "Point", coordinates: [2, 2] } }
])
```

// 3. Find locations near a specific point using geospatial queries.

```
db.locations.find({
  location: {
    $near: {
      $geometry: { type: "Point", coordinates: [0, 0] },
      $maxDistance: 100000 // in meters
    }
  }
})
```

Exercise 10: Aggregation Pipeline

1. Create a collection named `orders` with documents representing orders.
2. Use the aggregation pipeline to calculate the total revenue.

Answers :

// 1. Create a collection named `orders` with documents representing orders.

```
db.createCollection("orders")
db.orders.insertMany([
  { product: "A", quantity: 10, price: 5 },
  { product: "B", quantity: 5, price: 10 },
  { product: "A", quantity: 8, price: 6 }
])
```

// 2. Use the aggregation pipeline to calculate the total revenue.

```
db.orders.aggregate([
  { $project: { revenue: { $multiply: ["$quantity", "$price"] } } },
  { $group: { _id: null, totalRevenue: { $sum: "$revenue" } } }
])
```