MongoDB

Introduction

MongoDB is a NoSQL database known for its flexibility and scalability. It stores data in JSON-like documents, which makes it easy to work with nested and hierarchical data. This guide focuses on MongoDB's querying capabilities.

Create Collection using mongosh

Method 1

- You can create a collection using the createCollection() database method.

```
db.createCollection("posts")
```

Method 2

- You can also create a collection during the insert process.
- We are here assuming object is a valid JavaScript object containing post data:

```
db.posts.insertOne(object)
```

Insert Documents

There are 2 methods to insert documents into a MongoDB database.

1. **insertOne()**

- To insert a single document, use the insertOne() method.
- This method inserts a single object into the database.
- Example

```
db.posts.insertOne({
  title: "Post Title 1",
  body: "Body of post.",
  category: "News",
  likes: 1,
  tags: ["news", "events"],
  date: Date()
})
```

- Output

```
{
  acknowledged: true,
  insertedId: ObjectId("62c350dc07d768a33fdfe9b0")
}
Atlas atlas-8iy36m-shard-0 [primary] blog>
```

2. insertMany()

- To insert multiple documents at once, use the insertMany() method.
- This method inserts an array of objects into the database.
- Example

```
db.posts.insertMany([
  {
    title: "Post Title 2",
    body: "Body of post.",
    category: "Event",
    likes: 2,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 3",
    body: "Body of post.",
    category: "Technology",
    likes: 3,
    tags: ["news", "events"],
    date: Date()
  },
  {
    title: "Post Title 4",
    body: "Body of post.",
    category: "Event",
    likes: 4,
    tags: ["news", "events"],
    date: Date()
  }
])
```

- Output

```
{
  acknowledged: true,
```

```
  insertedIds: {
    '0': ObjectId("62c3513907d768a33fdfe9b1"),
    '1': ObjectId("62c3513907d768a33fdfe9b2"),
    '2': ObjectId("62c3513907d768a33fdfe9b3")
  }
}
```

Find Data

- There are 2 methods to find and select data from a MongoDB collection, find() and findOne().

1. find()

- To select data from a collection in MongoDB, we can use the find() method.
- This method accepts a query object. If left empty, all documents will be returned.
- Example

  db.posts.find()

```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    body: 'Body of post.',
    category: 'News',
    likes: 1,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  },
  {
    _id: ObjectId("62c3513907d768a33fdfe9b1"),
    title: 'Post Title 2',
    body: 'Body of post.',
    category: 'Event',
    likes: 2,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:44:41 GMT-0500 (Central Daylight Time)'
  },
  {
    _id: ObjectId("62c3513907d768a33fdfe9b2"),
    title: 'Post Title 3',
    body: 'Body of post.',
    category: 'Technology',
    likes: 3,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:44:41 GMT-0500 (Central Daylight Time)'
  },
  {
    _id: ObjectId("62c3513907d768a33fdfe9b3"),
    title: 'Post Title 4',
    body: 'Body of post.',
    category: 'Event',
    likes: 4,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:44:41 GMT-0500 (Central Daylight Time)'}]
```

2. findOne()

- To select only one document, we can use the findOne() method.
- This method accepts a query object. If left empty, it will return the first document it finds.

- Example

  db.posts.findOne()

```
{
  _id: ObjectId("62c350dc07d768a33fdfe9b0"),
  title: 'Post Title 1',
  body: 'Body of post.',
  category: 'News',
  likes: 1,
  tags: [ 'news', 'events' ],
  date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
}
```

Update Document

- To update an existing document we can use the updateOne() or updateMany() methods.
- The first parameter is a query object to define which document or documents should be updated.
- The second parameter is an object defining the updated data.

1. updateOne()

- The updateOne() method will update the first document that is found matching the provided query.
- Example

  db.posts.find({ title: "Post Title 1" })

```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    body: 'Body of post.',
    category: 'News',
    likes: 1,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  }
]
```

- update the "likes" on this post to 2. To do this, we need to use the $set operator.

  db.posts.updateOne({ title: "Post Title 1" }, { $set: { likes: 2 } })

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

- After updating

  db.posts.find({ title: "Post Title 1" })

```
[
  {
    _id: ObjectId("62c350dc07d768a33fdfe9b0"),
    title: 'Post Title 1',
    body: 'Body of post.',
    category: 'News',
    likes: 2,
    tags: [ 'news', 'events' ],
    date: 'Mon Jul 04 2022 15:43:08 GMT-0500 (Central Daylight Time)'
  }
```

```
]
```

2. updateMany()

- The updateMany() method will update all documents that match the provided query.
- Example
- Update likes on all documents by 1. For this we will use the $inc (increment) operator:

```
db.posts.updateMany({}, { $inc: { likes: 1 } })
```

```
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 5,
  modifiedCount: 5,
  upsertedCount: 0
}
```

Delete Documents

- We can delete documents by using the methods deleteOne() or deleteMany().
- These methods accept a query object. The matching documents will be deleted.

1. deleteOne()

- The deleteOne() method will delete the first document that matches the query provided.
- Example

```
db.posts.deleteOne({ title: "Post Title 5" })
```

- Output

```
{ acknowledged: true, deletedCount: 1 }
```

2. deleteMany()

- The deleteMany() method will delete all documents that match the query provided.
- Example

```
db.posts.deleteMany({ category: "Technology" })
```

- Output

```
{ acknowledged: true, deletedCount: 1 }
```

Query Operators

There are many query operators that can be used to compare and reference document fields.

Comparison

The following operators can be used in queries to compare values:

- $eq: Values are equal
- $ne: Values are not equal
- $gt: Value is greater than another value
- $gte: Value is greater than or equal to another value
- $lt: Value is less than another value

- $lte: Value is less than or equal to another value
- $in: Value is matched within an array

Example

```
db.users.find({ age: { $gt: 25 } })
```

Output

```
{ "_id": ObjectId("507f191e810c19729de860ea"), "name": "Jane Doe", "age": 30, "email": "jane@example.com" }
```

Logical

The following operators can logically compare multiple queries.

- $and: Returns documents where both queries match
- $or: Returns documents where either query matches
- $nor: Returns documents where both queries fail to match
- $not: Returns documents where the query does not match

Example

```
db.users.find({ $or: [{ age: { $lt: 25 } }, { name: "John Doe" }] })
```

Output

```
{ "_id": ObjectId("507f191e810c19729de860ea"), "name": "John Doe", "age": 29, "email": "john@example.com" }
{ "_id": ObjectId("507f191e810c19729de860eb"), "name": "Alex", "age": 22, "email": "alex@example.com" }
```

Evaluation

The following operators assist in evaluating documents.

- $regex: Allows the use of regular expressions when evaluating field values
- $text: Performs a text search
- $where: Uses a JavaScript expression to match documents

Example

```
db.collection.find({ key: { $regex: /pattern/, $options: 'i' } })
```

```
db.users.find({ name: { $regex: /^J/, $options: 'i' } })
```

Output

```
{ "_id": ObjectId("507f191e810c19729de860ea"), "name": "John Doe", "age": 29, "email": "john@example.com" }
{ "_id": ObjectId("507f191e810c19729de860eb"), "name": "Jane Doe", "age": 30, "email": "jane@example.com" }
```

Update Operators

There are many update operators that can be used during document updates.

Fields

The following operators can be used to update fields:

- $currentDate: Sets the field value to the current date

- $inc: Increments the field value
- $rename: Renames the field
- $set: Sets the value of a field
- $unset: Removes the field from the document

Array

The following operators assist with updating arrays.

- $addToSet: Adds distinct elements to an array
- $pop: Removes the first or last element of an array
- $pull: Removes all elements from an array that match the query
- $push: Adds an element to an array

Aggregation Pipelines

- Aggregation operations allow you to group, sort, perform calculations, analyze data, and much more.

- Aggregation pipelines can have one or more "stages". The order of these stages are important. Each stage acts upon the results of the previous stage.

- Example

```
db.posts.aggregate([
  {
    $match: { likes: { $gt: 1 } }
  },
  {
    $group: { _id: "$category", totalLikes: { $sum: "$likes" } }
  }
])
```

- Output

```
[ { _id: 'News', totalLikes: 3 }, { _id: 'Event', totalLikes: 8 }
```

Indexing & Search

MongoDB Atlas comes with a full-text search engine that can be used to search for documents in a collection.

➢ Creating an Index

1. From the Atlas dashboard, click on your **Cluster name** then the **Search** tab.
2. Click on the **Create Search Index** button.
3. Use the **Visual Editor** and click Next.
4. Name your index, choose the Database and Collection you want to index and click Next.
    - If you name your index "default" you will not have to specify the index name in the $search pipeline stage.
    - Choose the sample_mflix database and the movies collection.
5. Click **Create Search Index** and wait for the index to complete.

➢ Running a Query

To use our search index, we will use the $search operator in our aggregation pipeline.

- Example

```
db.movies.aggregate([
 {
  $search: {
   index: "default",
   text: {
    query: "star wars",
    path: "title"
   },
  },
 },
 {
  $project: {
   title: 1,
   year: 1,
  }
 }
])
```

- Output

```
[
  {
    _id: ObjectId("573a13c0f29313caabd62f62"),
    year: 2008,
    title: 'Star Wars: The Clone Wars'
  },
  {
    _id: ObjectId("573a13bdf29313caabd599ca"),
    title: 'Robot Chicken: Star Wars',
    year: 2007
  },
  {
    _id: ObjectId("573a1396f29313caabce37ff"),
    title: 'Star!',
    year: 1968
  },
  {
    _id: ObjectId("573a13a6f29313caabd17d08"),
    title: 'Star',
    year: 2001
  },
  {
    _id: ObjectId("573a1397f29313caabce68f6"),
    year: 1977,
    title: 'Star Wars: Episode IV - A New Hope'
  }
]
```