Lesson:

CRUD with MongoDB

PW SKILLS

Topics

- Introduction
- Prerequisites
- Create operation
- Read operation
- Update operation
- Delete operation

Introduction

MongoDB, a NoSQL database, supports the four fundamental operations: Create, Read, Update, and Delete, commonly called CRUD. This article will guide you through the essential concepts and implementation of CRUD operations using MongoDB.

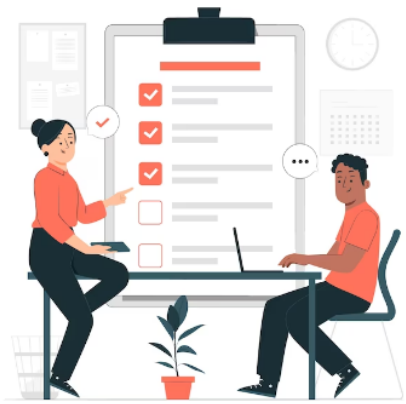
We will begin by creating a dedicated database named "my-db" to serve as the focal point for all the examples within this module. This database will act as the sandbox where we perform various CRUD operations, allowing us to explore and understand the functionalities of MongoDB in a focused and organized manner throughout this module.

Create the database using the “use” command in MongoDB shell
i.e

```
Unset
test> use my-db
switched to db my-db
my-db> show dbs
admin    40.00 KiB
config   72.00 KiB
local    76.00 KiB
my-db>
```

Note - The database created i.e. “my-db” will be visible after data or collections are added.

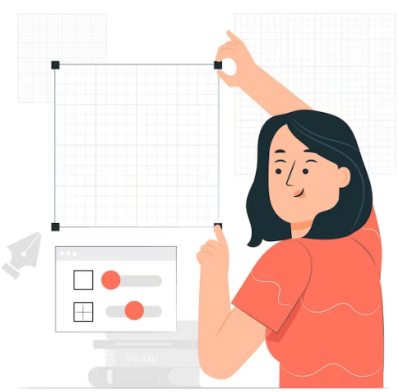
Pre-requisites



Before diving into CRUD operations with MongoDB, ensure you have the following prerequisites:

1. **MongoDB Installed:** Install MongoDB on your machine. You can find installation instructions on the official MongoDB website.
2. Basic knowledge of MongoDB key concepts
3. **MongoDB Shell installed:** The MongoDB Shell is a command-line interface to interact with MongoDB. It's a powerful tool for executing queries and commands directly.
4. **MongoDB Compass installed(Optional):** MongoDB Compass is a graphical user interface for MongoDB. Though optional, it can provide a visual representation of your data and aid in understanding the operations.

Create operation



In MongoDB, data is stored in documents. Learn how to perform the Create operation using the insertOne and insertMany methods. Dive into examples that illustrate the creation of single and multiple documents, understanding the nuances of document structure.

Syntax

```
Unset
// Insert a single document
db.collection.insertOne({
  key1: value1,
  key2: value2,
  // ... additional fields

});

// Insert multiple documents
db.collection.insertMany([
  {
    key1: value1,
    key2: value2,
  },
  // ... additional documents
]);
```

Examples of create operation

Inserting single documents -

```
Unset
my-db> db.users.insertOne({name: "john", age: 23})
{
  acknowledged: true,
  insertedId: ObjectId("65a23b9d9cd85549a1ed90d6")
}
my-db>
```

Inserting multiple documents -

```
Unset
my-db> db.users.insertMany([{name: "Tom", age: 22}, {name: "Jimmy", age: 14}])
{
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("65a23c389cd85549a1ed90d7"),
    '1': ObjectId("65a23c389cd85549a1ed90d8")
  }
}
my-db>
```

Read operation

In MongoDB, data in documents can be fetched using the find method. Discover how to find documents in a collection, apply filters, limit results, and sort data. Uncover the versatility of MongoDB queries to extract precisely the information you need.

Syntax -

```
Unset
// Find all documents in a collection
db.collection.find();

// Find documents that match a specific condition
db.collection.find({ key: value });

// Limit the number of returned documents
db.collection.find().limit(5);

// Sorting documents
db.collection.find().sort({ key: 1 }); // 1 for ascending, -1 for descending


// retreive single document
db.collection.findOne(
{key: value}, // query criteria

{key: value} // Projection
)
```

Example read operation

Find all documents in a collection

```
Unset
my-db> db.users.find()
[
  { _id: ObjectId("65a23b9d9cd85549a1ed90d6"), name: 'john', age: 23 },
  { _id: ObjectId("65a23c389cd85549a1ed90d7"), name: 'Tom', age: 22 },
  { _id: ObjectId("65a23c389cd85549a1ed90d8"), name: 'Jimmy', age: 14 }
]
my-db>
```

Find documents that match a specific condition, here find a user name with “john”

```
Unset
my-db> db.users.find({name: "john"})
[
  { _id: ObjectId("65a23b9d9cd85549a1ed90d6"), name: 'john', age: 23 }
]
my-db>
```

Limit the number of returned documents

```
Unset
my-db> db.users.find().limit(2)


[
  { _id: ObjectId("65a23b9d9cd85549a1ed90d6"), name: 'john', age: 23 },
  { _id: ObjectId("65a23c389cd85549a1ed90d7"), name: 'Tom', age: 22 }
]
my-db>
```

Sorting documents, 1 for ascending & -1 for descending, sorting based on _id field

```
Unset
my-db> db.users.find().sort({_id: 1})
[
  { _id: ObjectId("65a23b9d9cd85549a1ed90d6"), name: 'john', age: 23 },
  { _id: ObjectId("65a23c389cd85549a1ed90d7"), name: 'Tom', age: 22 },
  { _id: ObjectId("65a23c389cd85549a1ed90d8"), name: 'Jimmy', age: 14 }
]
my-db> db.users.find().sort({_id: -1})
[
  { _id: ObjectId("65a23c389cd85549a1ed90d8"), name: 'Jimmy', age: 14 },
  { _id: ObjectId("65a23c389cd85549a1ed90d7"), name: 'Tom', age: 22 },
  { _id: ObjectId("65a23b9d9cd85549a1ed90d6"), name: 'john', age: 23 }
]
my-db>
```

Retrieve a single document that matches the query

```
JavaScript
my-db> db.users.find()
[
 { _id: ObjectId("65a24b6e9cd85549a1ed90d9"), name: 'john', age: 22 },
 { _id: ObjectId("65a6191ca2c46added9391c9"), name: 'Tommy', age: 18 },
 { _id: ObjectId("65a6192ca2c46added9391ca"), name: 'Jerry', age: 28 },
 { _id: ObjectId("65a619dea2c46added9391cb"), name: 'Steve', age: 18 }
]
my-db> db.users.findOne({name: "Tommy"}, {age: 18})
{ _id: ObjectId("65a6191ca2c46added9391c9"), age: 18 }
my-db>
```

Update operation

Updating documents allows you to modify existing data, In MongoDB, the updateOne and updateMany methods along with the $set operator are used to modify specific fields within documents in the collections and database.

Syntax -

```
Unset
// Update a single document
db.collection.updateOne(
  { key: value }, // Filter criteria
  { $set: { newKey: newValue } } // Update operation
);

// Update multiple documents
db.collection.updateMany(
  { key: value }, // Filter criteria
  { $set: { newKey: newValue } } // Update operation
);
```

Example of Update operation

Updating single document

```
Unset
my-db> db.users.updateOne({name:"john"}, {$set: {emal: "john@gmail.com"}})
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
my-db> db.users.find({name: "john"})
[
  {
    _id: ObjectId("65a23b9d9cd85549a1ed90d6"),
    name: 'john',
    age: 23,
    emal: 'john@gmail.com'
  }
]
my-db>
```

Updating multiple documents

```
Unset
my-db> db.users.updateMany({ _id: ObjectId("65a23b9d9cd85549a1ed90d6") },{
$set: {
  name: "johnny",
  email: "johnny@gmail.com"
  }
 }
);
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 0,
  upsertedCount: 0

}
my-db> db.users.find()
[
  {
    _id: ObjectId("65a23b9d9cd85549a1ed90d6"),
    name: 'johnny',
    age: 23,
    emal: 'john@gmail.com',
    email: 'johnny@gmail.com'
  },
  { _id: ObjectId("65a23c389cd85549a1ed90d7"), name: 'Tom', age: 22 },
  { _id: ObjectId("65a23c389cd85549a1ed90d8"), name: 'Jimmy', age: 14 }
]
my-db>
```

Delete operation

Deleting documents removes data from a collection, In MongoDB the deleteOne and deleteMany methods are used to delete documents within the documents inside the collection and database.

Syntax -

```
Unset
// Delete a single document
db.collection.deleteOne({ key: value });

// Delete multiple documents
db.collection.deleteMany({ key: value });
```

Example of Update operation

Deleting single documents

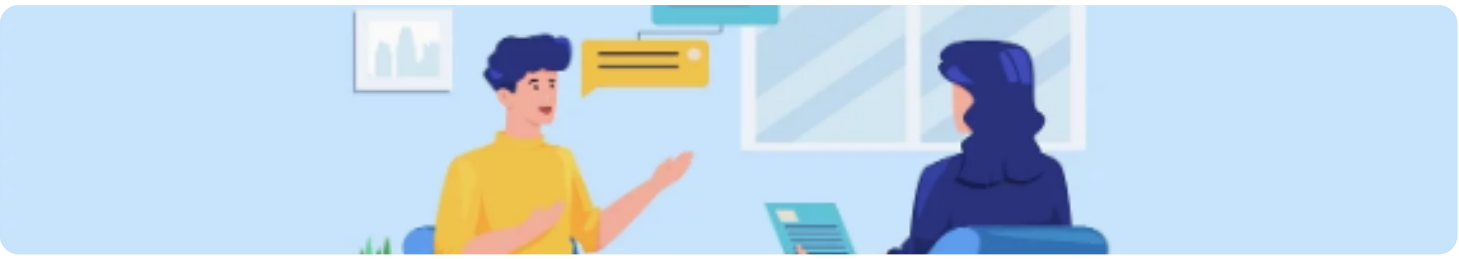
```
Unset
my-db> db.users.deleteOne({category: "pass"})
{ acknowledged: true, deletedCount: 1 }
my-db> db.users.find()
[
  {
    _id: ObjectId("65a23c389cd85549a1ed90d7"),
    name: 'Tom',
    age: 22,
    category: 'fail'
  },
  {
    _id: ObjectId("65a23c389cd85549a1ed90d8"),
    name: 'Jimmy',
    age: 14,
    category: 'fail'
  }
]
```

Deleting multiple documents

```
Unset
my-db> db.users.deleteMany({category: "fail"})
{ acknowledged: true, deletedCount: 2 }
my-db> db.users.find()

my-db>
```

Interview points



Question 1 - What is the purpose of the $unset operator in MongoDB updates?

Answer - The $unset operator removes a specified field from a document. It is useful when you need to eliminate a field or multiple fields from documents, effectively "unsetting" them.

Question 2 - How can you find documents in MongoDB where a specific field exists? Provide an example

Answer - You can use the $exists operator to find documents where a specific field exists:

```
Unset
my-db> db.users.find({name: {$exists: true}})

[
  { _id: ObjectId("65a24b6e9cd85549a1ed90d9"), name: 'john', age: 22 }
]
my-db>
```

This query retrieves documents where the field “name” or provided key name exists.

PW SKILLS

THANK YOU !