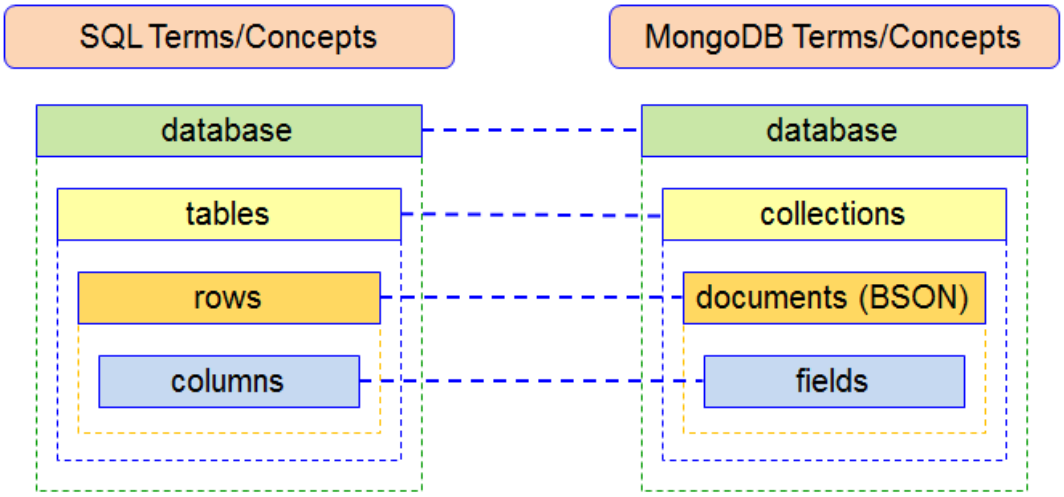
What is MongoDB?

MongoDB is a **NoSQL open-source, document-oriented database** developed for storing and processing high volumes of data. Compared to conventional relational databases, MongoDB makes use of collections and documents instead of tables consisting of rows and columns. The collections consist of several documents containing the basic units of data in terms of key and value pairs.

How is MongoDB Structured?

Perhaps the best place to start is to compare MongoDB with a traditional RDBMS.



The idea behind MongoDB is you start from the server which can host multiple databases. But unlike a SQL database that stores data in tables and rows, MongoDB stores data as collections of documents. So under your database, you get collections, and under your collections you get documents. Documents have fields that are simply key/value pairs.

Since JSON documents support embedded fields, related data and lists of data can be stored within the document as arrays/lists instead of having an external table. Therefore, developers can validate relationships between data without the need for spreading data across tables and performing joins, allowing them to work quickly and efficiently. To illustrate this concept, consider the following tabular structure:

name	quantity	size	status	tags	rating
postcard	75	8.5×11,cm	A	double-sided, white	10

From the example above, it is evident that there are fields in this table that have multiple values and would otherwise not be easy to query or display if modelled in a single column (e.g. size and tags). In an RDBMS you might need to model your data differently by creating a relational table for those columns.

A MongoDB document on the other hand might solve the same problem as follows:

```
{
 "name": "postcard",
 "qty": 75,
 "rating": 10,
 "size": { "height": 11, "width": 8.5, "unit": "cm" },
 "status": "A",
 "tags": [ "double-sided", "white"]
}
```

CRUD Operations with MongoDB are the methods that MongoDB exposes for storage management. **CRUD** is an acronym for Create, Read, Update, and Delete. You can use these four basic methods for viewing, searching, and changing resources in your database.

Image Source

The following is a brief overview of what each operation does:

- **Create**: This is the operation that is used to insert or add new documents to the database.
- **Read**: This operation is used to view or fetch documents from the database.
- **Update**: This operation should be used when modifying existing documents.
- **Delete**: As the name suggests, the delete operation is used to remove documents from the database.

Together these four CRUD operations make up the essential operations of interacting with your MongoDB server.

How to Perform CRUD Operations with MongoDB?

Now that you have connected to your MongoDB database, and you know how MongoDB works, as well as what CRUD operations with MongoDB are, it's time to take a look at how to manipulate documents in a MongoDB database using CRUD methods.

Let's dive into the processes of creating, reading, updating, and deleting documents in a MongoDB server.

Prerequisites

To follow along, you will need:

- MongoDB installed on your server.
- Basic familiarity with the MongoDB shell will be used throughout this guide.

Before you Begin

- Connect to your MongoDB server using the MongoDB shell.
- Create a new MongoDB database using the **use <database_name>** command, or use an existing database.

CRUD Operations with MongoDB: Create Operations

The first one in CRUD Operations with MongoDB is **Create**. When creating documents in a MongoDB collection, the most important commands are:

- insertOne()
- insertMany()

insertOne()

The **insertOne()** command allows you to insert one data object at a time into the collection. It takes some options in key/value pairs separated by a comma thus establishing the schema. Collections are created implicitly in MongoDB, therefore if a collection doesn't exist, MongoDB will create it.

For this example, you'll work with a collection that will be named "BooksDB". You can insert a single entry into the BooksDB collection by calling the **insertOne()** method. In the MongoDB shell, execute the following command:

```
db.BooksDB.insertOne({
   title: "Dead Silence",
   author: "S.A. Barnes",
   isbn: 1250819997,
   price: 13.99,
```

```
    available: true
})
```

If the create operation is successful, a new document is created and the output will acknowledge that it was executed successfully. MongoDB will also automatically generate a unique ObjectId for the new document.

```
db.BooksDB.insertOne({
    title: "Dead Silence",
    author: "S.A. Barnes",
    isbn: 1250819997,
    price: 13.99,
    available: true})
{
        "acknowledged" : true,
        "insertedId" : ObjectId("5fd989674e6b9ceb8665c57d")
}
```

insertMany()

Inserting many documents one at a time can quickly turn out to be an arduous task, especially for huge datasets. Therefore, MongoDB provides the **insertMany()** method for inserting multiple documents in one go. To do this, within the parentheses, you use square brackets ([and]) to signify that you are passing in an array of multiple documents to the collection. This is called nesting and the nested documents are separated by commas.

```
db.BooksDB.insertMany([{
    title: "Dead Silence",
    author: "S.A. Barnes",
    isbn: 1250819997,
    price: 13.99,
    available: true},
      {title: "Day Zero",
      author: "C. Robert Cargill",
      isbn: 0062405802,
      price: 27.99,
```

```
    available: true},
    {title: "Sea of Rust",
    author: "C. Robert Cargill",
    isbn: 0062405803,
    price: 21.99,
    available: false}])
```

This is what the output looks like:

```
db.BooksDB.insertMany([{ title: "Dead Silence", author: "S.A. Barnes", isbn: 1250819997,
price: 13.99, available: true}, {title: "Day Zero", author: "C. Robert Cargill",
isbn: 0062405802, price: 27.99, available: true}, {title: "Sea of Rust", author: "C. Robert
Cargill", isbn: 0062405803, price: 21.99, available: false}])

{
      "acknowledged" : true,
      "insertedIds" : [
            ObjectId("5fd98ea9ce6e8850d88270b4"),
            ObjectId("5fd98ea9ce6e8850d88270b5"),

"ObjectId("5fd98ea9ce6e8850d88270y3",

      ]
}
```

CRUD Operations with MongoDB: Read Operations

The next operation in CRUD Operations with MongoDB is **Read**. They are used to retrieve documents from a collection. You can perform read operations using the following methods provided by MongoDB CRUD:

- find()
- findOne()

Let’s now look at what each method does.

find()

To get all the documents from a collection, you can simply use the **find()** method on your chosen collection. Executing just the **find()** method with no arguments will return all records currently in the collection.

```
db.BooksDB.find()
```

This is what the output looks like:

```
{ "_id": "ObjectId("5fd98ea9ce6e8850d88270b4")", "title": "Dead Silence", "author": "S.A. Barnes", "isbn": 1250819997,
"price": 13.99, "available': true}
{"_id": "ObjectId("5fd98ea9ce6e8850d88270b5")", "title": "Day Zero", "author": "C. Robert Cargill",
"isbn": 0062405802, "price": 27.99, "available": true}
{"_id": "ObjectId("5fd98ea9ce6e8850d88270y3")", "title": "Sea of Rust", "author": "C. Robert Cargill",
"isbn": 0062405803, "price": 21.99, "available": false}
```

As you can see from the output, MongoDB has assigned an "_id" key to every record that maps to an "ObjectId" value.

When performing reads, it's possible to drill down to a more specific subsection of your records by filtering the results by the key/value pair.

```
db.BooksDB.find({"title":"Day Zero"})
{"_id": "ObjectId("5fd98ea9ce6e8850d88270b5")", "title": "Day Zero", "author": "C. Robert Cargill",
"isbn": 0062405802, "price": 27.99, "available": true}
```

findOne()

To retrieve only one record that satisfies specific search criteria, you can use the **findOne()** method.

So perhaps if you have more one than one record that satisfies the query, then the **findOne()** method will return only one record based on the order that the records were created. The function takes the following syntax.

```
db.{collection}.findOne({query}, {projection})
```

For example, when you run the following query:

```
{ "_id": "ObjectId("5fd98ea9ce6e8850d88270b4")", "title": "Dead Silence", "author": "S.A. Barnes", "isbn": 1250819997,
"price": 13.99, "available': true}
```

CRUD Operations with MongoDB: Update Operations

The third operation in CRUD Operations with MongoDB is **Update**. Over time, it's quite common for the records in your database to change. For example, a user might update their username, email address, credit card information, etc. Other times, you might need to evolve the data structure to keep up with the changing requirements of an application.

MongoDB provides update operations that enable users to change field values at the document level as well as enabling them to add new fields to the documents in a collection. An update operation typically takes a filter to choose the documents you want to update and an action. Note that updates cannot be rolled back and as such you take extra caution when running them.

MongoDB provides two methods of updating documents:

- updateOne()
- updateMany()

updateOne()

You can update a single document using the **updateOne()** method on a chosen collection. To update a single document in the BooksDB collection, you need to provide a filter as an argument as well as an update action.

The update filter specifies the items to update, and action defines how to update the items. Let's try updating the name of the author S.A. Barnes to her full name of Stacey Kade Barnes. To achieve this, use the **updateOne()** method which updates a single document:

```
db.BooksDB.updateOne({author: "S.A. Barnes"}, {$set:{author: "Stacey Kade Barnes"}})
```

This is what the output looks like:

```
{ "acknowledged" : true, "matchedCount" : 1, "modifiedCount" : 1 }


{ "_id": "ObjectId("5fd98ea9ce6e8850d88270b4")", "title": "Dead Silence", "author": "Stacey Kade Barnes", "isbn": 1250819997,
"price": 13.99, "available': true}
```

updateMany()

CRUD operations with MongoDB expose the **updateMany()** method to enable application developers to update multiple documents by passing in a list of items, similar to inserting multiple documents. It's unlike The **updateOne()** method that only updates the first document that meets the filter criteria. The **updateMany()** method has the same syntax as the **updateOne()** method.

```
db.BooksDB.updateMany({author:"C. Robert Cargill"}, {$set: {author: "Christopher Robert Cargill"}})
```

This is what the output looks like:

```
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```

Now running the **find()** command yields the following result:

```
db.BooksDB.find()


{"_id": "ObjectId("5fd98ea9ce6e8850d88270b5")", "title": "Day Zero", "author": "Christopher Robert Cargill",
"isbn": 0062405802, "price": 27.99, "available": true}
```

{"_id": "ObjectId("5fd98ea9ce6e8850d88270y3")", "title": "Sea of Rust", "author": "Christopher Robert Cargill",
"isbn": 0062405803, "price": 21.99, "available": false}

CRUD Operations with MongoDB: Delete Operations

The last operation in CRUD Operations with MongoDB is **Delete**. Delete operations operate on a single document. To delete a document from a collection, you need to provide a filter select the document. Delete has a similar syntax to read operations.

CRUD Operations with MongoDB provides two methods of deleting records from a collection:

- deleteOne()
- deleteMany()

deleteOne()

The **deleteOne()** method of Crud operations with MongoDB can be used to delete a document from a collection. A filter is normally required to select the specified item that you want to delete. Usually, only the first document that meets the criteria is deleted.

```
db.BooksDB.deleteOne({name:"Christopher Robert Cargill"})
```

This is what the output looks like:

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

deleteMany()

To delete multiple documents from a collection that match a filter, MongoDB CRUD provides the **deleteMany()** method used to delete multiple documents from a desired collection with a single delete operation. For example, to delete all books by the author Christopher, the syntax is quite to the **deleteOne()** method:

```
db.BooksDB.deleteMany({author:"Christopher Robert Cargill"})
```

This is what the output looks like:

```
{ "acknowledged" : true, "deletedCount" : 2 }
```

Conclusion

Congratulations on reading till the end. In this article, you looked at MongoDB's architecture, defined what CRUD operations with MongoDB are, and looked at some examples on how to use the CRUD operations with MongoDB. Hopefully, you now have a solid foundation of how to apply these concepts in your MongoDB project.

However, as a Developer, extracting complex data from a diverse set of data sources like Databases, CRMs, Project management Tools, Streaming Services, Marketing Platforms to your **MongoDB** Database can seem to be quite challenging.