

MongoDB

Mimanshu Maheshwari

September 3, 2024

Contents

1	Introduction	5
1.1	Getting Started with MongoDB Atlas	5
1.1.1	Creating and Deploying on Atlas Cluster	5
1.2	Introduction to MongoDB	6
1.2.1	Key Terms	6
1.3	MongoDB and The Document Model	6
1.3.1	The MongoDB Document Model	6
1.4	Managing Databases, Collections, and Documents in Atlas Data Explorer	6
2	Data Modeling	7
2.1	Introduction to Data Modeling	7
2.2	Types of Data Relationships	8
2.2.1	Different types of relationships data can have:	8
2.2.2	Two main ways to model these relationships:	8
2.3	Modeling Data Relationships	8
2.3.1	Embedding Data in Documents	8
2.3.2	Referencing Data in Documents	9
2.4	Scaling a Data Model	9
2.5	Using Atlas Tools for Schema Help	9
3	Connection to MongoDB Database	11
3.1	Using MongoDB Connection Strings	11
3.1.1	Connection String	11
3.2	Connecting to a MongoDB Atlas Cluster with the Shell	11
3.3	Connecting to a MongoDB Atlas Cluster with Compass	12
3.4	Connecting to a MongoDB Atlas Cluster from an Application	12
3.5	Troubleshooting MongoDB Atlas Connection Errors	12
3.6	Connecting to MongoDB in Java	12
3.6.1	Java driver for MongoDB	12
3.6.2	Configure the application to use the MongoDB Java Synchronous Driver	12
4	MongoDB Create Read Update Delete (CRUD) Operations: Insert and Find Documents	14
4.1	Inserting Documents in MongoDB Collections	14
4.2	Finding documents in MongoDB Collections	15
4.2.1	eq operator	15
4.2.2	in operator	15
4.3	Finding Documents by using Comparison Operators	15
4.3.1	Usage:	15
4.3.2	gt: greater than	16
4.3.3	lt: less than	16
4.3.4	gte: greater than or equal	16
4.3.5	lte: less than or equal	16

4.4	Querying on Array elements in MongoDB	16
4.5	Finding Documents by Using Logical Operators	16

List of Figures

Listings

1.1	account test data	5
1.2	filter on account id	5
1.3	load cluster	6
3.1	shell example	11
3.2	connect to db	11
3.3	pom.xml MongoDB Java Driver	12
3.4	Java connection	12
3.5	Java client database fetch documents	13
3.6	maven command	13
4.1	insertOne using mongosh	14
4.2	output of insertOne using mongosh	14
4.3	insertMany using mongosh	14
4.4	output of insertMany using mongosh	15
4.5	find method in mongosh	15

Chapter 1

Introduction

1.1 Getting Started with MongoDB Atlas

1.1.1 Creating and Deploying on Atlas Cluster

1. Create a Organization
2. Create a Project under Organization
3. Create a cluster → Advanced Settings → Shared M0 cluster → Add username and password → Add you ip as allowed origin.
4. Open Database tab and navigate to your cluster
5. Load default data
6. Navigate to collections tab to see loaded clusters
7. Sample database fetch from analytics accounts table

```
1 {
2   "_id": {
3     "$oid": "5ca4bbc7a2dd94ee5816238c"
4   },
5   "account_id": {
6     "$numberInt": "371138"
7   },
8   "limit": {
9     "$numberInt": "9000"
10  },
11  "products": [
12    "Derivatives",
13    "InvestmentStock"
14  ]
15 }
```

Listing 1.1: account test data

8. To filter on account id:

```
1 {
2   account_id: 794875
3 }
```

Listing 1.2: filter on account id

9. To login into atlas login use `atlas auth login`

10. create cluster with user and password:

```
1 atlas setup --clusterName myAtlasClusterEDU --provider WS --currentIp --skipSampleData --  
  ↪ username myAtlasDBUser --password myatlas-001 --projectId 66d19clf26ef8b512df3b41 |  
  ↪ tee atlas_cluster_details.txt
```

Listing 1.3: load cluster

11. Load sample data: `atlas clusters smapleData load myAtlasClusterEDU`

1.2 Introduction to MongoDB

1.2.1 Key Terms

Document

The basic using of data in MongoDB

Collections

A grouping of documents. Documents in collection are typically similar they don't necessarily have same structure.

Database

A container for Collections

1.3 MongoDB and The Document Model

1.3.1 The MongoDB Document Model

- MongoDB stores data in structures known as documents

Document

Docuement are displayed in JavaScript Object Notation (JSON) format but stored in Binary JavaScript Object Notation (BSON) format. BSON also supports additional data types that are unavailable in JSON

It can support All JSON data types as well as Dates, Numbers, Object Id's, and more!

ObjectId is used to create unique identifires. Every Document requires an *_id* field, which acts as a primary key. If an inserted document doesn't include the *_id* field, MongoDB automatically generates an ObjectId for the *_id* field. MongoDB supports flexible schema model and polymorphic data, this allows us to store documents with different structure together. Documents may contain fields. Fields may contain different types.

Optional Schema validation

Set of constraints on the structure of documents.

1.4 Managing Databases, Colections, and Documents in Atlas Data Explorer

Explore Atlas

Chapter 2

Data Modeling

2.1 Introduction to Data Modeling

Data Modeling is process of defining how data is stored and the relationships and the relationships that exists among different entities in your data.

We refer to Organization of data inside a database as a **Schema**.

To develop schema think about your database and you need to ask these questions:

- What does my application do?
- What data will I store?
- How will users access this data?
- What data will be most valuable to me?

Asking these questions will help you to:

- describe your task as well those of users.
- What you data looks like?
- the relationships among the data.
- The tooling you plan to have.
- The access patterns that might emerge.

A good data model can:

- Make it easier to manage data
- Make queries more efficient
- Use less memory and Computer Processing Unit (CPU)
- Reduce cost

As a general principal of MongoDB data that is accessed together should be stored together.

Collections do not enforce any document structure by default. That means document even those in the same collection can have different structures.

Each document at MongoDB can be different which is known as **Polymorphism**.

Embedded document model enables us to build complex relationships among data. You can normalize your data by using database references.

The key principal here is how your application will use the data rather than how it's stored in the database.

2.2 Types of Data Relationships

2.2.1 Different types of relationships data can have:

- One-to-one
- One-to-many
- Many-to-Many

One-to-One

A relationship where a data entity in one set is connected to exactly one data entity in another set.

One-to-Many

A relationship where a data entity in one set is connected to any number of entities in another set.

Many-to-Many

A relationship where any number of data entities in one set are connected to any number of data entities in another set.

2.2.2 Two main ways to model these relationships:

- Embedding
- Referencing

Embedding

We take related data and insert it into our document.

Referencing

When we refer to documents in another collection in our document. Reference is made by their respective `ObjectID`.

2.3 Modeling Data Relationships

2.3.1 Embedding Data in Documents

Also known as nested document.

- Avoids application joins.
- Provides better performance for read operations.
- Allows developers to update related data in a single write operations.

Issues:

- Embedding data into a single document can create large documents.
- Large documents have to be read into memory in full, which can result in a slow application performance for you end user.
- Continuesly adding data without limit creates **unbounded documents**.
- Unbounded Documents may exceed the BSON document threshold of 16MB. This is schema anti-patterns which you should avoid.

2.3.2 Referencing Data in Documents

When we want to store data in two different collections but also ensure that those collections are related, we can use **References**. References save the *_id* field of one document in another document as a link between the two. They are simple and sufficient for most use cases. Using references is called **linking** or **data normalization**. It avoids duplication of data and in most cases smaller documents, however referencing data you'll need to query from multiple documents costing extra resources and impacts read performance.

Embedding	Referencing
✓ Single query to retrieve data	✓ No duplication
✓ Single operation to update/delete data	✓ Smaller Documents
! Data Duplication	! Need to join data from multiple documents
! Large documents	

2.4 Scaling a Data Model

Optimum efficiency of:

- query result depends on
- memory usage
- CPU usage
- storage
- When creating a data model, avoid documents that are unbounded. Document size grows infinitely. Problems as the array grows larger in embedded document.
 - It will take up more space in memory
 - may impact write performance as entire document is rewritten into MongoDB data storage.
 - it will be difficult to perform pagination of that array.
 - Maximum document size of 16MB will lead to storage problems.
- benefit is that we can retrieve all document in single read.
- Avoid:
 - More than the document size limit of 16 MB.
 - Poor query performance.
 - Poor write performance.
 - Too much memory being used.

2.5 Using Atlas Tools for Schema Help

Schema design patterns are guidelines that help developers plan, organize, and model data. Schema anti-patterns result in suboptimal performance and non-scalable solutions. Eg:

- massive arrays.
- Massive number of collections.
- Bloated documents.
- Unnecessary indexes.

- Queries without indexes.
- Data that's accessed together, but stored in different collections.

Tools in Atlas are **Data Explorer** and **Performance Advisor**

Chapter 3

Connection to MongoDB Database

3.1 Using MongoDB Connection Strings

Connection string can be used to connect from MongoShell, MongoDB Compass, or any other application. It provides standered format and DNS seed list format.

Standered Format

Used to connect to standalone clusters, replica sets, or sharded clusters.

DNS Seed List Format

Released in MongoDB 3.6.

3.1.1 Connection String

It allows to provide a DNS server list to connection string. Gives more flexibility of deployment. Ability to change servers in rotation without reconfiguring clients.

In atlas it is present in Database tab → Connect → Connect to application.

Connection String: mongodb+srv://<username>:<password>@cluster0.usqsf.mongodb.net/?retryWrites=true

↪ &w=majority If port number is not specified MongoDB will default to 27017.

3.2 Connecting to a MongoDB Atlas Cluster with the Shell

mongosh: MongoDB shell is a nodejs read-eval-print loop (REPL) environment. It gives us access to JavaScript variables, functions, conditionals, loops and Control flow statements inside shell.

```
1 const greetingArray = ["hello", "world", "welcome"];
2 const loopArray = (array) => array.forEach(el => console.log(el));
3 loopArray(greetingArray);
```

Listing 3.1: shell example

In atlas it is present in Database tab → Connect → Connect with the MongoDB Shell.

```
1 atlas clusters connectionStrings describe myAtlasClusterEDU
2 MY_ATLAS_CONNECTION_STRING=$(atlas clusters connectionStrings describe myAtlasClusterEDU | sed "1
  ↪ d")
3 mongosh -u myAtlasDBUser -p myatlas-001 &MY_ATLAS_CONNECTION_STRING
```

Listing 3.2: connect to db

3.3 Connecting to a MongoDB Atlas Cluster with Compass

In atlas it is present in Database tab → Connect → Connect using MongoDB Compass.

3.4 Connecting to a MongoDB Atlas Cluster from an Application

MongoDB drivers

Connect our application to our database by using a connect string. You can find the [mongodb drivers list here](#).

3.5 Troubleshooting MongoDB Atlas Connection Errors

3.6 Connecting to MongoDB in Java

MongoDB maintains Java Driver for Synchronous and Asynchronous java application code.

3.6.1 Java driver for MongoDB

```
1 <properties>
2   <mongodb-driver-sync.version>4.7.1</mongodb-driver-sync.version>
3   <mongodb-crypt.version>1.4.1</mongodb-crypt.version>
4 </properties>
5 <dependencies>
6   <dependency>
7     <groupId>org.mongodb</groupId>
8     <artifactId>mongodb-driver-sync</artifactId>
9     <version>${mongodb-driver-sync.version}</version>
10  </dependency>
11  <dependency>
12    <groupId>org.mongodb</groupId>
13    <artifactId>mongodb-crypt</artifactId>
14    <version>${mongodb-crypt.version}</version>
15  </dependency>
16 </dependencies>
```

Listing 3.3: pom.xml MongoDB Java Driver

3.6.2 Configure the application to use the MongoDB Java Synchronous Driver

```
1 package com.mongodb.quickstart;
2
3 import com.mongodb.ConnectionString;
4 import com.mongodb.MongoClientSettings;
5 import com.mongodb.MongoException;
6 import com.mongodb.ServerApi;
7 import com.mongodb.ServerApiVersion;
8 import com.mongodb.client.MongoClient;
9 import com.mongodb.client.MongoClients;
10 import com.mongodb.client.MongoDatabase;
11 import org.bson.Document;
12
13 public class Connection {
14   public static void main(String[] args){
15     ConnectionString connectionString = new ConnectionString("mongodb+srv://student-joe:<password>
16       ↪ @sandbox.svlx7.mongodb.net/?retryWrites=true&w=majority");
17     MongoClientSettings settings = MongoClientSettings.builder()
18       .applyConnectionString(connectionString)
19       .serverApi(ServerApi.builder()
```

```

19     .version(ServerApiVersion.V1)
20     .build())
21     .build();
22     try(MongoClient mongoClient = MongoClient.create(settings)){
23         MongoDB database = mongoClient.getDatabase("admin");
24         database.runCommand(new Document("ping", 1));
25         System.out.println("Pinged your deployment. You successfully connected to MongoDB!");
26     } catch (MongoException e){
27         e.printStackTrace();
28     }
29 }
30 }
31 }

```

Listing 3.4: Java connection

```

1 package com.mongodb.quickstart;
2 import com.mongodb.client.MongoClient;
3 import com.mongodb.client.MongoClients;
4 import org.bson.Document;
5
6 import java.util.ArrayList;
7 import java.util.List;
8 public class Connection {
9     public static void main(String[] args){
10         String connectionString = System.getProperty("mongodb.uri");
11         try(MongoClient mongoClient = MongoClient.create(connectionString)){
12             MongoClientSettings settings = MongoClientSettings.builder()
13                 .listDatabases().into(new ArrayList<>());
14             database.forEach(db -> System.out.println(db.toJson()));
15         }
16     }
17 }

```

Listing 3.5: Java client database fetch documents

```

1 mvn compile exec:java -Dexec.mainClass="com.mongodb.quickstart.Connection" -Dmongodb.uri="mongodb
  ➔ +srv://m001-student:sandbox1995@cluster0.6sfcv.mongodb.net/test?w=majority"

```

Listing 3.6: maven command

Chapter 4

MongoDB CRUD Operations: Insert and Find Documents

4.1 Inserting Documents in MongoDB Collections

There are two methods `insertOne()` and `insertMany()`.

Usage: `db.<collection>.insertOne()`. If the collection doesn't exist then MongoDB creates the collection when we use `insertOne()`. Eg:

```
1 db.grades.insertOne({
2   student_id: 546799,
3   scores: [
4     {
5       type: "quiz",
6       score: 50,
7     },
8     {
9       type: "homework",
10      score: 70,
11    }
12  ]
13 })
```

Listing 4.1: insertOne using mongosh

```
1 {
2   acknowledged: true,
3   insertedId: ObjectId("...")
4 }
```

Listing 4.2: output of insertOne using mongosh

```
1 db.grades.insertMany([
2   {
3     student_id: 546799,
4     scores: [
5       {
6         type: "quiz",
7         score: 50,
8       },
9       {
10        type: "homework",
11        score: 70,
12      }
13    ]
14  }
15 ])
```

```

14 },
15 {
16   student_id: 546800,
17   scores: [
18     {
19       type: "quiz",
20       score: 50,
21     },
22     {
23       type: "homework",
24       score: 70,
25     }
26   ]
27 }
28 ])

```

Listing 4.3: insertMany using mongosh

```

1 {
2   acknowledged: true,
3   insertedIds: [
4     '0': ObjectId("..."),
5     '1': ObjectId("..."),
6   ]
7 }

```

Listing 4.4: output of insertMany using mongosh

4.2 Finding documents in MongoDB Collections

Usage: db.<collection>.find()

```

1 use training; // database
2 db.zips.find(); // zips are collection
3 // use "it" for iterating from more result

```

Listing 4.5: find method in mongosh

4.2.1 eq operator

```

1 { field: { $eq: <value> } }
2 # or
3 { field: <value> }

```

4.2.2 in operator

```

1 { field: { $in: [<value1>, <value2>, <value3>, ...] } }

```

4.3 Finding Documents by using Comparison Operators

4.3.1 Usage:

```

1 { field: { <operator>: <value> } }

```


4.3.2 gt: greater than

```
1 { field: { $gt : <value> } }
```

4.3.3 lt: less than

```
1 { field: { $lt : <value> } }
```

4.3.4 gte: greater than or equal

```
1 { field: { $gte : <value> } }
```

4.3.5 lte: less than or equal

```
1 { field: { $lte: <value> } }
```

4.4 Querying on Array elements in MongoDB

```
1 db.accounts.find({"products": "InvestmentStock"})
```

\$elemMatch ensures product field is an array.

```
1 db.accounts.find({
2   products: {
3     $elemMatch: {
4       <query1>,
5       <query2>,
6       <query3>,
7       ...
8     }
9   }
10 })
```

```
1 db.accounts.find({
2   products: {
3     $elemMatch: { $eq: "InvestmentStock" }
4   }
5 })
```

4.5 Finding Documents by Using Logical Operators