How to integrate with CinemaBooking's Movie Database

User: Integrator
Password: securepassword

In order to connect to the MongoDB database Mongosh must be installed on your machine. The official Mongo documentation provides detailed guides on how to install mongosh on your machine using numerous methods and operating systems (MacOS, Windows and Linux).

Examples:
MacOS using Homebrew:
https://www.mongodb.com/docs/mongodb-shell/install/?operating-system=macos&macos-installation-method=homebrew

Windows using the MSI Installer:
https://www.mongodb.com/docs/mongodb-shell/install/?operating-system=windows&windows-installation-method=msiexec

Linux and Ubuntu:
https://www.mongodb.com/docs/mongodb-shell/install/?operating-system=linux&linux-distribution=ubuntu&ubuntu-version=bionic

Once installed open your terminal and type in the following to connect remotely to the MongoDB connection containing the database:

mongosh --host db.cinemabooking.com -u [Username] -p [Password] --authenticationDatabase CinemaBooking

Replace [Username] and [Password] with the credentials provided at the top of this document.

Once connected through the shell access the database by typing in:

use CinemaBooking

After typing in the above the prompt in the terminal should change to **Cinemabooking>**.

If that is the case, you're now ready to query the database. Let's go over how the data is structured.

As of now the Database consists of two collections: 'members' and 'movies'

The provided login only allows you to read and write data in the ‘movies’ collection. You are not able to read, write nor delete data in the ‘members’ collection.

Business rules and Document Structure

MongoDB is a NoSQL document based database consisting of collections (E.g. ‘movies’) and documents (a specific movie). The data model is BSON, which should be very familiar if you know JSON. Movie documents have an hierarchical and nested structure. Meaning that all the relevant data are contained within the movie document. Here is a concrete example of how a movie document is structured:


```
_id: ObjectId('68426f8d9f12d100818c9325')
Director : Object
  Hash : "e31ef0fa1e3537a51a15dadabe74355880d009034f1fb3e5b305e56ba666121e"
  FirstName : "Frank"
  LastName : "Darabont"
Genres : Array (1)
  0: Object
    Name : "Drama"
Actors : Array (3)
  0: Object
    Hash : "41fc4cf53208ac335ae0e0ffb45850478477a33d2979fc8758a290e35292d513"
    FirstName : "Tim"
    LastName : "Robbins"
  1: Object
    Hash : "8d0ba104fb1ff719bbc2afdc276d13ba94fc7f0b3c9aa47a5b535be809b986e1"
    FirstName : "Morgan"
    LastName : "Freeman"
  2: Object
    Hash : "ea47056ec218fb0d62a64f42b57f3aa916c03dfc7ebc3d24cb5fceeee6e278ff"
    FirstName : "Bob"
    LastName : "Gunton"
Runtime : 142
Summary : "A banker convicted of uxoricide forms a friendship over a quarter cent…"
IMDBRating : 9.300000190734863
Year : 1994
Title : "The Shawshank Redemption"
```


At top level meta data about the movie is defined. These are all REQUIRED fields, meaning that if you ever want to create a new ‘movie’ document these must be provided. Additionally the document has some validation configured. Meaning that some fields are required, must be of a specific datatype and also have some business rules. If a movie document is not adhering to these validation rules an error will be thrown and the document won’t be added to collection. Here is an overview of the validation schema:

All fields below are required:
Top-level movie attributes:

1. Runtime: (integer) must be a number between 1 and 500. The number represents minutes.
2. Summary: (string) number of characters must be between 50 and 300.
3. IMDBRating: (double) must be a double between 0 and 10. Decimals are allowed.

4. Year: (integer) must be a number between 1800 and 2050.
5. Title: (string) number of characters must be between 1 and 179. Must be unique. Movies with the same title cannot be added.

Movies however can also contain nested objects. In this case ‘Director’, ‘Actors’ and ‘Genres’.

Director:
A document can as of now only have one director. Each director must have a first name and a last name:

Required attributes:

1. FirstName: (string) number of characters must be between 1 and 30.
2. LastName: (string) number of characters must be between 1 and 30.

When creating a movie document a director must be specified.

Actors:
A movie can have multiple actors and nested inside the movie document through the ‘actors’ array. When adding one or more actors to this array the following validation is in place:

Required attributes:
1. FirstName: (string) number of characters must be between 1 and 30.
2. LastName: (string) number of characters must be between 1 and 30.

A movie document can’t have actors with the same firstname and lastname.

Genres:
A movie can have multiple genres. Therefore a movie document also contains a ‘genres’ array. When adding a genre to this array the following validation is in place:

Required attributes:
1. Name: (string) number of characters must be between 1 and 20.

A movie document can't have genres with the same name.

Additional notes:
When creating a movie the ‘genres’ array and ‘actors’ must be set. However these arrays are not required to contain any objects. This is to allow users to create a movie document for announced movies where all info isn't available yet.

Querying the Database

Now that the validation rules and the ‘movie’ document structure has been specified, let's go through how to query the database.

After connecting and changing to the CinemaBooking database using ‘use CinemaBooking’, we can query the database.

Reading data:

Since MongoDB is schemaless the fields on every document might differ. Hence its a good idea to use db.movies.find() to get an idea of the document structure for the movie you want to query/edit.

To fetch all available info on a movie simply use:

```
db.movies.find({ Title: "Movie title" });
```

or:

```
db.movies.findOne({ Title: "Movie title" });
```

.find() returns all documents that matches the query. If you’re only interested in the first match - use findOne():

```
db.movies.find({ Title: "Inception" })
db.movies.findOne({ Title: "Inception" })
```

Fetching by another attribute

If you want to find documents by another attribute simply replace ‘Title’ with another field:

```
db.movies.find({ Year: 2010 })
```

This will return all movies released in the year 2010.

Specifying data to be returned from a query

If you’re only interested in returning specific data from a query, MongoDB allow u to specify what data to be returned:

```
db.movies.find(
  { Title: "Inception" }, // find by title (could be any other attribute)
  { IMDBRating: 1, _id: 0 } // Include IMDBRating and exclude _id
)
```

As above illustrates, if you want to include an attribute in the query result simply provide the name of the attribute (E.g. IMDBRating) and then the number ‘1’ to specify that u want it to be returned. _id is included by default - to exclude it we use the ‘0’. U can also include multiple attributes this way:

```
db.movies.find(
  { Title: "Inception" }, // find by title (could be any other attribute)
  { IMDBRating: 1, Title: 1, Actors: 1, _id: 0 } // Include IMDBRating, Title, Actors and exclude _id
)
```

Returning attributes from nested objects:
To specify what attributes from a nested objects you want returned from the query, u can use a couple of approaches depending on the use case.

If u only want to return a field for a nested object simply use the dot notation:

```
db.movies.find(
  { Title: "Inception" },
  { "Director.LastName": 1, _id: 0 }
)
```

This returns the last name of the director.

And for arrays:

```
db.movies.find(
  { Title: "Inception" },
  { "Actors.LastName": 1, _id: 0 }
)
```

This returns the last name of all actors.

Returning a specific field from a specific actor for a specific movie:
If u want to extract a specific field from a specific object inside the actors array u can do the following:

```
db.movies.aggregate([
  { $match: { Title: "Inception" } }, // movie where u want to find the actor
  { $unwind: "$Actors" }, // Should match the arrays you want to search
  { $match: { "Actors.LastName": "DiCaprio" } }, // Find actor by last name
  { $project: { _id: 0, FirstName: "$Actors.FirstName" } } // Return the actor’s first name
```

])

Creating New Movies

When inserting a movie remember the structure and validation rules described in the beginning of the guide. To insert a movie you can use this query for inspiration:

```
db.movies.insertOne({
  Title: "Wall-E",
  Summary: "In a distant future, Earth has been abandoned and left covered in trash. WALL-E, a small waste-collecting robot, continues to clean up the planet, until he discovers a new purpose when he meets the probe robot EVE.",
  Runtime: 98,
  IMDBRating: 8.4,
  Year: 2008,
  Director: {
    FirstName: "Andrew",
    LastName: "Stanton"
  },
  Genres: [
    { Title: "Animation" },
    { Title: "Adventure" },
    { Title: "Family" },
    { Title: "Sci-Fi" }
  ],
  Actors: [
    { FirstName: "Ben", LastName: "Burtt" },
    { FirstName: "Elissa", LastName: "Knight" }
  ]
})
```

If the Actors and genres are unknown, simply the arrays empty:

```
db.movies.insertOne({
  Title: "Wall-E",
  Summary: "In a distant future, Earth has been abandoned and left covered in trash. WALL-E, a small waste-collecting robot, continues to clean up the planet, until he discovers a new purpose when he meets the probe robot EVE.",
  Runtime: 98,
  IMDBRating: 8.4,
  Year: 2008,
  Director: {
```

```
    FirstName: "Andrew",
    LastName: "Stanton"
  },
  Genres: [],
  Actors: []
})
```

To add an actor to an existing movie document simply:

```
db.movies.updateOne(
  { Title: "Wall-E" },  // title of the movie
  {
    $push: {
      Actors: { FirstName: "Ben", LastName: "Burtt" } // insert actor
    }
  }
)
```

Use:
Genres: { Name: “Animation”} to insert an genre.

Updating a movie:

To update a movie field:

```
db.movies.updateOne(
  { Title: "Wall-E" },        // filter by title
  { $set: { IMDBRating: 8.5 } }  // change the IMDBRating
)
```

To update a nested object:

```
db.movies.updateOne(
  { Title: "Wall-E" }, // filter by title
  { $set: { "Director.LastName": "Doe" } } // set lastname to Doe
)
```

To update a nested object inside an array:

```
db.movies.updateOne(
 { Title: "Inception", "Actors.LastName": "DiCaprio" }, find by movie and actor lastname
  { $set: { "Actors.$.FirstName": "Leo" } }  set first name to Leo
)
```