MongoDB Design and Development Guide

Author: Richard Krasso
Course: WEB 335 Introduction to NoSQL
Date: May 28, 2024

Table of Contents

Guide 1: Setting up MongoDB Atlas Account

Every effort has been made to keep this guide current, but as with anything else in life, it is possible that the steps outlined in this guide could become outdated. To mitigate this, direct links to the official documentation have been provided in each section. These official resources are regularly updated and should be your go-to reference if you run into issues while following the steps in this guide. If you notice something is out of date, please send your instructor an email, so the document can be updated with the most current information.

MongoDB Atlas: Signup

1. Visit: https://account.mongodb.com/account/register and create an Atlas account.
2. Step-by-step instructions:
    a. Create an Atlas Account.

MongoDB Atlas: Deploy a Free Cluster

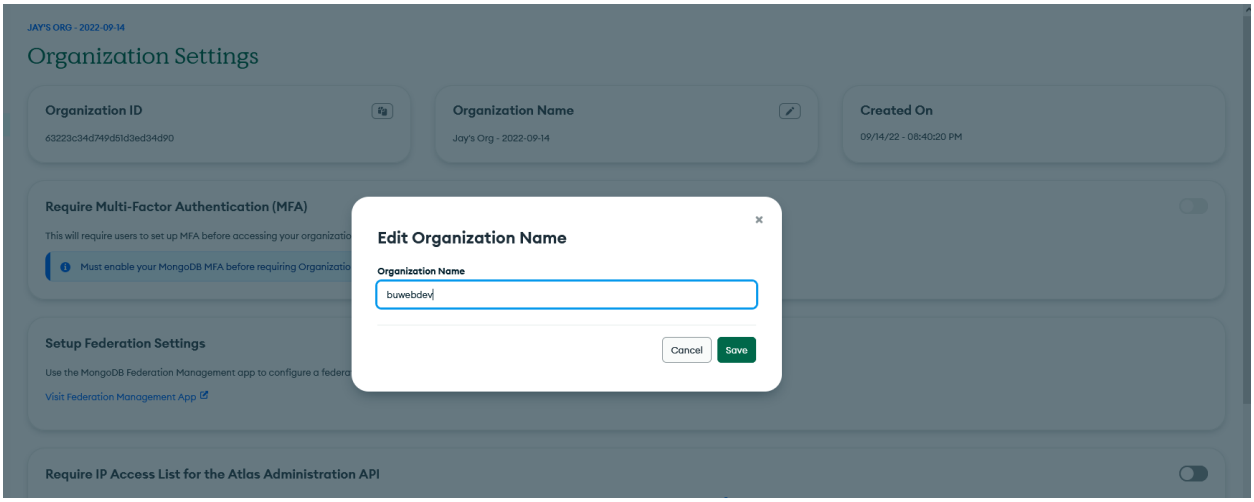
1. Step-by-step instructions:
    a. Deploy a Free Cluster
    b. Name the cluster: BellevueUniversity

Make sure you select the "Shared" deployment, which is their free cluster. MongoDB provides users with a free cluster, which is all you will need for this cohort.

MongoDB Atlas: Rename an Organization

1. Expand the "Organizations" menu in the navigation bar.
2. Click View All Organizations.
3. Rename an Organization
    a. Rename the organization buwebdev

Figure 1.

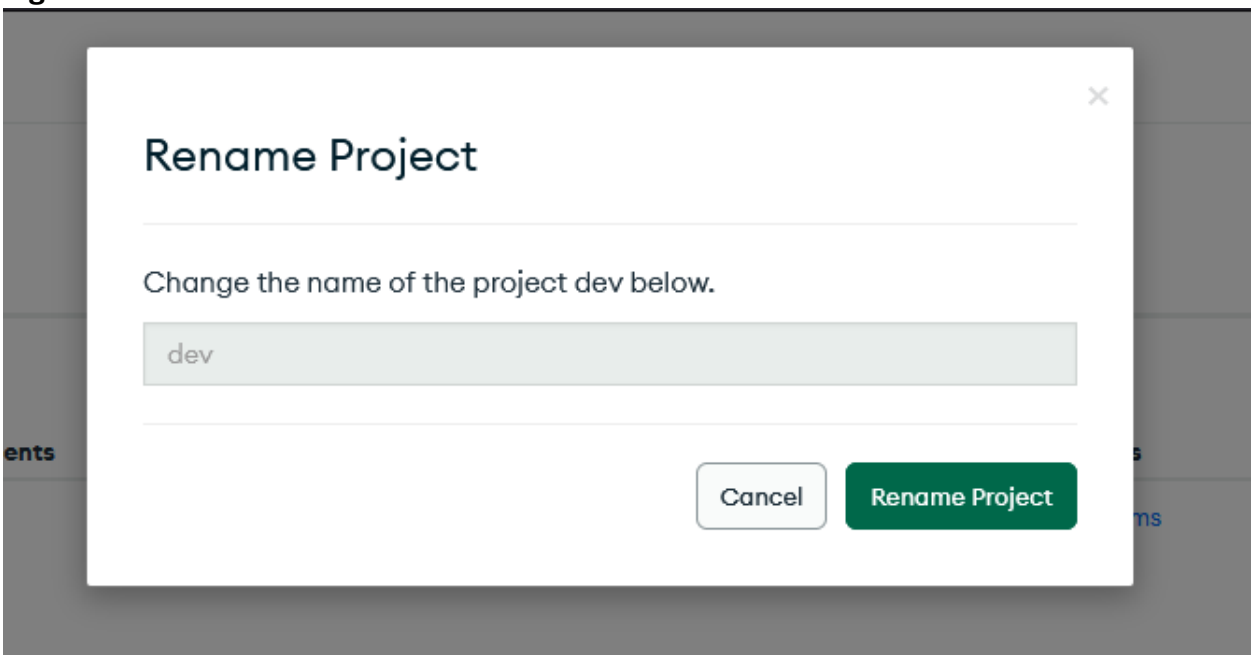


MongoDB Atlas: Rename Project

1. View Projects

2. Click on the ellipse icon and select “Edit Project.”
    a. Rename the project to dev

Figure 2.



If you run into issues or cannot locate the Edit Project option, you can always create a new project and delete the one that was created by default.

- Create a Project

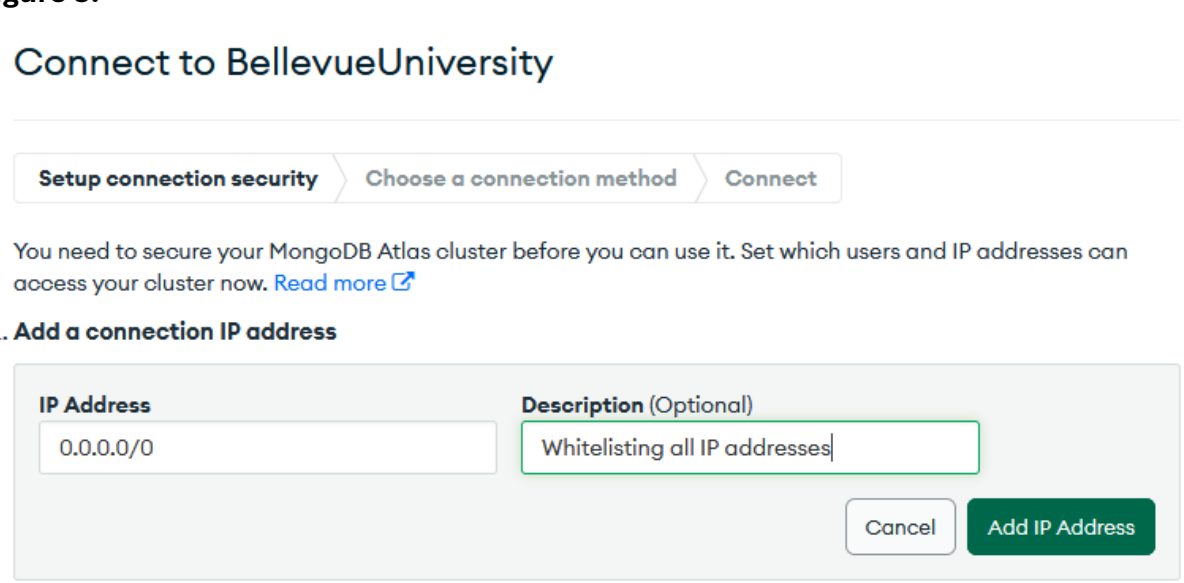
- Delete a Project

MongoDB Atlas: Whitelist IP Addresses

1. Select Database from the left-hand menu pane.
2. Step-by-step instructions:
    a. Add Your Connection IP Address to Your IP Access List
3. Make sure you select Allow Access from Anywhere.
    a. Verify the IP Address 0.0.0.0/0 is entered into the IP Address input field.
    b. Set a description to Whitelisting all IP addresses.

In a production database you would not do this. Instead, you would only add the IP addresses of the applications connecting to your cluster, which in our case would only be the Node.js applications we write in this cohort. I repeat, in a production database, do not do this!

Figure 3.



If you were not able to follow the above steps, because you could not locate the **Allow Access from Anywhere** button, you can access the IP settings through the **Network Access** link in the left-hand menu pane. Adding a new IP address is handled through the **ADD IP ADDRESS** button.

MongoDB Atlas: Create a new Database

1. Step-by-step instructions

    a. Create a Database

    b. Name the database using the following convention: [course/projectName]DB. For example, web340DB, web420DB, web450DB, nodebucketDB, bcrsDB, etc.,

    c. Collection names should be plural and be appropriate to the type of data they represent. For example, users, employees, fruits, etc.,

MongoDB Atlas: Create a Database User

1. Step-by-step instructions:

    a. Procedure

    b. Name the database user web340_admin

All database users in this cohort must use the following naming convention: [courseName]_admin. For example, web340_admin, web420_admin, and web450_admin.

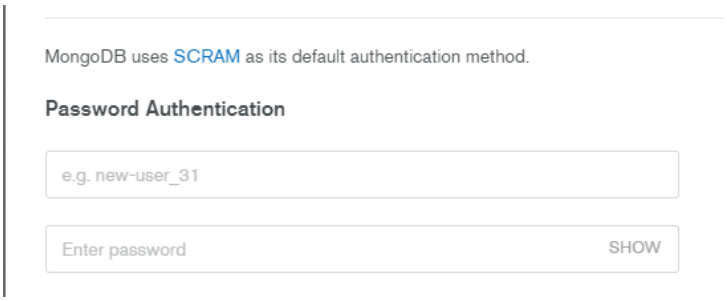
The following steps outline how to create a new database user in MongoDB Atlas. These steps are included because once the initial database user is created the above tutorial no longer applies. New database users will be created in WEB 335 and WEB 450.

1. Sign-in to MongoDB Atlas using your personal account.

2. Under the left-hand menu, there should be an option for Database Access. Select it and choose Add New Database User.

3. Under the Add New Database User dialog window, set the Authentication Method to Password and enter a username and password under the Password Authentication section (see Figure 4).

Figure 4.



4. Add a username/password of your choosing, but make sure you either write it down or it is easy to remember. Next, scroll down to the bottom of the dialog window, and select Add User.

5. The list of users should update with the user you just added.

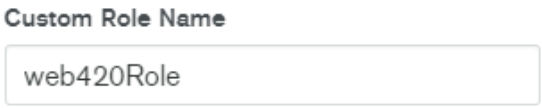
MongoDB Atlas: Custom Roles
The following steps outline how to create custom roles in MongoDB Atlas:

1. Sign-in to MongoDB Atlas using your personal account.

2. Under the left-hand menu, there should be an option for Database Access. Select it and choose the Custom Roles tab. Next, select Add New Custom Role.

3. Under the Add Custom Role dialog, enter a name for the custom role under the Custom Role Name input field (see Figure 5).

Figure 5.



All custom roles in this cohort must use the following naming convention: web340Role, web335Role, web420Role, and web450Role.

4. Under the Action or Role select menu, choose Collection Actions.

5. Under the Database input field enter the name of the database you want this custom role to be assigned to (remember, the name you enter must match an actual database in your cluster).

6. Finally, select Add Custom Role.

7. The list of custom roles should update with the role you just added.

MongoDB Atlas: Custom Role Assignment
The following steps outline how to assign a custom role to an existing MongoDB user:

1. Sign-in to MongoDB Atlas using your personal account.

2. Under the left-hand menu, there should be an option for Database Access. Select it and choose the Edit button next to the user you want the custom role assigned to.

3. Under the Database User Privileges section, expand the Custom Roles section. Next, click on the Add Custom Role button and a select menu should appear. Expand the menu and choose the role you want to assign to this user.

4. Update the user record by clicking on the Update User button.

Guide 2: Using the MongoDB Shell

Every effort has been made to keep this guide current, but as with anything else in life, it is possible that the steps outlined in this guide could become outdated. To mitigate this, direct links to the official documentation have been provided in each section. These official resources are regularly updated and should be your go-to reference if you run into issues while following the steps in this guide. If you notice something is out of date, please send your instructor an email, so the document can be updated with the most current information.

mongosh: MongoDB Connection

1. Sign-in to your MongoDB Atlas account.

2. In the left-hand menu pane, choose “Database.”

3. Select the “Connect” button.

4. Choose the “Shell.”

5. Under option 2, copy the connection string.

6. Open a new terminal window and paste in the copied string.

7. Replace the value <username> with the username of the user we created. In our case, we created a user named **web335_user**.

8. Once you press enter the program will prompt you to enter the password associated with the custom user we created. In this example, I created the user with a password of **s3cret**.

9. By default, mongosh will sign you into the "test" database. To switch between databases use: use <database_name>. For example, **use web335DB**. If enter a database name that does not exist, it will still connect you, but there will not be any collections. You can view a list of collections by using: **show collections;**

10. To exit the mongosh program you can either enter Ctrl+c or .exit

mongosh: Loading a db script

- Resource: https://www.mongodb.com/docs/mongodb-shell/write-scripts/

Before you load the users.js script make sure you have created a collection named users.

1. Save the script to a location of your choosing (note: keep track of where you saved the file, because in the next step you will need to use the CD command from the CLI to access the file's location).

2. Navigate to the file's location using the CD CLI command.

3. Sign-in to mongosh using the connection string from MongoDB Atlas (note: this is the connection you have been using to access your MongoDB database from mongosh).

4. Assuming you are in the current directory where the script was saved, you call the load() function. This use MongoDB's shell to load the specified script. If the script was loaded correctly, a value of true will be printed to the console window.

5. Test the script:

    a. **findOne()**: db.users.findOne({firstName: 'Claude'});

```
[Atlas Cluster0-shard-0 [primary] web335DB> db.users.findOne({firstName: 'Claude']
});
{
  _id: ObjectId('62b5f63a65d737ba9a792f60'),
  firstName: 'Claude',
  lastName: 'Debussy',
  employeeId: '1012',
  email: 'cdebussy@me.com',
  dateCreated: ISODate('2022-06-24T17:36:57.841Z')
}
```

b. **find()**: db.users.find();

```
Atlas Cluster0-shard-0 [primary] web335DB> db.users.find();
[
  {
    _id: ObjectId('62b5f63965d737ba9a792f5b'),
    firstName: 'Johann',
    lastName: 'Bach',
    employeeId: '1007',
    email: 'jbach@me.com',
    dateCreated: ISODate('2022-06-24T17:36:57.841Z')
  },
  {
    _id: ObjectId('62b5f63965d737ba9a792f5c'),
    firstName: 'Ludwig',
    lastName: 'Beethoven',
    employeeId: '1008',
    email: 'lbeethoven@me.com',
    dateCreated: ISODate('2022-06-24T17:36:57.841Z')
  },
  {
    _id: ObjectId('62b5f63965d737ba9a792f5d'),
    firstName: 'Wolfgang',
    lastName: 'Mozart',
    employeeId: '1009',
```

Guide 3: Executing MongoDB Queries

In this guide, I will demonstrate how to execute basic MongoDB queries through the Shell program. This guide assumes you have already completed the MongoDB Atlas and Mongosh setups. If you have not, complete those before proceeding with the examples in this guide.

1. Find all documents in a collection: db.[collectionName].find();

```
Atlas Cluster0-shard-0 [primary] web335DB> db.users.find();
```

There are two types of queries in MongoDB for retrieving documents. findOne() and find(). The findOne() query returns a single document matching the filter criteria (first actual parameter of the function call). The find() function returns a collection of documents matching the filter criteria (first actual parameter of the function call). In either operation, if no actual parameter is supplied for the first argument of the function, the entire data set is returned for find() and the first record is returned for the findOne(). The first argument in the find() and findOne() functions are officially

referred to as the “query document.” It is an object with a set of fields and values that MongoDB uses to search for matching documents in the collection.

2. Create a new document:

   user = {firstName: 'Bryce', lastName: 'Wane', employeeId: 'BW123', email: 'bryce.wane@supermail.com', dateCreated: new Date()};

   db.users.insertOne(user);

```
{
[  firstName: 'Bryce',
   lastName: 'Wane',
   employeeId: 'BW123',
   email: 'bryce.wane@supermail.com',
   dateCreated: ISODate('2024-05-28T20:49:14.375Z')
}
Atlas Cluster0-shard-0 [primary] web335DB> db.users.insertOne(user);
{
   acknowledged: true,
   insertedId: ObjectId('6656435b6ff598471d9c1099')
[}
```

   In this code example, notice how I am using JavaScript’s built-in **new Date()** function to assign today’s date to the **dateCreated** property of the user object. In MongoDB’s Shell, most built-in JavaScript functions are allowed. You will know the insertion was successful if you see a value of “true” for the “acknowledged” property and a returned auto-generated “ObjectId.”

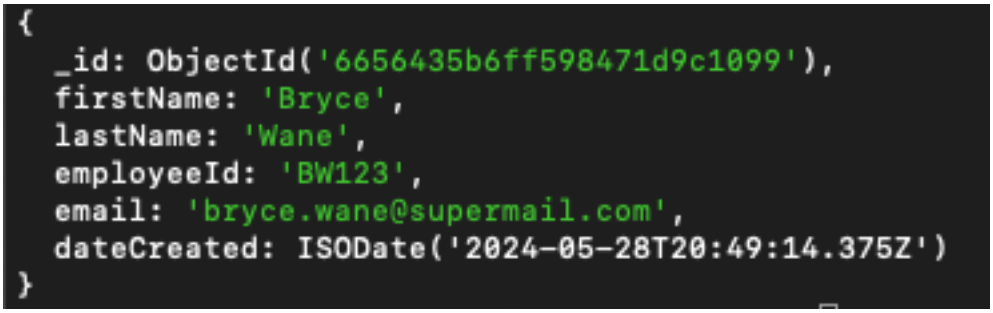
3. Query the user’s collection by firstName (this query searches the users collection for a user document with a firstName matching the filtered actual parameter).

   db.users.findOne({firstName: 'Bryce'});

```
{
   _id: ObjectId('6656435b6ff598471d9c1099'),
   firstName: 'Bryce',
   lastName: 'Wane',
   employeeId: 'BW123',
   email: 'bryce.wane@supermail.com',
   dateCreated: ISODate('2024-05-28T20:49:14.375Z')
}
```

4. Query the user’s collection by multiple fields:

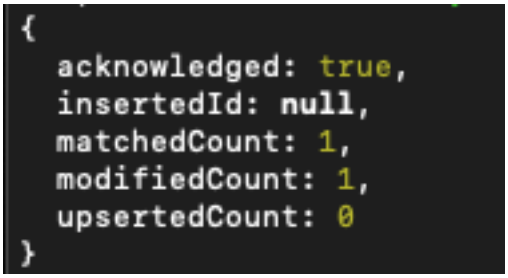
db.users.findOne({firstName: 'Bryce', lastName: 'Wane'});



The more fields you use in the query document, the more specific a search becomes. You can narrow down the results to only documents matching a stricter criterion.

5. updateOne():

db.users.updateOne({employeeId: 'BW123'}, {$set: {email: 'bryce.isnotbatman@supermail.com'}});

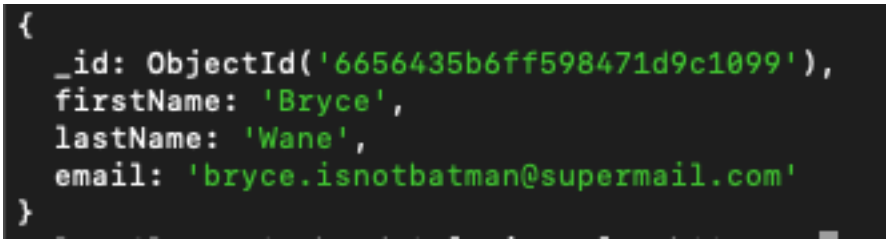


You will know the update was successful if you see a "1" next to the modified count. But, just to be sure, it is always advised to run a findOne() operation to confirm the update.

db.users.findOne({employeeId: 'BW123'});

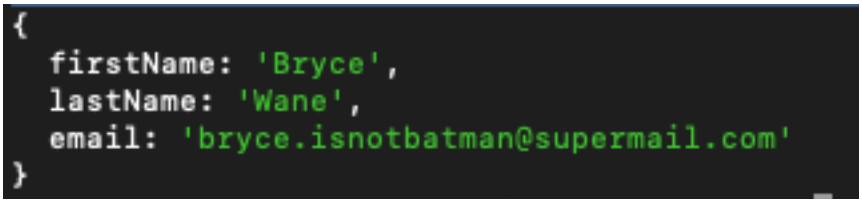
6. findOne() with projections:

db.users.findOne({employeeId: 'BW123'}, {firstName: 1, lastName: 1, email: 1});



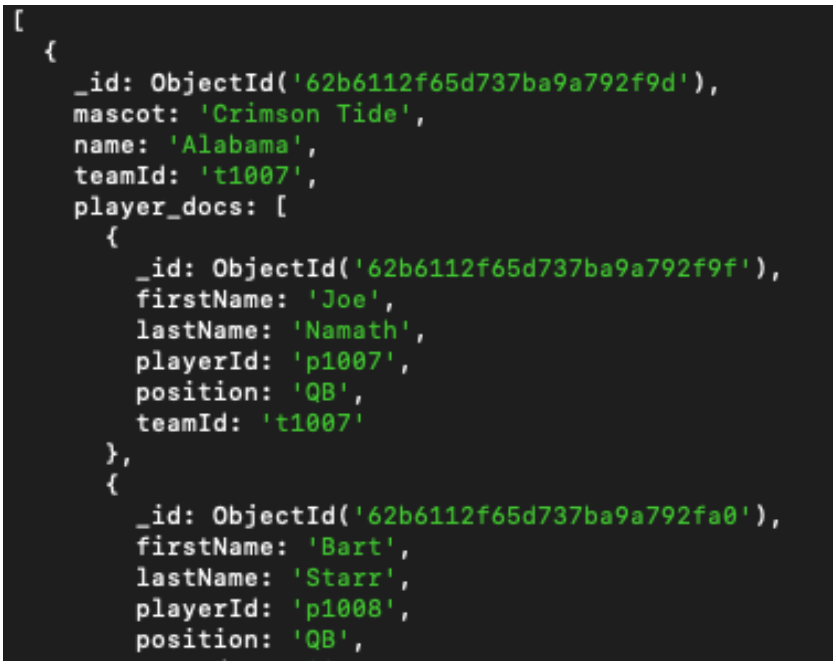
In this example, the auto-generated ObjectId is being returned. By default, the ObjectId is always returned. To omit it from the result set, set the _id property to 0. For example,

db.users.findOne({employeeId: 'BW123'}, {_id: 0, firstName: 1, lastName: 1, email: 1});



7. Aggregate query:

db.teams.aggregate([{$lookup: { from: 'players', localField: 'teamId', foreignField: 'teamId', as: 'player_docs'} }]);

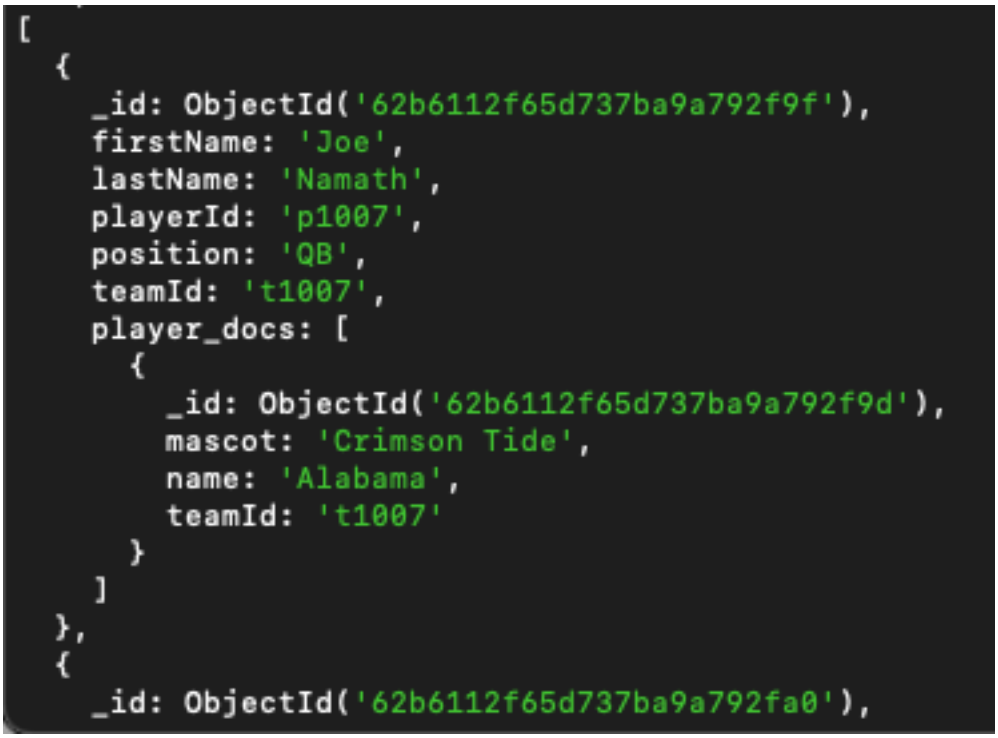


The position of the collections in the look query will determine how the data is printed out. In this example, we are trying to achieve a print out of:

Team
Players

To achieve this, we use the teams collection as the base query operator (db.teams.aggregate) and the players collection as the lookup. Let's reverse the order so you can see the difference.

db.players.aggregate([{$lookup: { from: 'teams', localField: 'teamId', foreignField: 'teamId', as: 'player_docs'} }]);

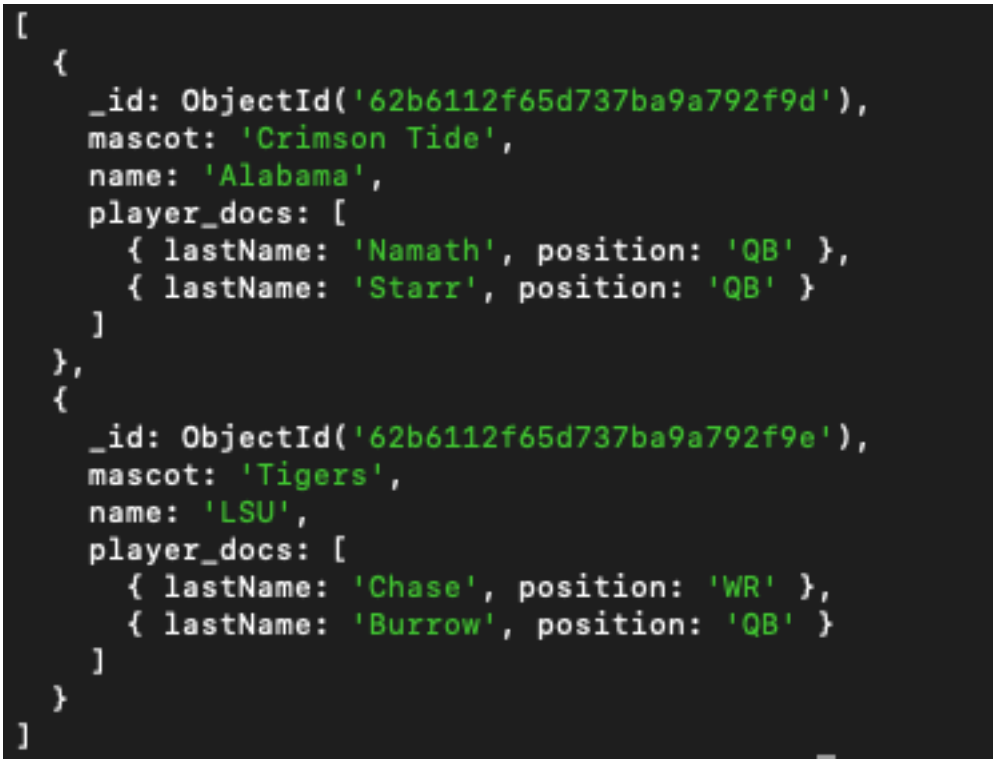


Notice now the player_docs property is the actual team assigned to the player, which is in reverse order. That is, each player will be listed and the team will be a nested document. Whenever you are using aggregates, always make sure the order of how the data is printed is appropriate.

In either example, the lookup operation is joining two collections by a matching field. In this example, the matching field is teamId. If there Is not a matching field in either document, you would not be able to use the lookup function to connect related collection documents.

8. Aggregate query with projections:

   db.teams.aggregate([{$lookup: { from: 'players', localField: 'teamId', foreignField: 'teamId', as: 'player_docs'} }, {$project: {'mascot': 1, 'name': 1, 'player_docs.lastName': 1, 'player_docs.position': 1 } }]);



```
[
  {
    _id: ObjectId('62b6112f65d737ba9a792f9d'),
    mascot: 'Crimson Tide',
    name: 'Alabama',
    player_docs: [
      { lastName: 'Namath', position: 'QB' },
      { lastName: 'Starr', position: 'QB' }
    ]
  },
  {
    _id: ObjectId('62b6112f65d737ba9a792f9e'),
    mascot: 'Tigers',
    name: 'LSU',
    player_docs: [
      { lastName: 'Chase', position: 'WR' },
      { lastName: 'Burrow', position: 'QB' }
    ]
  }
]
```

9. Aggregate query with projection and query document.

   db.teams.aggregate([{ $match: {'name': 'Alabama'} }, {$lookup: { from: 'players', localField: 'teamId', foreignField: 'teamId', as: 'player_docs'} }, {$project: {'mascot': 1, 'name': 1, 'player_docs.lastName': 1, 'player_docs.position': 1 } }]);

   In this code example, we use the $match operation to filter the returned documents matching the name “Alabama”. Also, as you can see from the query expression, the aggregate operation takes an array as an actual parameter. The array is a series of objects with instructions on how to aggregate the data. You can use as many objects as you like to aggregate the data. But, remember, while you can use as many stages as needed in your aggregation pipeline (official term), each additional stage can potentially increate the complexity of your query. This doesn’t necessarily mean it will slow down execute, performance depends on various factors such as the nature of the operations, the size of the data, and the use of indexes.

However, more complex queries can be harder to understand and maintain, so it's important to strike a balance between complexity, readability, formatting, and performance.

10. Delete query:

    db.users.deleteOne({employeeId: 'BW123'});

    You will know the document was deleted successfully if there is a "1" next to the deletedCount property. But, just to be sure, let's call the findOne() function to prove the document was removed.

    db.users.findOne({employeeId: 'BW123'});

    A value of "null" should be printed to the console window.

Hands-On 2.1: Building Your First ORD

In this hands-on project, you will be creating an Object Relational Diagram (ORD) for a BurgerStand business and converting it into a NoSQL data structure.

Instructions:

1. Using Microsoft Visio, create an ORD to represent the following business rules in BurgerStand.

    a. A CUSTOMER can order many BURGER(s).

    b. Many BURGER(s) are ordered by a CUSTOMER.

2. Be sure to include the following fields in your ORD:

    CUSTOMER:
    - _id
    - firstName
    - lastName

    BURGER:
    - _id

- burgerName
- price

3. Convert the proposed ORD into a NoSQL data structure. The NoSQL data structure must be a valid JSON object, which can be validated using jsonlint. You will need to supply your own values for each of the identified fields. NoSQL data structures with empty fields or field values like “test1”, “test2”, etc., will incur a point penalty.

Hints:

- Here is an example of a NoSQL data structure for a book with many attributes:

```
{
 "book": {
  "_id": "1",
  "title": "Moby Dick",
  "author": "Herman Melville"
 }
}
```

- Remember to represent the one-to-many relationship between CUSTOMER and BURGER in your ORD.
- In your NoSQL data structure, consider whether to use a reference or embedded documents to represent the relationship between CUSTOMER and BURGER.

Hands-On 3.1: Movie Night

In this hands-on project, you will be creating an ORD for a Movie Night event and converting it to a NoSQL data structure.

Instructions:

1. Using Microsoft Visio, create an ORD to represent the following business rules in Movie Night.

    a. A VIEWER can watch many MOVIE(s).

    b. Many MOVIE(s) can be watched by a VIEWER.

    c. Each VIEWER can give a RATING to each MOVIE they watch.

2. Be sure to include the following fields in your ORD:

    VIEWER:
    - _id

- firstName
- lastName

MOVIE:
- _id
- movieName
- Genre

RATING:
- _id
- ratingScore
- viewerId
- movieId

3. Convert the proposed ORD into a NoSQL data structure. The NoSQL data structure must be a valid JSON object, which can be validated using jsonlint. You will need to supply your own values for each of the fields. NoSQL data structures with empty fields or field values like “test1”, “test2”, etc., will be accepted.

Hints:
- Remember to represent the one-to-many relationship between VIEWER and MOVIE, and the one-to-one relationship between MOVIE and RATING in your ORD.
- In your NoSQL data structure, consider whether to use a reference or embedded documents to represent the relationships.

Hands-On 4.1: Cell Phone Charging

In this hands-on project, you will be creating an ORD for a Cell Phone Charging scenario and converting it to a NoSQL data structure.

Instructions:

1. Using Microsoft Visio, create an ORD to represent the following business rules:

    a. A USER can charge many DEVICE(s).

    b. Each DEVICE uses a specific CHARGER.

c. Each CHARGER can be used to charge many DEVICE(s).

2. Be sure to include the following fields in your ORD.

   USER:
   - _id
   - firstName
   - lastName

   DEVICE:
   - _id
   - deviceName
   - deviceType
   - charger_id

   CHARGER:
   - _id
   - chargerType
   - voltage

3. Convert the proposed ORD into a NoSQL data structure. The NoSQL data structure must be a valid JSON object, which can be validated using jsonlint. You will need to supply your own values for each of the fields. NoSQL data structures with empty fields or field values like “test1”, “test2”, etc., will be accepted.

Hints:
- Remember to represent the one-to-many relationship between USER and DEVICE, and the one-to-many relationships between CHARGER and DEVICE in your ORD.
- In your NoSQL data structure, consider whether to use a reference or embedded documents to represent the relationships.

Hands-On 4.2: MongoDB Database Setup and Querying with MongoDB Shell

In this hands-on project, you will be setting up a MongoDB Atlas account, creating a database, and performing some basic operations using the MongoDB Shell.

Instructions:

1. Review Guides 1, 2, and 3 in this document.

2. Create a new database named web335DB with a collection named users.

3. Create a new custom role named web335Admin.

4. Create a new database user named web335_user with a password of s3cret and assign it to the custom role from step 3.

5. Download and install mongosh. Test the software's installation by running the command: **mongosh --help**.

6. Load the user.js script from week four of the course's GitHub repository.

7. Write the following queries and take screenshots of the code you used to write the queries and their results.

    a. Display all users in the collection.

    b. Display the user with the email address **jbach@me.com**.

    c. Display the user with the last name **Mozart**.

    d. Display the user with the first name **Richard**.

    e. Display the user with employeeId **1010**.

Hints:

- If you run into issues, please refer to the Guides in this document.
- The queries you write in this hands-on project should be saved in a file named <yourLastName>-<assignmentName>.js and added to your GitHub repository under the current week's folder. Use standard programming comments above each query to explain what the query is doing.
- All images should be added to a single Word document and they must legible. If I cannot make out the text in the screenshot it will not be awarded points. Include your name, date, and assignment number.

Hands-On 5.1: MongoDB Document Manipulation and Projections

In this hands-on project, you will be performing various operations on the user's collection from Assignment 4.2 using MongoDB Shell.

Instructions:

1. Write the following queries and take screenshots of the code you used to write the queries and their results.

    a. Add a new user to the user's collection. Ensure the fields in the new document match the existing fields in the user's collection. Next, prove the new user was added successfully.

    b. Update Mozart's email address to mozart@me.com. Next, prove the document was updated successfully.

    c. Display all users in the collection. Use projections to only show the first name, last name, and email address.

Hints:

- If you run into issues, please refer to the Guides in this document.
- The queries you write in this hands-on project should be saved in a file named <yourLastName>-<assignmentName>.js and added to your GitHub repository under the current week's folder. Use standard programming comments above each query to explain what the query is doing.
- All images should be added to a single Word document and they must legible. If I cannot make out the text in the screenshot it will not be awarded points. Include your name, date, and assignment number.

Hands-On 6.1: Aggregate Queries

In this hands-on project, you will be performing various operations on the houses and student's collections using MongoDB Shell.

Instructions:

1. Load the houses.js script from week six of the courses GitHub repository.

2. Write the following queries and take screenshots of the code you use to write the query and their results:

    a. Display all students.

b. Add a new student. Ensure the fields in the new document match the existing fields in the collection. Next, prove the new student was added successfully.

c. Update one of the properties from the student you added in step b. Next, prove the property was updated successfully.

d. Delete the student you created in step b. Next, prove the student was removed successfully.

e. Display all students by house. The order should be:

   Houses
    Students

f. Display all students in house Gryffindor. The order should be:

   Gryffindor
    Students

g. Display all students in the house with an Eagle mascot. The order should be:

   House
    Students

Hints:

- If you run into issues, please refer to the Guides in this document.
- The queries you write in this hands-on project should be saved in a file named <yourLastName>-<assignmentName>.js and added to your GitHub repository under the current week's folder. Use standard programming comments above each query to explain what the query is doing.
- All images should be added to a single Word document and they must legible. If I cannot make out the text in the screenshot it will not be awarded points. Include your name, date, and assignment number.

References

Copilot. (n.d.). OpenAI. *Microsoft Copilot*. computer software. Retrieved December 19, 2023, from https://www.microsoft.com/en-us/microsoft-copilot.