CS 628: Full-Stack Development – Web App

City University of Seattle

School of Technology & Computing

Professor Sam Chung

## HOS07: MongoDB Atlas

Samantha Hipple

August 12, 2023

# PLEASE NOTE

Screenshots in this guide may differ from your environment (e.g., directory paths, version numbers, etc.). When choosing between a stable or most recent release, we advise you install the stable release rather than the best-testing version. Additionally, there may be subtle discrepancies along the steps, please use your best judgment to complete the tutorial. If you are unfamiliar with terminal, command line, and bash scripts, we recommend watching [this video](https://youtu.be/Dp7uw9c6QH8) prior to moving forward with this guide. Not all steps are fully explained. Lastly, we advise that you avoid copy-pasting code directly from the guide or GitHub repositories. Instead, type out the code yourself to improve familiarity.

More information on this guide can be found under the related module in [this repository](https://github.com/samchung0117/cs628-examples). Please save a screenshot of the app at the end of each section and save it in the current module folder with the relevant section number.

# SECTION CONTENTS

1. Accessing GitHub Codespaces
2. SQL vs NoSQL
3. Introduction to MongoDB
4. Signing up for MongoDB Atlas
5. Setting up a MongoDB database
6. Setting up a MongoDB environment
7. Learning MongoDB step-by-step

# SECTION 1. ACCESSING GITHUB CODESPACES

GitHub Codespaces is an online cloud-based development environment that allows users to easily write, run and debug code. Codespaces is fully integrated with your GitHub repository and provides a seamless experience for developers. In order to access Codespaces, users only need a GitHub account and an active internet connection.

After downloading the current HOS assignment, in the top-right corner of the repo, click on the **<> Code** drop-down menu and select **Create codespace on main** as shown in the following image. The free and pro GitHub subscriptions include free use of GitHub Codespaces *up to a fixed amount of usage each month*. In order to avoid unexpected charges, please review the [billing information](https://docs.github.com/en/billing/managing-billing-for-github-codespaces/about-billing-for-github-codespaces).

# SECTION 2. SQL VS NOSQL

SQL (Structured Query Language) databases are relational databases with fixed schemas, where data is stored in structured tables with rows and columns. These databases utilize ACID (Atomicity, Consistency, Isolation, Durability) transactions to ensure data integrity. They are optimal for handling structured data and maintaining well-defined relationships, with popular examples being MySQL, PostgreSQL, SQLite, and Oracle.

NoSQL (Not Only SQL) databases, on the other hand, offer flexible and often schema-less storage solutions. Their data can be stored in diverse formats such as documents, key-value pairs, or graphs. Designed for horizontal scalability and high performance, they are particularly suitable for unstructured or semi-structured data. Some renowned NoSQL databases include MongoDB, Cassandra, Redis, and Couchbase.

# SECTION 3. INTRODUCTION TO MONGODB

MongoDB is a document-oriented NoSQL database known for storing data in JSON-like documents. Unlike rigid structures, these documents can vary in structure, providing flexibility in how data is represented. Within MongoDB, there are key concepts to understand:

* **Collections** are akin to tables in relational databases and hold related documents.
* **Documents** serve as individual data records and use the BSON (Binary JSON) format.
* **Fields** within these documents can be likened to columns in SQL databases.

Some clear advantages of using MongoDB include its flexible schema, which adapts to changing data requirements, and its speed in read and write operations. In terms of functionalities, MongoDB supports CRUD (Create, Read, Update, Delete) operations, albeit with its unique syntax. The flexible query language is adept at retrieving specific documents, and users can perform tasks like filtering, sorting, and aggregation.

For advanced data operations, there's the aggregation framework. This involves processes like grouping and transforming data. Additionally, MongoDB allows the creation of indexes on specific fields to boost query performance and speed up read operations; and supports geospatial queries, facilitating location-based data queries. Given these capabilities, MongoDB is an ideal choice for various applications, such as websites, content management systems, cataloging, and real-time analytics.

# SECTION 4. SIGNING UP FOR MONGODB ATLAS

1. Open this link: <https://www.mongodb.com/atlas/database> and click on **Try Free**.
2. Either complete the registration form or sign up with a Gmail account.
3. Once registered, sign in and complete the welcome form as demonstrated below:

A screenshot of a computer

Description automatically generated

1. Press the **Finish** button below the form to submit.

# SECTION 5. SETTING UP A MONGODB DATABASE

1. Create a new deployment:

A screenshot of a computer

Description automatically generated

1. Select the **M0 FREE** template:

A close-up of a computer screen

Description automatically generated

1. Set **AWS** as the provider and provide a name for the new cluster:

A white background with black dots

Description automatically generated

1. Click **Create** once the above steps are completed:

A white background with black text

Description automatically generated

Once submitted, you will be redirected to a Security QuickStart screen.

1. Create a username and password for the MongoDB connection.

A screenshot of a computer screen

Description automatically generated

A screenshot of a computer screen

Description automatically generated

1. Then press the **Create User** button.

Remember the credentials for the new user account. They are needed to create a connection string further on in the tutorial. Next, we need to enable access to Atlas from anywhere. Ideally, we would only allow connections to MongoDB Atlas from trusted IP addresses. For now, we will use wild cards that allow universal access.

1. Select the **Network Access** tab in the side menu and click the **Add IP Address** button:

A screenshot of a computer

Description automatically generated

1. Select **ALLOW ACCESS FROM ANYWHERE**, then press the **Confirm** button.

A screenshot of a computer

Description automatically generated

# SECTION 6. SETTING UP A MONGOBD ENVIRONMENT

1. Install the **MongoDB for VS Code** extension in your development environment:

A screenshot of a computer

Description automatically generated

1. Navigate to the extension in the side menu, then click on **Add Connection**:

A screenshot of a computer

Description automatically generated

Next, we need to retrieve our connection string from the **Overview** section in our Atlas account.

1. Click on **CONNECT** for the deployment we just created:

A screenshot of a computer

Description automatically generated

1. Under **Access you data through tools**, select **MongoDB for VS Code**:

A screenshot of a computer

Description automatically generated

Follow the directions on the next screen, which are outlined in more detail in the following steps. We already completed step one by installing the extension.

1. Return to your VS Code window and navigate to the command palette.
2. Search for **MongoDB: Connect**, then select **Connect with Connection String**:

A screenshot of a computer

Description automatically generated

1. Copy the connection string prompt from Atlas to paste into the command palette.
2. Replace **<password>** with the password created for the new user we set up previously:

A screenshot of a computer

Description automatically generated

1. Press **Enter** to establish the connection:

A screenshot of a computer

Description automatically generated

**NOTE:** *If you user password contains special characters such as* ***: / # [ ]*** *or* ***@****, please follow the directions in* [*this link*](https://www.mongodb.com/docs/atlas/troubleshoot-connection/#incorrect-connection-string-format) *to encode them for your connection string.*

1. Click the **Create playground** button, then remove the generated script for a blank slate.
2. Save the blank file to proper directory for this module (**HOS07**):

A screenshot of a computer

Description automatically generated

# SECTION 7. LEARNING MONGODB STEP-BY-STEP

Finally, we are going to learn MongoDB commands by executing them one-by-one in the playground we just created. MongoDB playgrounds are interactive JavaScript (JS) environments that are designed for prototyping queries, aggregations, and MongoDB commands. These playgrounds provide useful syntax highlighting to assist in development. In MongoDB for VS Code, playgrounds are recognized by files bearing the **.mongodb.js** extension.

**NOTE:** *In order to run the commands in the following steps, press the play button found in the top right corner of the playground file. Capture screenshots of the playground results as you go for submission.*

1. Display a list of available databases from the current MongoDB server instance:



***Playground output:***

A black rectangular object with a black border

Description automatically generated

A black rectangular object with a black border

Description automatically generated

1. Switch to or create a specific database using the command below:



***Playground output:***



If the database does not exist, MongoDB will create it.

1. Display the name of the currently selected database:



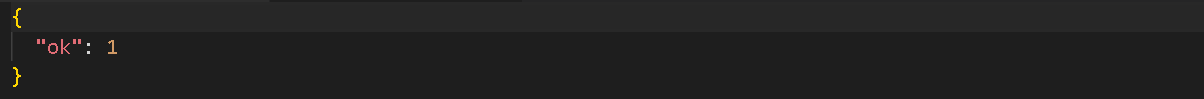
***Playground output:***



1. Create a new collection named **users** within the current database:



***Playground output:***



1. Insert multiple user profiles into the **users** collection:

A screen shot of a computer

Description automatically generated

***Playground output:***

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

Each user document contains three fields: **name**, **age**, and **email**.

1. Display all documents from the **users** collection:



***Playground output:***

A screenshot of a computer

Description automatically generated

1. Display only user documents whose **age** field contains a value of less than **30**:



***Playground output:***

A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

Other operators similar to **$lt** (less than) in MongoDB can be found in the chart below.

|  |  |  |
| --- | --- | --- |
| operator | matches | criteria |
| $gt | values | that are greater than a specified value |
| $lte | values | that are less than or equal to a specified value |
| $gte | values | that are greater than or equal to a specified value |
| $eq | values | that are equal to a specified |
| $ne | values | that are not equal to a specified value |
| $in | values | that exist in a specified array |
| $nin | values | that do not exist in a specified array |
| $exists | documents | that have the specified field |
| $type | documents | where the value of a field has a specific data type |

1. Update a single user document in the **users** collection:



***Playground output:***

A black rectangular object with a black strip

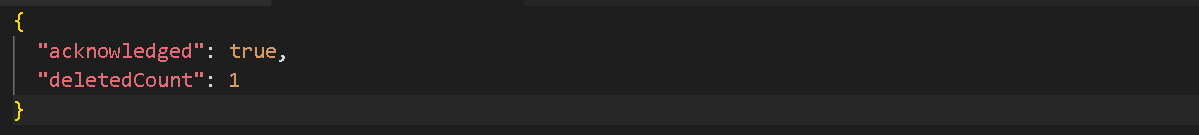
Description automatically generated

The command above finds the document with the name **user1** and sets the **age** field to **31**.

1. Delete a single user document from the **users** collection:



***Playground output:***



This command finds and removes the document with the name **user2** from the **users** collection.

1. Calculate the average age of the user documents in the **users** collection:



***Playground output:***

A black rectangular object with a black border

Description automatically generated

This command groups all documents due to the **\_id: null** parameter and calculates the average age using the **$avg** aggregation operator on the **age** field.

1. Create an index for the **email** field in the **users** collection:

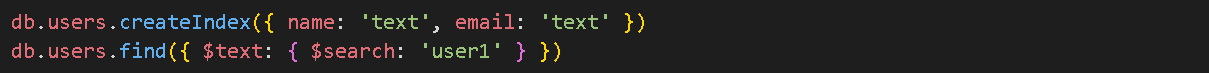


***Playground output:***



This command creates an ascending index for the **email** field, which enables faster searches.

1. Create a text index on the **name** and **email** fields in the **users** collection:



***Playground output:***

A black screen with green text

Description automatically generated

Creating a text index on the **name** and **email** fields of the **users** collection enables efficient text-based queries, such as searching for specific terms like **user1**. The command **db.users.find({ $text: { $search: 'user1' } })** is used to perform a text search on the **users** collection. When executed, it looks for documents where the text index matches the term **user1**, which could span across multiple fields in the collection.

1. Display the documents from the **users** collection in descending order by **age**:



***Playground output:***

A screenshot of a computer

Description automatically generated

1. Display some documents from the **users** collection, skipping the first result:



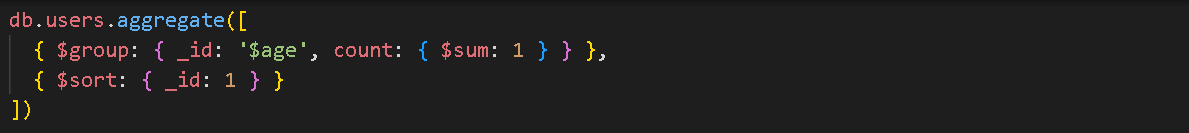
***Playground output:***

A black screen with green text

Description automatically generated

The **limit(2)** command specifies the max number of documents to return, while the **skip(1)** command tells MongoDB to skip the very first document. This approach is commonly applied when implementing pagination in database queries.

1. Group and count user documents in the **users** collection based on the **age** field:



***Playground output:***

A black screen with a black background

Description automatically generated

The **db.users.aggregate([...])** command, when provided with a specific aggregation pipeline, organizes user profiles from the **users** collection by their age. The **$group** stage groups the documents using the **age** field, while the **$sum** aggregation operator calculates the number of profiles for each age. Subsequently, the **$sort** stage arranges these grouped results in ascending order based on age.

1. Identify the oldest and youngest users in the **users** collection:



***Playground output:***

A black rectangular object with a red border

Description automatically generated

The provided command employs a specific aggregation pipeline to discern the ages of the oldest and youngest users from the **users** collection. Within the **$group** stage, the **$max** aggregation operator calculates the highest age, and the **$min** operator determines the lowest age. Both operations are executed without specifically grouping the data by any field. As an outcome, you receive a singular document that showcases both the calculated maximum and minimum ages.

1. Using geospatial queries, locate places within a specific radius from a given point:

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

***Playground output:***

A screen shot of a computer

Description automatically generated

Geospatial queries can be utilized to identify places located within a certain radius from a designated point. In this context, the locations of places are represented as coordinates and are indexed using the **2dsphere** index. When the correct queries are run, they search for and retrieve locations based on the proximity criteria provided.

1. Locate users whose names start with **u** or **C**:

A black background with colorful dots

Description automatically generated

***Playground output:***

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

In the command above, an advanced query using the **$or** operator is combined with regular expressions. The **$options: ‘i’** line ensures case-insensitive matching, making the search process both flexible and comprehensive.

1. Calculate the total age of the users in the **users** collection:



***Playground output:***

A black rectangular object with orange dots

Description automatically generated

Here, all users are grouped together by passing the null identifier and their ages are summed by passing the **$sum** aggregation operator.

1. Remove the **users** collection from the database:



***Playground output:***



1. Display the list of collections available in the current database:



***Playground output:***

A black screen with a black background

Description automatically generated with medium confidence

**PUSH YOUR WORK TO GITHUB TO SUBMIT**