**Modern Web Technologies – Lab 4:**

**Using MongoDB and MongoSH**

Breakdown for the lab:

1. A cheat sheet with some information on some of the commands you can use with mongo
2. The lab questions
   1. I will request you to make a few queries and I want you to submit screenshots for the results

Just to recall, here is the breakdown for MongoDB

**NoSQL: DB** contains **Collections** contains **Documents** contains **Fields**

**SQL:** **DB** contains **Tables** containing **Entries** containing **Data**

**MongoSH Commands:**

1. **Database Management:**
   * **show dbs**
     1. Lists all the databases
     2. Example:
        1. Users
        2. Product
        3. Order
        4. Inventory
        5. Messaging
   * **use** **db\_name**
     1. Switch to the specified database
     2. If it doesn’t exist, it will be created once data is inserted
   * **db.dropDatabase()**
     1. Drop the currently selected database
2. **Collection Management:**
   * **show collections**
     1. List all collections in the current database
   * **db.createCollection("collection\_name")**
     1. Creates a new collection with the given collection name
   * **db.collection\_name.drop()**
     1. Drops the "laptop" collection.
3. **Document Operations (with examples):**
   * **Insert data**
     1. Insert movie data into a moveCollection

|  |
| --- |
| db.movieCollection.insertOne({  title: "Inception",  director: "Christopher Nolan",  year: 2010,  genre: "Science Fiction"  })  **In MongoDB, if you insert documents without specifying an "\_id" field, MongoDB will automatically generate a unique "\_id" for each document.**  db.movies.insertMany([  {  title: "The Shawshank Redemption",  director: "Frank Darabont",  year: 1994,  genre: ["Drama"]  },  {  title: "The Godfather",  director: "Francis Ford Coppola",  year: 1972,  genre: ["Crime", "Drama"]  }  ]) |

* + **Find, and FindAll -> this is like the select option in SQL**

db.nameCollection.find({})

its like the select option in SQL -> SQL SELECT \* FROM nameCollection

* + **Find specific**
    1. Breakdown for this feature: **db.collectionName.find(query, projection)**
       1. Query is like the WHERE statement is for SQL
       2. Projection is finding the fields we want to include or exclude. Its like the SELECT clause for SQL

db.nameCollection.find({“title”: “Spider-man(2000)”})

More complex example:

|  |
| --- |
| db.movies.find(  {  director: "Christopher Nolan",  year: { $gt: 2005 }  },  {  title: 1,  year: 1,  "cast.name": 1  }  ) |

* The **query** part **{ director: "Christopher Nolan", year: { $gt: 2005 } }** filters movies directed by Christopher Nolan and released after 2005.
* The **projection** part **{ title: 1, year: 1, "cast.name": 1 }** specifies to include the title, year, and the names of the cast members in the result. The **1** indicates inclusion, while **0** would indicate exclusion.
* Notice the way we check for year; we added another clause $gt which refers to greater than. When you want to find data for a specific range, you need to add it in like this.
  + year: { $gt: 2005 } -> means: year greater than 2005
  + **Find items with LT (less than x)**

db.movieCollection.find({“year”: {$lt: 2020})

* + **Updating an item/multiple items**
    1. Similarly with insert, we have updateOne and updateMany
    2. There are additional **Logical** flags you can add to help with updating items
    3. Example:

|  |
| --- |
| db.movieCollection.updateOne(  {“title”: “Spider-man(2000)”},  {$set: {“title”: “Spider Man(2000)”},  }) |

* + **Deleting an item**

db.movieCollection.deleteMany({“rating”: {$gt: 10}})

**MongoDB command breakdown (important for next class)**

**Db.collection.operation({Filtering Options}, {Other options})**

Once we isolate for the collection (table) we want to look at, we choose an **operation** we want to run. Typically FIND or INSERT. If you want to add filters to isolate what you find, we add filtering objects inside an object. This will be important for anyone building a search + filter, you might want to look at this.

**Here are a few examples:**

Find laptops

Db.laptops.find({}) -> “{}” are optional here

Find laptops cheaper than $1000 -> we store the price in a field call “price”

Db.laptops.find({ price: { $lt: 1000 }}))

In here, we add a filter object, with a key word: Price, and a filter for “less than 1000”

**Operators in MongoSH**

In MongoDB, operators in mongosh (MongoDB Shell) refer to special symbols or keywords used within queries and commands to perform specific operations on data. These operators allow you to filter, manipulate, and work with data in your MongoDB collections. MongoDB provides a wide range of operators to handle different types of operations.

**Comparison Operators:**

* $eq: Matches values that are **equal to** a specified value.
* $ne: Matches values that are **not equal to** a specified value.
* $gt: Matches values that are **greater than** a specified value.
* $gte: Matches values that are **greater than or equal to** a specified value.
* $lt: Matches values that are **less than** a specified value.
* $lte: Matches values that are **less than or equal to** a specified value.

**Logical Operators:**

* $or: Performs a logical **OR** operation on an array of two or more expressions and selects documents that satisfy at least one of the expressions.
* $and: Performs a logical **AND** operation on an array of two or more expressions and selects documents that satisfy all the expressions.
* $not: **Inverts the effect of a query** expression and selects documents that do not match the expression.
* $nor: Performs a logical **NOR** operation on an array of two or more expressions and selects documents that do not satisfy any of the expressions.

**Element Operators:**

* $exists: Matches documents that contain a specified field, regardless of its value.
* $type: Selects documents if a field is of the specified BSON data type. Array Operators:
* $in: Matches any of the values specified in an array.
* $nin: Matches none of the values specified in an array.
* $all: Matches arrays that contain all elements specified in an array.
* $size: Matches arrays with a specific number of elements.

**Regex Operators:**

* $regex: Provides regular expression matching to select documents where a field matches a specified pattern.
* $options: Allows you to specify options for the $regex operator. Text Search Operators:
* $text: Performs full-text search on text indexes.
* $meta: Returns metadata about the text search score. Geospatial Operators:
* $geoWithin: Selects documents with geospatial data that is within a specified shape.
* $geoIntersects: Selects documents with geospatial data that intersects a specified shape.

**Array Update Operators:**

* $addToSet: Adds elements to an array if they do not already exist in the array.
* $pop: Removes the first or last element from an array.
* $pull: Removes all matching values from an array.
* $push: Appends values to an array.

Operators in mongosh are an essential part of constructing queries and performing various operations on MongoDB data. They provide flexibility and control when working with MongoDB collections and documents.

**Lab 4: Using MongoDB and MongoSH for a laptop store.**

For this lab, I just want you to practice making queries with mongoSH. Think about your shopping apps (amazon, bestbuy, etzy), what do they filter through.

**I have posted a json file you can upload to compass inside the same Lab 4 area**

Here is the document structure:

{

"brand": "Dell",

"model": "XPS 13",

"processor": "Intel Core i7",

"ram": "16GB",

"storage": "512GB SSD",

"price": 1299.99,

"stock": 10

},

Here is a list of queries I want you to try:

* Comparison queries **[Complete 4 of 6]**
  + Find laptops with the price equal to $999.99.
  + Find laptops with prices not equal to $1299.99.
  + Find laptops with a price greater than $1500.
  + Find laptops with a price greater than or equal to $1399.99.
  + Find laptops with a price less than $1000.
  + Find laptops with a price less than or equal to $1099.99.









* Logical queries **[Complete 3 of 4]**
  + Find laptops with a price less than $1000 OR with a stock greater than 10.



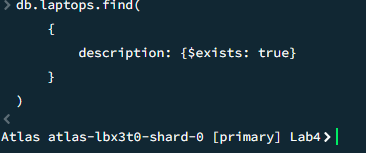
* + Find laptops with a price less than $1000 AND with a stock greater than 10.



* + Find laptops with a price NOT equal to $999.99.



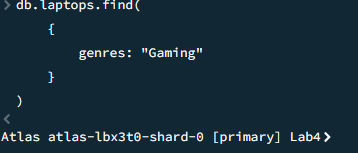
* + Find laptops with a price NOT equal to $999.99 NOR with stock NOT equal to 8.
* Element operators: **[Complete 2 of 2]**
  + Find laptops with the "description" field using **exists**.



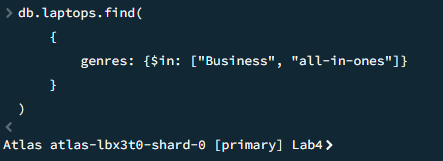
* + Find laptops with the "reviews" field missing.
    - Hint: $exist, you can use 0 and 1 or true and false



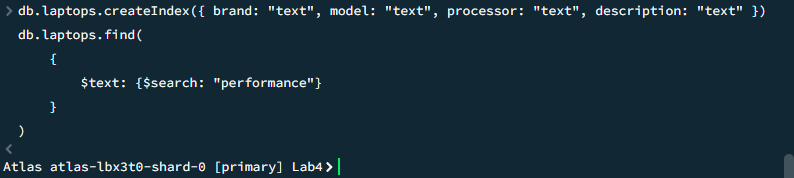
* Array operators: **[Complete 2 of 2]**
  + Find laptops with a genre of "Gaming" in their genres array.



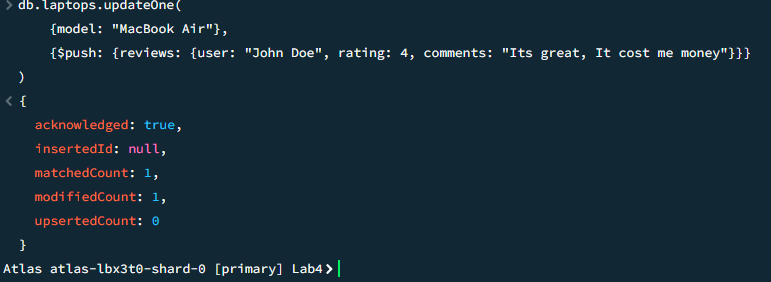
* + Find laptops with the genre "Business" using $in.



* Text Search Operators: **[Complete 1 of 1]**
  + Perform a text search for laptops with the word "performance".
    - Hint: use the $text tag and the $search tag



* Array Update Operators: **[Complete 1 of 1]**
  + Add a new review to a laptop with a specific title.
    - (e.g., "Laptop Title", review: user: John Doe, rating: 4)
    - Hint: use the update command and the $push to append a new field of data
    - The review is an array holding the fields user and rating.



**Submission:**

**Provide me 1 screenshot with each of the query (upwards of 17 total)**

**Provide me a list of the queries you have made, just place them in order of the above order**

**No GitHub Submission required this time**

Grading Scheme:

|  |  |
| --- | --- |
| **Query** | Marks |
| **Comparison** | 8 (2 mark per query completed) |
| **Logical** | 6 (2 marks per query completed) |
| **Element** | 4 (2 marks per query completed) |
| **Array** | 4 (2 marks per query completed) |
| **Text Search** | 4 |
| **Array** | 4 |
| **Total** | 30 |

**Overall Grade weightage: 4 Marks**