MongoDB Project

Private Function

Malvica Philomina Lewis
A00303932
Masters in Data Analytics
5th May 2023
Advanced Databases

Contents

Introduction

MongoDB is a popular document-oriented NoSQL database management system that stores data in a flexible, JSON-like format known as BSON. MongoDB is designed to be scalable and can handle large amounts of data, making it a popular choice for big data applications.

A MongoDB project would typically involve the following steps:

- **Defining the data model**: In MongoDB, data is stored in collections of documents. The first step in a MongoDB project is to define the data model by identifying the documents that will be stored in the collections, and the fields that will be included in each document.
- **Setting up the database**: The next step is to set up the MongoDB database and create the necessary collections and indexes. This involves installing the MongoDB server and client software, configuring the server, and creating the necessary users and roles to manage access to the data.
- **Importing data**: Once the database is set up, data can be imported from various sources, such as CSV files, JSON files, or relational databases.
- **Querying the data**: MongoDB provides a powerful query language that allows users to retrieve data from the database based on specific criteria. Queries can be simple or complex and can include aggregation and sorting functions.
- **Analyzing the data**: MongoDB also provides a range of tools for analyzing and visualizing data, such as the MongoDB Compass GUI, which allows users to explore the data and create visualizations.
- **Scaling the database**: MongoDB is designed to be scalable and can handle large amounts of data and high levels of traffic. As the database grows, it may be necessary to add more servers to the cluster or to partition the data across multiple nodes.

Overall, a MongoDB project can involve a range of tasks, from designing the data model to setting up the database, importing data, querying and analyzing the data, and scaling the database as needed.

Background to the Dataset

The data pertains to private functions, and includes the following fields:

- **Private_Function_Id**: a unique identifier for each private function.
- **Year_Held**: the year in which the function was held.

- **Function_Type**: the type of function, which can be one of Birthday, Retirement, Wedding, or Corporate Event.
- **Food_Choice**: an array of one or more food choices for the event. These could include Finger Food, Fish, Beef, Lamb, Turkey, Chicken, Burgers, Chips, Vegetarian, Salad, Soup, Tea, Sandwiches, and so on.
- **Num_Guests**: the number of guests who attended the event.
- **Alcohol**: an array of drinks served at the event, which could include Guinness, Heineken, Gin, Vodka, Wine, Whiskey, and so on.
- **Soft_Drinks**: an array of soft drinks served at the event, which could include Coca Cola, Lemonade, Lilt, Orange Juice, Water, and so on.
- **Duration**: the number of hours the event lasted.
- **Disc_Jockey**: an embedded document containing details about the Disc Jockey who provided music at the event. This could include the DJ's name, fee, and availability.
- **Venue**: an embedded document containing details about the venue where the event was held. This could include the venue's name, city, type (such as Sports Stadium, Park, Private Estate, Racecourse, Theatre, Multi-Purpose Hall, Concert Hall, and so on), capacity (how many people the venue can hold), and acoustics (Poor, Fair, or Good).

Overall, this data could be used to analyze private functions and their characteristics, such as the types of food and drinks served, the size of the events, the duration of the events, and the quality of the venues and entertainment provided. This information could be used by event planners and managers to improve the quality of their services and better cater to their customers' needs.

Understanding Embedded Documents in MongoDB

An embedded document is a data structure within a larger document in a NoSQL database, such as MongoDB. It is a way to store related data together in a single document, rather than splitting it into multiple tables in a relational database.

In MongoDB, an embedded document is stored as a nested JSON object within the main document. For example, in the provided Private_Function data, the Disc_Jockey and Venue fields are embedded documents. The details of the disc jockey and venue are stored within the same document as the Private_Function, making it easier to retrieve all the relevant information about the function in a single query.

In comparison to SQL databases, which use tables to store data in a structured manner, NoSQL databases like MongoDB use documents to store data in a semi-structured or unstructured manner. This means that data can be stored in a more flexible and dynamic format, with the

ability to store embedded documents, arrays, and other complex data types. In SQL databases, related data is typically split across multiple tables, which can make it more difficult to retrieve all the relevant data for a specific query.

Overall, the use of embedded documents in NoSQL databases provides a more flexible and efficient way to store related data, compared to the more rigid structure of SQL databases.

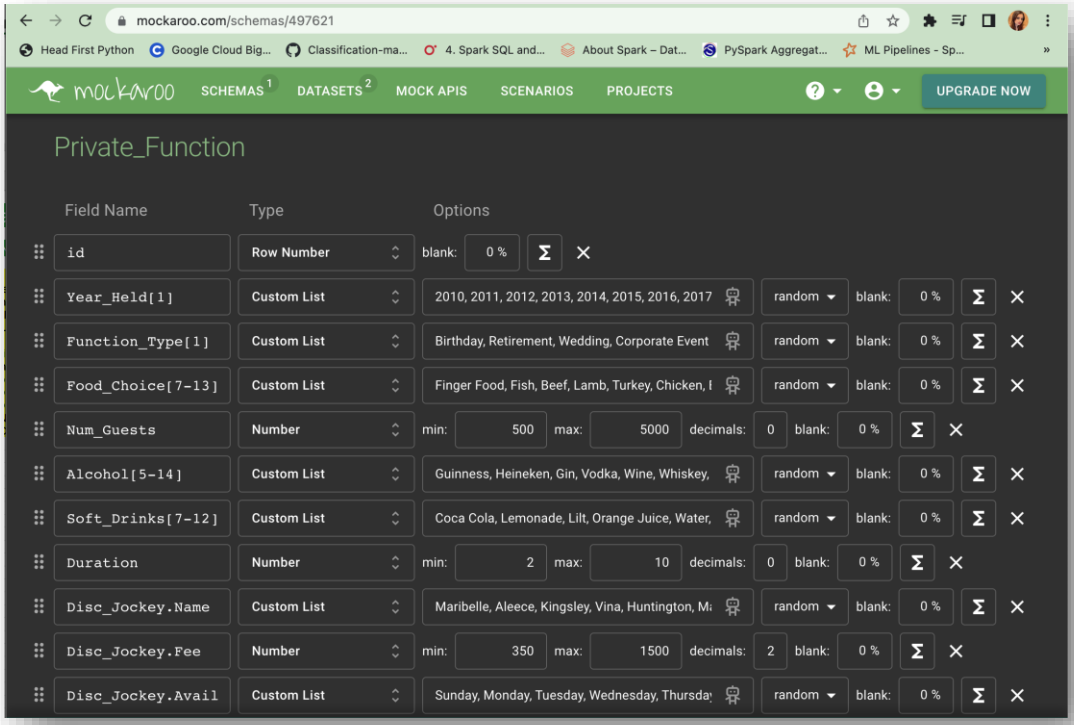
Generating Data Using Mockaroo

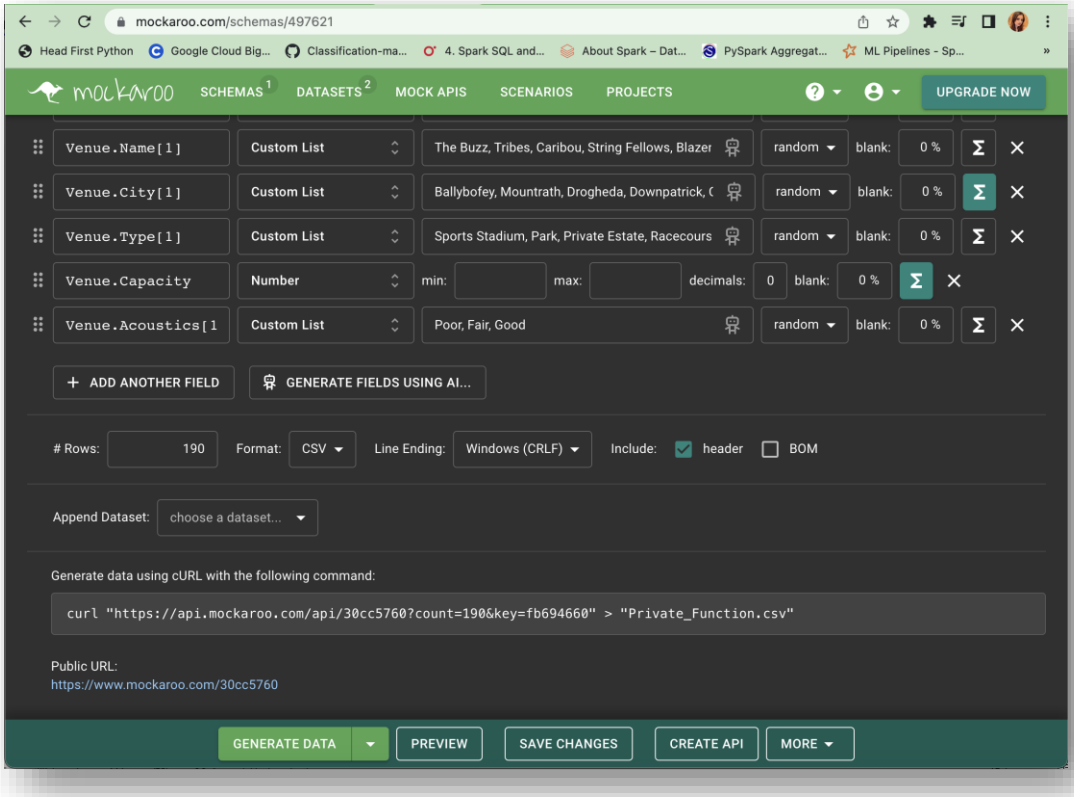
Mockaroo is a web-based tool that allows you to generate realistic mock data for testing and development purposes. Here is an overview of how to generate data in Mockaroo:

- Sign up for a free account on the Mockaroo website (if you haven't already).
- Once you are logged in, click on the "New Dataset" button to create a new dataset.
- Choose a data type for your dataset, such as CSV, JSON, SQL, or Excel, and then select the appropriate file format.
- Define the schema of your dataset by adding fields and specifying their data types, formats, and any other relevant properties.
- Configure any options for your dataset, such as the number of records to generate, any constraints or dependencies between fields, or any custom formulas or expressions to use.
- Preview your dataset to make sure it looks correct and meets your requirements.
- Generate your data by clicking the "Download" button, which will create and download a file containing your mock data in the specified format.
- Use the generated data for testing or development purposes, or export it to your preferred application or database.

Mockaroo also provides several advanced features, such as data anonymization, data generation based on regular expressions, and data import/export from various sources, which can help you generate more complex and realistic mock data.

A schema named Private_Function generated using Mockaroo, is shown below:


mockaroo.com/schemas/497621
Head First Python
Google Cloud Big...
Classification-ma...
4. Spark SQL and...
About Spark – Dat...
PySpark Aggregat...
ML Pipelines - Sp...
mockaroo
SCHEMAS
DATASETS
MOCK APIS
SCENARIOS
PROJECTS
UPGRADE NOW
Private_Function
Field Name
Type
Options
id
Row Number
blank: 0 %
Year_Held[1]
Custom List
2010, 2011, 2012, 2013, 2014, 2015, 2016, 2017
random
blank: 0 %
Function_Type[1]
Custom List
Birthday, Retirement, Wedding, Corporate Event
random
blank: 0 %
Food_Choice[7-13]
Custom List
Finger Food, Fish, Beef, Lamb, Turkey, Chicken, I
random
blank: 0 %
Num_Guests
Number
min: 500
max: 5000
decimals: 0
blank: 0 %
Alcohol[5-14]
Custom List
Guinness, Heineken, Gin, Vodka, Wine, Whiskey,
random
blank: 0 %
Soft_Drinks[7-12]
Custom List
Coca Cola, Lemonade, Lilt, Orange Juice, Water,
random
blank: 0 %
Duration
Number
min: 2
max: 10
decimals: 0
blank: 0 %
Disc_Jockey.Name
Custom List
Maribelle, Aleece, Kingsley, Vina, Huntington, Mi
random
blank: 0 %
Disc_Jockey.Fee
Number
min: 350
max: 1500
decimals: 2
blank: 0 %
Disc_Jockey.Avail
Custom List
Sunday, Monday, Tuesday, Wednesday, Thursda
random
blank: 0 %


mockaroo.com/schemas/497621
Head First Python
Google Cloud Big...
Classification-ma...
4. Spark SQL and...
About Spark – Dat...
PySpark Aggregat...
ML Pipelines - Sp...
mockaroo
SCHEMAS
DATASETS
MOCK APIS
SCENARIOS
PROJECTS
UPGRADE NOW
Venue.Name[1]
Custom List
The Buzz, Tribes, Caribou, String Fellows, Blazer
random
blank: 0 %
Venue.City[1]
Custom List
Ballybofey, Mountrath, Drogheda, Downpatrick, (
random
blank: 0 %
Venue.Type[1]
Custom List
Sports Stadium, Park, Private Estate, Racecours
random
blank: 0 %
Venue.Capacity
Number
min:
max:
decimals: 0
blank: 0 %
Venue.Acoustics[1
Custom List
Poor, Fair, Good
random
blank: 0 %
ADD ANOTHER FIELD
GENERATE FIELDS USING AI...
Rows: 190
Format: CSV
Line Ending: Windows (CRLF)
Include: header BOM
Append Dataset: choose a dataset...
Generate data using cURL with the following command:
curl "https://api.mockaroo.com/api/30cc5760?count=190&key=fb694660" > "Private_Function.csv"
Public URL:
https://www.mockaroo.com/30cc5760
GENERATE DATA
PREVIEW
SAVE CHANGES
CREATE API
MORE

Installing Mongo Shell on MacBook Pro M1

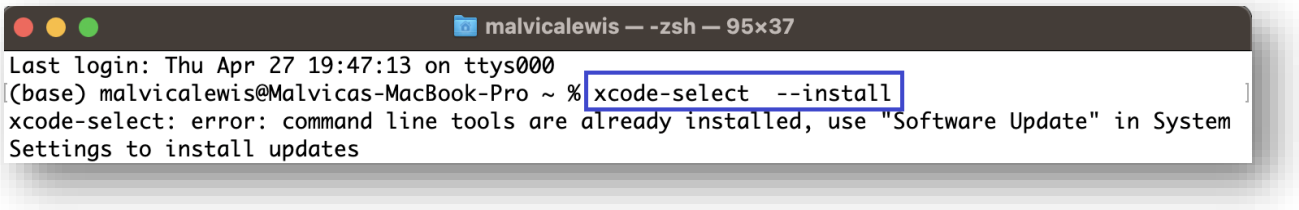
The installation of Mongo Shell on MacBook Pro M1 includes the following installations:

- Xcode Command-Line Tools
- Homebrew
- MongoDB Community Edition

Installing Xcode Command-Line Tools

To utilize Homebrew on your system, it is necessary to have the Xcode command-line tools installed. These tools are part of Apple's Xcode, which is an Integrated Development Environment (IDE) containing a collection of software development tools specifically designed for MacOS.

- Open your MacOS terminal.
- Run the following command in terminal:

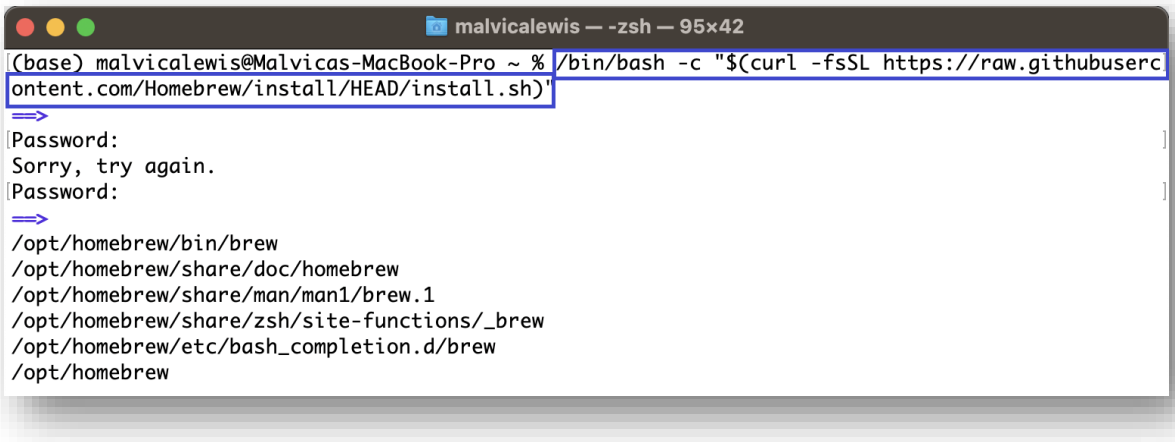


The installation process will initiate once you start it, and during the process, you will be asked to accept a software license before the tools are downloaded and automatically installed. This was already installed in my system.

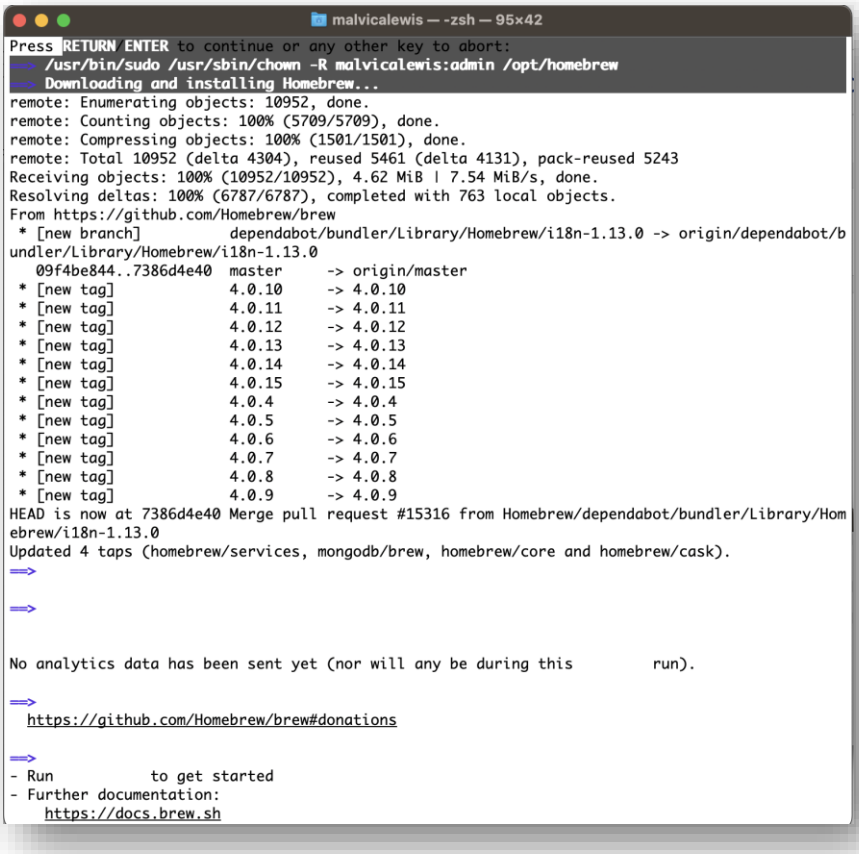
Installing Homebrew

In MacOS, the Homebrew package is not pre-installed, so to use it, you need to first download the installation script and then run it to complete the installation process.

- Open your MacOS terminal.
- Paste the following code to the terminal - **/bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/HEAD/install.sh)"** and enter the password when prompted.



- Press **Return** or **Enter** to continue the installation, else press any other key to abort the process.

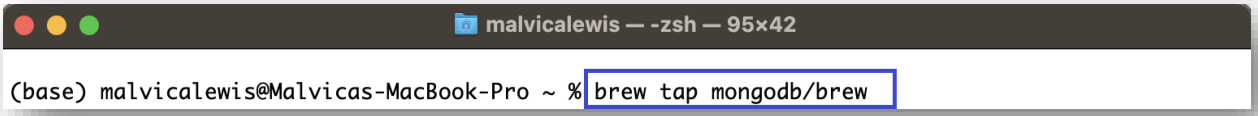


By running the command mentioned above, you can obtain the Homebrew script from Homebrew's Git repository hosted on GitHub. This script is then interpreted to initiate the installation of Homebrew on your system.

Installing MongoDB Community Edition on MacOS Using Shell

Follow the steps below to install the MacOS Mongo Shell.

- To obtain the official Homebrew formula for MongoDB and its associated database tools, you can navigate to the MongoDB Homebrew Tap on Homebrew and execute a command in the terminal to download them.

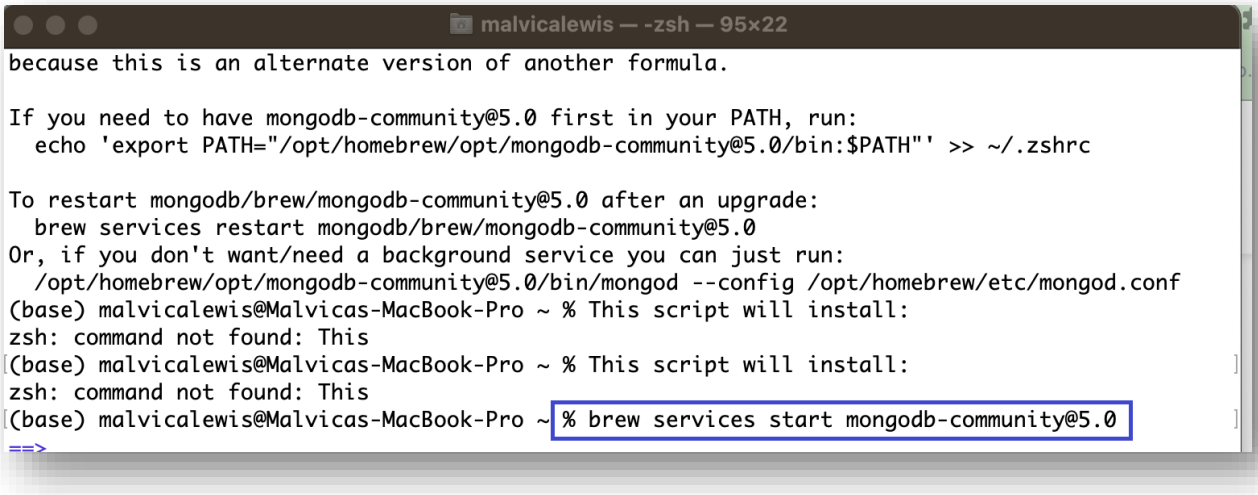


- To install MongoDB on MacOS, you can use the terminal to run a specific command. This command allows you to install different versions of MongoDB and even manage multiple versions simultaneously.

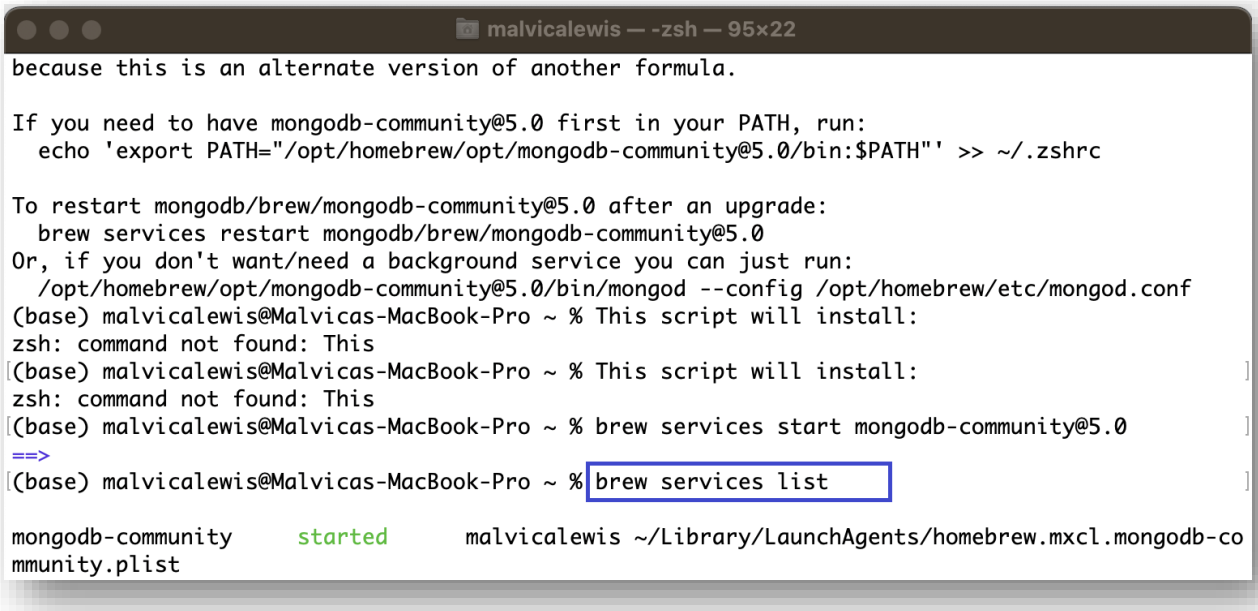
Running MongoDB Community Edition on MacOS Using Shell

Follow the steps below to run the MongoDB Community Edition on MacOS.

- Using **brew**, you have the option to run MongoDB as a MacOS service which will ensure that the appropriate system **ulimit** values are set automatically. To initiate the MongoDB **mongod** server as a MacOS service, you can execute the command provided below:



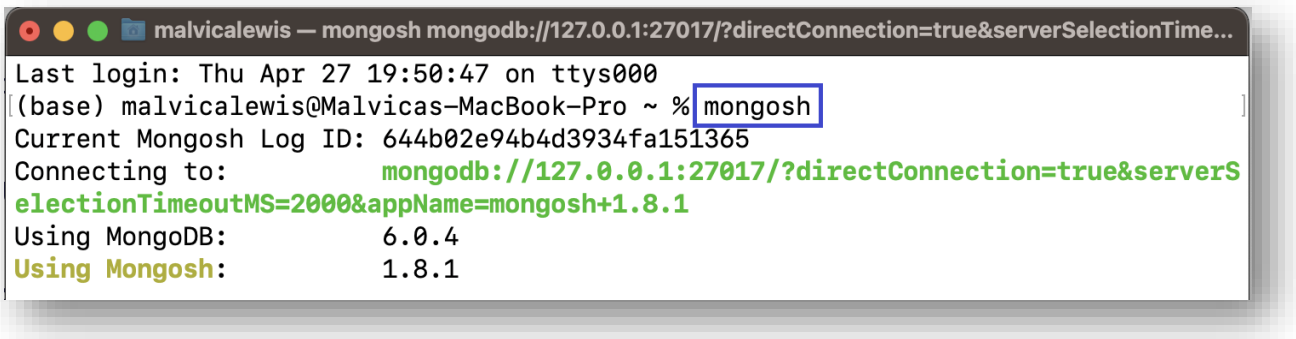
- To verify if MongoDB is running, enter the following command:



It will show the **mongo-community edition** listed as **started**.

Connecting and Using MongoDB on MacOS

To begin utilizing MongoDB, you must establish a connection between **mongosh** and the active instances. You can achieve this by opening a new terminal and typing in "**mongosh**" to connect and start using it.



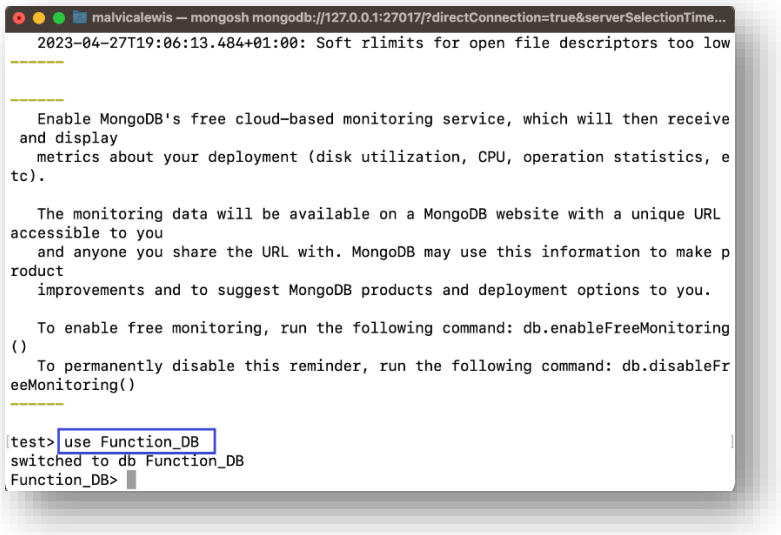
The installed versions of MongoDB and Mongosh are displayed.

Loading and Executing JavaScript files in Mongosh

The following section explains how to load and execute the JavaScript files that have embedded documents in Mongosh. It includes the following steps:

Creating a new database using Mongosh

To create a new database, enter ***use <database name>*** in Mongosh. For this project, we will use the database name '**Function_DB**'.



Viewing the database in Mongosh

To view the databases in Mongosh, type “show databases” as shown below:

Loading the JavaScript files in Mongosh

To load the JavaScript files, use **load** command and include the path of the javascript file as shown below:

When you load a JavaScript file in a system, the feedback generated at the end of the operation will be a single word - "**true**". This indicates that the contents of the file have been successfully executed and loaded into the system.

Executing the JavaScript files in Mongosh

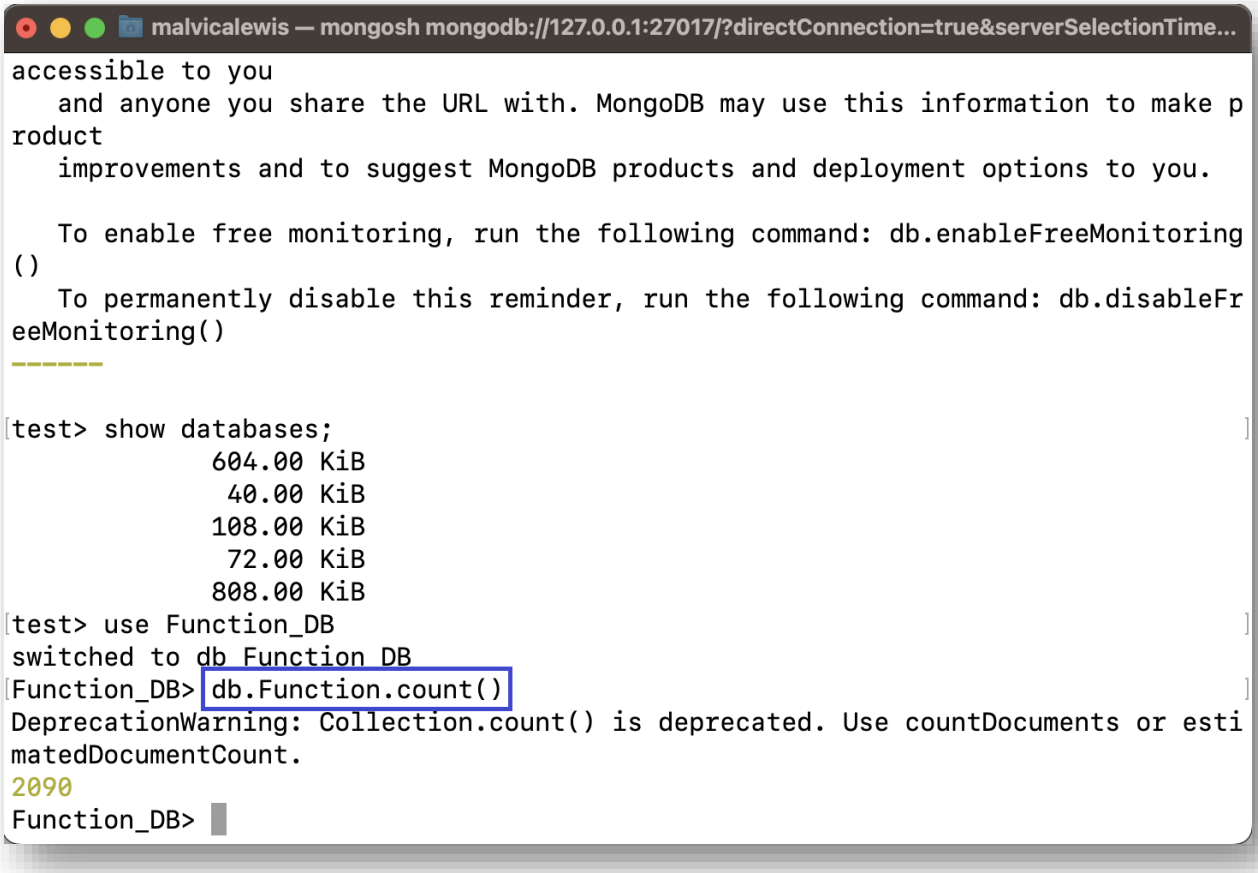
We can execute a few commands at the shell to establish that the data is safely loaded in the **Function_DB** database.

Using the count() function

In MongoDB, count() is a method used to return the number of documents that match a specific query criteria in a collection. The count() method takes one argument, which is the query that

defines the selection criteria for the documents to be counted. It returns a count of the documents that match the specified query.

Let's use the **count()** to find the number of documents in the database.



We can see that there are **2090** documents in the **Function_DB** database.

Using the find() and pretty() functions

`**find()**` is a method in MongoDB that is used to retrieve documents from a collection. It accepts a query as an argument and returns a cursor to the documents that match the query. The syntax of the `find()` method is as follows:

db.collection.find(query, projection)

Here, `**db.collection**` refers to the name of the collection from which we want to retrieve documents, `**query**` is an optional parameter that specifies the selection criteria for the

documents to retrieve, and `**projection**` is also an optional parameter that specifies which fields to include or exclude from the returned documents.

`**pretty()**` is a method that is used to format the output of a MongoDB shell method or cursor to make it more human-readable. When called on a cursor object or a document returned by a shell method, the `**pretty()**` method adds line breaks and indentation to make the output easier to read. The syntax to use `**pretty()**` is as follows:

db.collection.find(query, projection).pretty()

Here, `**db.collection**`, `**query**`, and `**projection**` are the same as described above for the `**find()**` method.

Getting Started with MongoDB Simple Queries

1. $ne, $not, $in, $gt

Question: What are the events held in 2022 that:

- did not have a disc jockey available on Friday, Saturday or Sunday
- had a food choice other than vegetarian or soup, and
- had more than 100 guests?

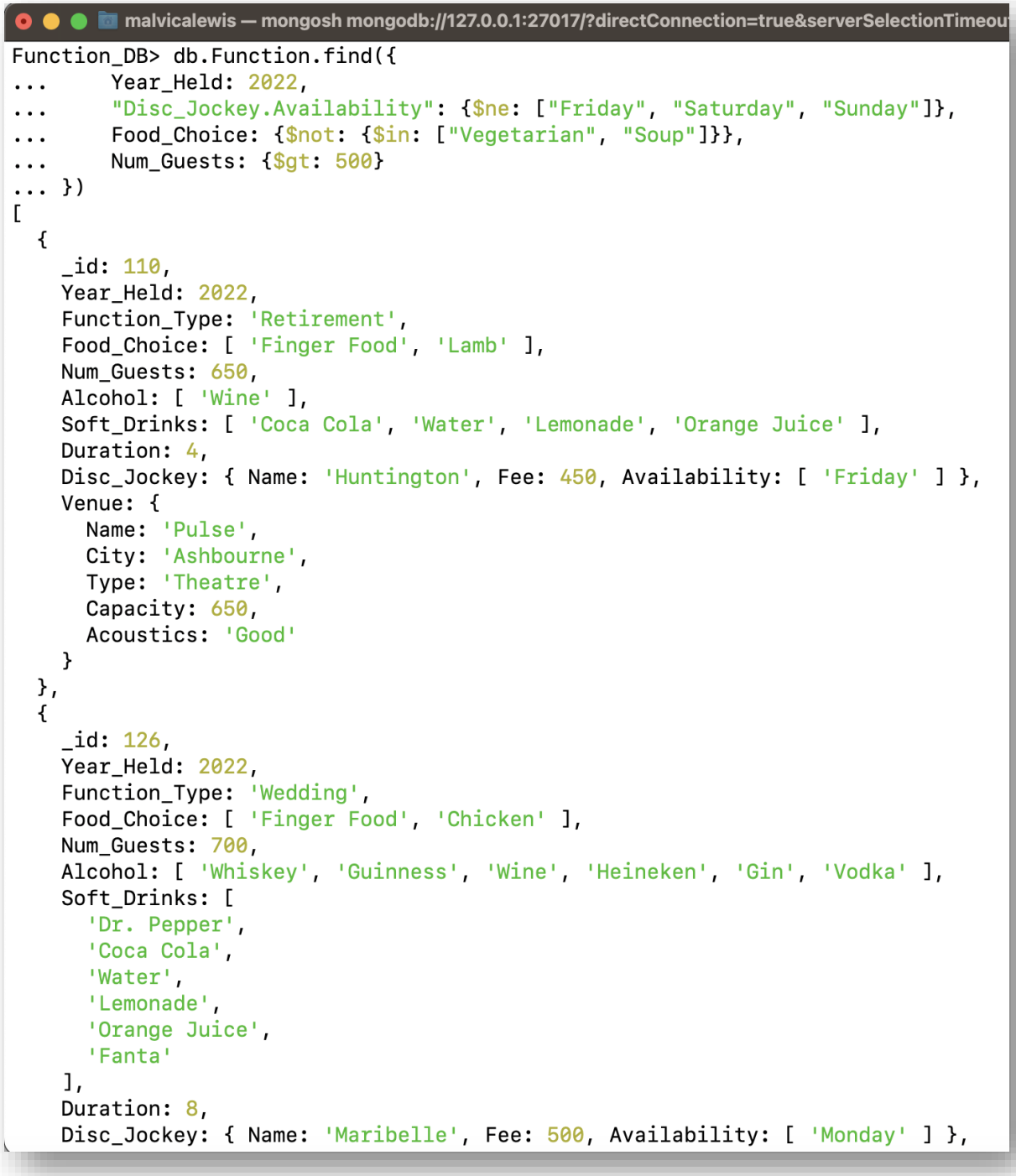
Query:

```
db.Function.find({
  Year_Held: 2022,
  "Disc_Jockey.Availability": {$ne: ["Friday", "Saturday", "Sunday"]},
  Food_Choice: {$not: {$in: ["Vegetarian", "Soup"]}},
  Num_Guests: {$gt: 500}
})
```

The query uses the following functions:

- **Year_Held**: This is used to filter the documents based on the year the function was held, and it is set to **2022**.
- **Disc_Jockey.Availability**: This function is used to filter the documents where the disc jockey is not available on **Friday**, **Saturday**, or **Sunday**.
- **Food_Choice**: This function is used to filter the documents where the food choice is not **Vegetarian** or **Soup**.
- **Num_Guests**: This function is used to filter documents where the number of guests is greater than **500**.
- All these functions are used together with the **$and** operator to ensure that all the conditions are met for a document to be returned.

Query Output:



Query Count:

```
db.Function. countDocuments ({
  Year_Held: 2022,
  "Disc_Jockey.Availability": {$ne: ["Friday", "Saturday", "Sunday"]},
  Food_Choice: {$not: {$in: ["Vegetarian", "Soup"]}},
  Num_Guests: {$gt: 500}
})
```

2. $and, $lte, $not

Question: What are the venues that

- have good acoustics,
- a capacity lesser than or equal to 100, and
- where Maribelle, Irwinn, Brett, Charity, Serge, Crawford, Chane, Baxie, Shaun and Mattheus are not available as a disc jockey?

Query:

```
db.Function.find({
  $and: [
            { "Venue.Acoustics": "Good",
                "Venue.Capacity": {$lte: 100},
              },
            {"Disc_Jockey.Name": {$not: {$in: ["Maribelle", "Irwinn",
            "Brett", "Charity", "Serge", "Crawford", "Chane", "Baxie",
            "Shaun", "Mattheus"]}},
                }
      ]
})
```

The query has two main parts, each containing multiple conditions:

1. The first part of the query contains two conditions, which are combined using the **$and** operator:
   - **"Venue.Acoustics": "Good"**: This condition checks if the value of the "**Venue.Acoustics**" field is equal to "**Good**".
   - **"Venue.Capacity": {$lte: 100}**: This condition checks if the value of the "**Venue.Capacity**" field is less than or equal to **100**.
   - These two conditions are combined using the **$and** operator, which means that both conditions must be true for a document to be returned by the query.
2. The second part of the query also contains two conditions, which are combined using the **$and** operator:
   - **"Disc_Jockey.Name": {$not: {$in: ["Maribelle", "Irwinn", "Brett", "Charity", "Serge", "Crawford", "Chane", "Baxie", "Shaun", "Mattheus"]}}**: This condition checks if the value of the "**Disc_Jockey.Name**" field is not in the list of names specified. The **$in** operator checks if the value of the field matches any of the values in the list, and the $not operator negates the result so that documents where the value of the field is in the list are excluded.

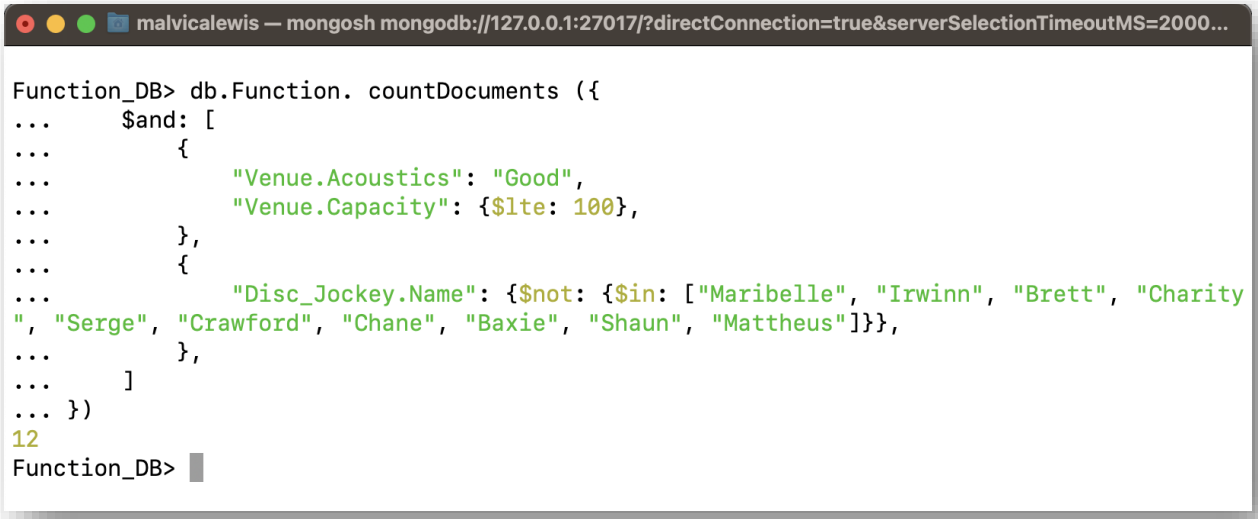
- The two conditions in this part of the query are also combined using the **$and** operator, which means that both conditions must be true for a document to be returned by the query.

3. Finally, the two parts of the query are combined using the **$or** operator, which means that a document will be returned if it satisfies at least one of the two parts of the query.

Query Output:

```
malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000...
Function_DB> db.Function.find({
...     $and: [ {
...             "Venue.Acoustics": "Good",
...             "Venue.Capacity": {$lte: 100},
...         },
...         {
...             "Disc_Jockey.Name": {$not: {$in: ["Maribelle", "Irwinn", "Brett", "Charity
", "Serge", "Crawford", "Chane", "Baxie", "Shaun", "Mattheus"]}},
...         }]
... })
[
  {
    _id: 51,
    Year_Held: 2011,
    Function_Type: 'Corporate Event',
    Food_Choice: [ 'Chicken' ],
    Num_Guests: 60,
    Alcohol: [ 'Whiskey', 'Wine', 'Gin', 'Vodka' ],
    Soft_Drinks: [ 'Coca Cola', 'Fanta' ],
    Duration: 3,
    Disc_Jockey: { Name: 'Kingsley', Fee: 700, Availability: [ 'Wednesday' ] },
    Venue: {
      Name: 'Athetic Grounds',
      City: 'Drumcliff',
      Type: 'Sports Venue',
      Capacity: 100,
      Acoustics: 'Good'
    }
  },
  {
    _id: 90,
    Year_Held: 2014,
    Function_Type: 'Retirement',
    Food_Choice: [ 'Beef', 'Chicken', 'Lamb' ],
```

Query Count:


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000...
Function_DB> db.Function. countDocuments ({
... $and: [
... {
... "Venue.Acoustics": "Good",
... "Venue.Capacity": {$lte: 100},
... },
... {
... "Disc_Jockey.Name": {$not: {$in: ["Maribelle", "Irwinn", "Brett", "Charity
", "Serge", "Crawford", "Chane", "Baxie", "Shaun", "Mattheus"]}},
... },
...]
... })
12
Function_DB>

3. $elemMatch, $or, $gte, $ne, $exists

Question: What venues had events

- with number of guests greater than or equal to 700,
- with acoustics that were not considered "Fair" or "Good", or
- where a disc jockey was hired for the event?

Query:

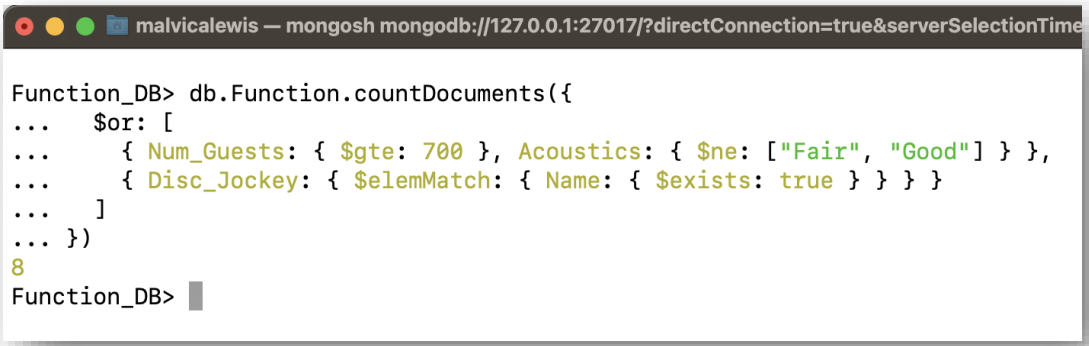
```
db.Function.find({
 $or: [
    { Num_Guests: { $gte: 700 },
          Acoustics: { $ne: ["Fair", "Good"] }
       },
    { Disc_Jockey: { $elemMatch: { Name: { $exists: true } } }
       }
     ]
})
```

The query uses the following functions:

- Documents where the **Num_Guests** field is greater than or equal to **700**, and the Acoustics field is not "**Fair**" or "**Good**". This is accomplished using the comparison operator **$gte** to check the value of **Num_Guests**, and the logical operator **$ne** with an array to check that the value of Acoustics is not "**Fair**" or "**Good**".
- Documents where the **Disc_Jockey** field contains an object with a Name field that exists. This is accomplished using the **$elemMatch** operator to search for an object within the **Disc_Jockey** array that has a Name field that exists.
- Both these criteria are combined using the logical operator **$or**, which means that the query will return any documents that meet either one of these criteria.

Query Output:

Query Count:



4. $gte, $in, $lt

Question: What venues held corporate events

- in 2015 for weddings at Sports Venue
- with a capacity of at least 300 guests,
- had a disc jockey available on Tuesday and Thursday
- with a fee less than 1500?

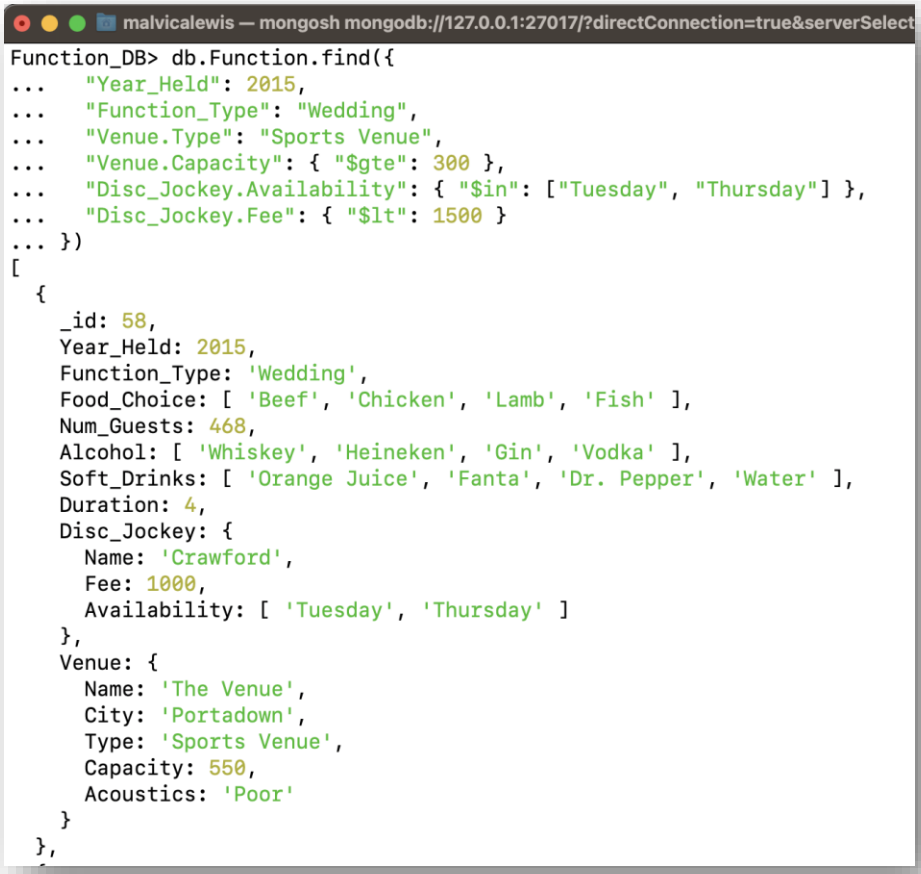
Query:

```
db.Function.find({
                    "Year_Held": 2015,
                    "Function_Type": "Wedding",
                    "Venue.Type": "Sports Venue",
                    "Venue.Capacity": { "$gte": 300 },
                    "Disc_Jockey.Availability": { "$in": ["Tuesday", "Thursday"]
                  },
                    "Disc_Jockey.Fee": { "$lt": 1500 }
                  })
```

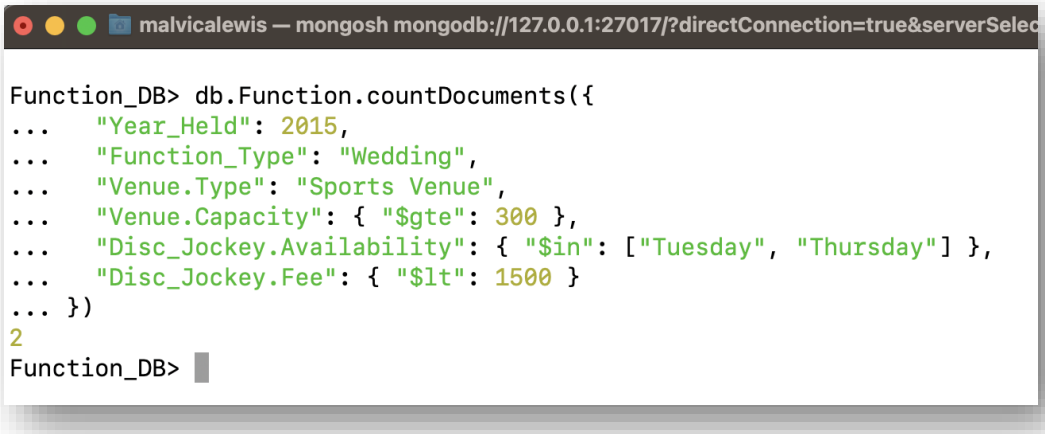
The query uses the following functions:

- **"Year_Held": 2015** specifies that the year of the function should be 2015.
- **"Function_Type": "Wedding"** specifies that the type of the function should be Wedding.
- **"Venue.Type": "Sports Venue"** specifies that the type of the venue should be Sports Venue.
- **"Venue.Capacity": { "$gte": 300 }** specifies that the capacity of the venue should be greater than or equal to 300.
- **"Disc_Jockey.Availability": { "$in": ["Tuesday", "Thursday"] }** specifies that the availability of the disc jockey should be either Tuesday or Thursday.
- **"Disc_Jockey.Fee": { "$lt": 1500 }** specifies that the fee of the disc jockey should be less than 1500.

Query Output:



Query Count:



5. $regex, $and

Question: What are the venues that held events where

- soft drinks that contain the word "Orange" in their name were served
- along with Finger Food as food choice?

Query:

```
db.Function.find(
        {$and: [
            {"Food_Choice": ["Finger Food"]},
            {"Soft_Drinks": {$regex: "Orange"}},
            {"Venue.City": /^D/}
            ]
        },
        {"Venue.Name": 1}
        )
```

The query uses the following functions:

- **$and**: combines multiple conditions and all conditions must be satisfied for a document to be included in the result.
- **$regex**: It is a regular expression operator which matches the word "**Orange**" in the field "**Soft_Drinks**". The caret **^** in the regular expression **^D** matches the beginning of the string for the field "**Venue.City**".

Query Output:

Query Count:

6. $strLenCP, $and, $expr, $gte, $eq, $mod

Question: Find the documents where

- the length of the city name is greater than or equal to 10
- the year held is divisible by 4 and the remainder when divided by 3 is equal to 1

Query:

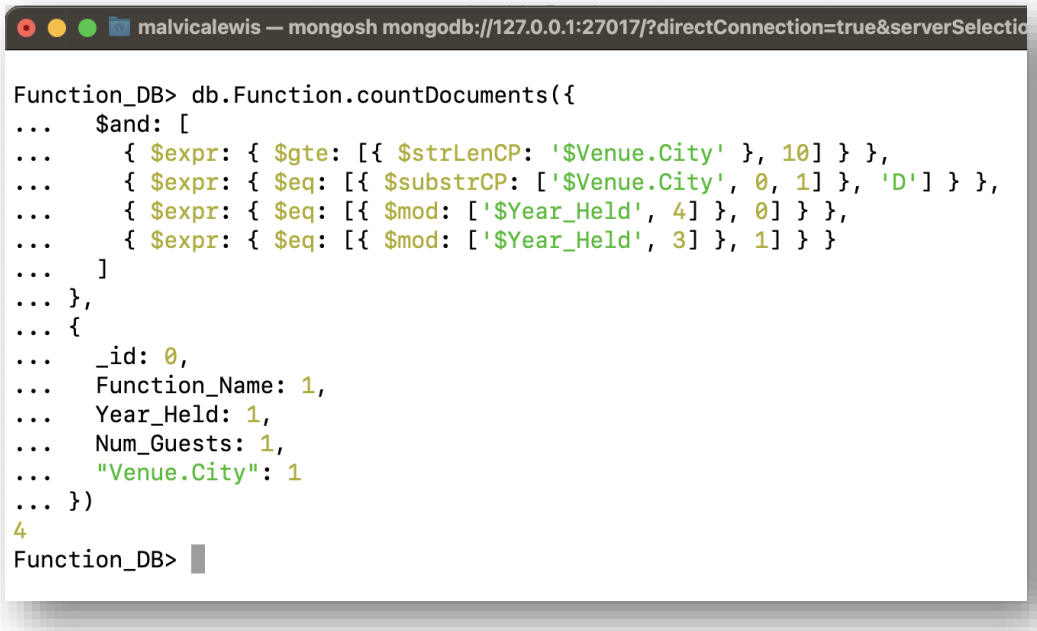
```
db.Function.find(
        {$and: [
                    { $expr: { $gte: [{ $strLenCP: '$Venue.City' }, 10] } },
                    { $expr: { $eq: [{ $substrCP: ['$Venue.City', 0, 1] }, 'D'] } },
                    { $expr: { $eq: [{ $mod: ['$Year_Held', 4] }, 0] } },
                    { $expr: { $eq: [{ $mod: ['$Year_Held', 3] }, 1] } }
                ]
        },
        {
          _id: 0,
          Function_Name: 1,
          Year_Held: 1,
          Num_Guests: 1,
          "Venue.City": 1
        })
```

The query uses the following functions:

- **$and**: This operator combines multiple query conditions and requires that all conditions be true to match a document.
- **$expr**: This operator allows the use of aggregation expressions within the query language.
- **$gte**: This operator compares the length of the "**Venue.City**" string to a value of 10, and returns true if the length is greater than or equal to 10.
- **$substrCP**: This operator extracts a substring from the "**Venue.City**" string. The first argument specifies the string to extract from, the second argument specifies the starting position of the substring, and the third argument specifies the length of the substring.
- **$eq**: This operator checks whether the first argument is equal to the second argument.
- **$mod**: This operator divides the first argument by the second argument and returns the remainder.

Query Output:

Query Count:



7. $mod, $all, $in, $ne

Question: Find the documents where

- the number of guests who attended the event is a multiple of 9
- food served included "Beef" and "Fish"
- disc jockey's performed on "Monday" or "Friday", and
- the venue type is not "Sports Venue"

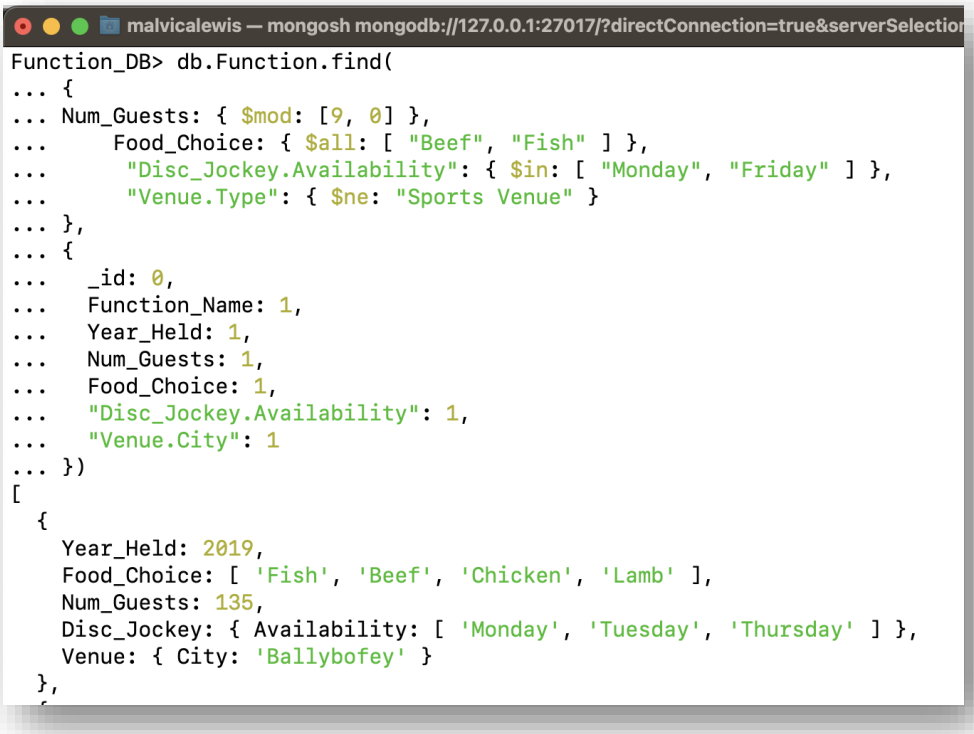
Query:

```
db.Function.find(
        {
        Num_Guests: { $mod: [9, 0] },
    Food_Choice: { $all: [ "Beef", "Fish" ] },
     "Disc_Jockey.Availability": { $in: [ "Monday", "Friday" ] },
     "Venue.Type": { $ne: "Sports Venue" }
        },
        {
         _id: 0,
         Function_Name: 1,
         Year_Held: 1,
         Num_Guests: 1,
         Food_Choice: 1,
         "Disc_Jockey.Availability": 1,
         "Venue.City": 1
        })
```

The query uses the following functions:

- **$mod**: returns the remainder of a division operation. In this case, **{ $mod: [9, 0] }** means that the **Num_Guests** field should be divisible by 9.
- **$all**: checks if all elements in an array match a set of specified values. **{ $all: ["Beef", "Fish"] }** checks that the **Food_Choice** field contains both "**Beef**" and "**Fish**".
- **$in**: checks if a value matches any of the specified values in an array. **{"Disc_Jockey.Availability": { $in: ["Monday", "Friday"] } }** means that the **Disc_Jockey.Availability** field should be either "**Monday**" or "**Friday**".
- **$ne**: checks if a value is not equal to the specified value. **{"Venue.Type": { $ne: "Sports Venue" } }** means that the **Venue.Type** field should not be equal to "**Sports Venue**".

Query Output:



Query Count:

8. $where, $and, $regex, $exists, $gt

Question: Find the events where

- the type of function has 7 characters,
- a food choice must exist,
- the duration must be greater than 0, and
- a venue capacity that can accommodate the number of guests for the duration of the function

Query:

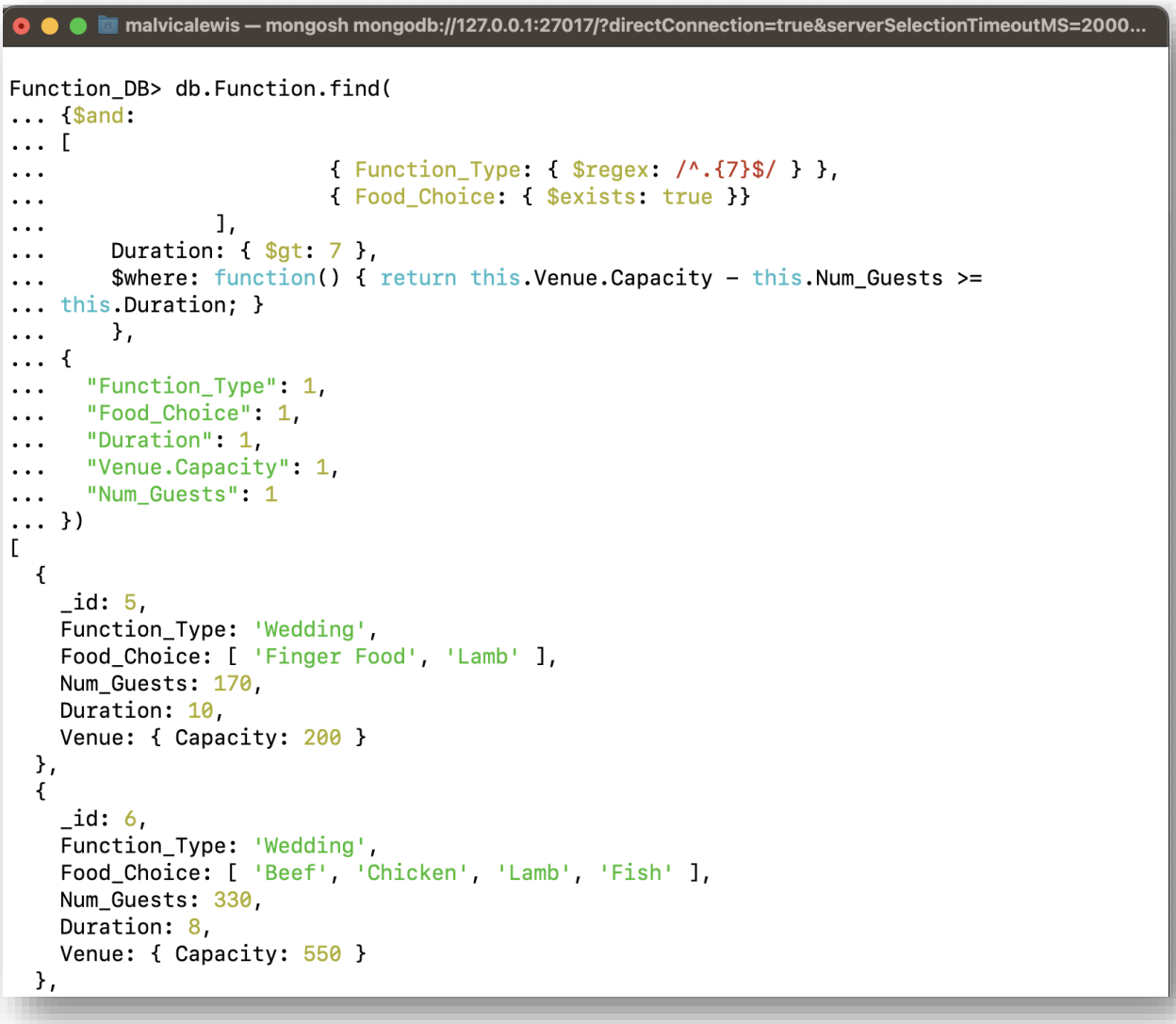
```
db.Function.find(
        {$and:
                [
            { Function_Type: { $regex: /^.{7}$/ } },
            { Food_Choice: { $exists: true }}
                ],
    Duration: { $gt: 7 },
    $where: function() { return this.Venue.Capacity - this.Num_Guests >=
                                                        this.Duration; }
    },
        {
          "Function_Type": 1,
          "Food_Choice": 1,
          "Duration": 1,
          "Venue.Capacity": 1,
          "Num_Guests": 1
        })
```

The query uses the following functions:

- **$and**: This function specifies that all conditions enclosed in the array must be true for a document to be returned.
- **$regex**: This function allows for regular expression matching. In this case, it is matching the **Function_Type** field to a regex that looks for any string of exactly **7** characters in length.
- **$exists**: This function checks if a field exists in the document.
- **$gt**: This function checks if the value of the **Duration** field is greater than **7**.

- **$where**: This function allows for arbitrary JavaScript code to be executed on the server. In this case, the code checks if the difference between the **Capacity** field of the Venue subdocument and the **Num_Guests** field is greater than or equal to the **Duration** field.

Query Output:



9. $setIntersection, $and, $expr, $gt, $size, $ne, $where

Question: Find the events

- which were held after 2014,
- the capacity of the venue is greater than or equal to the number of guests,
- the food choice must have at least two items in common with the array ["Fish", "Beef", "Chicken"],
- the type of venue should only include "Theatre"

Query:

```
db.Function.find(
        {$and:
                [
                {$expr: {$gt:
                                [
                                {$size: {$setIntersection: [["Fish", "Beef",
                                "Chicken"],"$Food_Choice"]}},2
                                ]
                }},
                { "Venue.Type": { $ne: "Theatre" } }
                ],
            Year_Held: { $gt: 2014 },
            $where: function() { return this.Venue.Capacity >=
                                                    this.Num_Guests }
        },
        {
          "Function_Type": 1,
          "Year_Held": 1,
          "Venue.Type": 1,
          "Venue.Capacity": 1,
          "Num_Guests": 1,
          "Food_Choice": 1
        })
```

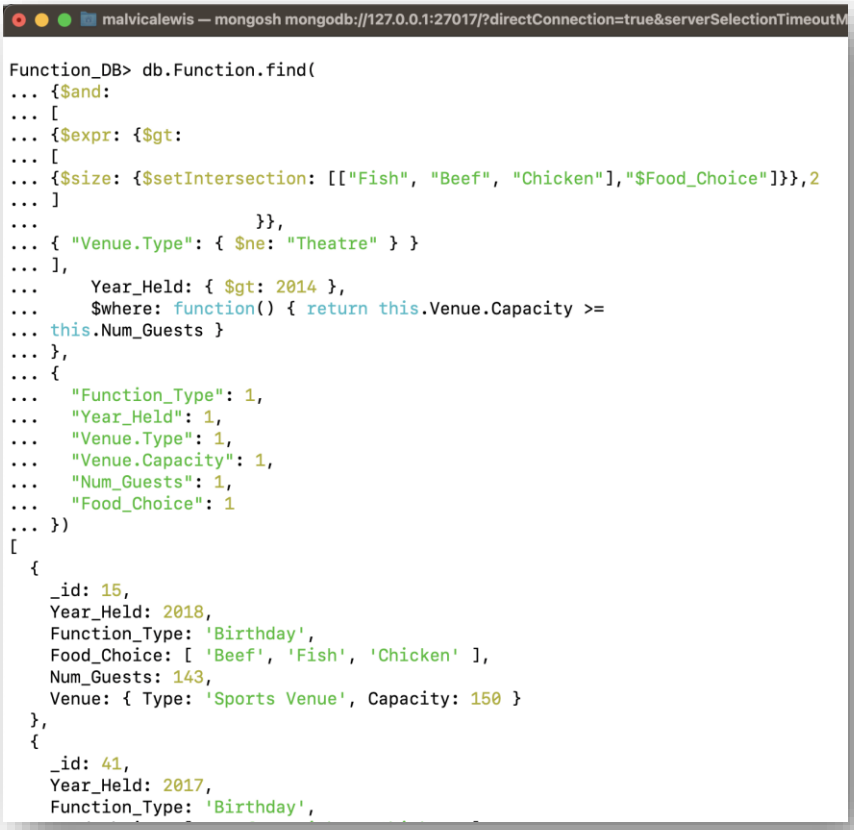
The query uses the following functions:

- **$setIntersection**: This function is used to find the intersection between two arrays, in this case, the array **["Fish", "Beef", "Chicken"]** and the array stored in the **Food_Choice**

field of the documents. The resulting array contains the elements that are present in both arrays.

- **$size**: This function returns the number of elements in an array. It is used to get the size of the array resulting from the **$setIntersection** operation.
- **$expr**: This function is used to evaluate a query expression in the context of the current document. In this case, it is used to compare the size of the resulting array from the **$setIntersection** operation to the value **2**.
- **$ne**: This function is used to check if the value of a field is not equal to a specified value. It is used to filter documents where the **Venue.Type** field is not equal to "**Theatre**".
- **$gt**: This function is used to check if the value of a field is greater than a specified value. It is used to filter documents where the **Year_Held** field is greater than **2014**.
- **$where**: This function is used to evaluate a JavaScript function on the server for each document in the collection. In this case, it is used to filter documents where the **Venue.Capacity** field is greater than or equal to the **Num_Guests** field.

Query Output:



10. $gt, $expr, $gte, $add, $size, $ne

Question: Find the events

- where the sum of number of guests who attend an event and the count of types of alcohol served is greater than or equal to 500 ,
- the type of event is not a wedding,
- the soft drinks served for the customers must include water, and
- the fee for the disc jockey is less than or equal to 950

Query:

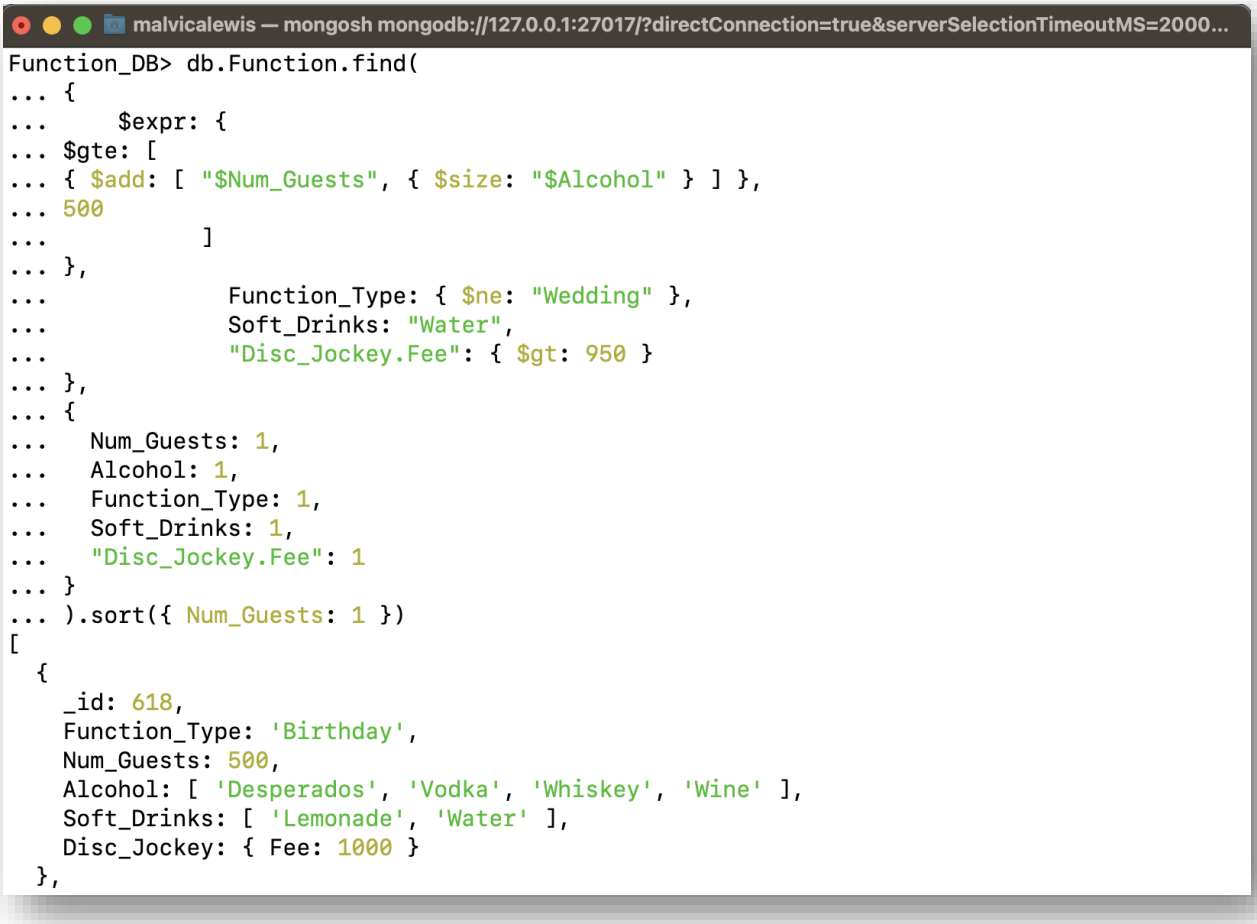
```
db.Function.countDocuments(
        {
          $expr: {
                $gte: [
                        { $add: [ "$Num_Guests", { $size: "$Alcohol" } ] },
                        500
                      ]
                },
                Function_Type: { $ne: "Wedding" },
                Soft_Drinks: "Water",
                "Disc_Jockey.Fee": { $gt: 950 }
        },
        {
          Num_Guests: 1,
          Alcohol: 1,
          Function_Type: 1,
          Soft_Drinks: 1,
          "Disc_Jockey.Fee": 1
        }
).sort({ Num_Guests: 1 })
```

The query uses the following functions:

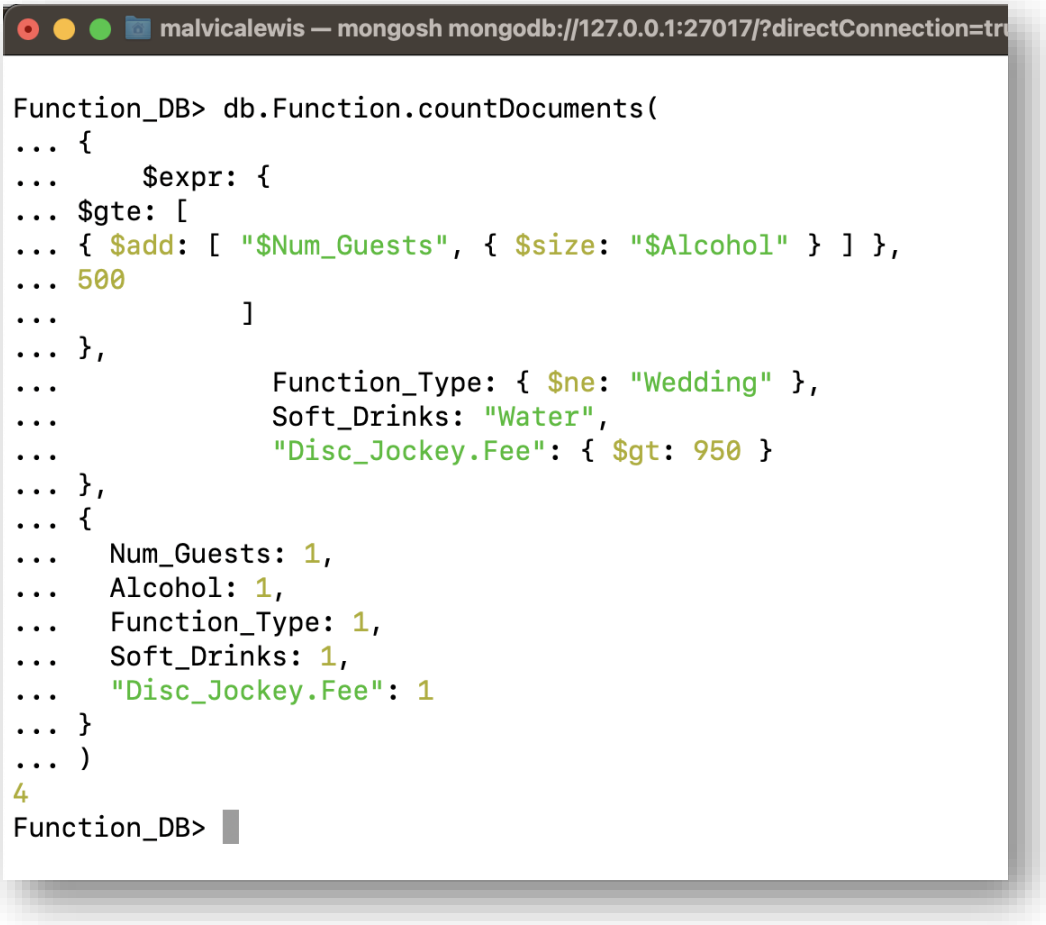
- **$expr: { $gte: [{ $add: ["$Num_Guests", { $size: "$Alcohol" }] }, 500] }**: This expression compares the sum of **Num_Guests** and the size of the Alcohol array to **500**. Only documents where this value is greater than or equal to **500** will be returned.

- **Function_Type: { $ne: "Wedding" }**: This condition excludes documents where the **Function_Type** field is equal to "**Wedding**".
- **Soft_Drinks: "Water"**: This condition only includes documents where the Soft_Drinks field is equal to "**Water**".
- **"Disc_Jockey.Fee": { $gt: 950 }**: This condition only includes documents where the **Disc_Jockey.Fee** field is greater than **950**.
- Finally, the **sort()** method is applied to sort the results by the "**Num_Guests**" field in ascending order.

Query Output:



Query Count:


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=tr
Function_DB> db.Function.countDocuments(
... {
... $expr: {
... $gte: [
... { $add: ["$Num_Guests", { $size: "$Alcohol" }] },
... 500
...   ]
... },
...   Function_Type: { $ne: "Wedding" },
...   Soft_Drinks: "Water",
...   "Disc_Jockey.Fee": { $gt: 950 }
... },
... {
...   Num_Guests: 1,
...   Alcohol: 1,
...   Function_Type: 1,
...   Soft_Drinks: 1,
...   "Disc_Jockey.Fee": 1
... }
...)
4
Function_DB>

11. $lte, $expr, $abs, $subtract, $in, $gte

Question: Find the events

- where the difference between the number of guests and the capacity of the venue is less than or equal to 100,
- the fee for the disc jockey is lesser than or equal to 300,
- the type of event is either "Corporate Event" or "Retirement", and
- the venue acoustics are "Fair"

Query:

```
db.Function.find(
 {
  $expr: {
   $lte: [
    { $abs: { $subtract: ["$Num_Guests", "$Venue.Capacity"] } },
    100
   ]
  },
  "Disc_Jockey.Fee": { $lte: 300 },
  Function_Type: { $in: ["Corporate Event", "Retirement"] },
  "Venue.Acoustics": "Fair"
 },
 {
  Num_Guests: 1,
 "Venue.Capacity": 1,
  "Disc_Jockey.Fee": 1,
  Function_Type: 1,
  "Venue.Acoustics": 1
 }
)
```

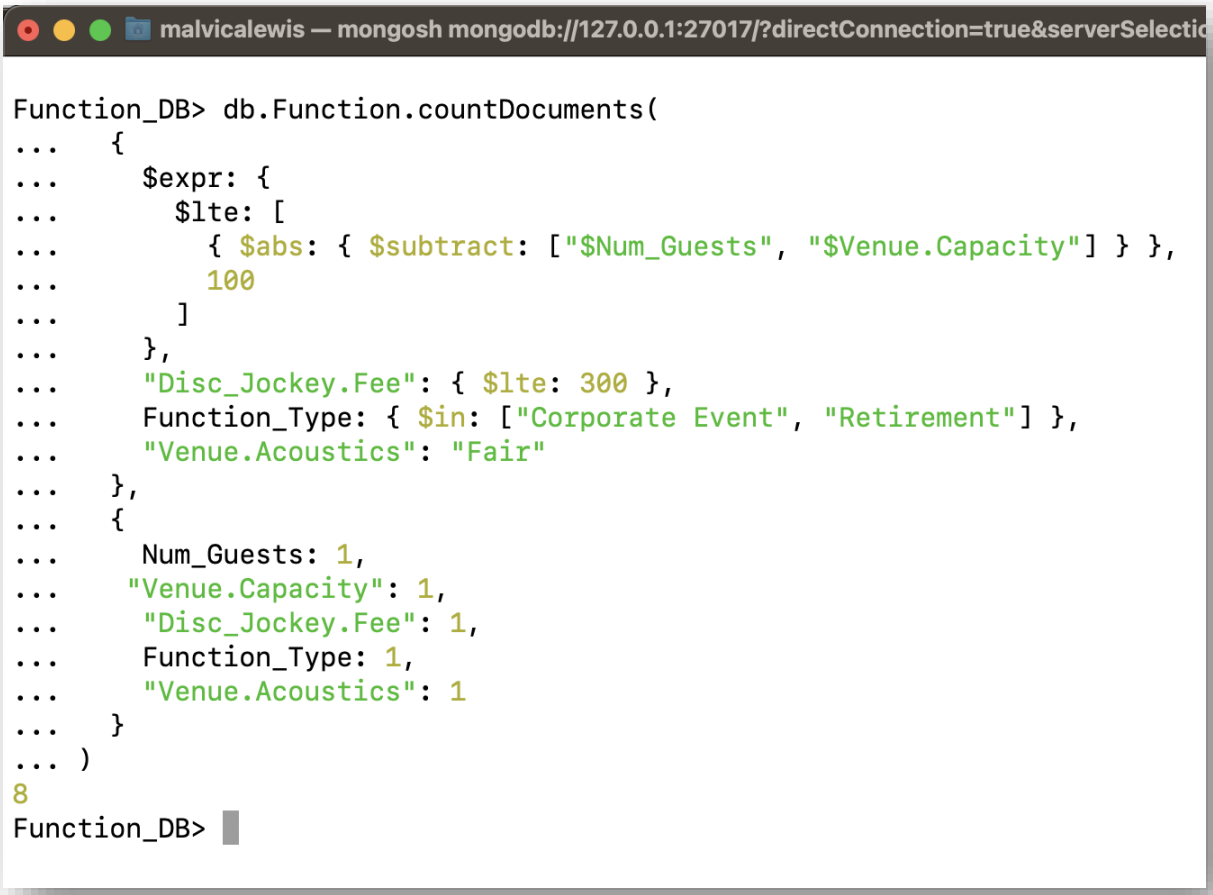
The query uses the following functions:

- **$expr**: This is an operator that allows the use of aggregation expressions inside the query language. In this case, it allows the use of **$abs** and **$subtract** aggregation expressions to compute the absolute value of the difference between **$Num_Guests** and **$Venue.Capacity**.

- **$lte**: This is a comparison operator that tests whether the first operand is less than or equal to the second operand. In this case, it checks if the absolute value of the difference between **$Num_Guests** and **$Venue.Capacity** is less than or equal to **100**.
- **$abs**: This is an aggregation expression that returns the absolute value of a number. In this case, it computes the absolute value of the difference between **$Num_Guests** and **$Venue.Capacity**.
- **$subtract**: This is an aggregation expression that subtracts the second operand from the first operand. In this case, it subtracts **$Venue.Capacity** from **$Num_Guests** to get the difference between them.
- **"Disc_Jockey.Fee"**: This is a field specifier that specifies the **Disc_Jockey** subdocument's Fee field. It limits the results to documents where the Fee value is less than or equal to **300**.
- **Function_Type**: This is a field specifier that specifies the **Function_Type** field. It limits the results to documents where the **Function_Type** value is either "**Corporate Event**" or "**Retirement**".
- **"Venue.Acoustics"**: This is a field specifier that specifies the Venue subdocument's **Acoustics** field. It limits the results to documents where the **Acoustics** value is "**Fair**".

Query Output:

Query Count:


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectio
Function_DB> db.Function.countDocuments(
... {
... $expr: {
... $lte: [
... { $abs: { $subtract: ["$Num_Guests", "$Venue.Capacity"] } },
... 100
...]
... },
... "Disc_Jockey.Fee": { $lte: 300 },
... Function_Type: { $in: ["Corporate Event", "Retirement"] },
... "Venue.Acoustics": "Fair"
... },
... {
... Num_Guests: 1,
... "Venue.Capacity": 1,
... "Disc_Jockey.Fee": 1,
... Function_Type: 1,
... "Venue.Acoustics": 1
... }
...)
8
Function_DB>

12. $expr, $strLenCP, $gt, $gte

Question: Find the events

- where the length of the type of function is greater than the length of the venue city,
- the number of guests is greater than or equal to 350,
- the alcohol served to the customers includes Guiness", and
- the venue type is a multi-purpose hall.

Query:

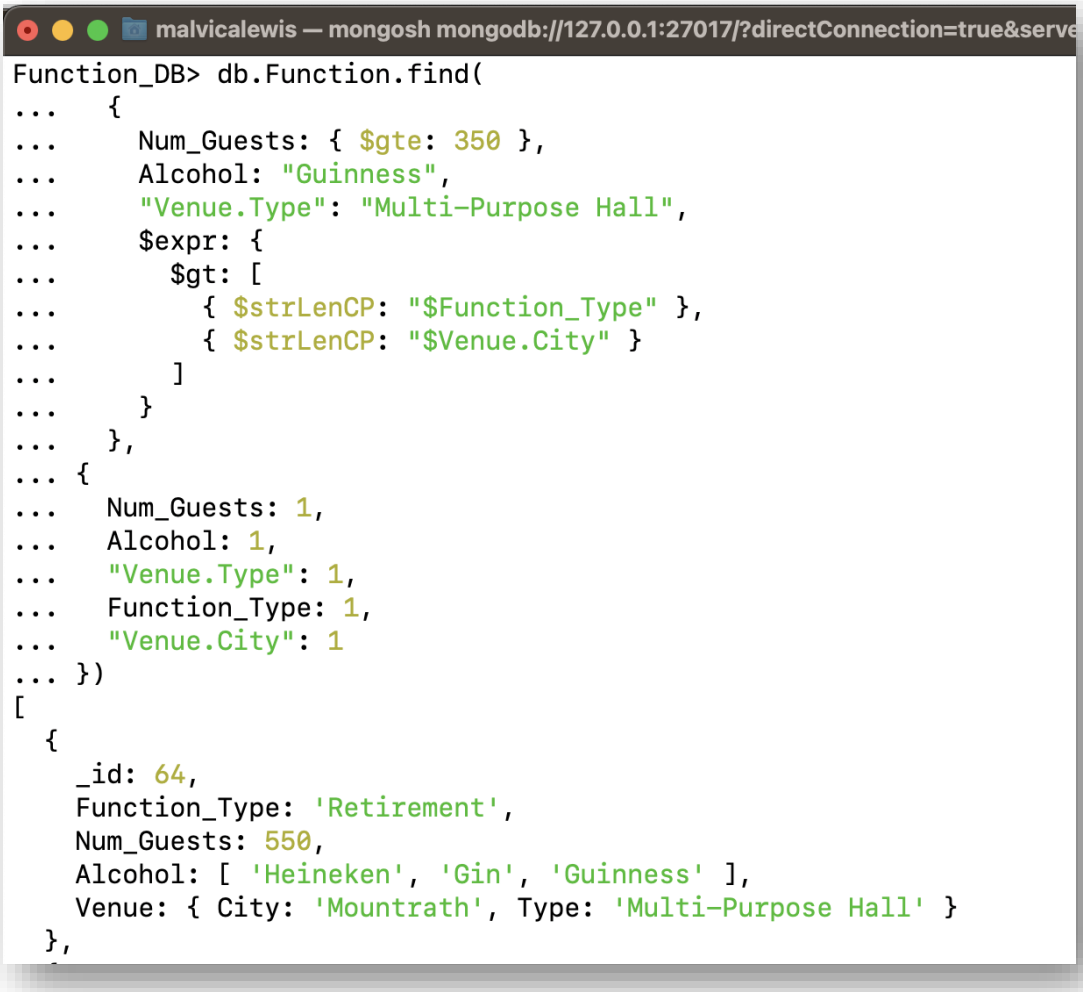
```
db.Function. find (
 {
  Num_Guests: { $gte: 350 },
  Alcohol: "Guinness",
  "Venue.Type": "Multi-Purpose Hall",
  $expr: {
   $gt: [
    { $strLenCP: "$Function_Type" },
    { $strLenCP: "$Venue.City" }
   ]
  }
 },
 {
  Num_Guests: 1,
  Alcohol: 1,
  "Venue.Type": 1,
  Function_Type: 1,
  "Venue.City": 1
 })
```

The query uses the following functions:

- **$gte** - This function stands for "greater than or equal to" and is used to specify a condition where the value of the **Num_Guests** field must be greater than or equal to **350**.
- **$strLenCP** - This function stands for "string length code points" and is used to return the length of the string in code points. It is used twice in this query to compare the length of two strings: **Function_Type** and **Venue.City**.

- **$expr** - This function allows the use of aggregation expressions within the query language. In this query, it is used to compare the length of the two strings mentioned above.
- **$gt** - This function stands for "greater than" and is used to specify a condition where the value of the first argument (in this case, the length of **Function_Type**) must be greater than the value of the second argument (in this case, the length of **Venue.City**).
- **"Venue.Type": "Multi-Purpose Hall"** - This is a simple equality check that specifies that the value of the Type field within the Venue subdocument must be "**Multi-Purpose Hall**".
- **Alcohol: "Guinness"** - This is also a simple equality check that specifies that the value of the Alcohol field must be "**Guinness**".

Query Output:



Query Count:


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directCor
Function_DB> db.Function. countDocuments (
... {
... Num_Guests: { $gte: 350 },
... Alcohol: "Guinness",
... "Venue.Type": "Multi-Purpose Hall",
... $expr: {
... $gt: [
... { $strLenCP: "$Function_Type" },
... { $strLenCP: "$Venue.City" }
...]
... }
... },
... {
... Num_Guests: 1,
... Alcohol: 1,
... "Venue.Type": 1,
... Function_Type: 1,
... "Venue.City": 1
... })
15
Function_DB>

13. $or, $gt, $options, $in, $lt

Question: Find the events

- where the disc jockey's name ends with "s" or contains the letter "B" (case-insensitive),
- the number of guests is greater 100,
- the alcohol served to the customers is either "Gin" or "Vodka",
- soft drinks are either "Coca Cola" or "Fanta", and
- the duration of the event is less than 4 hours.

Sort the output in descending order of number of guests.

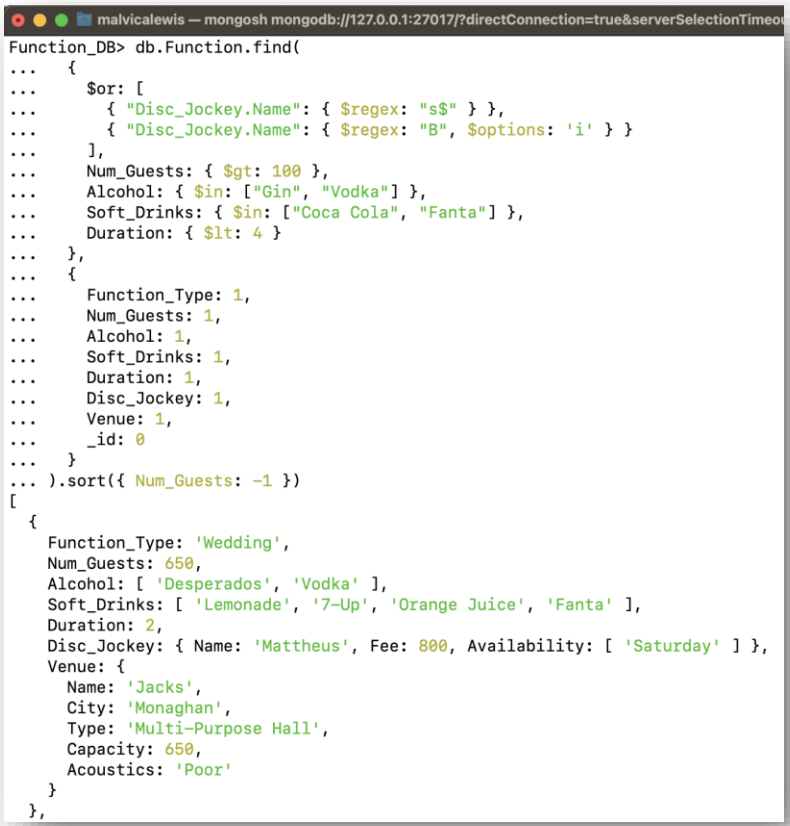
Query:

```
db.Function.find(
 {
  $or: [
   { "Disc_Jockey.Name": { $regex: "s$" } },
   { "Disc_Jockey.Name": { $regex: "B", $options: 'i' } }
  ],
  Num_Guests: { $gt: 100 },
  Alcohol: { $in: ["Gin", "Vodka"] },
  Soft_Drinks: { $in: ["Coca Cola", "Fanta"] },
  Duration: { $lt: 4 }
 },
 {
  Function_Type: 1,
  Num_Guests: 1,
  Alcohol: 1,
  Soft_Drinks: 1,
  Duration: 1,
  Disc_Jockey: 1,
  Venue: 1,
  _id: 0
 }
).sort({ Num_Guests: -1 })
```

The query uses the following functions:

- **$or**: This operator specifies an array of one or more query conditions. The query returns documents that match any of the conditions in the array.
- **$regex**: This operator performs a regular expression match on a specified field. In this query, it searches for DJ names that end with "**s**" or contain the letter "**B**" (case-insensitive).
- **$gt**: This operator selects documents where the value of the "**Num_Guests**" field is greater than the specified value (in this case, **100**).
- **$in**: This operator selects documents where the value of the specified field matches any of the values in the specified array.
- **$lt**: This operator selects documents where the value of the specified field is less than the specified value (in this case, **4**).
- **sort**: This function sorts the documents returned by the query by the value of the "**Num_Guests**" field in descending order.

Query Output:



Query Count:


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&ser
Function_DB> db.Function.countDocuments(
... {
... $or: [
... { "Disc_Jockey.Name": { $regex: "s$" } },
... { "Disc_Jockey.Name": { $regex: "B", $options: 'i' } }
...],
... Num_Guests: { $gt: 100 },
... Alcohol: { $in: ["Gin", "Vodka"] },
... Soft_Drinks: { $in: ["Coca Cola", "Fanta"] },
... Duration: { $lt: 4 }
... },
... {
... Function_Type: 1,
... Num_Guests: 1,
... Alcohol: 1,
... Soft_Drinks: 1,
... Duration: 1,
... Disc_Jockey: 1,
... Venue: 1,
... _id: 0
... }
...)
9
Function_DB>

14. $divide, $and, $expr, $gte, $in

Question: Find the events

- where the number of guests is at least 80% of the venue capacity,
- the venue is in Mountrath,
- the function includes "Fanta" as soft drinks, and
- lasts for more than 8 hours.

Query:

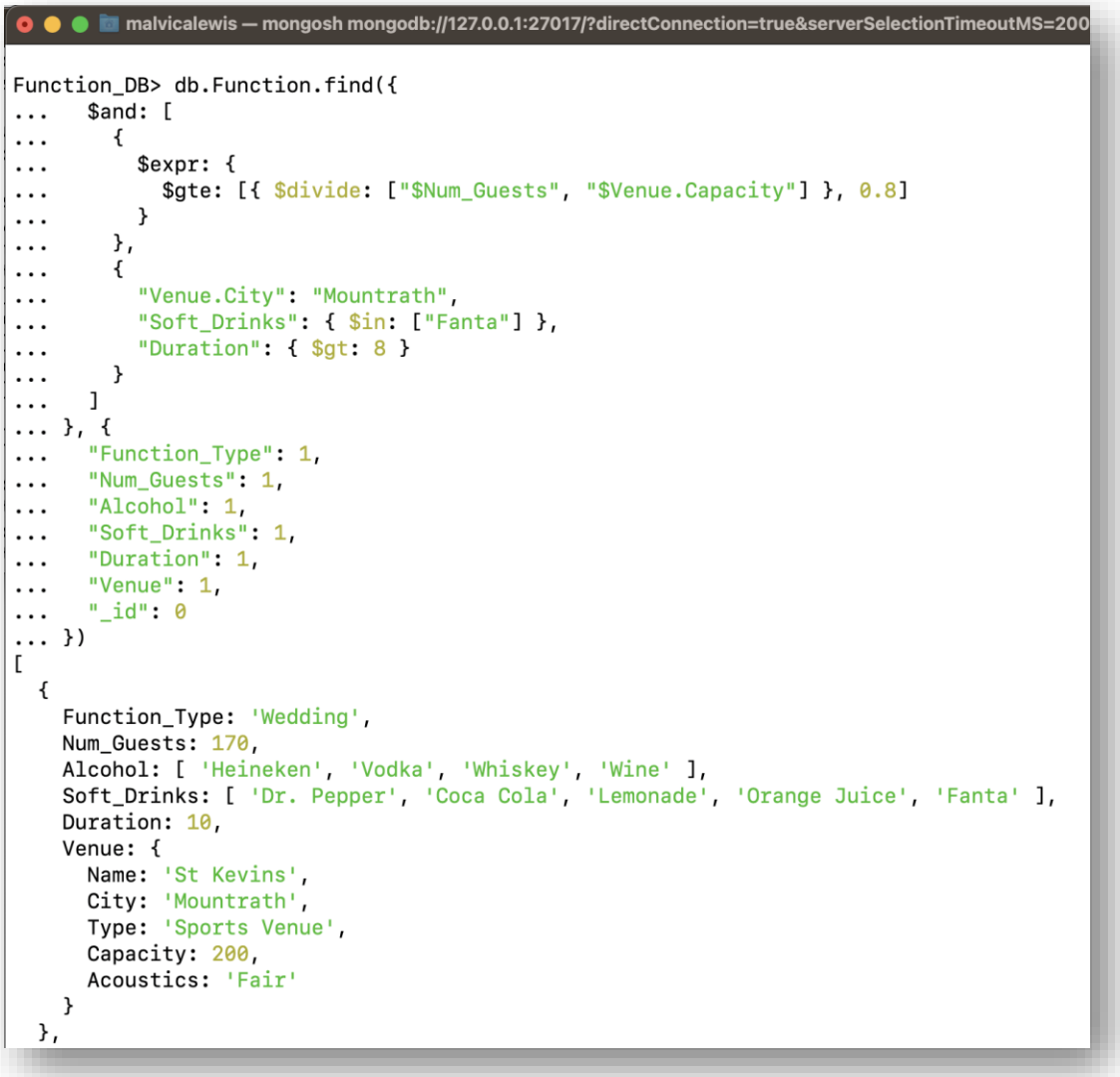
```
db.Function.find({
 $and: [
  {
   $expr: {
    $gte: [{ $divide: ["$Num_Guests", "$Venue.Capacity"] }, 0.8]
   }
  },
  {
   "Venue.City": "Mountrath",
   "Soft_Drinks": { $in: ["Fanta"] },
   "Duration": { $gt: 8 }
  }
 ]
},
{
 "Function_Type": 1,
 "Num_Guests": 1,
 "Alcohol": 1,
 "Soft_Drinks": 1,
 "Duration": 1,
 "Venue": 1,
 "_id": 0
})
```

The query uses the following functions:

- **db.Function.find()**: searches the Function collection for documents that match the specified criteria.
- **$and**: specifies that both enclosed conditions must be true for a document to be returned.

- **$expr**: allows the use of aggregation expressions within the query language. In this case, it calculates the ratio of **Num_Guests** to **Venue.Capacity** and checks if it's greater than or equal to **0.8**.
- **$divide**: divides the first operand by the second operand.
- **"Venue.City": "Mountrath"**: filters documents based on whether the **Venue.City** field equals "**Mountrath**".
- **"Soft_Drinks": { $in: ["Fanta"] }**: filters documents based on whether the **Soft_Drinks** field contains "**Fanta**".
- **"Duration": { $gt: 8 }**: filters documents based on whether the **Duration** field is greater than **8**.

Query Output:



Query Count:


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelecti
Function_DB> db.Function.countDocuments({
... $and: [
... {
... $expr: {
... $gte: [{ $divide: ["$Num_Guests", "$Venue.Capacity"] }, 0.8]
... }
... },
... {
... "Venue.City": "Mountrath",
... "Soft_Drinks": { $in: ["Fanta"] },
... "Duration": { $gt: 8 }
... }
...]
... }, {
... "Function_Type": 1,
... "Num_Guests": 1,
... "Alcohol": 1,
... "Soft_Drinks": 1,
... "Duration": 1,
... "Venue": 1,
... "_id": 0
... })
10
Function_DB>

15. $multiply, $size, $lte, $gte, $expr, $and

Question: Find the events

- where the duration of the function is lesser than or equal to the number of food choices, and
- the disc jockey fee is lesser than or equal to 40% of the venue capacity.

Query:

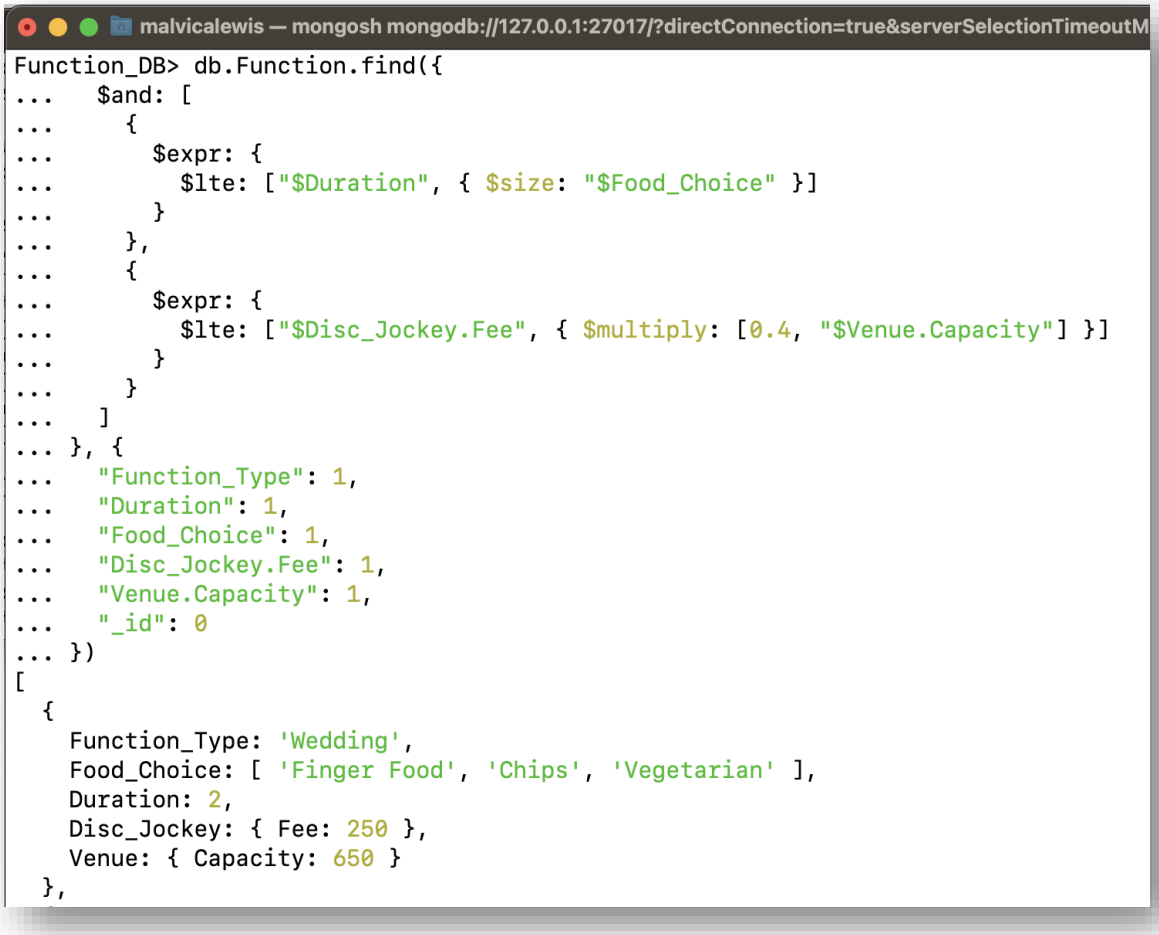
```
db.Function.find({
 $and: [
  {
   $expr: {
    $lte: ["$Duration", { $size: "$Food_Choice" }]
   }
  },
  {
   $expr: {
    $lte: ["$Disc_Jockey.Fee", { $multiply: [0.4, "$Venue.Capacity"] }]
   }
  }
 ]
}, {
 "Function_Type": 1,
 "Duration": 1,
 "Food_Choice": 1,
 "Disc_Jockey.Fee": 1,
 "Venue.Capacity": 1,
 "_id": 0
})
```

The query uses the following functions:

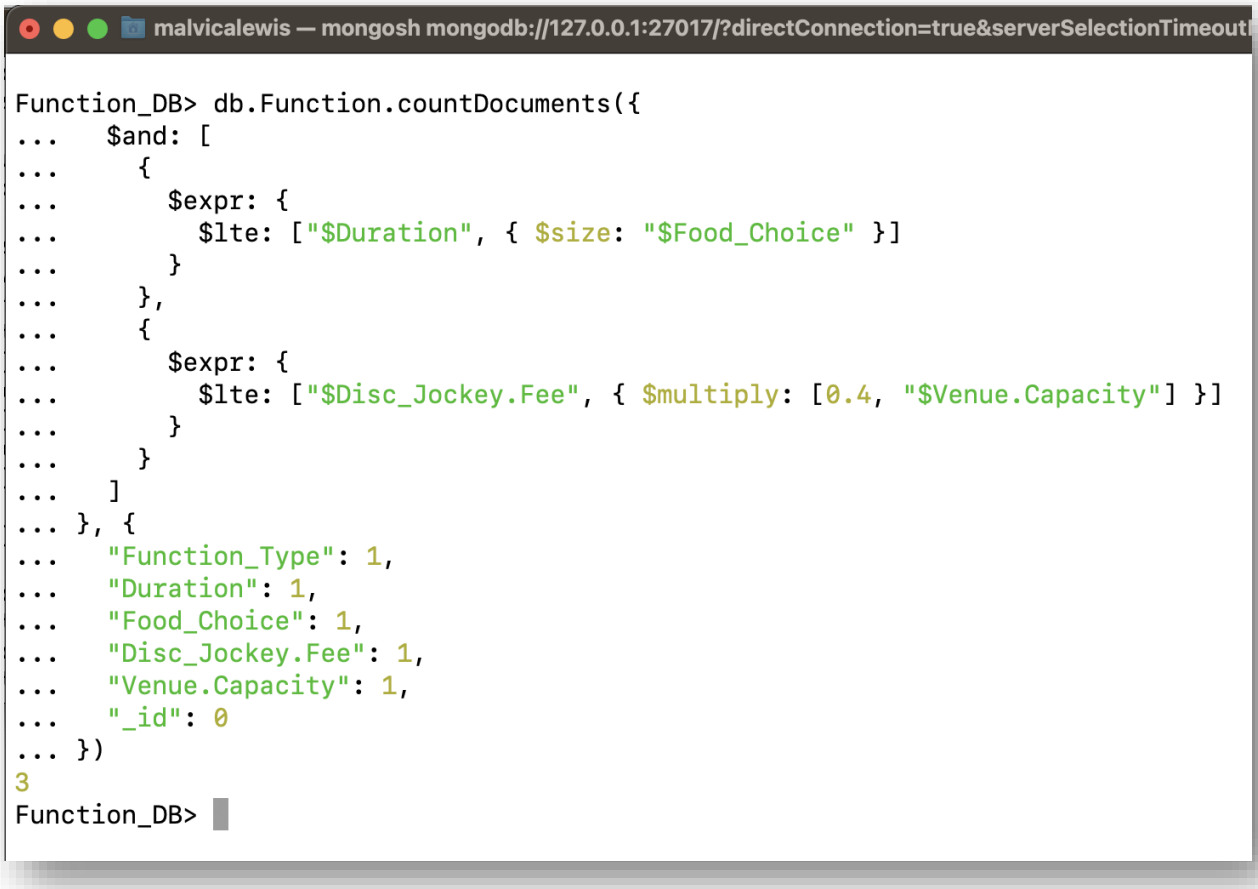
- **$and**: This is a logical operator that combines two or more expressions and returns true only if all of the expressions are true. In this case, it combines two expressions that are contained in two separate **$expr** operators.
- **$expr**: This is an aggregation operator that allows the use of aggregation expressions inside a query. In this case, it is used to compare two fields within a document.
- **$lte**: This is a comparison operator that stands for "less than or equal to". It returns true if the first operand is less than or equal to the second operand.

- **$Duration** and **$Food_Choice**: These are field names in the Function collection. **$Duration** represents the duration of the function and **$Food_Choice** is an array of food choices.
- **{ $size: "$Food_Choice" }**: This expression is used to retrieve the size of the **Food_Choice** array. **$size** is an aggregation operator that returns the number of elements in an array.
- **$Disc_Jockey.Fee** and **$Venue.Capacity**: These are nested field names in the Function collection. **$Disc_Jockey.Fee** represents the fee for a disc jockey and **$Venue.Capacity** represents the capacity of the venue.
- **{ $multiply: [0.4, "$Venue.Capacity"] }**: This expression is used to multiply the **Venue.Capacity** value by **0.4**. $multiply is an aggregation operator that returns the product of two or more values.

Query Output:



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeout
Function_DB> db.Function.countDocuments({
... $and: [
... {
... $expr: {
... $lte: ["$Duration", { $size: "$Food_Choice" }]
... }
... },
... {
... $expr: {
... $lte: ["$Disc_Jockey.Fee", { $multiply: [0.4, "$Venue.Capacity"] }]
... }
... }
...]
... }, {
... "Function_Type": 1,
... "Duration": 1,
... "Food_Choice": 1,
... "Disc_Jockey.Fee": 1,
... "Venue.Capacity": 1,
... "_id": 0
... })
3
Function_DB>

16. $size, $mod, $and, $gt, $expr, $strLenCP

Question: Find the events

- where the variety of Alcohol served is greater than 25% of the variety of soft drinks,, and
- the length of the city of the venue is an even number.

Query:

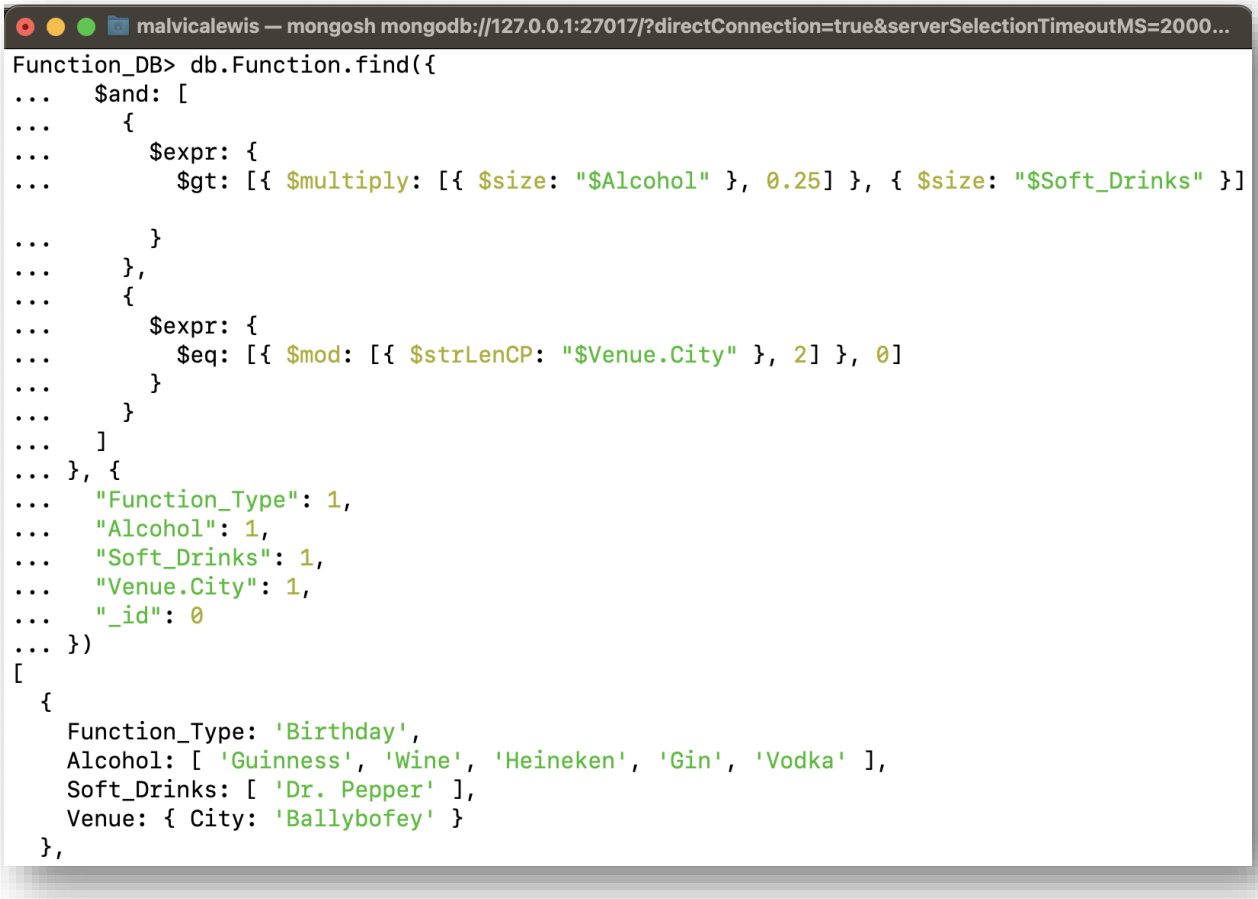
```
db.Function.find({
 $and: [
  {
   $expr: {
    $gt: [{ $multiply: [{ $size: "$Alcohol" }, 0.25] }, { $size:
"$Soft_Drinks" }]
   }
  },
  {
   $expr: {
    $eq: [{ $mod: [{ $strLenCP: "$Venue.City" }, 2] }, 0]
   }
  }
 ]
}, {
 "Function_Type": 1,
 "Alcohol": 1,
 "Soft_Drinks": 1,
 "Venue.City": 1,
 "_id": 0
})
```

The query uses the following functions:

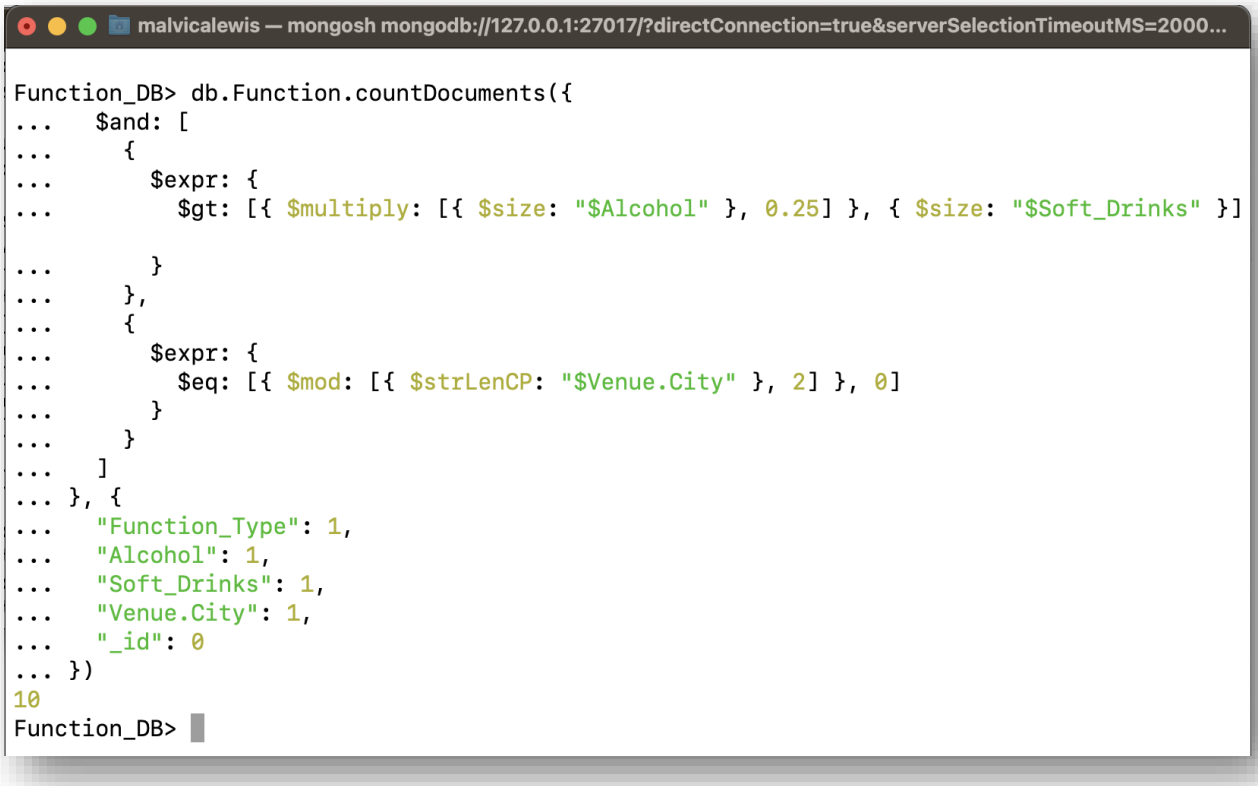
- **$size**: Returns the number of elements in an array. In this query, it is used to compare the sizes of the "**Alcohol**" and "**Soft_Drinks**" arrays.
- **$expr**: Allows the use of aggregation expressions within the query language. In this query, it is used to compare the sizes of the "**Alcohol**" and "**Soft_Drinks**" arrays.
- **$gt**: Performs a greater-than comparison and returns true if the first argument is greater than the second argument. In this query, it is used to compare the sizes of the "**Alcohol**" and "**Soft_Drinks**" arrays.

- **$strLenCP**: Returns the number of UTF-8 encoded code points in a string. In this query, it is used to calculate the length of the "**City**" field in the "**Venue**" subdocument.
- **$mod**: Returns the remainder of dividing the first argument by the second argument. In this query, it is used to check if the length of the "**City**" field in the "**Venue**" subdocument is an even number.
- **$eq**: Performs an equality comparison and returns true if the two arguments are equal. In this query, it is used to check if the length of the "**City**" field in the "**Venue**" subdocument is an even number.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000...
Function_DB> db.Function.countDocuments({
... $and: [
... {
... $expr: {
... $gt: [{ $multiply: [{ $size: "$Alcohol" }, 0.25] }, { $size: "$Soft_Drinks" }]
... }
... },
... {
... $expr: {
... $eq: [{ $mod: [{ $strLenCP: "$Venue.City" }, 2] }, 0]
... }
... }
...]
... }, {
... "Function_Type": 1,
... "Alcohol": 1,
... "Soft_Drinks": 1,
... "Venue.City": 1,
... "_id": 0
... })
10
Function_DB>

17. $options, $size, $divide, $all, $not, $regex, $mod, $strLenCP, $expr, $and, $gt

Question: Find the events

- That includes only weddings,
- with an even number of guests,
- where the length of the venue name is greater than half the number of alcohol choices,
- the food choice includes fish,
- there is more than one food choice, and
- the venue type ends with "hall" regardless of case

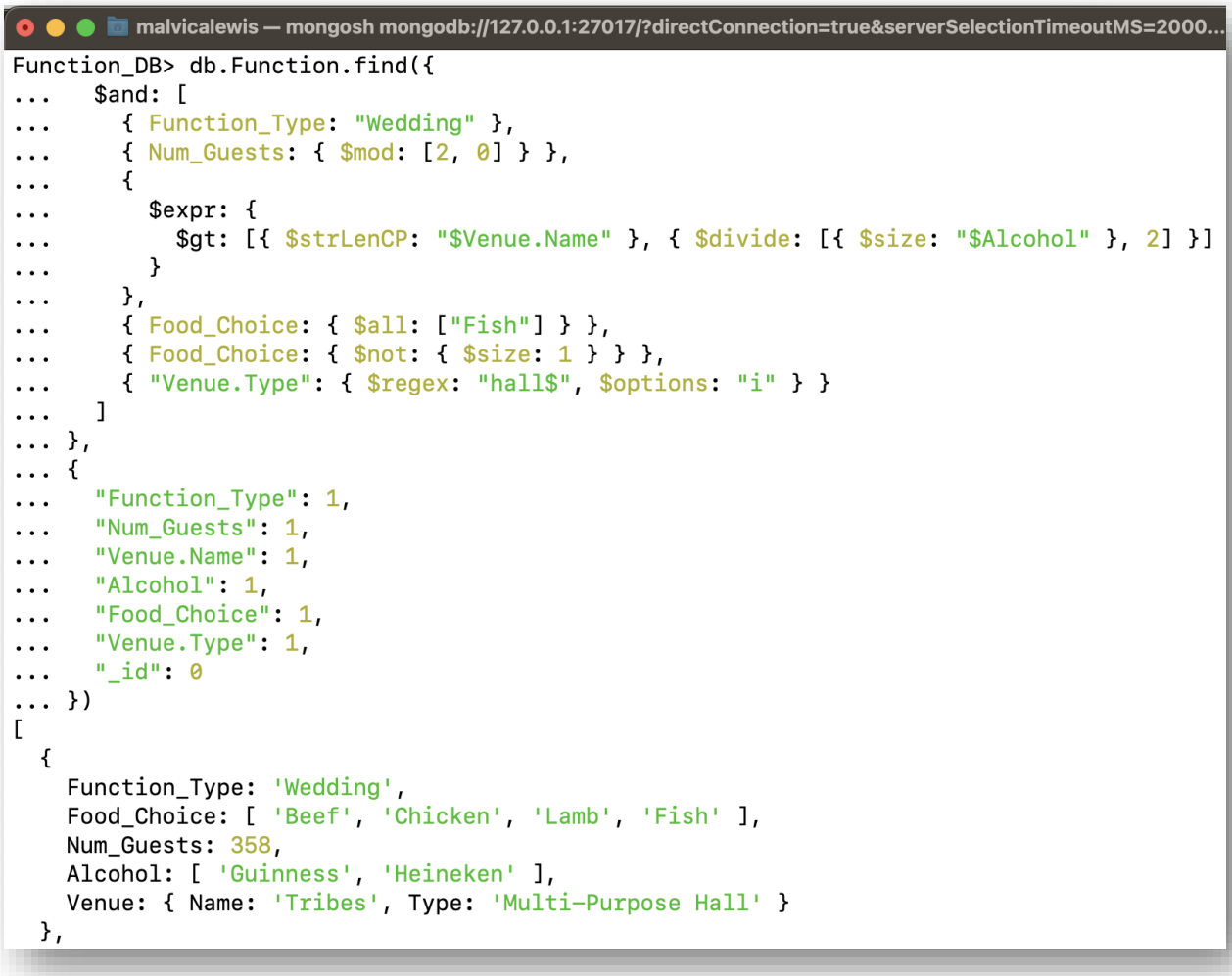
Query:

```
db.Function.find({
 $and: [
  { Function_Type: "Wedding" },
  { Num_Guests: { $mod: [2, 0] } },
  {
   $expr: {
    $gt: [{ $strLenCP: "$Venue.Name" }, { $divide: [{ $size: "$Alcohol" },
2] }]
   }
  },
  { Food_Choice: { $all: ["Fish"] } },
  { Food_Choice: { $not: { $size: 1 } } },
  { "Venue.Type": { $regex: "hall$", $options: "i" } }
 ]
},
{
 "Function_Type": 1,
 "Num_Guests": 1,
 "Venue.Name": 1,
 "Alcohol": 1,
 "Food_Choice": 1,
 "Venue.Type": 1,
 "_id": 0
})
```

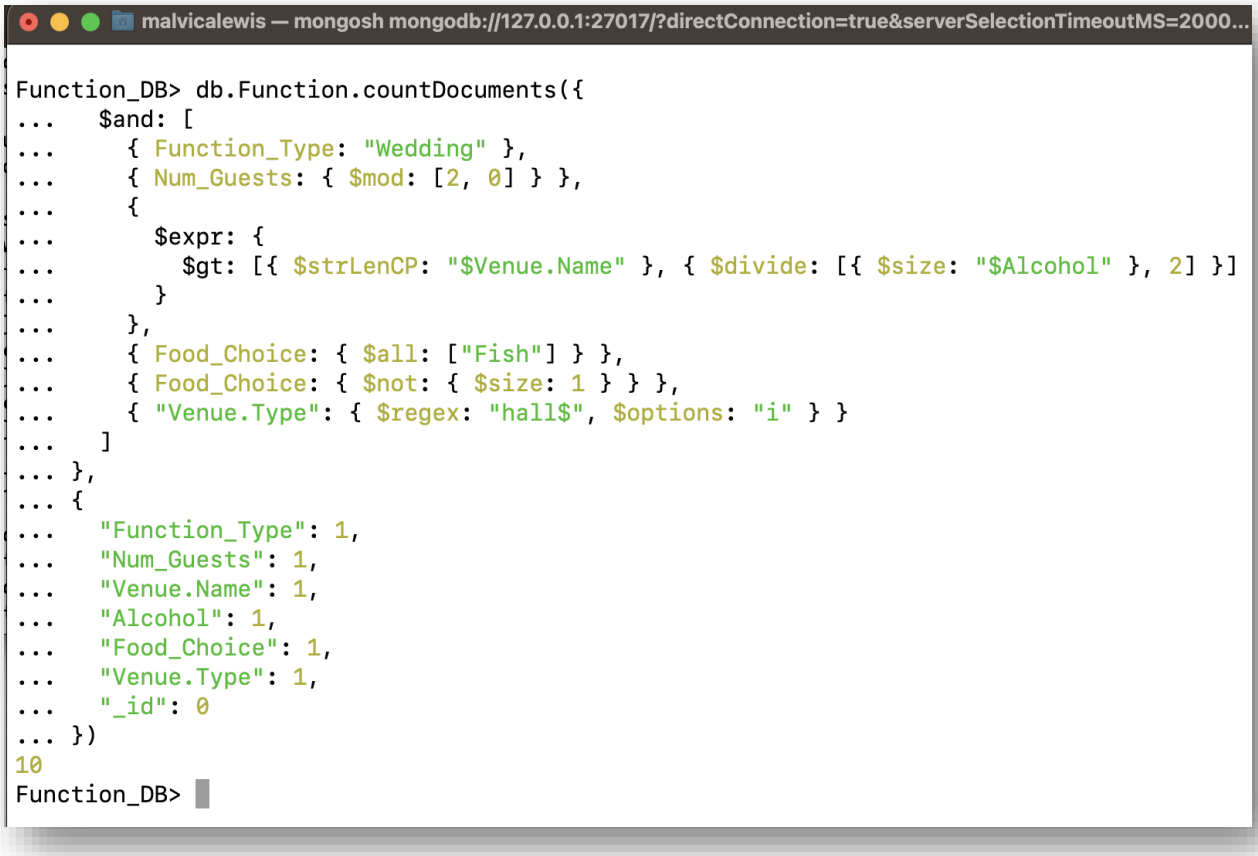
The query uses the following functions:

- **db.Function.find()** - This function is used to retrieve documents from the "**Function**" collection.
- **$and** - This is a logical operator used to combine multiple conditions in a single query.
- **{ Function_Type: "Wedding" }** - This specifies the first condition, which requires the "**Function_Type**" field to have the value "**Wedding**".
- **{ Num_Guests: { $mod: [2, 0] } }** - This specifies the second condition, which requires the "**Num_Guests**" field to be an even number (i.e., the remainder of dividing by **2** is **0**). The **$mod** operator is used for this purpose.
- **$expr** - This is an aggregation expression that allows the use of aggregation functions within a query.
- **{ $gt: [{ $strLenCP: "$Venue.Name" }, { $divide: [{ $size: "$Alcohol" }, 2] }] }** - This specifies the third condition, which compares the length of the "**Venue.Name**" field to half the size of the "**Alcohol**" array. The **$strLenCP** operator is used to get the length of the string and the **$divide** operator is used to divide the size of the "**Alcohol**" array by **2**. The **$gt** operator is used to check if the length of the "**Venue.Name**" field is greater than half the size of the "**Alcohol**" array.
- **{ Food_Choice: { $all: ["Fish"] } }** - This specifies the fourth condition, which requires the "**Food_Choice**" array to contain the value "**Fish**". The **$all** operator is used for this purpose.
- **{ Food_Choice: { $not: { $size: 1 } } }** - This specifies the fifth condition, which requires the "**Food_Choice**" array to have a size greater than **1**. The **$not** operator is used to negate the expression, and the **$size** operator is used to get the size of the array.
- **{ "Venue.Type": { $regex: "hall$", $options: "i" } }** - This specifies the sixth condition, which requires the "**Venue.Type**" field to end with the string "**hall**" (case-insensitive). The **$regex** operator is used for this purpose.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2000...
Function_DB> db.Function.countDocuments({
... $and: [
... { Function_Type: "Wedding" },
... { Num_Guests: { $mod: [2, 0] } },
... {
... $expr: {
... $gt: [{ $strLenCP: "$Venue.Name" }, { $divide: [{ $size: "$Alcohol" }, 2] }]
... }
... },
... { Food_Choice: { $all: ["Fish"] } },
... { Food_Choice: { $not: { $size: 1 } } },
... { "Venue.Type": { $regex: "hall$", $options: "i" } }
...]
... },
... {
... "Function_Type": 1,
... "Num_Guests": 1,
... "Venue.Name": 1,
... "Alcohol": 1,
... "Food_Choice": 1,
... "Venue.Type": 1,
... "_id": 0
... })
10
Function_DB>

18. $divide, $size, $expr, $and, $gte, $lte

Question: Find the events where

- the number of items of alcohol served is at least 6,
- the number of items of soft drinks served is not more than 2
- the duration of the event should be greater than or equal to the 1% of total number of guests who attended the event.

Query:

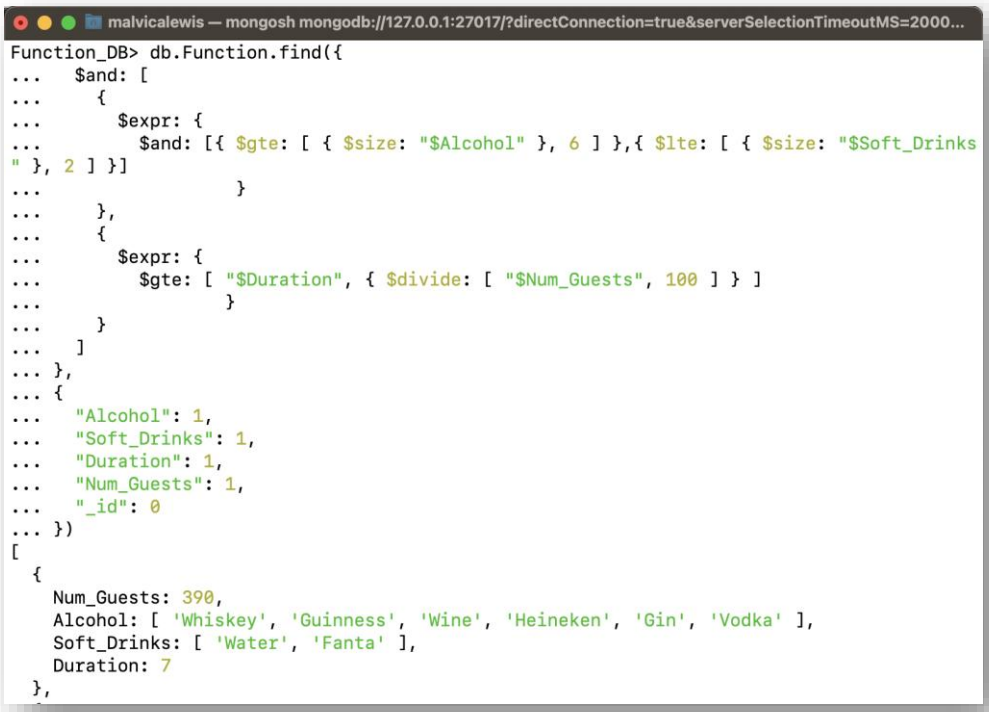
```
db.Function.find({
 $and: [
  {
   $expr: {
    $and: [{ $gte: [ { $size: "$Alcohol" }, 6 ] },{ $lte: [ { $size:
"$Soft_Drinks" }, 2 ] }]
          }
  },
  {
   $expr: {
    $gte: [ "$Duration", { $divide: [ "$Num_Guests", 100 ] } ]
         }
  }
 ]
},
{
 "Alcohol": 1,
 "Soft_Drinks": 1,
 "Duration": 1,
 "Num_Guests": 1,
 "_id": 0
})
```

The query uses the following functions:

- The first condition is an **$and** operator that specifies two **$expr** expressions:
  - The first **$expr** expression checks if the size of the "**Alcohol**" array is greater than or equal to **6**, and the size of the "**Soft_Drinks**" array is less than or equal to **2**.
- The second **$expr** expression checks if the "**Duration**" field is greater than or equal to the "**Num_Guests**" field divided by **100**.

- The query returns documents that satisfy both the conditions.

Query Output



Query Count

19. $all, $strLenCP, $expr, $and, $lt, $in

Question: Find the events where

- the events are either of type "Retirement",
- have a disc jockey available on both "Monday" and "Friday",
- their function type length is less than the length of their venue type.

Query:

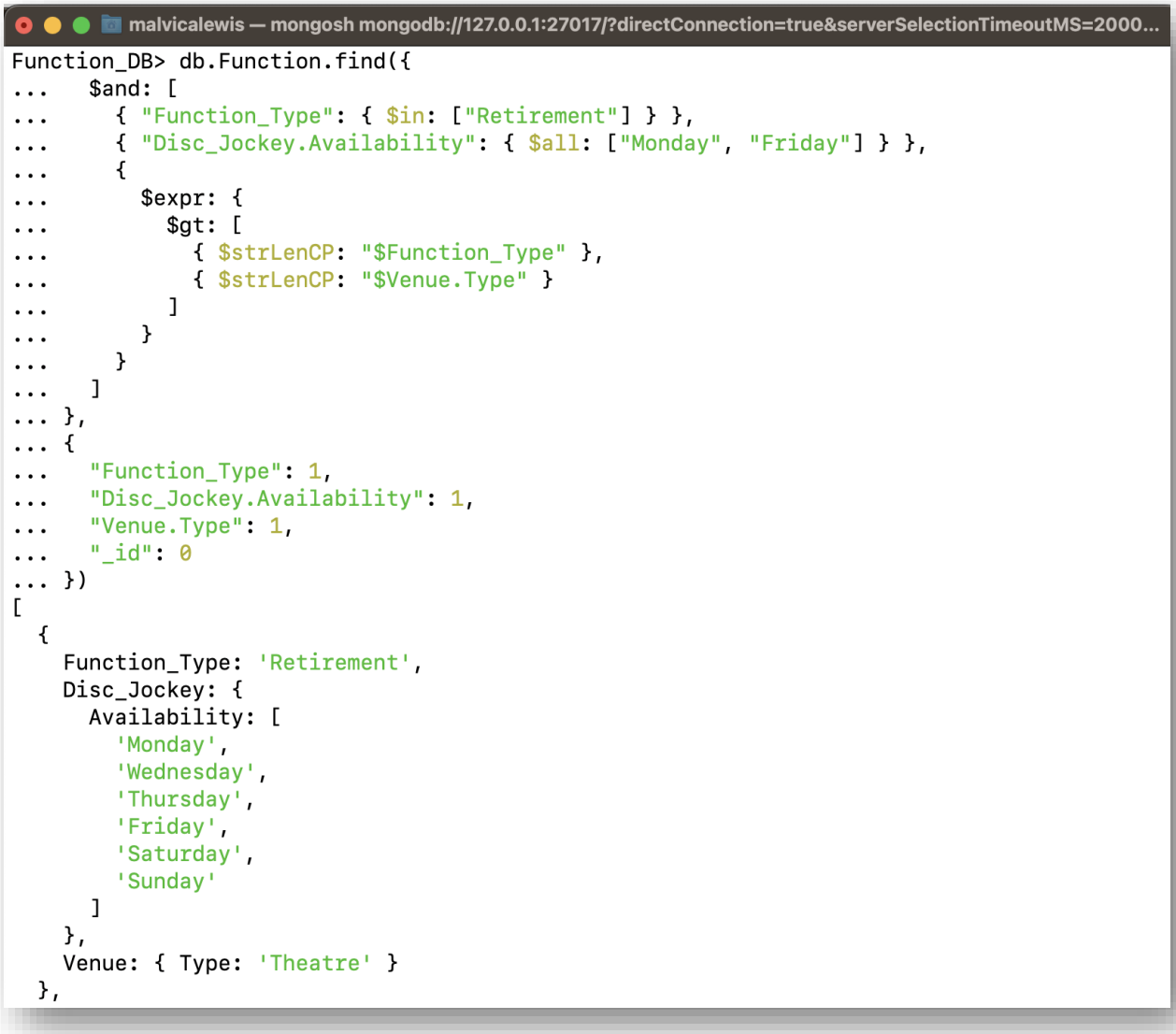
```
db.Function.find({
 $and: [
  { "Function_Type": { $in: ["Retirement"] } },
  { "Disc_Jockey.Availability": { $all: ["Monday", "Friday"] } },
  {
   $expr: {
    $gt: [
     { $strLenCP: "$Function_Type" },
     { $strLenCP: "$Venue.Type" }
    ]
   }
  }
 ]
},
{
 "Function_Type": 1,
 "Disc_Jockey.Availability": 1,
 "Venue.Type": 1,
 "_id": 0
})
```

The query uses the following functions:

- The first function, **{ $in: ["Retirement"] }**, checks if the value of the "**Function_Type**" field is equal to "**Retirement**". The **$in** operator matches any of the values specified in an array.
- The second function, **{ $all: ["Monday", "Friday"] }**, checks if the "**Disc_Jockey.Availability**" field contains both "**Monday**" and "**Friday**". The **$all** operator matches arrays that contain all elements specified in an array.

- The third function, **$gt: [{ $strLenCP: "$Function_Type" },{ $strLenCP: "$Venue.Type" }]**, checks if the length of the string value of the "**Function_Type**" field is greater than the length of the string value of the "**Venue.Type**" field. The **$strLenCP** operator returns the length of a string in UTF-8 code points.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelec
Function_DB> db.Function.countDocuments({
... $and: [
... { "Function_Type": { $in: ["Retirement"] } },
... { "Disc_Jockey.Availability": { $all: ["Monday", "Friday"] } },
... {
... $expr: {
... $gt: [
... { $strLenCP: "$Function_Type" },
... { $strLenCP: "$Venue.Type" }
...]
... }
... }
...]
... },
... {
... "Function_Type": 1,
... "Disc_Jockey.Availability": 1,
... "Venue.Type": 1,
... "_id": 0
... })
7
Function_DB>

20. $nin, $divide, $gte, $expr, $and, $gt

Question: Find the events where

- held in a sports venue,
- with at least 5 hours duration,
- but not on Monday, and
- the disc jockey fee is more than 12 euros per guest.

Query:

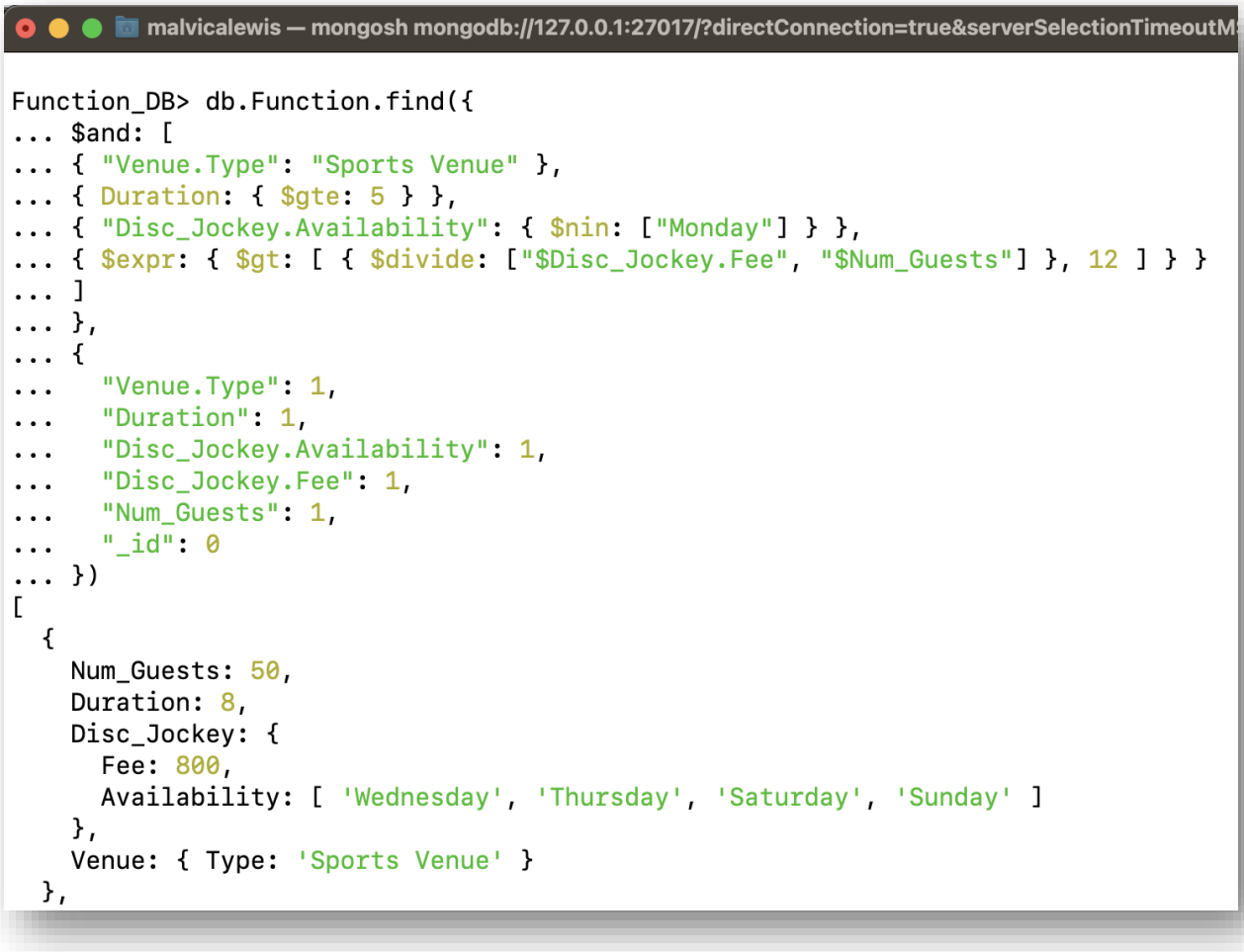
```
db.Function.find({
$and: [
{ "Venue.Type": "Sports Venue" },
{ Duration: { $gte: 5 } },
{ "Disc_Jockey.Availability": { $nin: ["Monday"] } },
{ $expr: { $gt: [ { $divide: ["$Disc_Jockey.Fee", "$Num_Guests"] }, 12 ] }
}
]
},
{
 "Venue.Type": 1,
 "Duration": 1,
 "Disc_Jockey.Availability": 1,
 "Disc_Jockey.Fee": 1,
 "Num_Guests": 1,
 "_id": 0
})
```

The query uses the following functions:

- **$and**: This operator allows you to combine multiple conditions and require that all of them are true for a document to match the query.
- **"Venue.Type": "Sports Venue"**: This is a condition that requires the value of the **Venue.Type** field to be exactly "**Sports Venue**".
- **Duration: { $gte: 5 }**: This is a condition that requires the value of the Duration field to be greater than or equal to **5**.
- "**Disc_Jockey.Availability": { $nin: ["Monday"] }**: This is a condition that requires the value of the **Disc_Jockey.Availability** field to not include "**Monday**".
- **$expr**: This operator allows you to use aggregation expressions in the query.

- **{ $gt: [{ $divide: ["$Disc_Jockey.Fee", "$Num_Guests"] }, 12] }**: This is an expression that compares the result of dividing the value of the **Disc_Jockey.Fee** field by the value of the **Num_Guests** field to **12**. The result of this comparison is true if the division result is greater than **12**, meaning the DJ fee is more than **12** euros per guest.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutM
Function_DB> db.Function.countDocuments({
... $and: [
... { "Venue.Type": "Sports Venue" },
... { Duration: { $gte: 5 } },
... { "Disc_Jockey.Availability": { $nin: ["Monday"] } },
... { $expr: { $gt: [{ $divide: ["$Disc_Jockey.Fee", "$Num_Guests"] }, 12] } }
...]
... },
... {
...   "Venue.Type": 1,
...   "Duration": 1,
...   "Disc_Jockey.Availability": 1,
...   "Disc_Jockey.Fee": 1,
...   "Num_Guests": 1,
...   "_id": 0
... })
5
Function_DB>

21. $regex, $and, $nin, $gte

Question: Find the events where

- the venue has a capacity of 400,
- the function type is not a wedding or a corporate event,
- the disc jockey's name starts with "E",
- the duration is at least 5 hours, and
- the alcohol served is Heineken.

Query:

```
db.Function.find({
 $and: [
  {
   "Venue.Capacity": 400,
   Function_Type: { $nin: ["Wedding", "Corporate Event"] },
   "Disc_Jockey.Name": { $regex: /^E/i }
  },
  {
   Duration: { $gte: 5 },
   Alcohol: "Heineken"
  }
 ]
},
{
 "Venue.Capacity": 1,
 "Function_Type": 1,
 "Disc_Jockey.Name": 1,
 "Duration": 1,
 "Alcohol": 1,
 "_id": 0
})
```

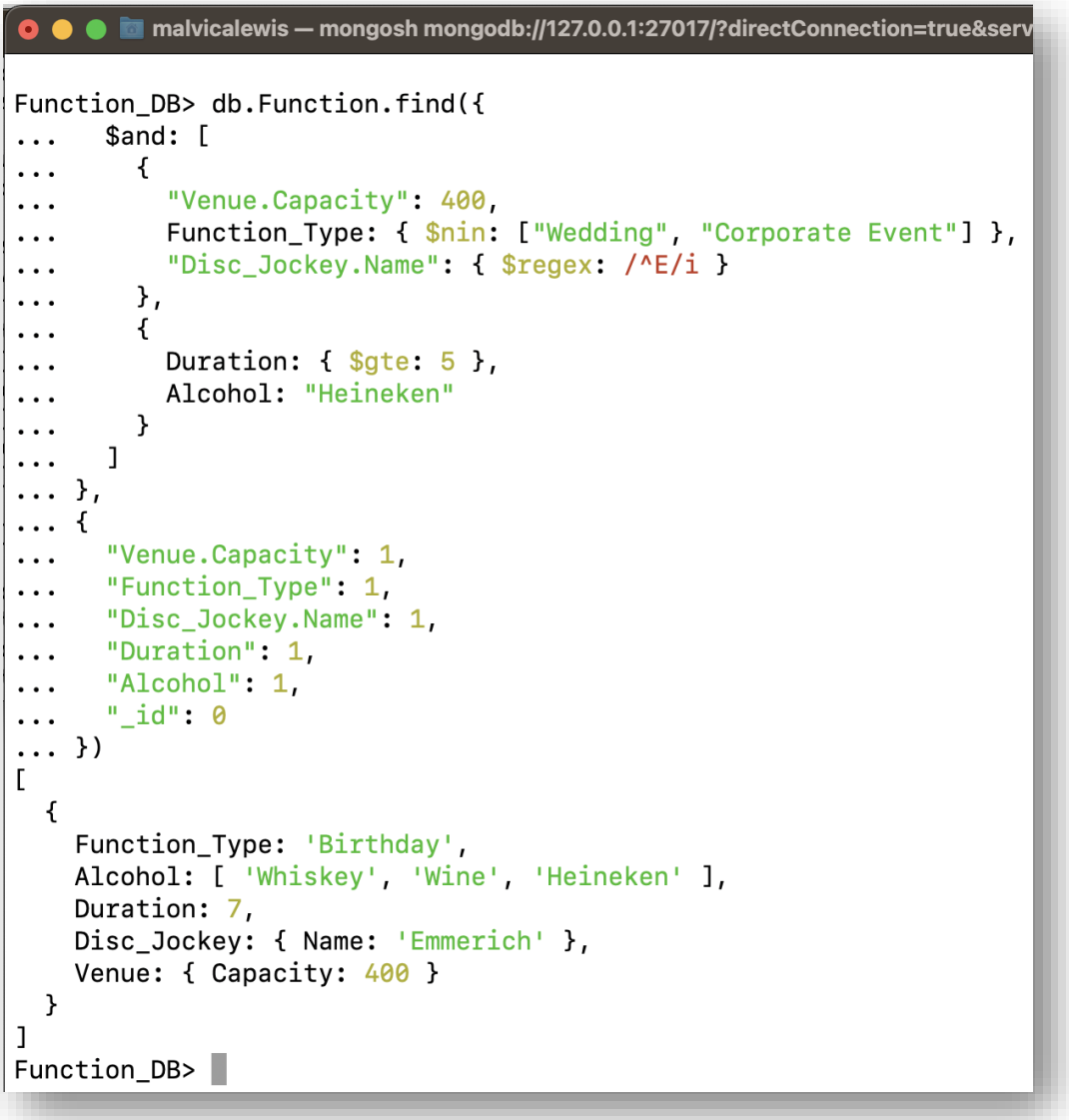
The query uses the following functions:

- The first **$and** operator includes three conditions that must all be true:
  - **"Venue.Capacity": 400**: The capacity of the venue must be **400**.
  - **Function_Type: { $nin: ["Wedding", "Corporate Event"] }**: The function type must not be "**Wedding**" or "**Corporate Event**".

- "**Disc_Jockey.Name": { $regex: /^E/i }**: The name of the disc jockey must start with "**E**" (case-insensitive).

- The second **$and** operator includes two conditions that must both be true:
  - **Duration: { $gte: 5 }**: The duration of the function must be greater than or equal to **5**.
  - **Alcohol: "Heineken"**: The alcohol served at the function must be "**Heineken**".

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serve
Function_DB> db.Function.countDocuments({
... $and: [
... {
... "Venue.Capacity": 400,
... Function_Type: { $nin: ["Wedding", "Corporate Event"] },
... "Disc_Jockey.Name": { $regex: /^E/i }
... },
... {
... Duration: { $gte: 5 },
... Alcohol: "Heineken"
... }
...]
... },
... {
... "Venue.Capacity": 1,
... "Function_Type": 1,
... "Disc_Jockey.Name": 1,
... "Duration": 1,
... "Alcohol": 1,
... "_id": 0
... })
1
Function_DB>

22. $or, $all, $expr, $lt, $strLenCP

Question: Find the events

- that are of retirement events,
- with a disc jockey available on both Monday and Saturday, and
- where the length of the function type is greater than the length of the venue type.

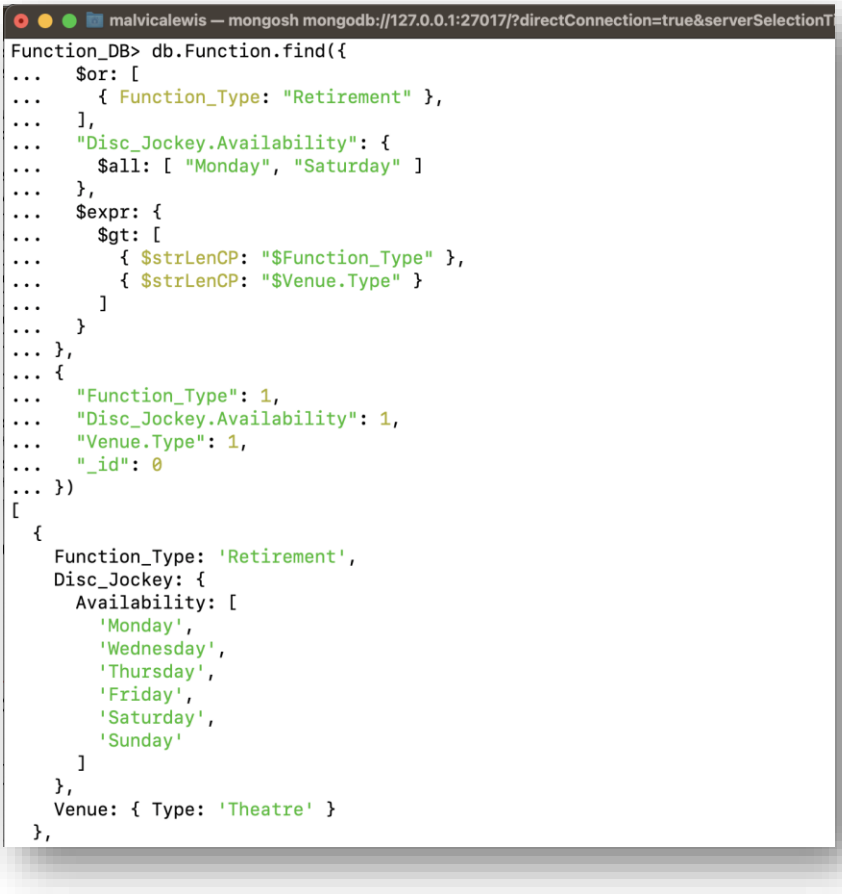
Query:

```
db.Function.find({
 $or: [
  { Function_Type: "Retirement" },
 ],
 "Disc_Jockey.Availability": {
  $all: [ "Monday", "Saturday" ]
 },
 $expr: {
  $gt: [
   { $strLenCP: "$Function_Type" },
   { $strLenCP: "$Venue.Type" }
  ]
 }
},
{
 "Function_Type": 1,
 "Disc_Jockey.Availability": 1,
 "Venue.Type": 1,
 "_id": 0
})
```

The query uses the following functions:

- **$or**: This function specifies that at least one of the conditions in the array should be true for the document to be included in the result set. In this case, the condition is that the value of the "**Function_Type**" field should be "**Retirement**".
- **"Disc_Jockey.Availability"**: This specifies that the value of the "**Availability**" field in the "**Disc_Jockey**" subdocument should contain both "**Monday**" and "**Saturday**".
- **$expr**: This specifies that a comparison between two expressions should be made to determine if a document should be included in the result set. In this case, the two expressions are the length of the "**Function_Type**" field and the length of the "**Venue.Type**" field.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConne
Function_DB> db.Function.countDocuments({
... $or: [
... { Function_Type: "Retirement" },
...],
... "Disc_Jockey.Availability": {
... $all: ["Monday", "Saturday"]
... },
... $expr: {
... $gt: [
... { $strLenCP: "$Function_Type" },
... { $strLenCP: "$Venue.Type" }
...]
... }
... },
... {
... "Function_Type": 1,
... "Disc_Jockey.Availability": 1,
... "Venue.Type": 1,
... "_id": 0
... })
8
Function_DB>

23. $size, $ne, $not, $multiply, $in, $and, $expr, $gt

Question: Find the events

- that offered either beef or lamb,
- did not offer wine,
- had more guests than 60% of the venue capacity,
- had fair or good acoustics, and
- had a disc jockey with a number of available days not equal to 3.

Query:

```
db.Function.find({
  $and: [
    { Food_Choice: { $in: ["Beef", "Lamb"] } },
    { Alcohol: { $ne: "Wine" } },
    { $expr: { $gt: ["$Num_Guests", { $multiply: ["$Venue.Capacity", 0.98]
}] } },
    { "Venue.Acoustics": { $in: ["Fair", "Good"] } },
    { "Disc_Jockey.Availability": { $not: { $size: 3 } } }
  ]
},
{
  "Food_Choice": 1,
  "Alcohol": 1,
  "Num_Guests": 1,
  "Venue.Capacity": 1,
  "Venue.Acoustics": 1,
  "Disc_Jockey.Availability": 1,
  "_id": 0
})
```

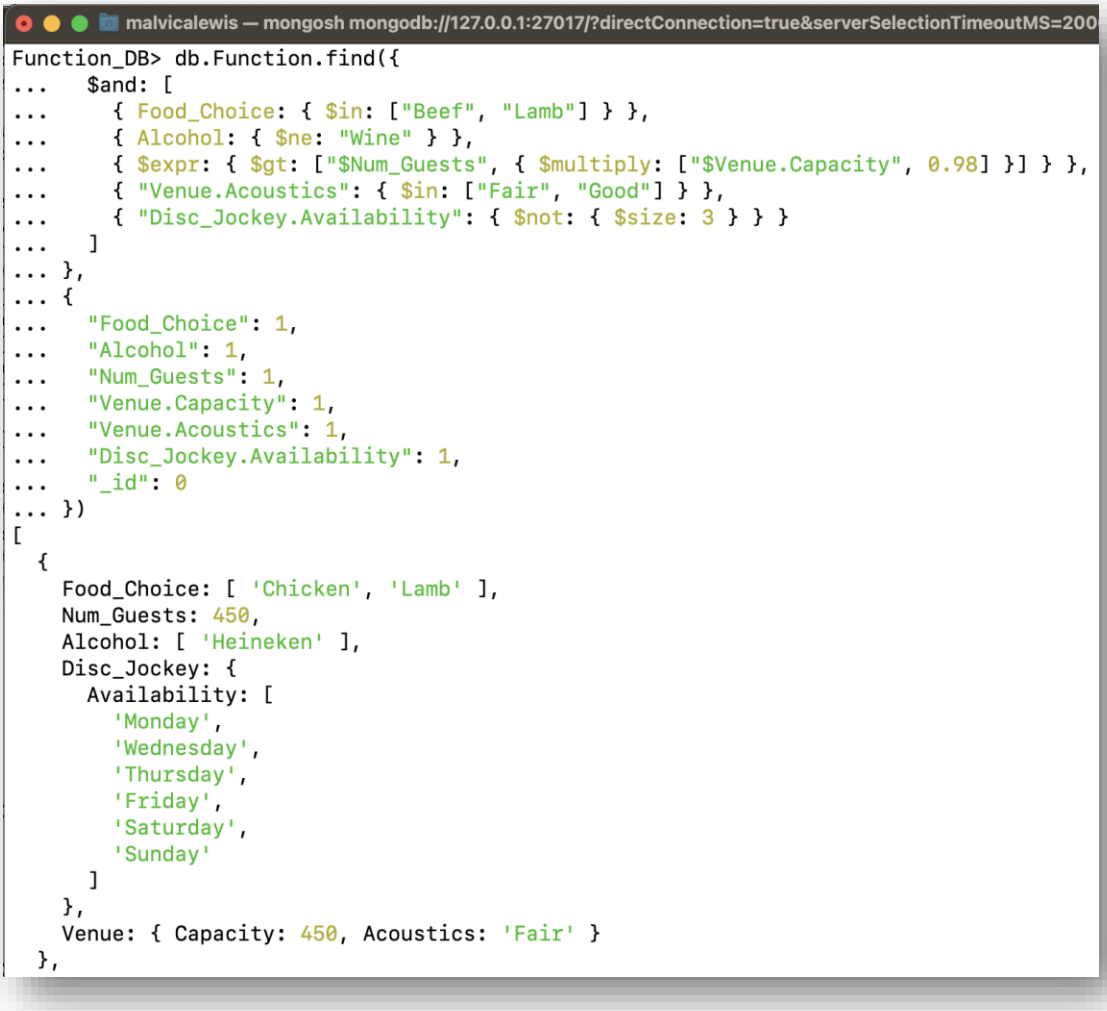
The query uses the following functions:

- **$and**: This operator combines multiple conditions that must all be true for a document to be returned by the query.
- **{ Food_Choice: { $in: ["Beef", "Lamb"] } }**: This specifies that the **Food_Choice** field must be either "**Beef**" or "**Lamb**".
- **{ Alcohol: { $ne: "Wine" } }**: This specifies that the Alcohol field must not be "**Wine**".
- **{ $expr: { $gt: ["$Num_Guests", { $multiply: ["$Venue.Capacity", 0.98] }] } }**: This uses the **$expr** operator to compare the **Num_Guests** field to a calculation that multiplies the

Capacity field of the **Venue** subdocument by **0.98** (i.e., 98%) and returns documents where **Num_Guests** is greater than this value.

- **{ "Venue.Acoustics": { $in: ["Fair", "Good"] } }**: This specifies that the **Acoustics** field of the **Venue** subdocument must be either "**Fair**" or "**Good**".
- **{ "Disc_Jockey.Availability": { $not: { $size: 3 } } }**: This specifies that the **Availability** field of the **Disc_Jockey** subdocument must not be an array with exactly **3** elements.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=200
Function_DB> db.Function.countDocuments({
... $and: [
... { Food_Choice: { $in: ["Beef", "Lamb"] } },
... { Alcohol: { $ne: "Wine" } },
... { $expr: { $gt: ["$Num_Guests", { $multiply: ["$Venue.Capacity", 0.98] }] } },
... { "Venue.Acoustics": { $in: ["Fair", "Good"] } },
... { "Disc_Jockey.Availability": { $not: { $size: 3 } } }
...]
... },
... {
... "Food_Choice": 1,
... "Alcohol": 1,
... "Num_Guests": 1,
... "Venue.Capacity": 1,
... "Venue.Acoustics": 1,
... "Disc_Jockey.Availability": 1,
... "_id": 0
... })
13
Function_DB>

24. $elemMatch $or, $and, $lte, $nin, $in, $gt, $eq, $not

Question: Find the events that match the criteria of having either:

- a venue in Athlone with a capacity of 320 or less and a disc jockey fee of 900 or less;
- a venue in Listowel with finger food as a food choice but no beef, and a disc jockey available on Tuesday; or
- an event in 2015 with more than 100 guests, held at a venue not in Drumcliff, serving lamb but not finger food.

Query:

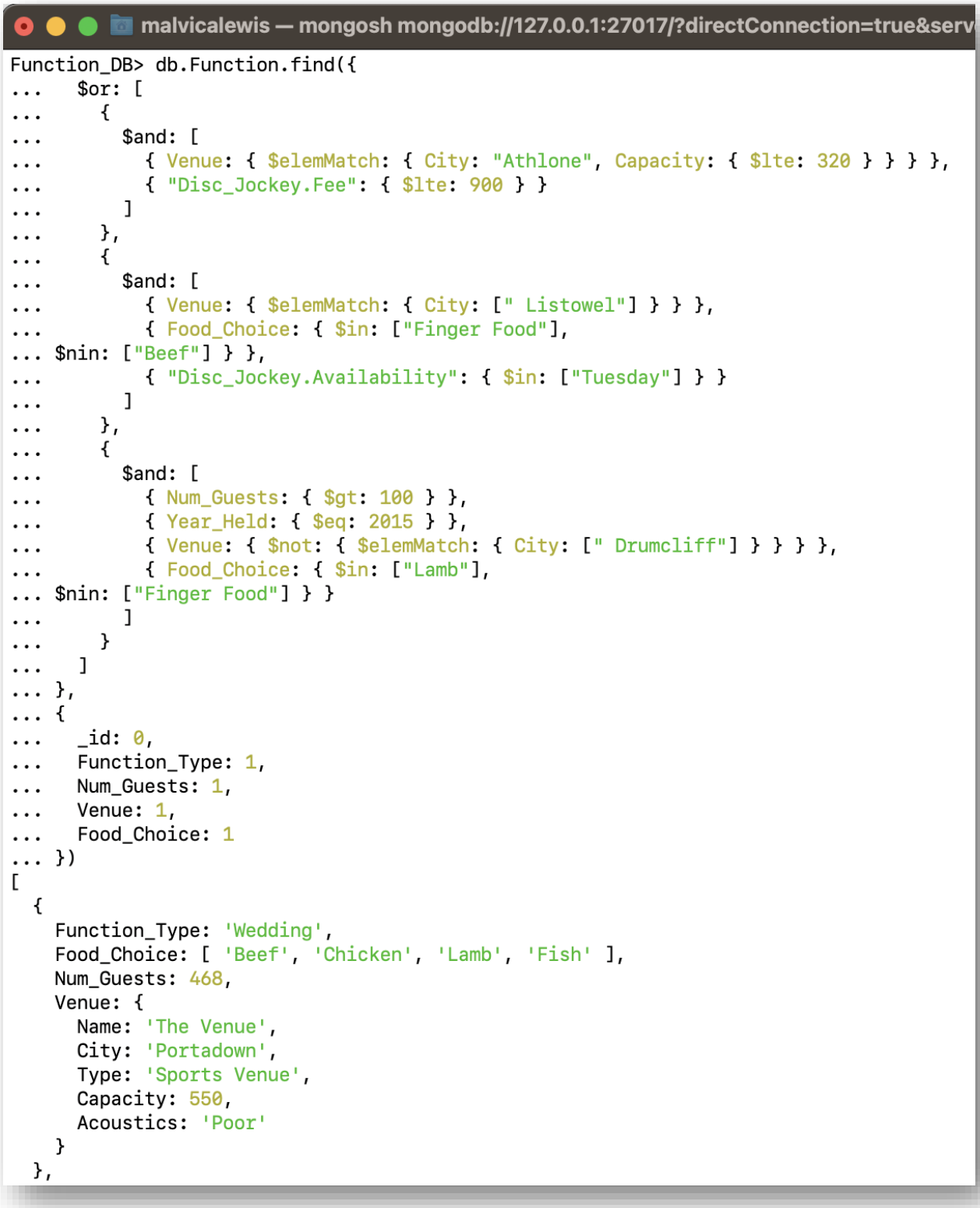
```
db.Function.find({
 $or: [
  {
   $and: [
    { Venue: { $elemMatch: { City: "Athlone", Capacity: { $lte: 320 } } } },
    { "Disc_Jockey.Fee": { $lte: 900 } }
   ]
  },
  {
   $and: [
    { Venue: { $elemMatch: { City: [" Listowel"] } } },
    { Food_Choice: { $in: ["Finger Food"],
                     $nin: ["Beef"] } },
    { "Disc_Jockey.Availability": { $in: ["Tuesday"] } }
   ]
  },
  {
   $and: [
    { Num_Guests: { $gt: 100 } },
    { Year_Held: { $eq: 2015 } },
    { Venue: { $not: { $elemMatch: { City: [" Drumcliff"] } } } },
    { Food_Choice: { $in: ["Lamb"],
                     $nin: ["Finger Food"] } }
   ]
  }
 ]
},
```

```
{
  _id: 0,
  Function_Type: 1,
  Num_Guests: 1,
  Venue: 1,
  Food_Choice: 1
})
```

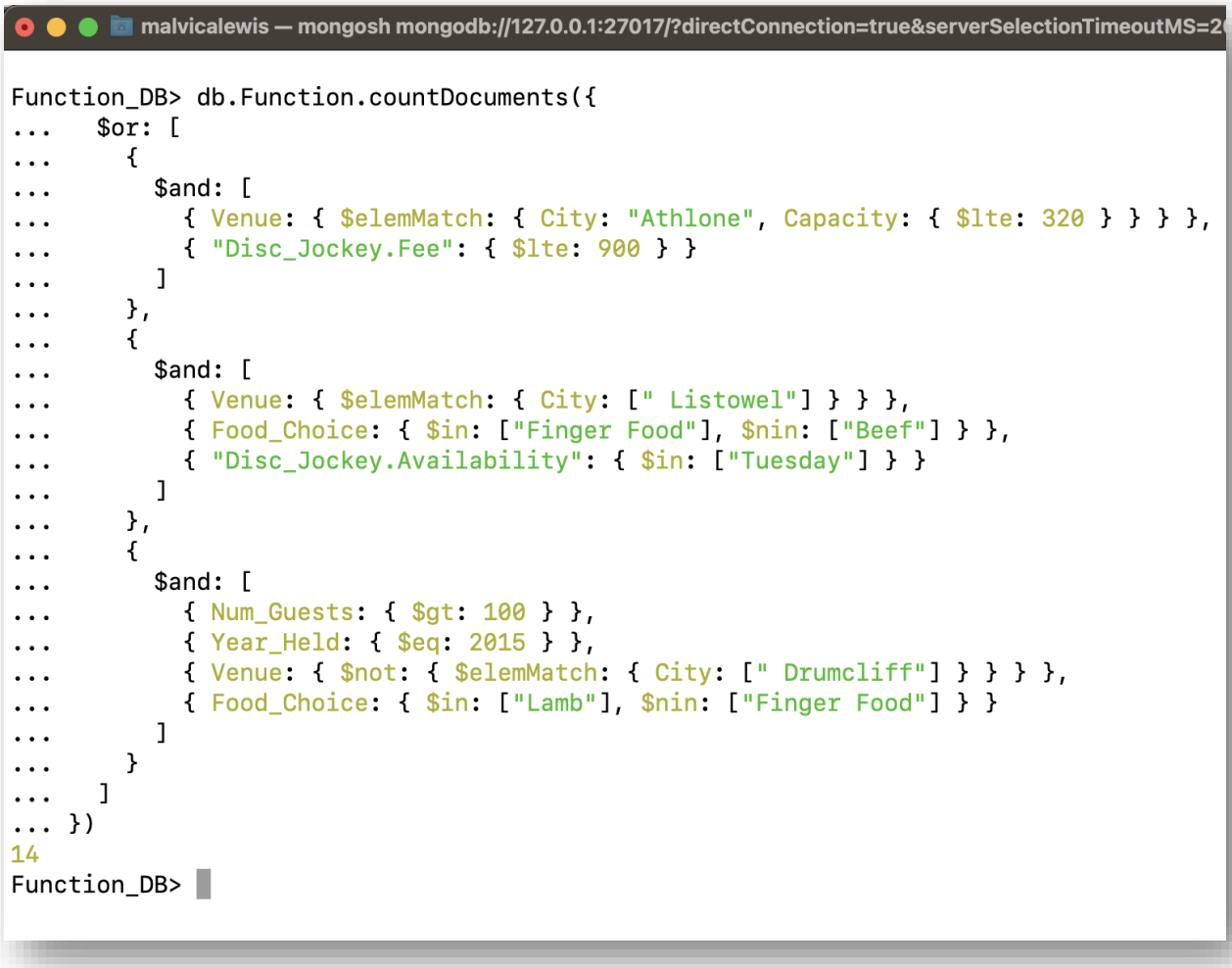
The query uses the following functions:

- The query consists of three **$or** clauses, each of which contains multiple **$and** clauses with different conditions:
    - The first **$or** clause retrieves documents where the **Venue** array contains an element with a **City** value of "**Athlone**" and a **Capacity** value less than or equal to **320**, and where the **Disc_Jockey.Fee** value is less than or equal to **900**.
    - The second **$or** clause retrieves documents where the **Venue** array contains an element with a **City** value of "**Listowel**", and where the **Food_Choice** value is "**Finger Food**" but not "**Beef**", and where the **Disc_Jockey.Availability** value is "**Tuesday**".
    - The third **$or** clause retrieves documents where the **Num_Guests** value is greater than **100**, the **Year_Held** value is **2015**, the **Venue** array does not contain an element with a **City** value of "**Drumcliff**", and the **Food_Choice** value is "**Lamb**" but not "**Finger Food**".

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS=2
Function_DB> db.Function.countDocuments({
... $or: [
... {
... $and: [
... { Venue: { $elemMatch: { City: "Athlone", Capacity: { $lte: 320 } } } },
... { "Disc_Jockey.Fee": { $lte: 900 } }
...]
... },
... {
... $and: [
... { Venue: { $elemMatch: { City: [" Listowel"] } } },
... { Food_Choice: { $in: ["Finger Food"], $nin: ["Beef"] } },
... { "Disc_Jockey.Availability": { $in: ["Tuesday"] } }
...]
... },
... {
... $and: [
... { Num_Guests: { $gt: 100 } },
... { Year_Held: { $eq: 2015 } },
... { Venue: { $not: { $elemMatch: { City: [" Drumcliff"] } } } },
... { Food_Choice: { $in: ["Lamb"], $nin: ["Finger Food"] } }
...]
... }
...]
... })
14
Function_DB>

25. $exists, $regex, $and, $eq, $ne, $gt, $nin, $expr

Question: What are the functions and food choices for events:

- held in 2019 with over 300 guests,
- excluding retirement functions,
- where the food choices
  - contain the string "Fish" anywhere within it,
  - the Food_Choice field exists,
  - does not equal an empty string or null, and
  - has at least 3 elements, and
- the city name of the venue is longer than 5 characters.

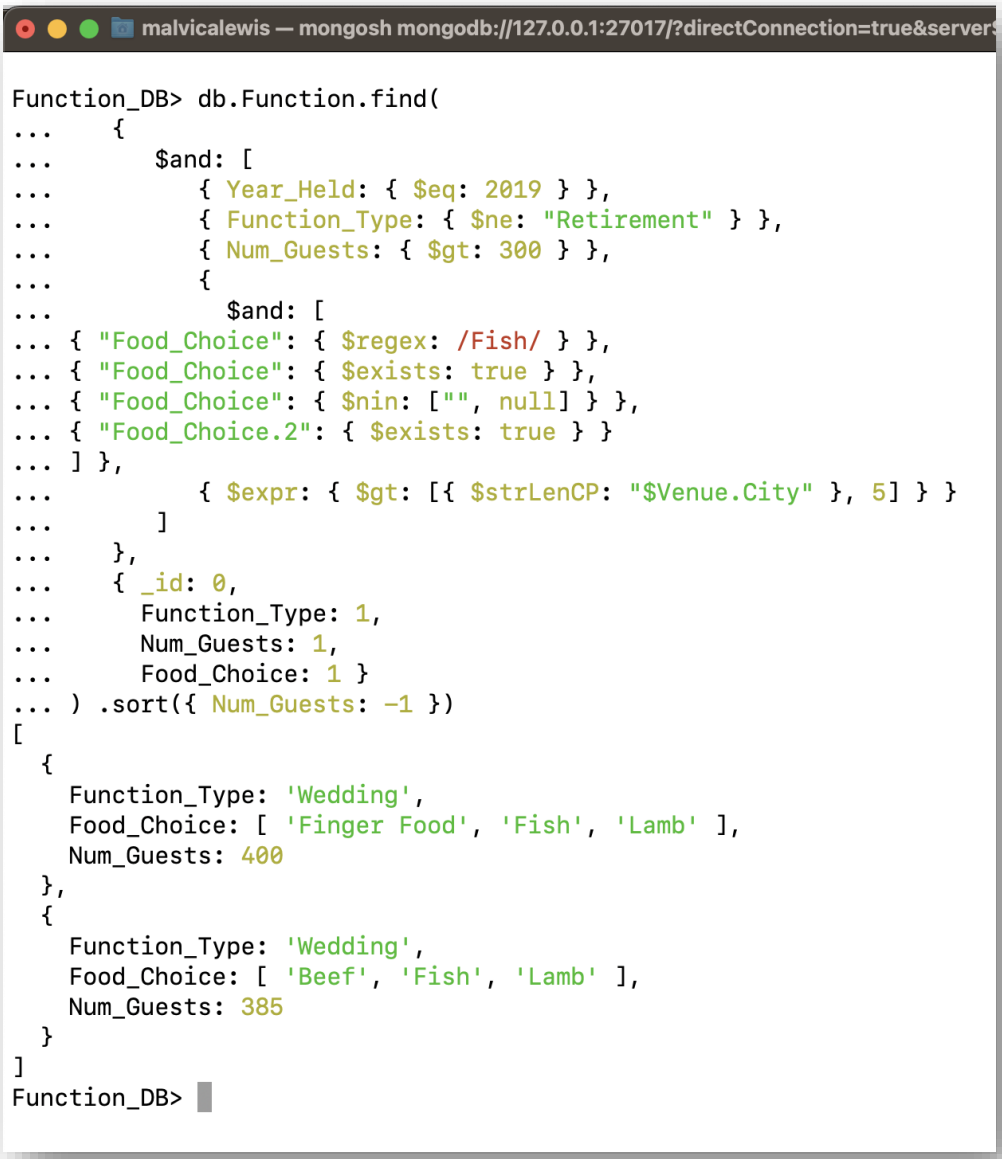
Query:

```
db.Function.find(
  {
    $and: [
      { Year_Held: { $eq: 2019 } },
      { Function_Type: { $ne: "Retirement" } },
      { Num_Guests: { $gt: 300 } },
      {
        $and: [
{ "Food_Choice": { $regex: /Fish/ } },
{ "Food_Choice": { $exists: true } },
{ "Food_Choice": { $nin: ["", null] } },
{ "Food_Choice.2": { $exists: true } }
] },
        { $expr: { $gt: [{ $strLenCP: "$Venue.City" }, 5] } }
    ]
  },
  { _id: 0,
    Function_Type: 1,
    Num_Guests: 1,
    Food_Choice: 1 }
) .sort({ Num_Guests: -1 })
```

The query uses the following functions:

- **db.Function.find()**: This is a MongoDB function that searches for documents in the Function collection that match the specified criteria.
- **$and**: This is a logical operator that combines multiple conditions, all of which must be true for a document to be returned.
- **{ Year_Held: { $eq: 2019 } }**: This condition matches documents where the **Year_Held** field is equal to **2019**.
- **{ Function_Type: { $ne: "Retirement" } }**: This condition matches documents where the **Function_Type** field is not equal to "**Retirement**".
- **{ Num_Guests: { $gt: 300 } }**: This condition matches documents where the **Num_Guests** field is greater than **300**.
- **$and: [{ "Food_Choice": { $regex: /Fish/ } }, { "Food_Choice": { $exists: true } }, { "Food_Choice": { $nin: ["", null] } }, { "Food_Choice.2": { $exists: true } }]**: This condition matches documents where the **Food_Choice** field contains the string "**Fish**" anywhere within it, and where **Food_Choice** exists and is not an empty string or null, and where the **Food_Choice** array has at least **3** elements (i.e. the third element exists).
- **{ $expr: { $gt: [{ $strLenCP: "$Venue.City" }, 5] } }**: This condition matches documents where the length of the **City** field within the **Venue** array is greater than **5**. **$strLenCP** is a string function that returns the length of a string in code points.
- **sort({ Num_Guests: -1 })**: This sorts the results in descending order based on the **Num_Guests** field. The **-1** argument specifies descending order.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelect
Function_DB> db.Function.countDocuments(
... {
... $and: [
... { Year_Held: { $eq: 2019 } },
... { Function_Type: { $ne: "Retirement" } },
... { Num_Guests: { $gt: 300 } },
... {
... $and: [
... { "Food_Choice": { $regex: /Fish/ } },
... { "Food_Choice": { $exists: true } },
... { "Food_Choice": { $nin: ["", null] } },
... { "Food_Choice.2": { $exists: true } }
...] },
... { $expr: { $gt: [{ $strLenCP: "$Venue.City" }, 5] } }
...]
... },
... { _id: 0,
... Function_Type: 1,
... Num_Guests: 1,
... Food_Choice: 1 }
...)
2
Function_DB>

26. insertOne(), deleteOne()

Inserting a new record with the id number as 1001.

Query:

```
// Insert the data
db.function.insertOne({
 "_id":1001,
 "Year_Held": 2013,
 "Function_Type": "Wedding",
 "Food_Choice": ["Beef", "Chicken", "Lamb", "Fish"],
 "Num_Guests": 358,
 "Alcohol": ["Guinness", "Heineken"],
 "Soft_Drinks": ["Water", "Fanta"],
 "Duration": 5,
 "Disc_Jockey": {
  "Name": "Emmerich",
  "Fee": 600,
  "Availability": ["Monday", "Tuesday", "Wednesday"]
 },
 "Venue": {
  "Name": "Tribes",
  "City": "Mountrath",
  "Type": "Multi-Purpose Hall",
  "Capacity": 550,
  "Acoustics": "Good"
 }
})

// Delete the inserted document with _id: 1001
db.function.deleteOne({ "_id": 1001 });
```

Query Output

27. insertMany(), deleteMany()

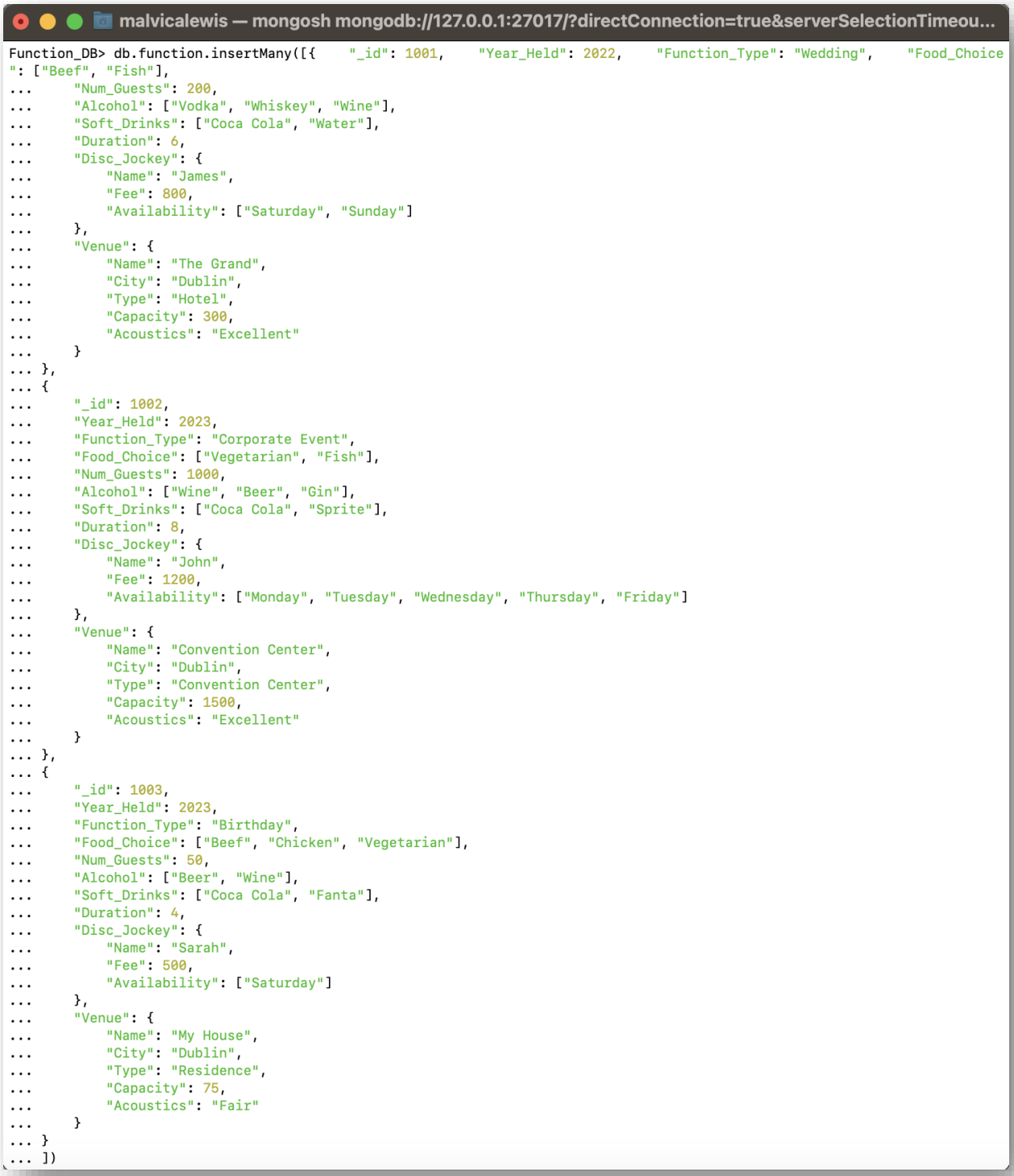
Inserting a new record with the id number as 1001.

Query:

```
// Insert the data
db.function.insertMany([{   "_id": 1001,   "Year_Held": 2022,
"Function_Type": "Wedding",   "Food_Choice": ["Beef", "Fish"],
  "Num_Guests": 200,
  "Alcohol": ["Vodka", "Whiskey", "Wine"],
  "Soft_Drinks": ["Coca Cola", "Water"],
  "Duration": 6,
  "Disc_Jockey": {
    "Name": "James",
    "Fee": 800,
    "Availability": ["Saturday", "Sunday"]
  },
  "Venue": {
    "Name": "The Grand",
    "City": "Dublin",
    "Type": "Hotel",
    "Capacity": 300,
    "Acoustics": "Excellent"
  }
},
{
  "_id": 1002,
  "Year_Held": 2023,
  "Function_Type": "Corporate Event",
  "Food_Choice": ["Vegetarian", "Fish"],
  "Num_Guests": 1000,
  "Alcohol": ["Wine", "Beer", "Gin"],
  "Soft_Drinks": ["Coca Cola", "Sprite"],
  "Duration": 8,
  "Disc_Jockey": {
    "Name": "John",
    "Fee": 1200,
    "Availability": ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday"]
```

```
    },
    "Venue": {
      "Name": "Convention Center",
      "City": "Dublin",
      "Type": "Convention Center",
      "Capacity": 1500,
      "Acoustics": "Excellent"
    }
  },
  {
    "_id": 1003,
    "Year_Held": 2023,
    "Function_Type": "Birthday",
    "Food_Choice": ["Beef", "Chicken", "Vegetarian"],
    "Num_Guests": 50,
    "Alcohol": ["Beer", "Wine"],
    "Soft_Drinks": ["Coca Cola", "Fanta"],
    "Duration": 4,
    "Disc_Jockey": {
      "Name": "Sarah",
      "Fee": 500,
      "Availability": ["Saturday"]
    },
    "Venue": {
      "Name": "My House",
      "City": "Dublin",
      "Type": "Residence",
      "Capacity": 75,
      "Acoustics": "Fair"
    }
  }
])

// Delete the inserted documents with id numbers – 1001,1002, 1003
db.function.deleteMany({ _id: { $in: [1001, 1002, 1003] } })
```

Query Output




malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnecti
... "Availability": ["Saturday"]
... },
... "Venue": {
... "Name": "My House",
... "City": "Dublin",
... "Type": "Residence",
... "Capacity": 75,
... "Acoustics": "Fair"
... }
... }
...])
{
acknowledged: true,
insertedIds: { '0': 1001, '1': 1002, '2': 1003 }
}
Function_DB>
Function_DB> // Delete the inserted documents with id numbers – 1001,1002, 1003
Function_DB> db.function.deleteMany({ _id: { $in: [1001, 1002, 1003] } })
{ acknowledged: true, deletedCount: 3 }
Function_DB>

Using Advanced Aggregate Function in MongoDB: $match, $group, $unwind, $lookup, $sum, $min, $max, $avg, $addToSet, $cond, $project, $limit, $skip, $slice, $round

In MongoDB, aggregate functions are used to perform a series of data processing operations on collections of documents. These operations allow you to transform and analyze data, and return computed results based on specified criteria.

Aggregate functions in MongoDB are performed using the **aggregate()** method, which takes an array of one or more stages as its argument. Each stage in the pipeline represents a single operation that is performed on the documents in the collection. There are many stages available in MongoDB, each with its own set of operators and options that can be used to customize the behavior of the stage.

Some common aggregate stages in MongoDB include:

- **$match**: Filters the documents in the collection based on specified criteria.
- **$group**: Groups the documents in the collection based on specified fields and performs aggregation operations on each group.
- **$project**: Reshapes the documents in the collection by including, excluding, or renaming fields.
- **$sort**: Sorts the documents in the collection based on specified fields.
- **$limit**: Limits the number of documents returned by the pipeline.
- **$skip**: Skips a specified number of documents in the collection before returning the remaining documents.

Aggregate functions in MongoDB are very powerful and can be used to perform complex data transformations and analytics. They are particularly useful for working with large collections of data, where traditional query methods may be too slow or inefficient.

1. Find the maximum number of guests for each year

```
db.Function.aggregate([
 {
  $group: {
   _id: "$Year_Held",
   max_guests: { $max: "$Num_Guests" }
  }
 },
 {
$sort: { max_guests: -1 }
}
])
```

The query uses the following functions:

- **db.Function.aggregate([])** is a function that performs aggregation operations on the documents in the "**Function**" collection of the current database.
- **{ $group: { _id: "$Year_Held", max_guests: { $max: "$Num_Guests" } } }** is an aggregation pipeline stage that groups the documents by the "**Year_Held**" field and calculates the maximum value of "**Num_Guests**" for each group. The **$group** function is used to group the documents, and the **$max** function is used to calculate the maximum value.
- **.sort({ max_guests: -1 })** is a function that sorts the output of the previous stage in descending order based on the value of the "**max_guests**" field. The **sort()** function is used with the **-1** option to sort in descending order.
- So, in summary, this MongoDB query groups the documents in the "**Function**" collection by year and calculates the maximum number of guests for each year. The output is then sorted in descending order based on the maximum number of guests.

Query Output

Query Count

2. Find the total number of guests and the average duration of each function type held in the city of Mountrath:

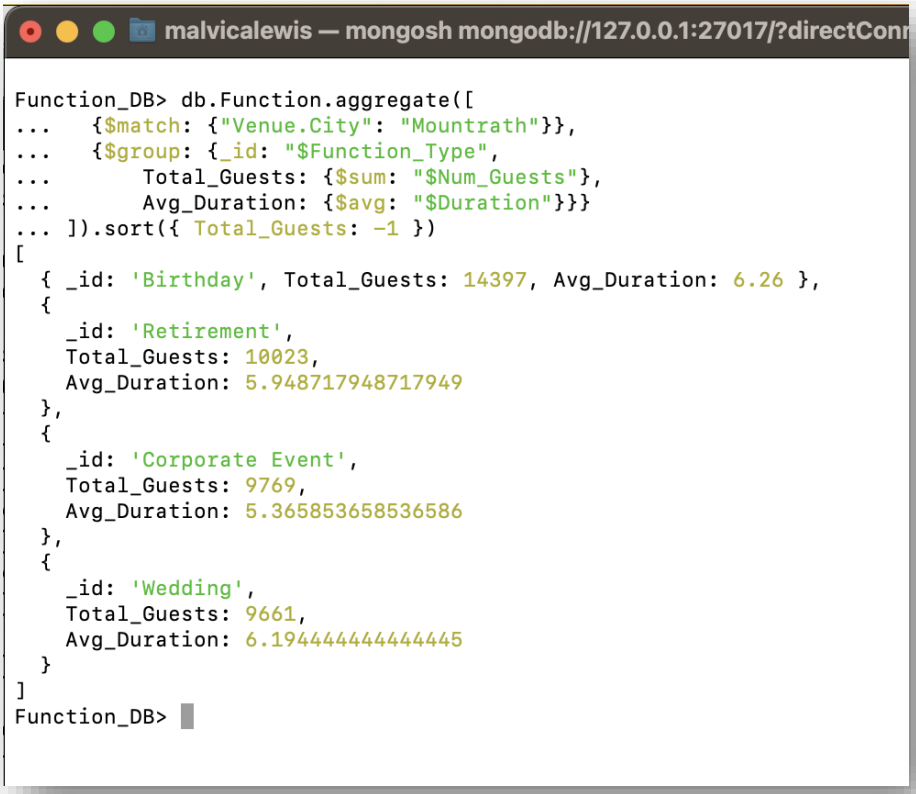
```
db.Function.aggregate([
  {
    $match: { "Venue.City": "Mountrath" }
  },
  {
    $group: {
      _id: "$Function_Type",
      Total_Guests: { $sum: "$Num_Guests" },
      Avg_Duration: { $avg: "$Duration" }
    }
  },
  {
    $sort: { Total_Guests: -1 }
  }
])
```

The query uses the following functions:

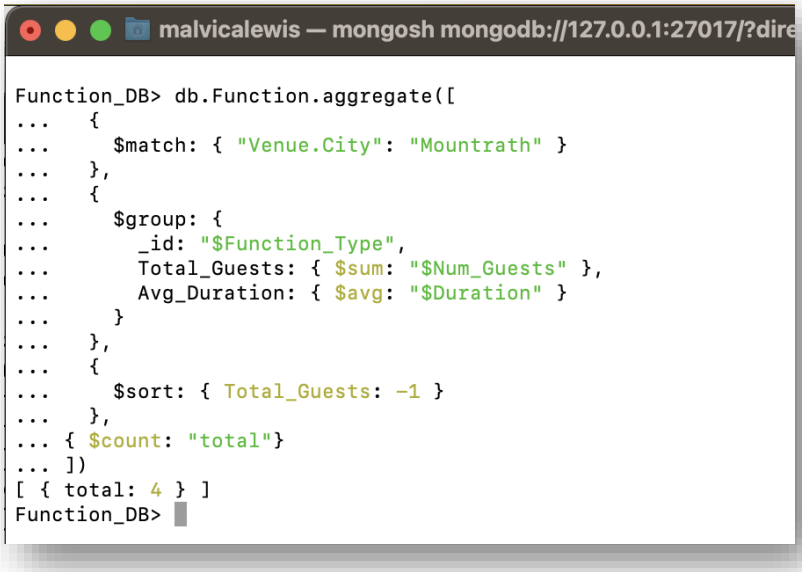
- **$match**: This function is used to filter the documents in the "**Function**" collection based on a given condition. In this case, it filters documents where the "**Venue.City**" field is equal to "**Mountrath**".
- **$group**: This function is used to group the filtered documents by a given field or set of fields. In this case, it groups documents by the "**Function_Type**" field and calculates the total number of guests and the average duration for each group.
- **$sort**: This function is used to sort the grouped results in ascending or descending order based on a given field. In this case, it sorts the grouped results in descending order based on the "**Total_Guests**" field.

So, the final output of this aggregation pipeline will be a list of documents, where each document represents a unique "**Function_Type**" group for events held in Mountrath. The documents will include the total number of guests and the average duration for each group, sorted in descending order based on the total number of guests.

Query Output



Query Count



3. Find the average fee for each disc jockey who performed at Athlone and have a fee less than 500 euros, sorted in ascending order.

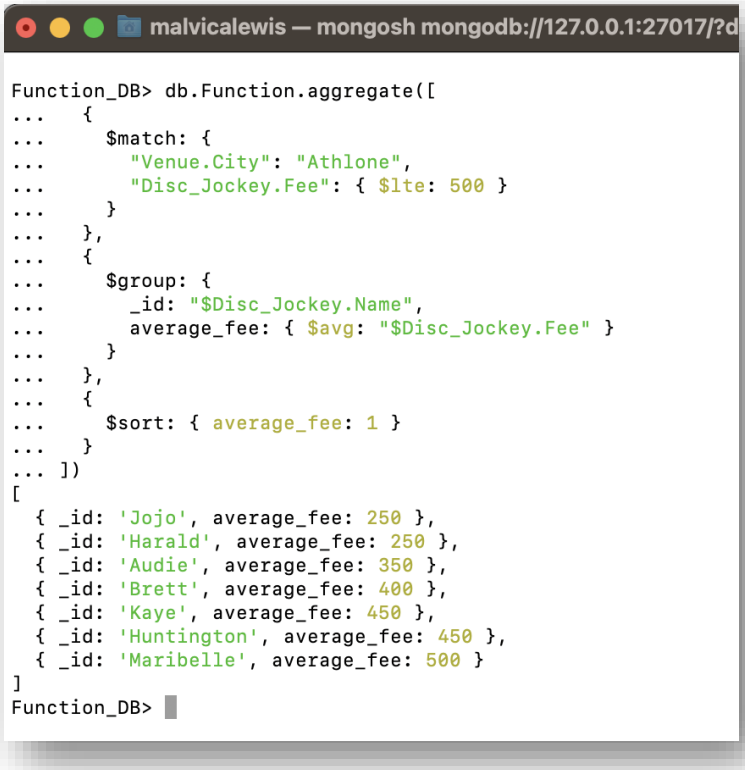
```
db.Function.aggregate([
 {
  $match: {
   "Venue.City": "Athlone",
   "Disc_Jockey.Fee": { $lte: 500 }
  }
 },
 {
  $group: {
   _id: "$Disc_Jockey.Name",
   average_fee: { $avg: "$Disc_Jockey.Fee" }
  }
 },
 {
  $sort: { average_fee: 1 }
 }
])
```

The query uses the following functions:

- **$match**: This function filters the input documents by selecting only those that match the specified criteria. In this case, it selects documents where the **Venue.City** is "**Athlone**" and the **Disc_Jockey.Fee** is less than or equal to **500**.
- **$group**: This function groups the selected documents by the **Disc_Jockey.Name** field and calculates the average fee for each group using the **$avg** aggregation function.
- **$sort**: This function sorts the output documents in ascending order based on the **average_fee** field.

Overall, the query selects only documents where the **Venue.City** is "**Athlone**" and the **Disc_Jockey.Fee** is less than or equal to **500**. It then groups the selected documents by the **Disc_Jockey.Name** field and calculates the average fee for each group. Finally, it sorts the output documents in ascending order based on the **average_fee** field**.**

Query Output



Query Count

4. Find the number of functions held in each venue type, sorted in descending order.

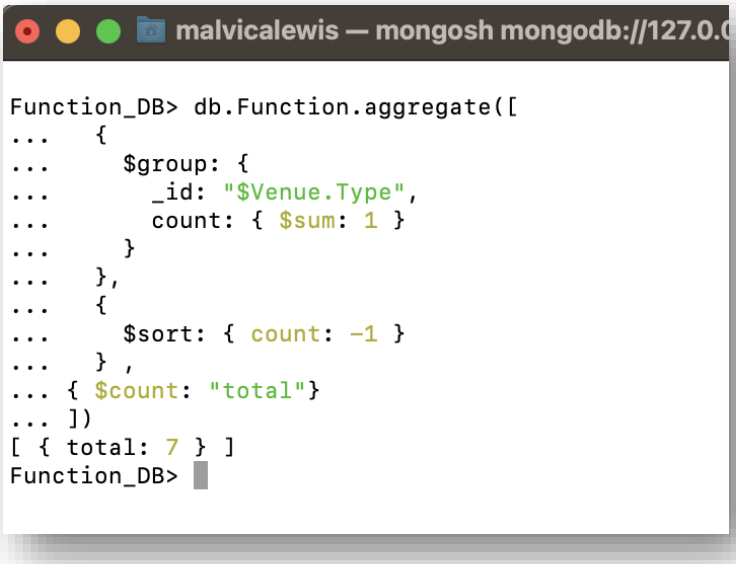
```
db.Function.aggregate([
  {
    $group: {
      _id: "$Venue.Type",
      count: { $sum: 1 }
    }
  },
  {
    $sort: { count: -1 }
  }
])
```

The query uses the following functions:

- **$group** stage: This stage groups documents in the collection by the **Venue.Type** field and calculates the count of documents in each group using the **$sum** aggregation operator. The output of this stage will be documents that have **_id** field as the unique value of the **Venue.Type** field and a count field with the total number of documents with that **Venue.Type**.
- **$sort** stage: This stage sorts the output documents from the previous stage in descending order based on the count field value. This means that the **Venue.Type** with the greatest number of documents in the collection will be displayed first.

Query Output

Query Count



5. Find the total count of events held in each city where the venue was Park, and only include cities where more than 1 event has been held.

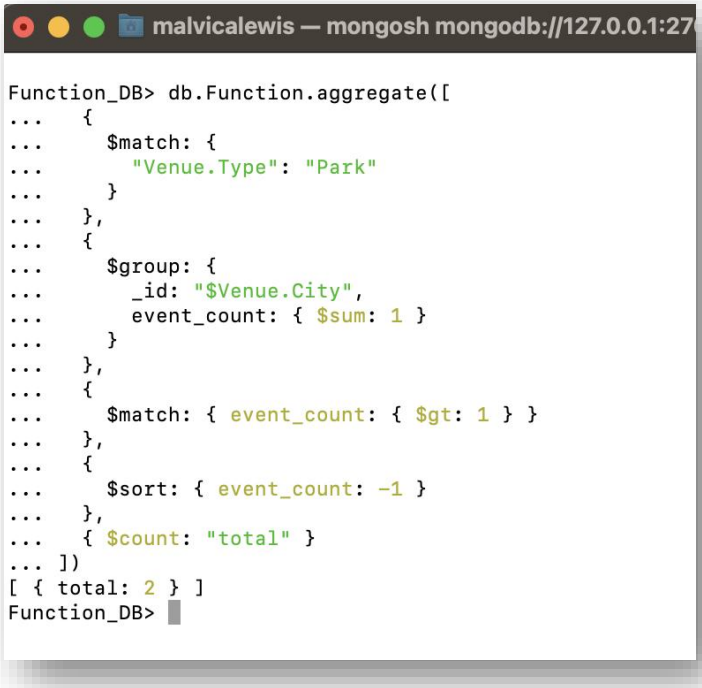
```
db.Function.aggregate([
 {
  $match: {
   "Venue.Type": "Park"
  }
 },
 {
  $group: {
   _id: "$Venue.City",
   event_count: { $sum: 1 }
  }
 },
 {
  $match: { event_count: { $gt: 1 } }
 },
 {
  $sort: { event_count: -1 }
 }
])
```

The query uses the following functions:

- **$group**: This is the first aggregation pipeline stage that groups the documents in the collection by the **Venue.City** field and calculates the total count of documents in each group. The **_id** field is set to **Venue.City** so that each group is identified by the city name.
- **$match**: This is the second aggregation pipeline stage that filters out the groups that have **event_count** less than or equal to **1**. This is done using the **$match** operator with a filter condition that checks if the **event_count** is greater than **1**.
- **$sort**: This is the third aggregation pipeline stage that sorts the remaining groups by the **event_count** field in descending order. The **-1** value in the **$sort** operator represents descending order.

Query Output

Query Count



6. Find the average fee for a disc jockey, grouped by the day of the week they are available on.

```
db.Function.aggregate([
 {
  $unwind: "$Disc_Jockey.Availability"
 },
 {
  $group: {
   _id: "$Disc_Jockey.Availability",
   average_fee: { $avg: "$Disc_Jockey.Fee" }
  }
 },
 {
  $sort: { average_fee: -1 }
 }
])
```

The query uses the following functions:

- **$unwind**: This stage is used to break down the array field **Disc_Jockey.Availability** into separate documents. Each document in the pipeline is duplicated for each element in the array. This makes it possible to group by the availability of each DJ later in the pipeline.
- **$group**: This stage groups the documents based on the **_id** field, which is set to **Disc_Jockey.Availability**. Then, it calculates the average fee for each group using the **$avg** operator.
- **$sort**: This stage sorts the output based on the **average_fee** field in descending order (**-1**).

Overall, this query calculates the average fee for each availability slot of the disc jockeys and sorts them in descending order of their average fee.

Query Output

Query Count

7. Find venues with good acoustics and a capacity greater than or equal to 400, sorted in ascending order of capacity.

```
db.Function.aggregate([
 {
  $match: {
   "Venue.Acoustics": "Good",
   "Venue.Capacity": { $gte: 400 }
  }
 },
 {
  $project: {
   "venue_name": "$Venue.Name",
   "venue_capacity": "$Venue.Capacity",
   _id: 0
  }
 },
 {
  $group: {
   _id: { "venue_name": "$venue_name" },
   "venue_capacity": { $first: "$venue_capacity" }
  }
 },
 {
  $sort: { "venue_capacity": 1 }
 }
])
```

The query uses the following functions:

- **$match**: This function filters the documents based on two conditions: "**Venue.Acoustics**" must equal "**Good**" and "**Venue.Capacity**" must be greater than or equal to **400**.
- **$project**: This function creates a new document with two fields: "**venue_name**" and "**venue_capacity**". The "**venue_name**" field is set to the value of the "**Venue.Name**" field, and the "**venue_capacity**" field is set to the value of the "**Venue.Capacity**" field. The "**_id**" field is excluded from the document.
- **$group**: This function groups the documents based on the "**venue_name**" field and calculates the maximum value of "**venue_capacity**" for each group. The "**_id**" field is set to an object containing the "**venue_name**" field.

- **$sort**: This function sorts the documents in descending order based on the "**venue_capacity**" field.

In summary, the query filters the "**Function**" collection to only include documents where the "**Venue.Acoustics**" field is "**Good**" and the "**Venue.Capacity**" field is greater than or equal to **400**. Then, it creates a new document with the "**Venue.Name**" and "**Venue.Capacity**" fields and groups the documents by "**Venue.Name**", calculating the maximum "**Venue.Capacity**" for each group. Finally, the documents are sorted in ascending order based on the "**Venue.Capacity**" field.

Query Output

Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?c
Function_DB> db.Function.aggregate([
... {
... $match: {
... "Venue.Acoustics": "Good",
... "Venue.Capacity": { $gte: 400 }
... }
... },
... {
... $project: {
... "venue_name": "$Venue.Name",
... "venue_capacity": "$Venue.Capacity",
... _id: 0
... }
... },
... {
... $group: {
... _id: { "venue_name": "$venue_name" },
... "venue_capacity": { $first: "$venue_capacity" }
... }
... },
... {
... $sort: { "venue_capacity": 1 }
... },
... { $count: "total"}
...])
[{ total: 6 }]
Function_DB>

8. What are the top three most popular types of events held each year, based on the average number of guests and duration, as well as the unique food and alcohol choices served at each event?

```
db.Function.aggregate([
 {
  $group: {
   _id: { year: "$Year_Held", functionType: "$Function_Type" },
   avgNumGuests: { $avg: "$Num_Guests" },
   avgDuration: { $avg: "$Duration" },
   foodChoices: { $addToSet: "$Food_Choice" },
   alcohol: { $addToSet: "$Alcohol" }
  }
 },
 {
  $sort: { "_id.year": -1 }
 },
 {
  $limit: 3
 },
 {
  $project: {
   year: "$_id.year",
   functionType: "$_id.functionType",
   avgNumGuests: 1,
   avgDuration: 1,
   foodChoices: 1,
   alcohol: 1,
   _id: 0
  }
 }
])
```

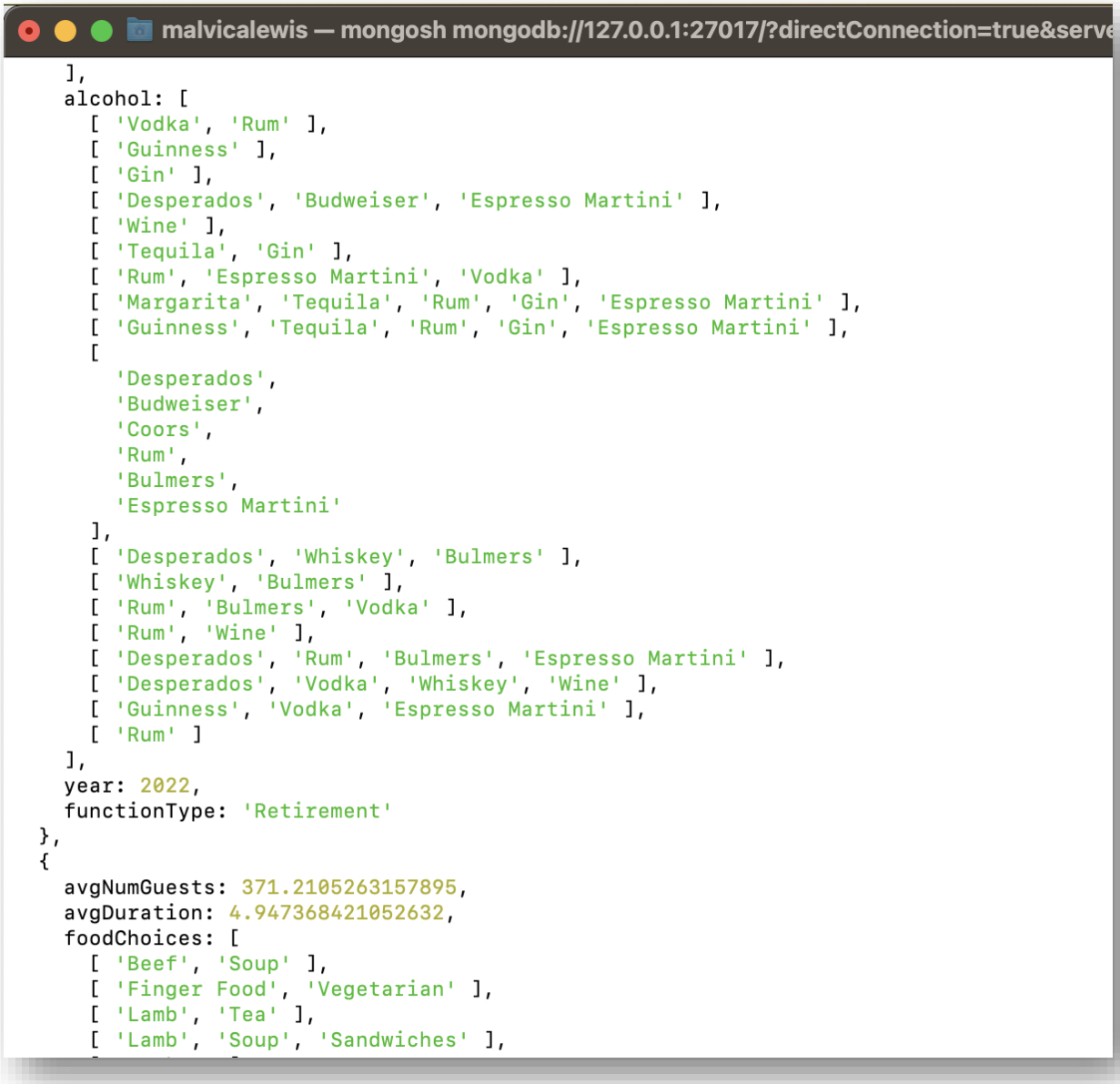
The query uses the following functions:

- **$group**: groups documents by year and function type and calculates the average number of guests and duration of events for each group. It also adds the distinct values of "**Food_Choice**" and "**Alcohol**" fields to an array for each group.
- **$sort**: sorts the results by descending order of the year field.
- **$limit**: limits the number of returned documents to **3**.

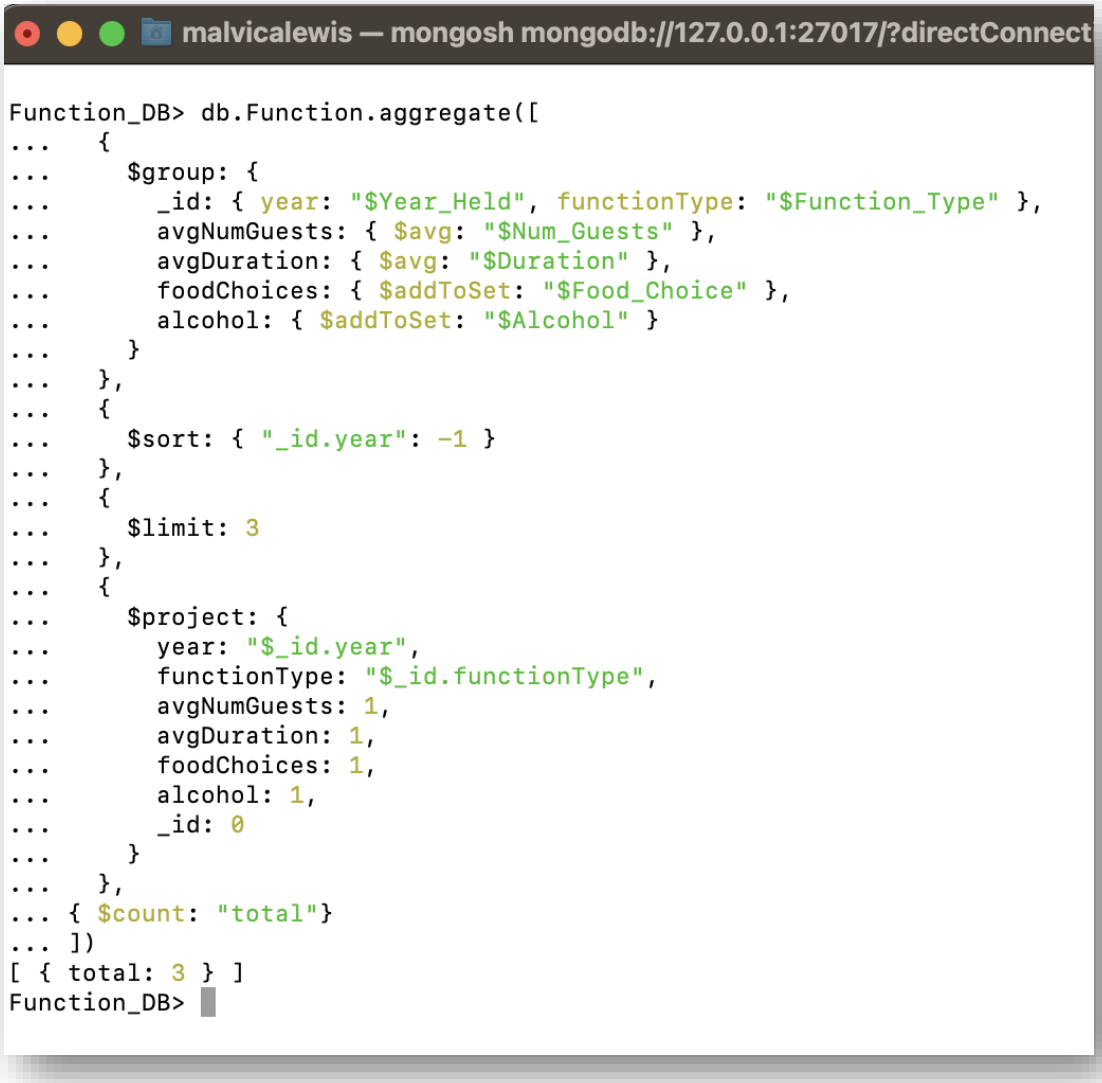
- **$project**: reshapes the output document to include only the specified fields and exclude the "**_id**" field.

The output of this query will include three documents, each representing a unique year and function type combination, with the average number of guests, average duration, food choices, and alcohol options. The documents will be sorted in descending order based on the year field.

Query Output


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serve
],
alcohol: [
['Vodka', 'Rum'],
['Guinness'],
['Gin'],
['Desperados', 'Budweiser', 'Espresso Martini'],
['Wine'],
['Tequila', 'Gin'],
['Rum', 'Espresso Martini', 'Vodka'],
['Margarita', 'Tequila', 'Rum', 'Gin', 'Espresso Martini'],
['Guinness', 'Tequila', 'Rum', 'Gin', 'Espresso Martini'],
[
'Desperados',
'Budweiser',
'Coors',
'Rum',
'Bulmers',
'Espresso Martini'
],
['Desperados', 'Whiskey', 'Bulmers'],
['Whiskey', 'Bulmers'],
['Rum', 'Bulmers', 'Vodka'],
['Rum', 'Wine'],
['Desperados', 'Rum', 'Bulmers', 'Espresso Martini'],
['Desperados', 'Vodka', 'Whiskey', 'Wine'],
['Guinness', 'Vodka', 'Espresso Martini'],
['Rum']
],
year: 2022,
functionType: 'Retirement'
},
{
avgNumGuests: 371.2105263157895,
avgDuration: 4.947368421052632,
foodChoices: [
['Beef', 'Soup'],
['Finger Food', 'Vegetarian'],
['Lamb', 'Tea'],
['Lamb', 'Soup', 'Sandwiches'],

Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnect
Function_DB> db.Function.aggregate([
... {
... $group: {
... _id: { year: "$Year_Held", functionType: "$Function_Type" },
... avgNumGuests: { $avg: "$Num_Guests" },
... avgDuration: { $avg: "$Duration" },
... foodChoices: { $addToSet: "$Food_Choice" },
... alcohol: { $addToSet: "$Alcohol" }
... }
... },
... {
... $sort: { "_id.year": -1 }
... },
... {
... $limit: 3
... },
... {
... $project: {
... year: "$_id.year",
... functionType: "$_id.functionType",
... avgNumGuests: 1,
... avgDuration: 1,
... foodChoices: 1,
... alcohol: 1,
... _id: 0
... }
... },
... { $count: "total"}
...])
[{ total: 3 }]
Function_DB>

9. What is the breakdown of event statistics for each city in the database including the total number of events, the average, and the maximum number of guests? Additionally, what is the average fee for disc jockeys at events in each city and what are the top three food choices for events in each city?

```
db.Function.aggregate([
 {
  $group: {
   _id: "$Venue.City",
   numEvents: { $sum: 1 },
   avgGuests: { $avg: "$Num_Guests" },
   maxGuests: { $max: "$Num_Guests" }
  }
 },
 {
  $lookup: {
   from: "Function",
   localField: "_id",
   foreignField: "Venue.City",
   as: "events"
  }
 },
 {
  $project: {
   _id: 1,
   numEvents: 1,
   avgGuests: { $round: ["$avgGuests", 2] },
   maxGuests: 1,
   discJockeyFee: { $avg: "$events.Disc_Jockey.Fee" },
   foodChoices: { $slice: ["$events.Food_Choice", 3] }
  }
 },
 {
  $limit: 2
 }
])
```

The query uses the following functions:

- **$group**: This function groups the documents in the collection by the **Venue.City** field and calculates the total number of events in each city using the **$sum** operator. It also calculates the average number of guests per event using the **$avg** operator, and the maximum number of guests using the **$max** operator.
- **$lookup**: This function performs a left outer join with the same collection (Function). It looks up all the documents with the same **Venue.City** as the **_id** of the previous **$group** operation, and adds a new field to the output called events that contains an array of matching documents.
- **$project**: This function shapes the output of the aggregation. It includes the **_id** field from the previous **$group** operation, and includes **numEvents**, **avgGuests**, and **maxGuests** fields from the previous **$group** operation. It calculates the average fee of the disc jockey at each event by averaging the **Disc_Jockey.Fee** field from the events array using the **$avg** operator. It also includes the first three elements of the **Food_Choice** array from the events array using the **$slice** operator and rounds the **avgGuests** field to two decimal places using the **$round** operator.

Query Output

Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnect
Function_DB> db.Function.aggregate([
... {
... $group: {
... _id: "$Venue.City",
... numEvents: { $sum: 1 },
... avgGuests: { $avg: "$Num_Guests" },
... maxGuests: { $max: "$Num_Guests" }
... }
... },
... {
... $lookup: {
... from: "Function",
... localField: "_id",
... foreignField: "Venue.City",
... as: "events"
... }
... },
... {
... $project: {
... _id: 1,
... numEvents: 1,
... avgGuests: { $round: ["$avgGuests", 2] },
... maxGuests: 1,
... discJockeyFee: { $avg: "$events.Disc_Jockey.Fee" },
... foodChoices: { $slice: ["$events.Food_Choice", 3] }
... }
... },
... {
... $limit: 2
... } ,
... { $count: "total"}
...])
[{ total: 2 }]
Function_DB>

10. What are the top 10 food choices for weddings with at least 300 guests, and how many times was each food choice served and what is the average number of guests for each choice?

```
db.Function.aggregate([

 // Match documents where Function_Type is "Wedding" and Num_Guests is
greater than or equal to 300
 {
  $match: {
   Function_Type: "Wedding",
   Num_Guests: { $gte: 300 }
  }
 },

 // Unwind the Food_Choice array to get individual food choices
 {
  $unwind: "$Food_Choice"
 },

 // Group by food choice and calculate the count and average number of
guests
 {
  $group: {
   _id: "$Food_Choice",
   count: { $sum: 1 },
   avgGuests: { $avg: "$Num_Guests" }
  }
 },

 // Sort by descending count
 {
  $sort: {count: -1}
 },

 // Limit to the top 10 results
 {
  $limit: 10
 },
```

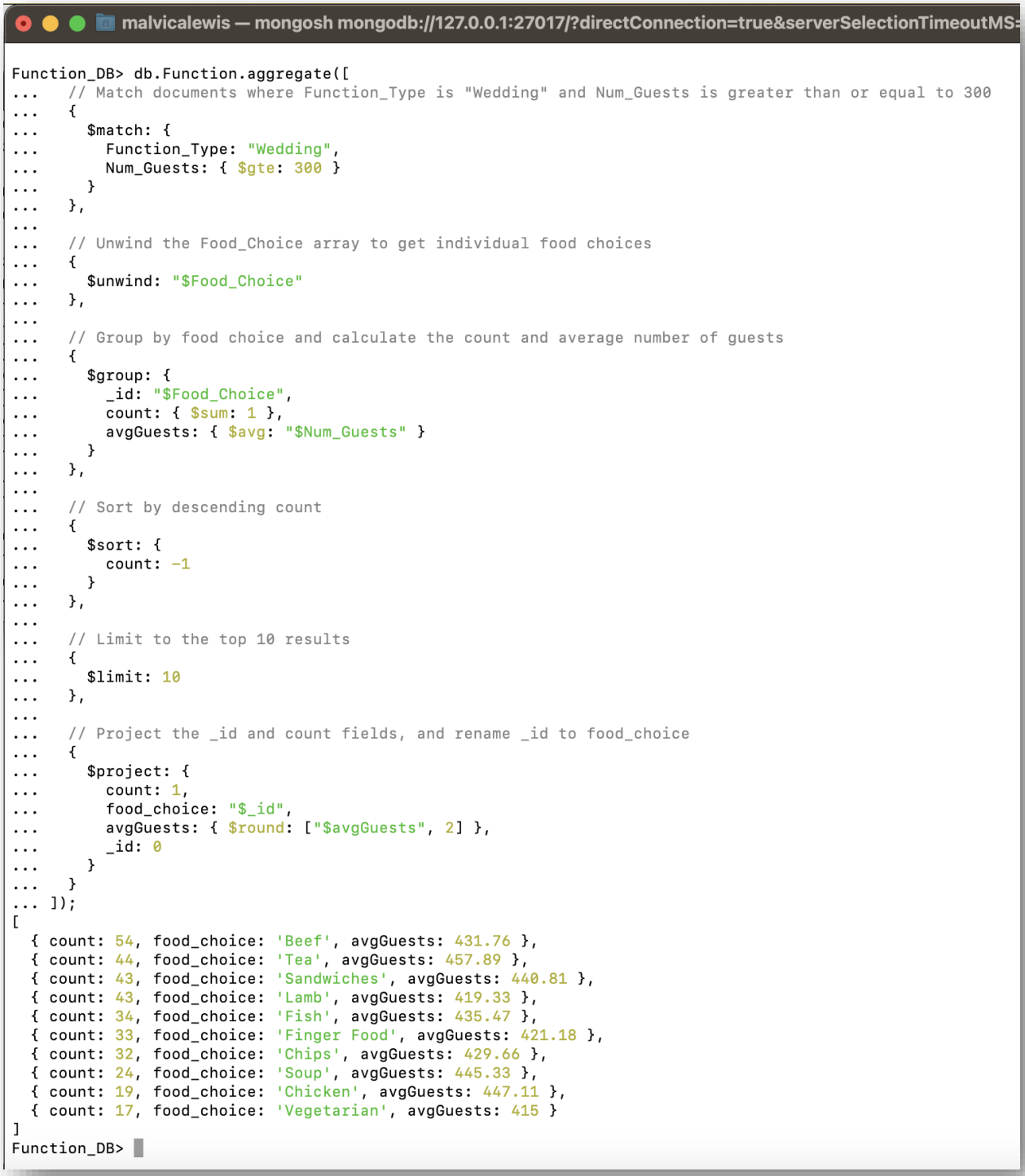
```
  // Project the _id and count fields, and rename _id to food_choice
  {
    $project: {
      count: 1,
      food_choice: "$_id",
      avgGuests: { $round: ["$avgGuests", 2] },
      _id: 0
    }
  }
])
```

The query uses the following functions:

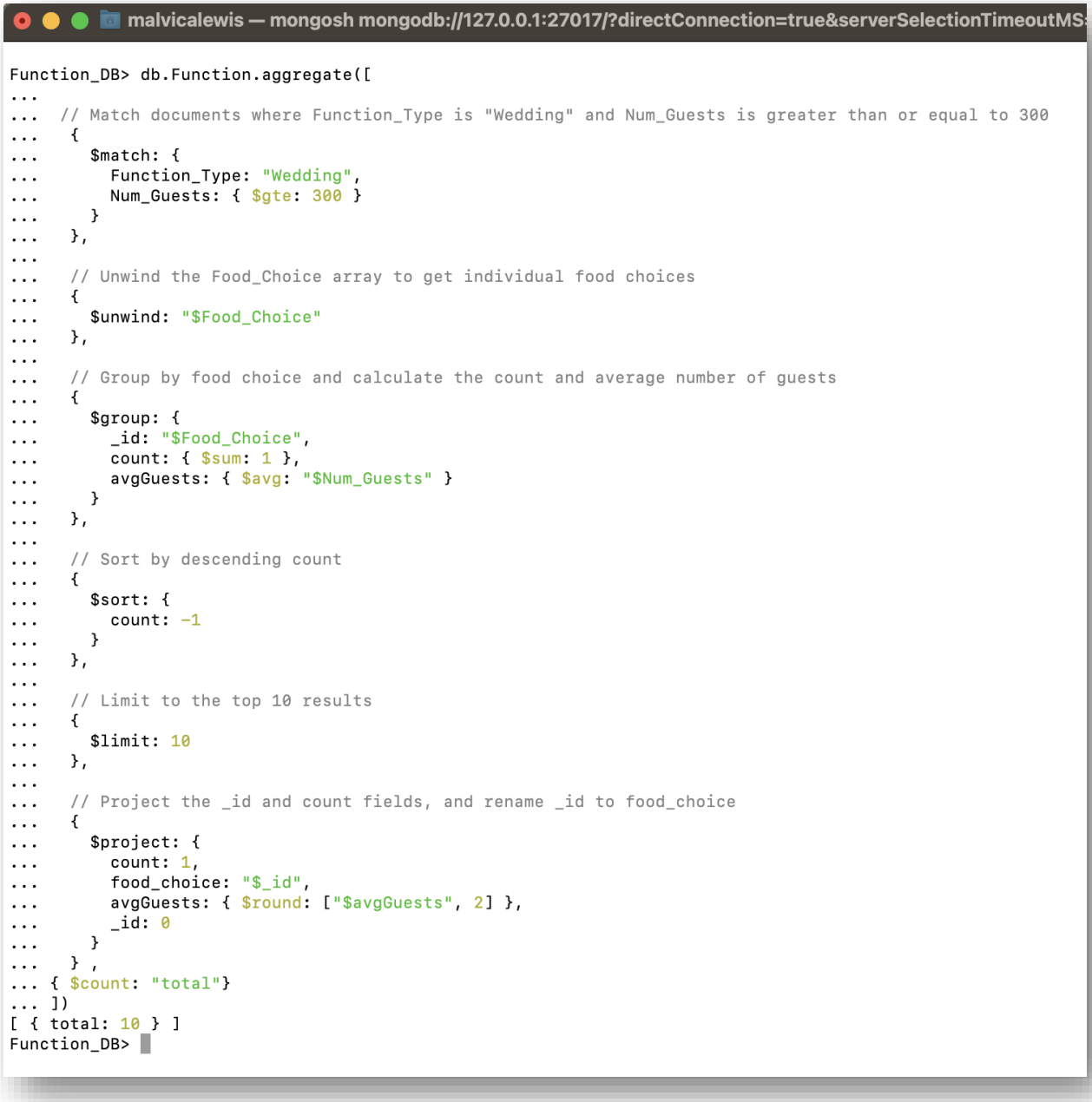
- **$match**: Filters the documents based on the given criteria. Here, it matches documents where the "**Function_Type**" field is "**Wedding**" and the "**Num_Guests**" field is greater than or equal to **300**.
- **$unwind**: Deconstructs the "**Food_Choice**" array field from the input documents to output a document for each element.
- **$group**: Groups the documents by the "**Food_Choice**" field and calculates the count and average number of guests for each group. The **_id** field is used to group the documents by the unique values of "**Food_Choice**".
- **$sort**: Sorts the documents by the "**count**" field in descending order.
- **$limit**: Limits the number of documents to **10**.
- **$project**: Projects the output fields to include only the "**count**", "**food_choice**", and "**avgGuests**" fields. The **_id** field is excluded from the output and the "**avgGuests**" field is rounded to two decimal places.

The result of this aggregation pipeline will be a list of the top **10** food choices for weddings with more than or equal to **300** guests, along with the count of events that had each food choice and the average number of guests for each food choice.

Query Output



Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSelectionTimeoutMS
Function_DB> db.Function.aggregate([
... // Match documents where Function_Type is "Wedding" and Num_Guests is greater than or equal to 300
... {
... $match: {
... Function_Type: "Wedding",
... Num_Guests: { $gte: 300 }
... }
... },
... // Unwind the Food_Choice array to get individual food choices
... {
... $unwind: "$Food_Choice"
... },
... // Group by food choice and calculate the count and average number of guests
... {
... $group: {
... _id: "$Food_Choice",
... count: { $sum: 1 },
... avgGuests: { $avg: "$Num_Guests" }
... }
... },
... // Sort by descending count
... {
... $sort: {
... count: -1
... }
... },
... // Limit to the top 10 results
... {
... $limit: 10
... },
... // Project the _id and count fields, and rename _id to food_choice
... {
... $project: {
... count: 1,
... food_choice: "$_id",
... avgGuests: { $round: ["$avgGuests", 2] },
... _id: 0
... }
... } ,
... { $count: "total"}
...])
[{ total: 10 }]
Function_DB>

11. What are the various statistics and characteristics of events held in 2022 such as:

- the number of events,
- average number of guests,
- total number of guests,
- minimum and maximum number of guests,
- most and least popular food choices,
- most and least popular alcoholic drinks,
- most and least popular soft drinks,
- average duration of events,
- total cost of disc jockeys and venues, and
- the names of all venues with good acoustics?

  How do these statistics and characteristics vary across different years and function types?

```
db.Function.aggregate([

 // Filter records for the year 2022
 {
  $match: {
   Year_Held: 2022
  }
 },

 // Group by year and function type
 {
  $group: {
   _id: {
    year: "$Year_Held",
    functionType: "$Function_Type"
   },

   // Count the number of events per year and function type
   numEvents: { $sum: 1 },

   // Calculate the average number of guests per event
   avgGuests: { $avg: "$Num_Guests" },
```

```
    // Calculate the total number of guests for all events
    totalGuests: { $sum: "$Num_Guests" },

    // Calculate the minimum and maximum number of guests per event
    minGuests: { $min: "$Num_Guests" },
    maxGuests: { $max: "$Num_Guests" },

    // Calculate the most and least popular food choices
    mostPopularFood: { $max: "$Food_Choice" },
    leastPopularFood: { $min: "$Food_Choice" },

    // Calculate the most and least popular alcoholic drinks
    mostPopularAlcohol: { $max: "$Alcohol" },
    leastPopularAlcohol: { $min: "$Alcohol" },

    // Calculate the most and least popular soft drinks
    mostPopularSoftDrink: { $max: "$Soft_Drinks" },
    leastPopularSoftDrink: { $min: "$Soft_Drinks" },

    // Calculate the average duration of the events
    avgDuration: { $avg: "$Duration" },

    // Calculate the total cost of disc jockeys and venues
    totalCost: {
      $sum: {
        $add: ["$Disc_Jockey.Fee", 1000, { $cond: [{ $eq: ["$Venue.Type",
              "Sports Venue"] }, 2000, 0] }]
      }
    },

    // List the names of all venues with good acoustics
    goodAcousticsVenues: { $addToSet: { $cond: [{ $eq:
                ["$Venue.Acoustics", "Good"] }, "$Venue.Name", null] } }
  }
},

// Sort by year and function type
{
  $sort: {
```

```
    "_id.year": -1,
    "_id.functionType": 1
  }
 }
])
```

This is a complex aggregation pipeline with several stages:

1. **$group**: Group the documents by year held and function type. Calculate several aggregate values, including the number of events per year and function type (`**numEvents**`), the average number of guests per event (`**avgGuests**`), the total number of guests for all events (`**totalGuests**`), the minimum and maximum number of guests per event (`**minGuests**` and `**maxGuests**`, respectively), the most and least popular food choices (`**mostPopularFood**` and `**leastPopularFood**`, respectively), the most and least popular alcoholic drinks (`**mostPopularAlcohol**` and `**leastPopularAlcohol**`, respectively), the most and least popular soft drinks (`**mostPopularSoftDrink**` and `**leastPopularSoftDrink**`, respectively), the average duration of the events (`**avgDuration**`), the total cost of disc jockeys and venues (`**totalCost**`), and a list of the names of all venues with good acoustics (`**goodAcousticsVenues**`).

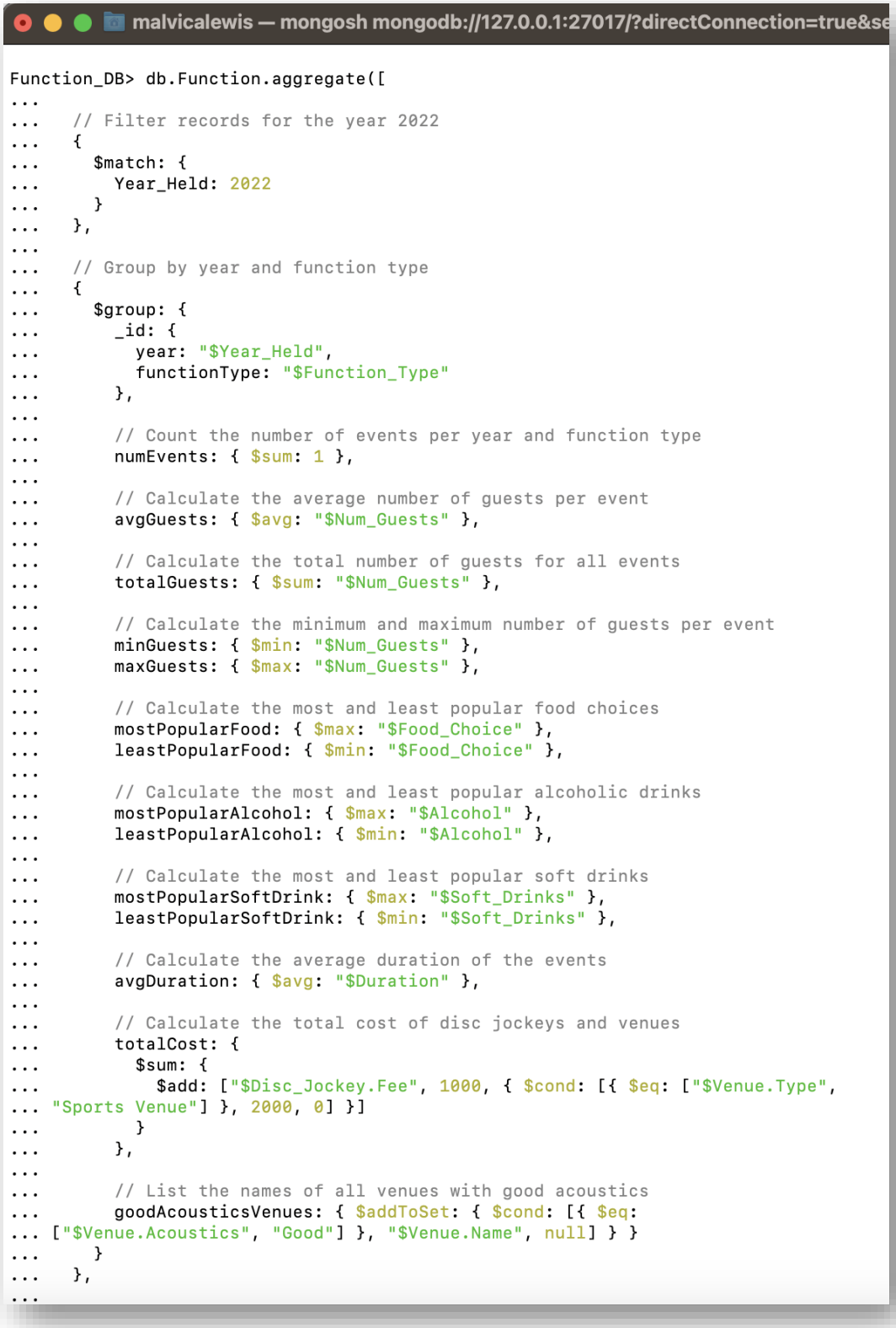
2. `**$sort**`: Sort the results by year held in descending order and function type in ascending order.

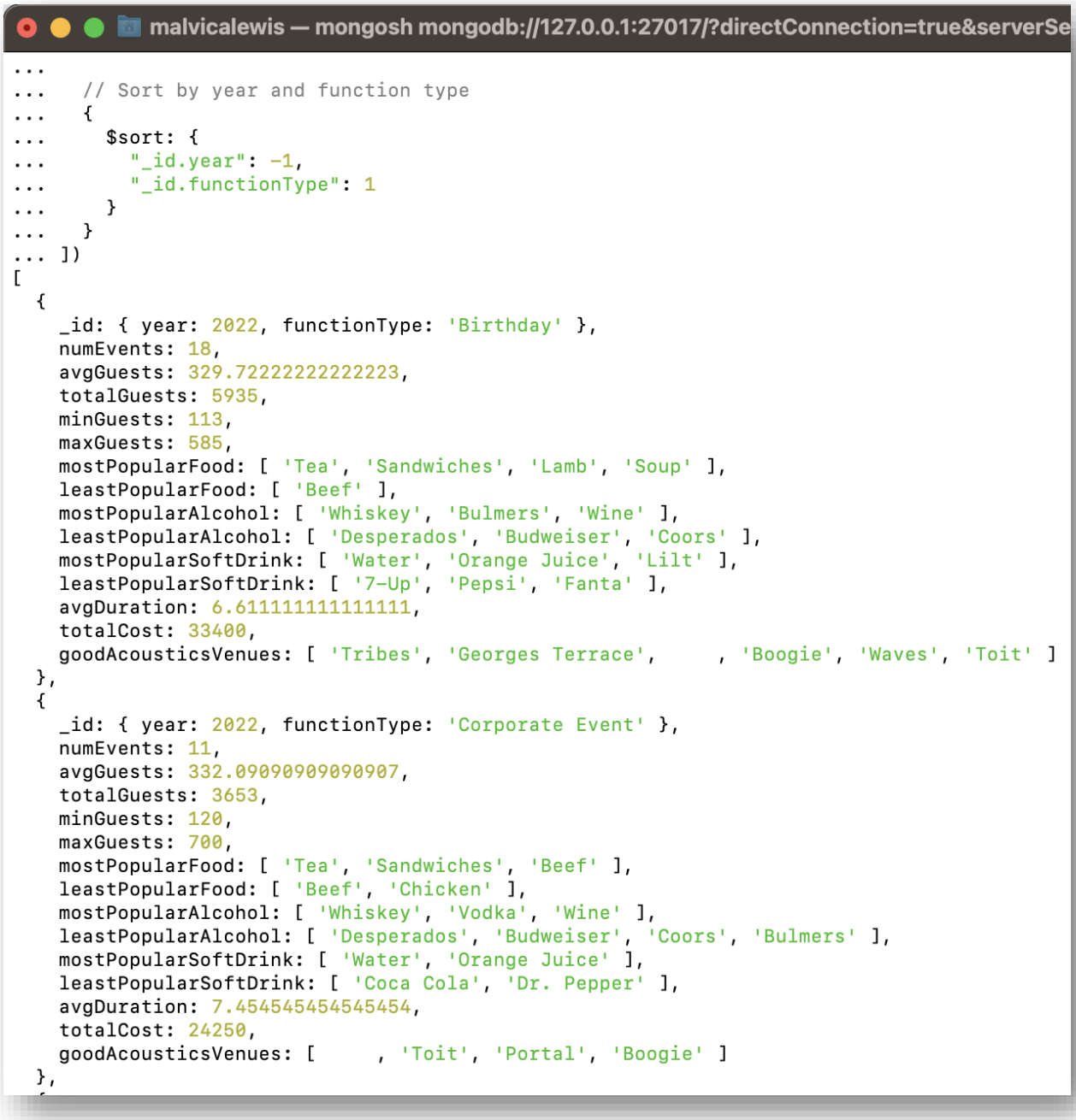
Here is a breakdown of each aggregate value:

- `**numEvents**`: This field is calculated using the `**$sum**` operator to count the number of events per year and function type.
- `**avgGuests**`: This field is calculated using the `**$avg**` operator to calculate the average number of guests per event for each year and function type.
- `**totalGuests**`: This field is calculated using the `**$sum**` operator to calculate the total number of guests for all events for each year and function type.
- `**minGuests**`: This field is calculated using the `**$min**` operator to find the minimum number of guests per event for each year and function type.
- `**maxGuests**`: This field is calculated using the `**$max**` operator to find the maximum number of guests per event for each year and function type.
- `**mostPopularFood**`: This field is calculated using the `**$max**` operator to find the most popular food choice for each year and function type.
- `**leastPopularFood**`: This field is calculated using the `**$min**` operator to find the least popular food choice for each year and function type.

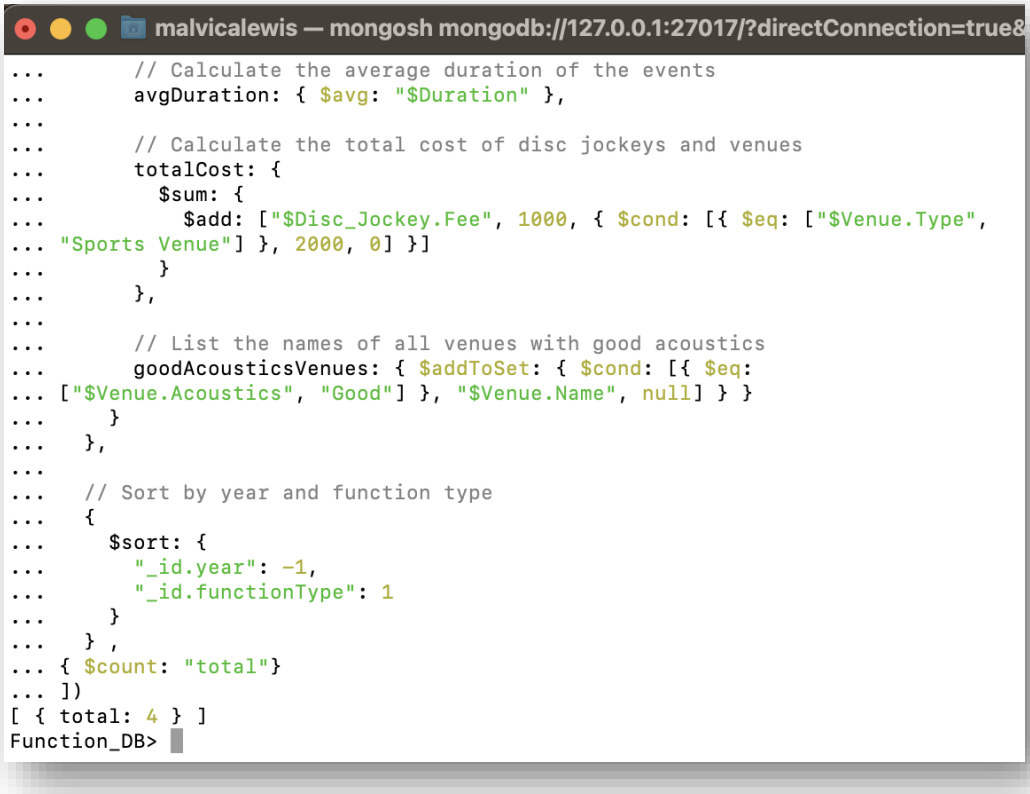
- `**mostPopularAlcohol**`: This field is calculated using the `**$max**` operator to find the most popular alcoholic drink for each year and function type.
- `**leastPopularAlcohol**`: This field is calculated using the `**$min**` operator to find the least popular alcoholic drink for each year and function type.
- `**mostPopularSoftDrink**`: This field is calculated using the `**$max**` operator to find the most popular soft drink for each year and function type.
- `**leastPopularSoftDrink**`: This field is calculated using the `**$min**` operator to find the least popular soft drink for each year and function type.
- `**avgDuration**`: This field is calculated using the `**$avg**` operator to find the average duration of events for each year and function type.
- `**totalCost**`: This field is calculated using the `**$sum**` operator and some conditional logic to find the total cost of disc jockeys and venues for each year and function type.
- `**goodAcousticsVenues**`: This field is calculated using the `**$addToSet**` operator and some conditional logic to create a list of the names of all venues with good acoustics for each year and function type.

Query Output




malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&serverSe
... // Sort by year and function type
... {
... $sort: {
... "_id.year": -1,
... "_id.functionType": 1
... }
... }
...])
[
{
_id: { year: 2022, functionType: 'Birthday' },
numEvents: 18,
avgGuests: 329.72222222222223,
totalGuests: 5935,
minGuests: 113,
maxGuests: 585,
mostPopularFood: ['Tea', 'Sandwiches', 'Lamb', 'Soup'],
leastPopularFood: ['Beef'],
mostPopularAlcohol: ['Whiskey', 'Bulmers', 'Wine'],
leastPopularAlcohol: ['Desperados', 'Budweiser', 'Coors'],
mostPopularSoftDrink: ['Water', 'Orange Juice', 'Lilt'],
leastPopularSoftDrink: ['7-Up', 'Pepsi', 'Fanta'],
avgDuration: 6.611111111111111,
totalCost: 33400,
goodAcousticsVenues: ['Tribes', 'Georges Terrace', , 'Boogie', 'Waves', 'Toit']
},
{
_id: { year: 2022, functionType: 'Corporate Event' },
numEvents: 11,
avgGuests: 332.09090909090907,
totalGuests: 3653,
minGuests: 120,
maxGuests: 700,
mostPopularFood: ['Tea', 'Sandwiches', 'Beef'],
leastPopularFood: ['Beef', 'Chicken'],
mostPopularAlcohol: ['Whiskey', 'Vodka', 'Wine'],
leastPopularAlcohol: ['Desperados', 'Budweiser', 'Coors', 'Bulmers'],
mostPopularSoftDrink: ['Water', 'Orange Juice'],
leastPopularSoftDrink: ['Coca Cola', 'Dr. Pepper'],
avgDuration: 7.454545454545454,
totalCost: 24250,
goodAcousticsVenues: [, 'Toit', 'Portal', 'Boogie']
},

Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=true&
... // Calculate the average duration of the events
... avgDuration: { $avg: "$Duration" },
...
... // Calculate the total cost of disc jockeys and venues
... totalCost: {
... $sum: {
... $add: ["$Disc_Jockey.Fee", 1000, { $cond: [{ $eq: ["$Venue.Type",
... "Sports Venue"] }, 2000, 0] }]
... }
... },
...
... // List the names of all venues with good acoustics
... goodAcousticsVenues: { $addToSet: { $cond: [{ $eq:
... ["$Venue.Acoustics", "Good"] }, "$Venue.Name", null] } }
... }
... },
...
... // Sort by year and function type
... {
... $sort: {
... "_id.year": -1,
... "_id.functionType": 1
... }
... } ,
... { $count: "total"}
...])
[{ total: 4 }]
Function_DB>

12. What are the venues with good acoustics that have hosted at least 50 events, and what are the average number of guests and acoustic ratings for each city in which they are located? Also, how much alcohol and soft drinks have been served at these events in each city?

```
db.Function.aggregate([
 { $match: { "Venue.Name": { $exists: true } } },
 { $group: { _id: "$Venue.Name", numEvents: { $sum: 1 } } },
 { $match: { numEvents: { $gte: 50 } } },
 {
  $lookup: {
   from: "Function",
   let: { venueName: "$_id" },
   pipeline: [
    { $match: { $expr: { $eq: [ "$Venue.Name", "$$venueName" ] } } },
    {
     $group: {
      _id: "$Venue.City",
      acoustics: {
       $avg: {
        $switch: {
         branches: [
          { case: { $eq: [ "$Venue.Acoustics", "Poor" ] }, then: 1 },
          { case: { $eq: [ "$Venue.Acoustics", "Fair" ] }, then: 2 },
          { case: { $eq: [ "$Venue.Acoustics", "Good" ] }, then: 3 }
         ],
         default: 0
        }
       }
      },
      avgGuests: { $avg: "$Num_Guests" },
      totalAlcohol: { $sum: { $size: "$Alcohol" } },
      totalSoftDrinks: { $sum: { $size: "$Soft_Drinks" } }
     }
    },
    {
     $group: {
      _id: null,
      cities: {
```

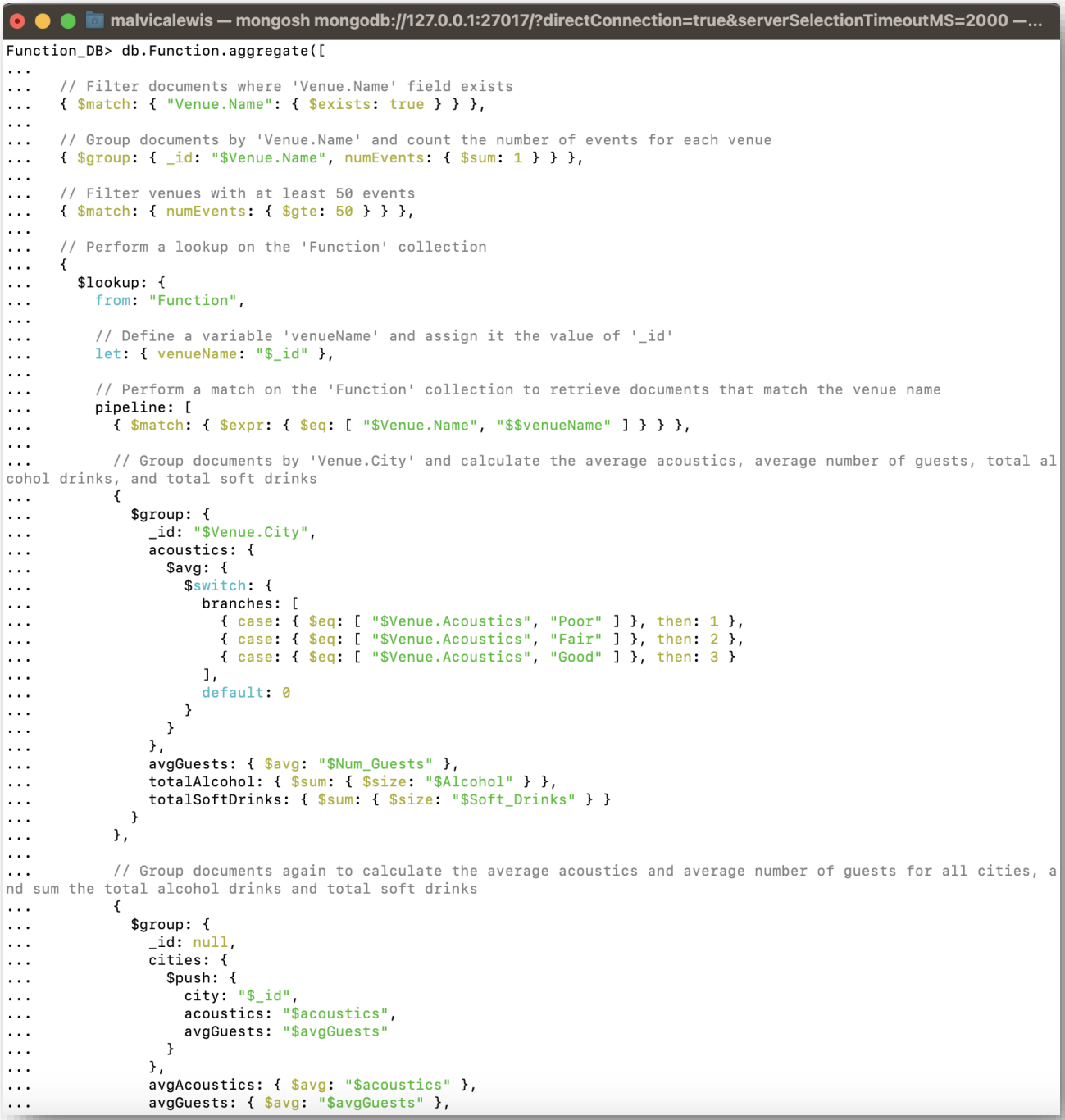
```
          $push: {
            city: "$_id",
            acoustics: "$acoustics",
            avgGuests: "$avgGuests"
          }
        },
        avgAcoustics: { $avg: "$acoustics" },
        avgGuests: { $avg: "$avgGuests" },
        totalAlcohol: { $sum: "$totalAlcohol" },
        totalSoftDrinks: { $sum: "$totalSoftDrinks" }
      }
    },
    { $unwind: "$cities" },
    {
      $project: {
        _id: 0,
        city: "$cities.city",
        acoustics: "$cities.acoustics",
        avgGuests: "$cities.avgGuests",
        avgAcoustics: 1,
        avgGuestsAll: "$avgGuests",
        totalAlcohol: 1,
        totalSoftDrinks: 1
      }
    }
  ],
  as: "cityStats"
 }
},
{
 $project: {
  _id: "$_id",
  numEvents: 1,
  cityStats: 1
  }
},
{ $sort: { "cityStats.acoustics": -1, "cityStats.avgGuests": -1 } }
])
```

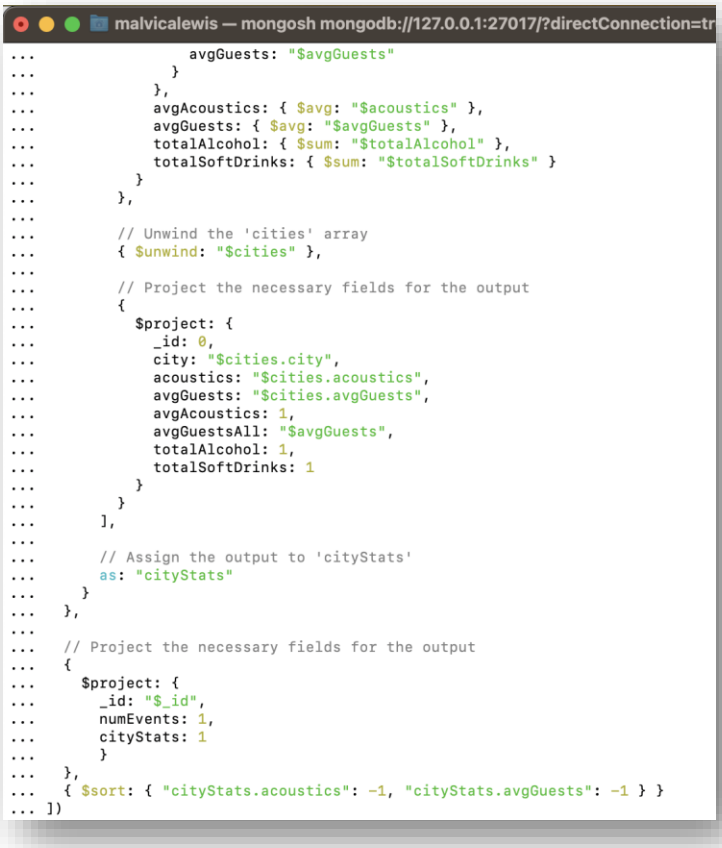
This is an aggregation query in MongoDB with several stages:

- The **$match** stage filters documents where the **Venue.Name** field exists.
- The **$group** stage groups the documents by **Venue.Name** and counts the number of events for each venue, creating a field **numEvents** with the sum of **1** for each event.
- The **$match** stage filters the venues that have at least **3** events.
- The **$lookup** stage performs a lookup on the Function collection, joining the documents with the venues that match the **_id** field.
- The **$match** stage filters the documents in the Function collection to retrieve only the ones that match the venue name.
- The **$group** stage groups the documents by **Venue.City** and calculates the average acoustics, average number of guests, total alcohol drinks, and total soft drinks for each city.
- The **$group** stage groups the documents again to calculate the average acoustics and average number of guests for all cities and sum the total alcohol drinks and total soft drinks.
- The **$unwind** stage unwinds the cities array.
- The **$project** stage projects the necessary fields for the output.
- The **$project** stage projects the necessary fields for the output.
- The **$sort** stage sorts the output documents in descending order of the **acoustics** and **avgGuests** fields of the **cityStats** sub-document.

The query is used to retrieve information about venues and their events, as well as statistics on their acoustics, number of guests, and drinks consumed. The output is sorted in descending order of the acoustics and average number of guests fields of the **cityStats** sub-document.

Query Output




malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=tr
... avgGuests: "$avgGuests"
... }
... },
... avgAcoustics: { $avg: "$acoustics" },
... avgGuests: { $avg: "$avgGuests" },
... totalAlcohol: { $sum: "$totalAlcohol" },
... totalSoftDrinks: { $sum: "$totalSoftDrinks" }
... }
... },
... // Unwind the 'cities' array
... { $unwind: "$cities" },
... // Project the necessary fields for the output
... {
... $project: {
... _id: 0,
... city: "$cities.city",
... acoustics: "$cities.acoustics",
... avgGuests: "$cities.avgGuests",
... avgAcoustics: 1,
... avgGuestsAll: "$avgGuests",
... totalAlcohol: 1,
... totalSoftDrinks: 1
... }
... }
...],
... // Assign the output to 'cityStats'
... as: "cityStats"
... }
... },
... // Project the necessary fields for the output
... {
... $project: {
... _id: "$_id",
... numEvents: 1,
... cityStats: 1
... }
... },
... { $sort: { "cityStats.acoustics": -1, "cityStats.avgGuests": -1 } }
...])


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnect
[
{
numEvents: 51,
cityStats: [
{
avgAcoustics: 3,
totalAlcohol: 131,
totalSoftDrinks: 162,
city: 'Mountrath',
acoustics: 3,
avgGuests: 429.96078431372547,
avgGuestsAll: 429.96078431372547
}
],
_id: 'Tribes'
},
{
numEvents: 53,
cityStats: [
{
avgAcoustics: 1,
totalAlcohol: 164,
totalSoftDrinks: 172,
city: 'Portadown',
acoustics: 1,
avgGuests: 405.9622641509434,
avgGuestsAll: 405.9622641509434
}
],
_id: 'The Venue'
}
]
Function_DB>

Query Count


malvicalewis — mongosh mongodb://127.0.0.1:27017/?directConnection=tru
... _id: 0,
... city: "$cities.city",
... acoustics: "$cities.acoustics",
... avgGuests: "$cities.avgGuests",
... avgAcoustics: 1,
... avgGuestsAll: "$avgGuests",
... totalAlcohol: 1,
... totalSoftDrinks: 1
... }
... }
...],
...
... // Assign the output to 'cityStats'
... as: "cityStats"
... }
... },
...
... // Project the necessary fields for the output
... {
... $project: {
... _id: "$_id",
... numEvents: 1,
... cityStats: 1
... }
... },
... { $sort: { "cityStats.acoustics": -1, "cityStats.avgGuests": -1 } } ,
... { $count: "total"}
...])
[{ total: 2 }]
Function_DB>

MongoDB Compass Interface

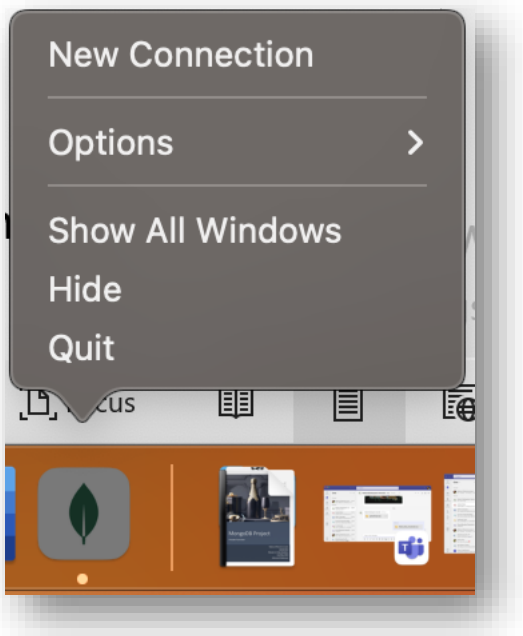
MongoDB Compass is a graphical user interface (GUI) tool for MongoDB, the popular document-oriented NoSQL database system. Compass provides a user-friendly interface to interact with MongoDB, allowing users to easily view, query, and manipulate data stored in their MongoDB databases. It supports all features of MongoDB, including querying, indexing, and aggregation, and provides a set of powerful visualization tools to explore data.

With Compass, users can connect to their MongoDB databases, view and edit documents, create and modify indexes, and run queries and aggregations. Compass also includes a built-in schema analysis tool that helps users to understand the structure of their data and relationships between collections. Additionally, Compass provides a real-time performance analysis feature that monitors the performance of MongoDB queries and operations, helping users to optimize their database performance.

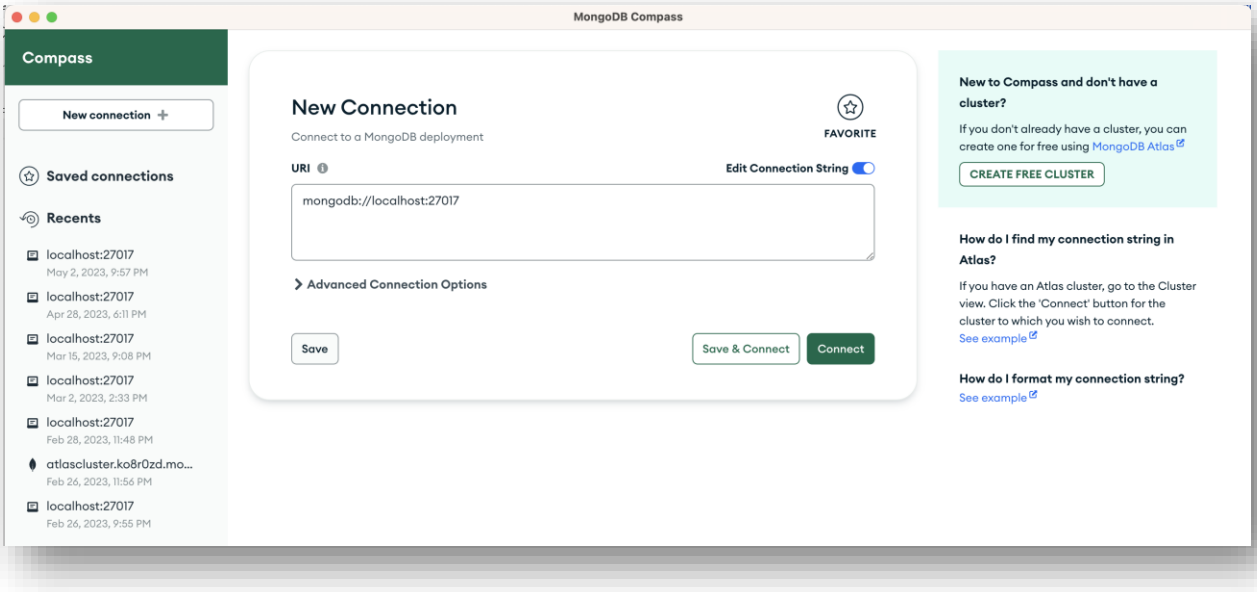
Connecting to a localhost MongoDB deployment

To connect to a MongoDB deployment running on your local machine, follow these steps in MongoDB Compass:

1. Open MongoDB Compass and click on the "**New Connection**" button. You can also right-click MongoDB Compass application on the toolbar and select the "**New Connection**" option.



2. In the "**New Connection**" tab, set the following:
    a. **Hostname**: Enter "**localhost**" or "**127.0.0.1**" to connect to the local host.
    b. **Port**: Enter the port number that MongoDB is running on. The default port is **27017**.
    c. **Authentication**: If you have set up authentication for your MongoDB deployment, enter your username and password.



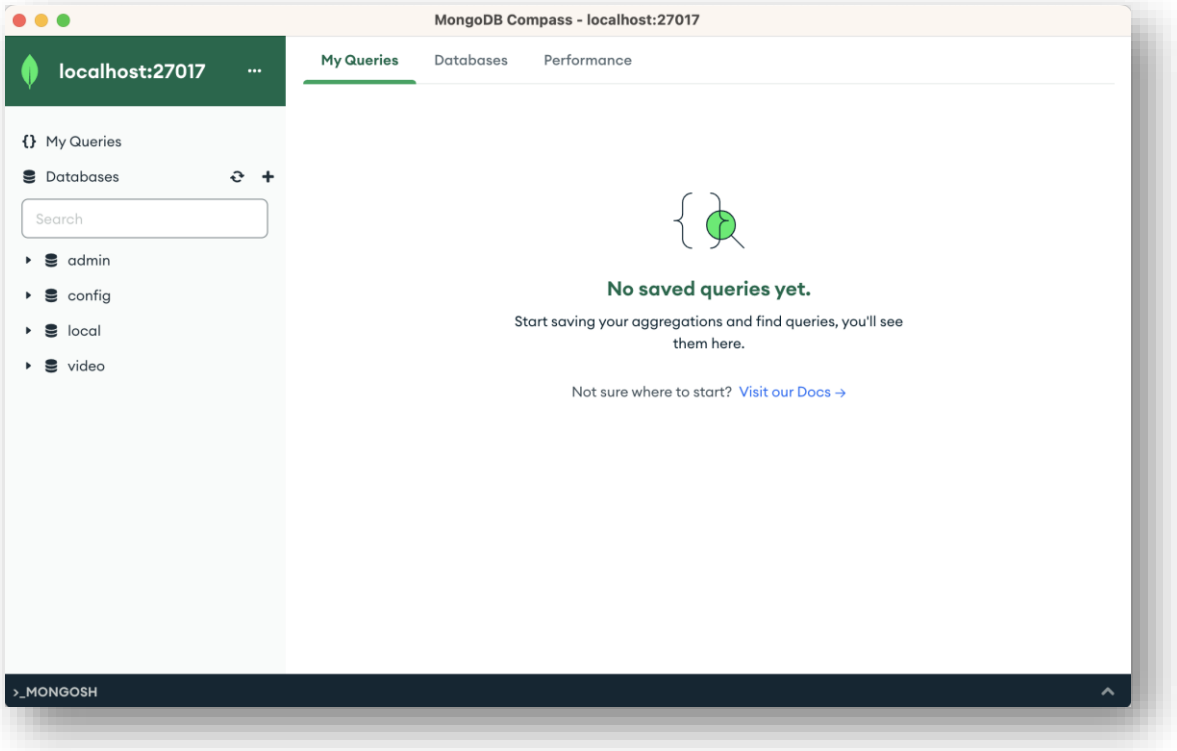
3. Click the "**Connect**" button to connect to the MongoDB deployment running on your local machine.

If the connection is successful, you will be taken to the MongoDB Compass home screen, where you can start working with your databases, collections, and documents.

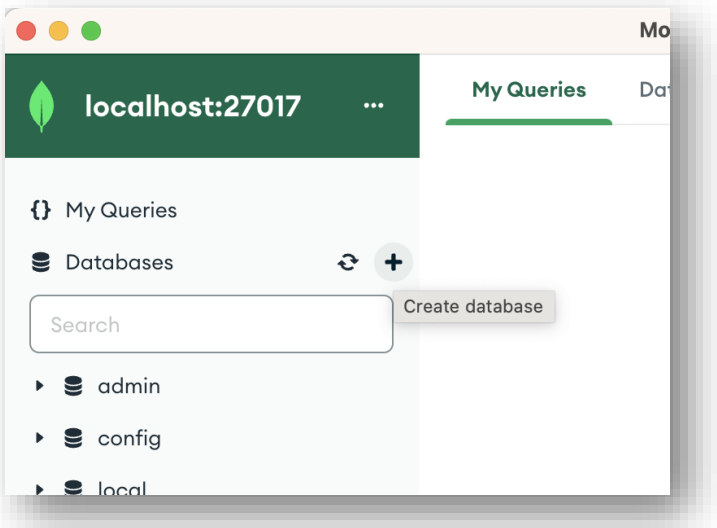
Creating a database in MongoDB compass using a JSON file

You can follow these steps to create a database in MongoDB Compass using a JSON file:

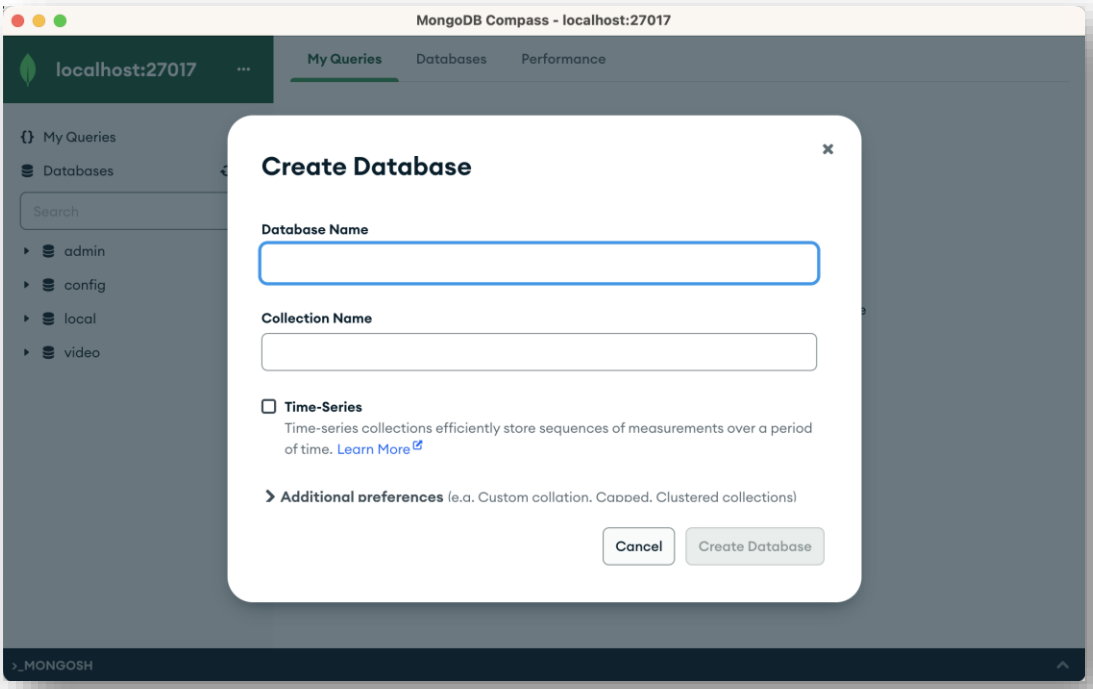
1. Open MongoDB Compass and connect to your MongoDB server.



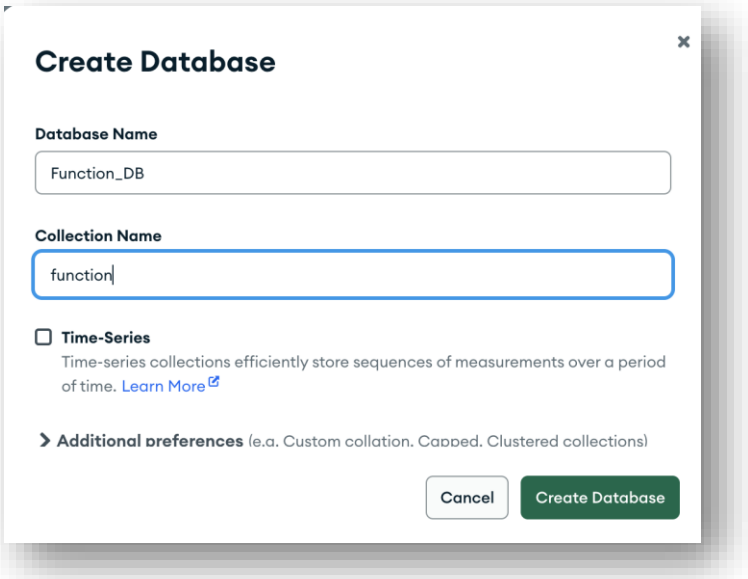
2. Click on the "**Create Database**" button in the top-left corner of the screen.



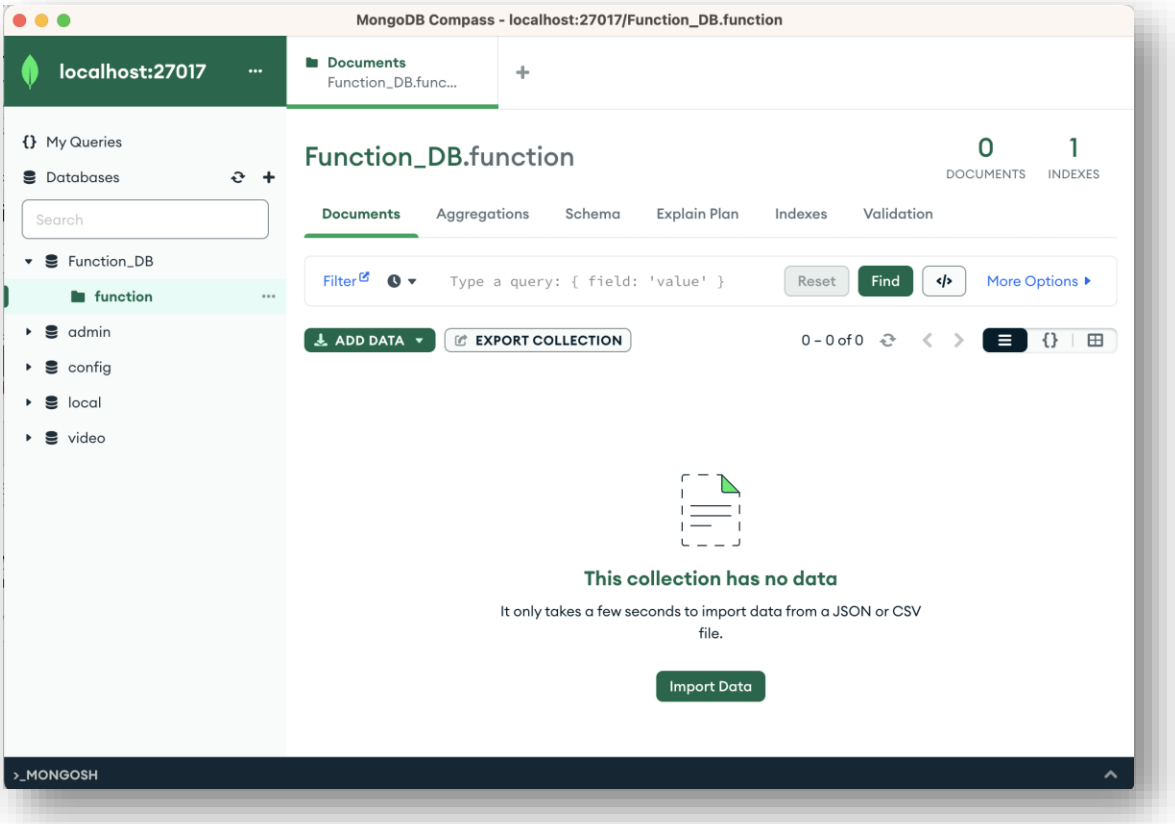
3. Enter a name for your database in the "**Database Name**" text box. We'll enter **Function_DB** as the database name for this project.



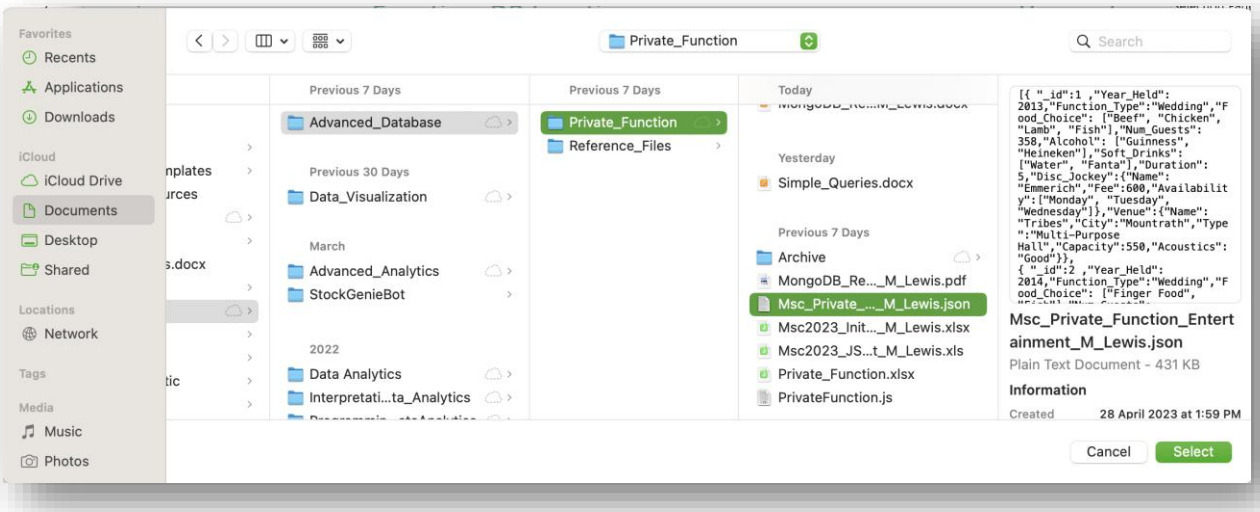
4. Enter a name for your collection in the "**Collection Name**" text box. We've used **function** as the collection name.
5. Click on the "**Create Database**" button.



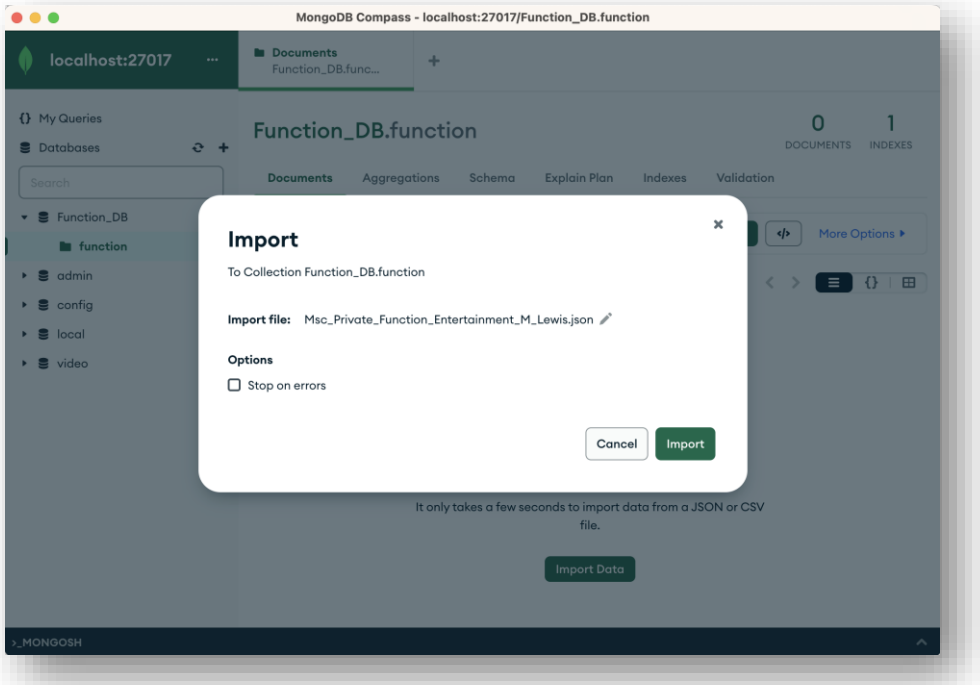
6. Click on the "**Import Data**" button in the bottom corner of the screen.



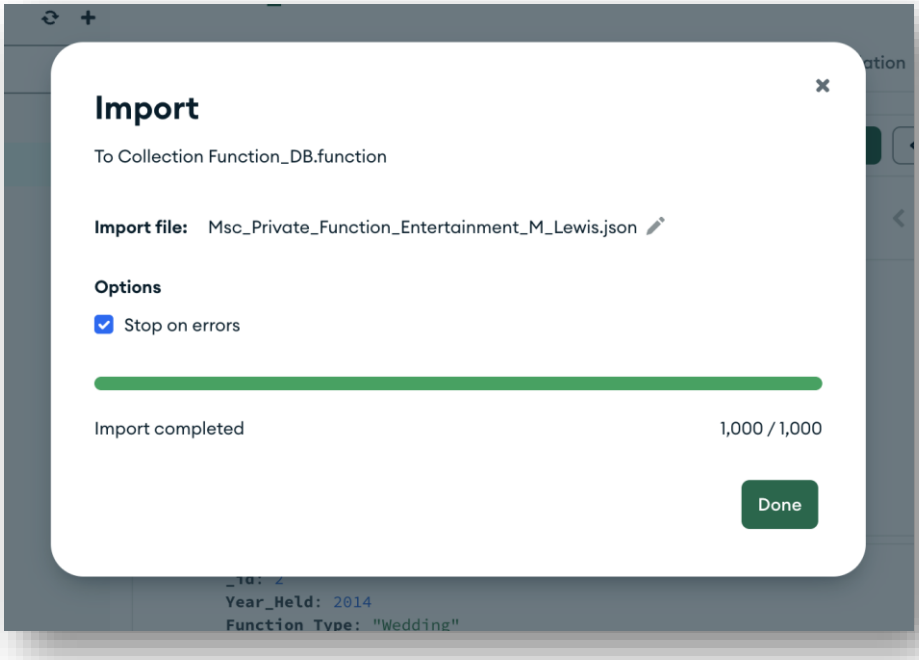
7. Select the JSON file that you want to import and click on the "**Import**" button.



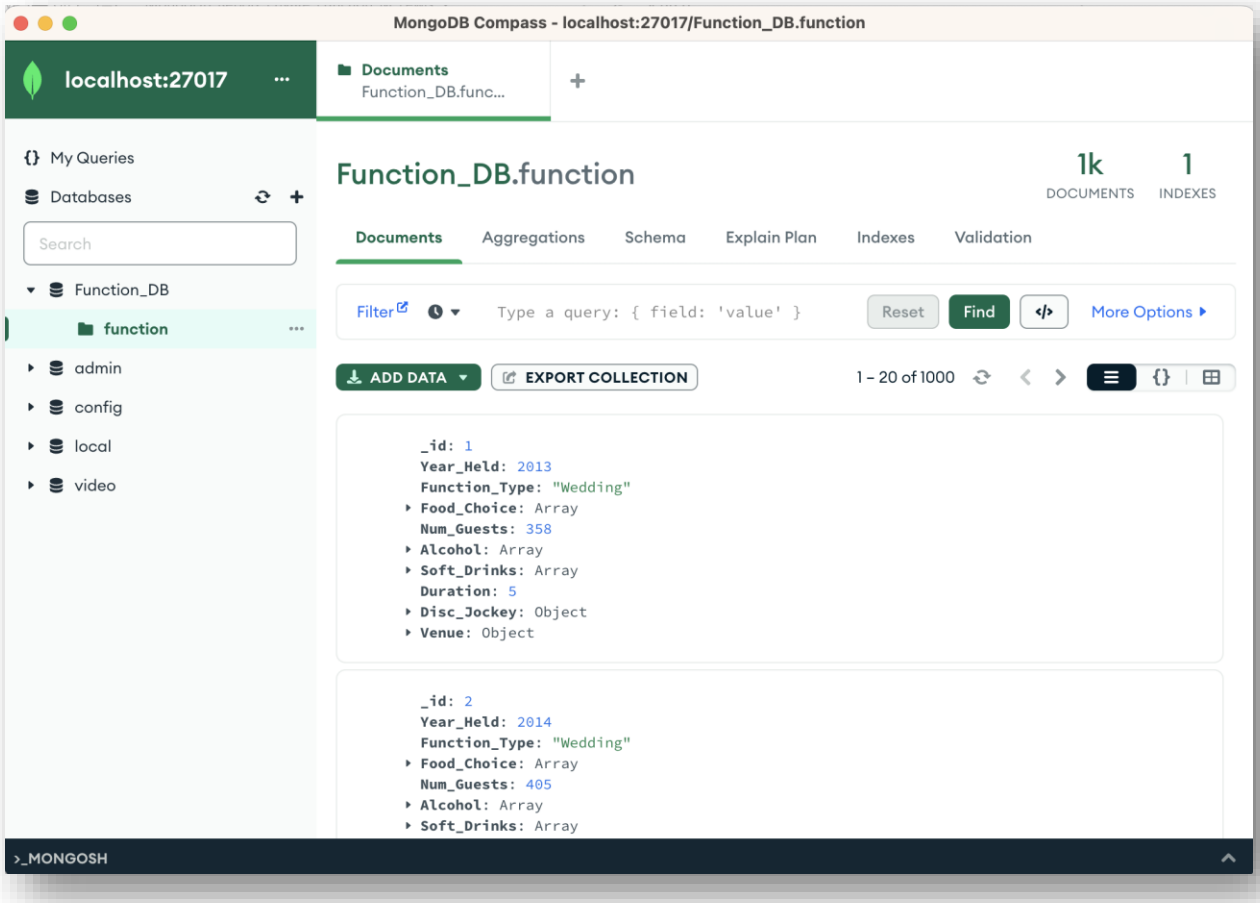
8. Click on the "**Import**" button to start the import process.



9. Once the import process is complete, the following screen is displayed. Click on the "**Done**" button to close the screen.



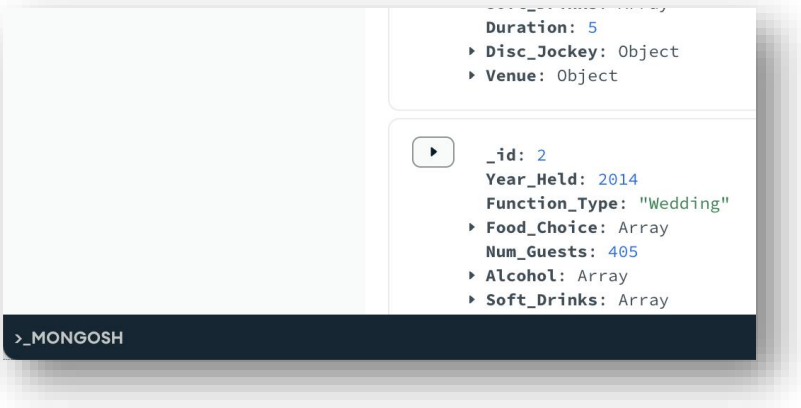
10. You can view your data in the "**Documents**" tab of your collection.



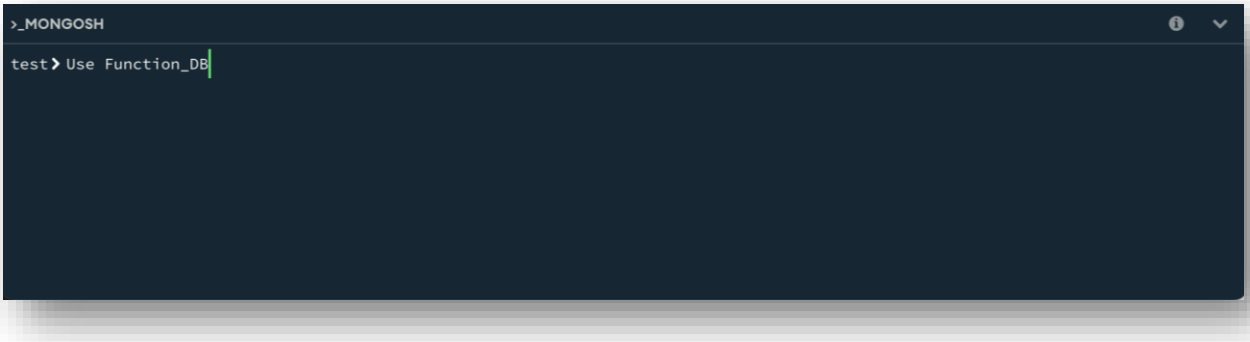
Running a basic query using Mongosh in Compass

You can follow these steps to run a basic query using Mongosh in Compass:

1. Open MongoDB Compass and connect to your MongoDB server.
2. Click on the "Mongosh" button in the bottom-left corner of the screen to open the Mongosh shell.



3. Switch to the **Function_DB** database by running the following command, and press **Enter** to run the query.



   This command will either create a new database with the name **Function_DB** if it doesn't already exist or switch to an existing database with that name.

4. To find the documents in the **function** collection, run the following command:

```
>_MONGOSH

Function_DB > db.function.find()
```

5. The results of the query will be displayed in the Mongosh shell.

```
>_MONGOSH

> db.function.find()
< {
    _id: 1,
    Year_Held: 2013,
    Function_Type: 'Wedding',
    Food_Choice: [
      'Beef',
      'Chicken',
      'Lamb',
      'Fish'
    ],
    Num_Guests: 358,
    Alcohol: [
      'Guinness',
      'Heineken'
    ],
    Soft_Drinks: [
      'Water',
      'Fanta'
    ],
```

Let's run the following query to find the events,

- that includes only weddings,
- with an even number of guests,
- where the length of the venue name is greater than half the number of alcohol choices,
- the food choice includes fish,
- there is more than one food choice, and
- the venue type ends with "hall" regardless of case

Query:

```
db.function.find({
 $and: [
  { Function_Type: "Wedding" },
  { Num_Guests: { $mod: [2, 0] } },
  {
   $expr: {
    $gt: [{ $strLenCP: "$Venue.Name" }, { $divide: [{ $size: "$Alcohol" },
2] }]
   }
  },
  { Food_Choice: { $all: ["Fish"] } },
  { Food_Choice: { $not: { $size: 1 } } },
  { "Venue.Type": { $regex: "hall$", $options: "i" } }
 ]
},
{
 "Function_Type": 1,
 "Num_Guests": 1,
 "Venue.Name": 1,
 "Alcohol": 1,
 "Food_Choice": 1,
 "Venue.Type": 1,
 "_id": 0
})
```

Query Output

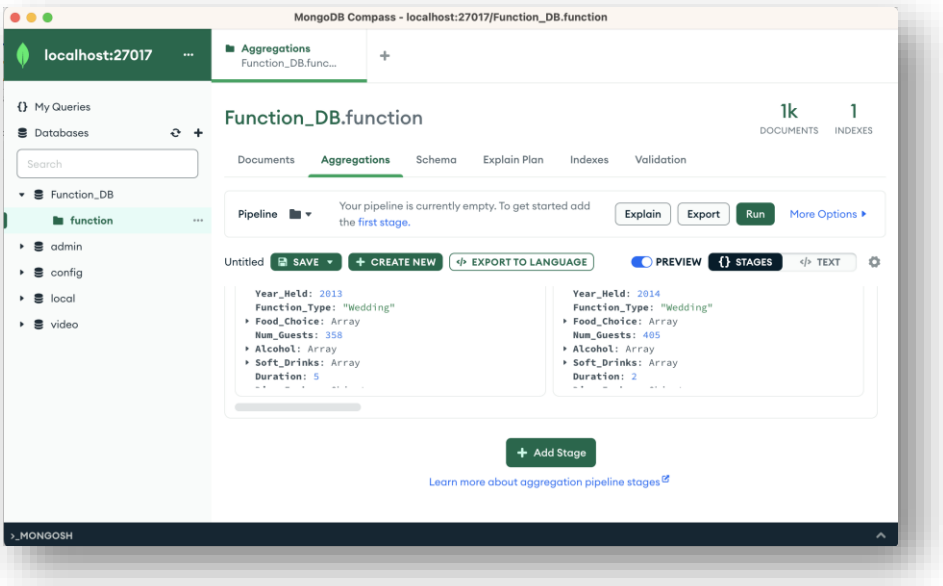
```
>_MONGOSH

> db.function.find({
    $and: [
      { Function_Type: "Wedding" },
      { Num_Guests: { $mod: [2, 0] } },
      {
        $expr: {
          $gt: [{ $strLenCP: "$Venue.Name" }, { $divide: [{ $size: "$Alcohol" }, 2] }]
        }
      },
      { Food_Choice: { $all: ["Fish"] } },
      { Food_Choice: { $not: { $size: 1 } } },
      { "Venue.Type": { $regex: "hall$", $options: "i" } }
    ]
  },
  {
    "Function_Type": 1,
    "Num_Guests": 1,
    "Venue.Name": 1,
    "Alcohol": 1,
    "Food_Choice": 1,
    "Venue.Type": 1,
    "_id": 0
  })
```

```
>_MONGOSH
{
  Function_Type: 'Wedding',
  Food_Choice: [
    'Beef',
    'Chicken',
    'Lamb',
    'Fish'
  ],
  Num_Guests: 358,
  Alcohol: [
    'Guinness',
    'Heineken'
  ],
  Venue: {
    Name: 'Tribes',
    Type: 'Multi-Purpose Hall'
  }
}
{
  Function_Type: 'Wedding',
  Food_Choice: [
    'Beef',
    'Fish',
    'Lamb'
  ],
  Num_Guests: 98,
  Alcohol: [
    'Heineken',
```

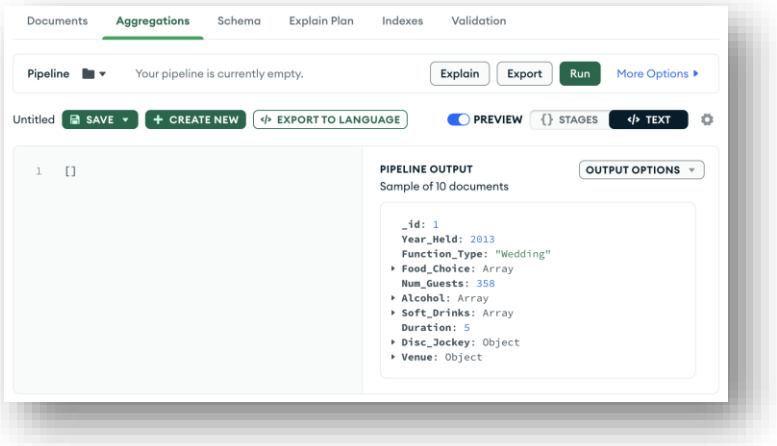
Running an aggregate query in MongoDB Compass

To run the aggregate query in Compass and understand the various stages, you can follow these steps:

1. Open MongoDB Compass and connect to your MongoDB server.
2. Select the database and collection that you want to perform the aggregation on.
3. Click on the "**Aggregations**" tab in the top navigation bar.



4. Click on the "**Add Stage**" button to start building your aggregation pipeline, or you can click on the **</> TEXT** button in the top-right corner of the screen, to enter the query on the text editor. It will automatically convert the query to various stages based on the aggregate functions used in the query.



5. Let's run a simple aggregate query to understand how the stages work in Compass:

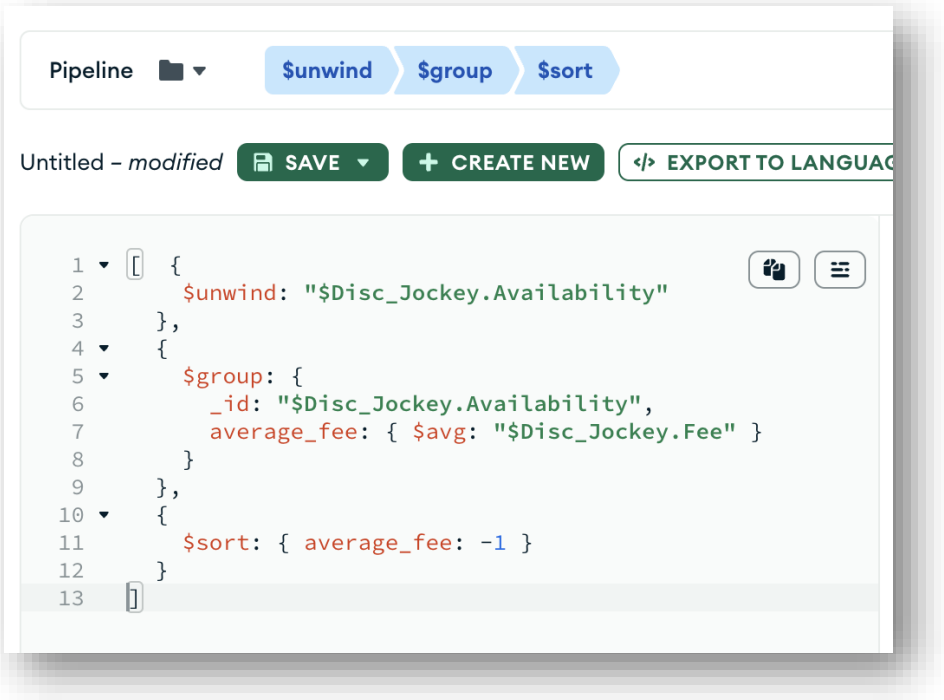
   Simple Aggregate Query

   Find the average fee for a disc jockey, grouped by the day of the week they are available on.

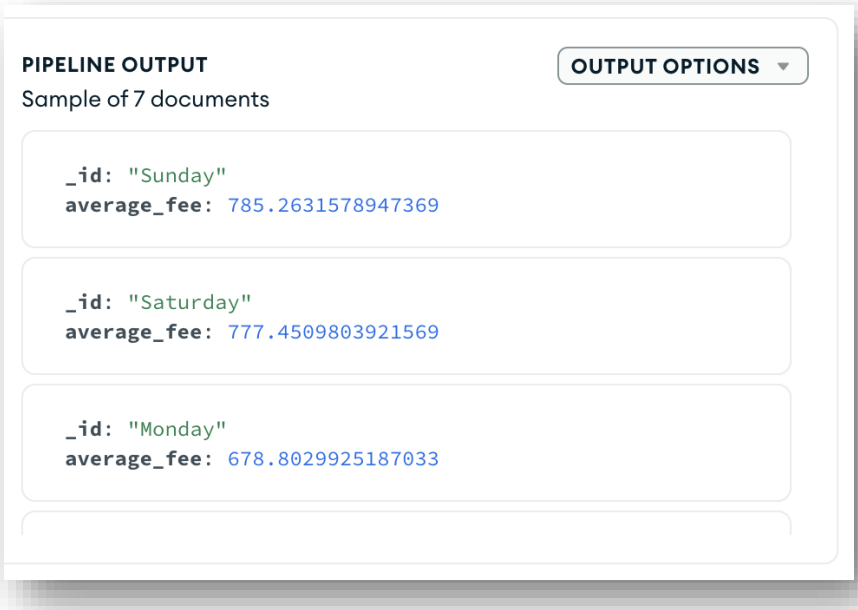
```
db.Function.aggregate([
 {
  $unwind: "$Disc_Jockey.Availability"
 },
 {
  $group: {
   _id: "$Disc_Jockey.Availability",
   average_fee: { $avg: "$Disc_Jockey.Fee" }
  }
 },
 {
  $sort: { average_fee: -1 }
 }
])
```

6. Paste the above simple aggregate query within the **db.Function.aggregate([ ])** on the text editor.

The Pipeline contains the various aggregate functions split into various stages as shown in the following screen:

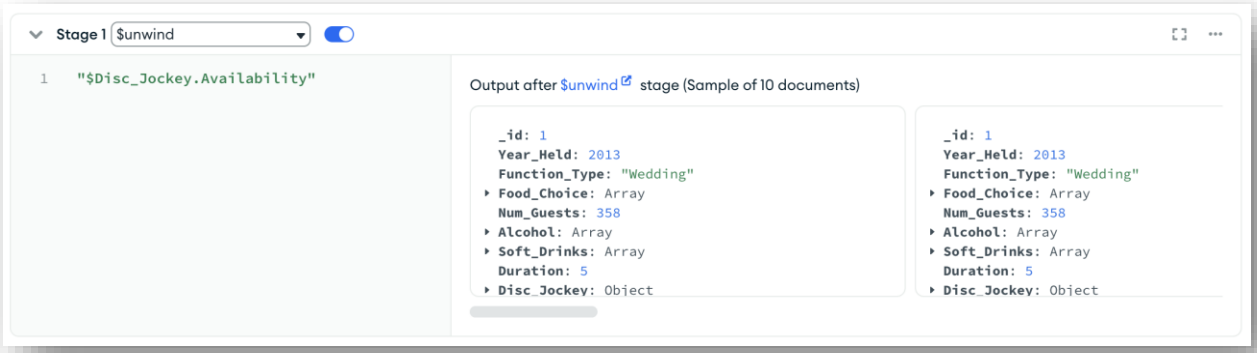


The output of the query is displayed in the "**PIPELINE OUTPUT**" section at the top-right corner.



7. Click on the {} STAGES button to view the various stages of the pipeline and their individual outputs.

Stage 1:



$unwind: This stage deconstructs the **Disc_Jockey.Availability** array and creates a new document for each element in the array, duplicating the values of the other fields in the original document. This allows subsequent stages to group and aggregate the data based on individual availability slots.

Stage 2:

$group: This stage groups the documents by the **Disc_Jockey.Availability** field and calculates the average fee for each availability slot using the **$avg** aggregation operator. The resulting documents contain the **_id** field with the availability slot value and the **average_fee** field with the calculated average fee.

Stage 3:

$sort: This stage sorts the resulting documents by the **average_fee** field in descending order, so that the availability slots with the highest average fees are listed first.

Overall, this query calculates the average fee for each availability slot in the Disc_Jockey subdocument and returns the results in descending order of average fee.

In the upcoming section, we will understand the functioning of each stage.

Stages in Aggregations: A Detailed Explanation of Each Stage in MongoDB Compass

Let's consider two complex aggregate queries that answers the following questions:

Query 1:

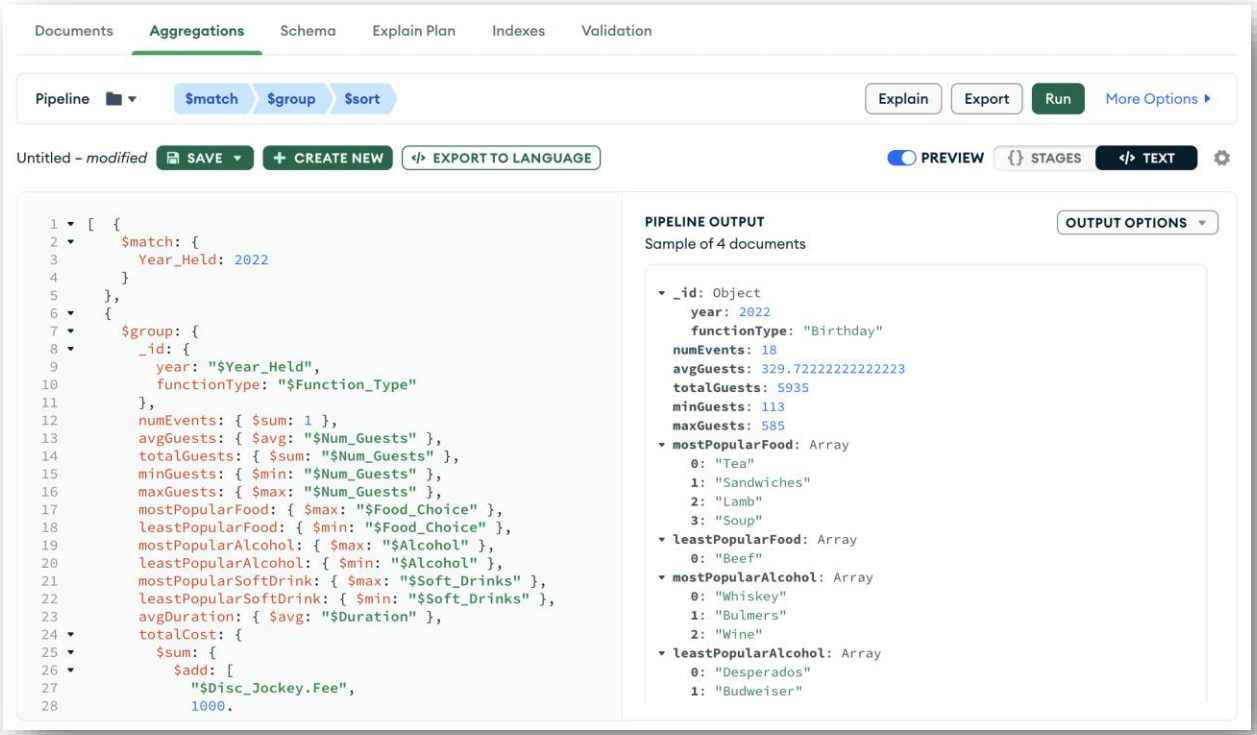
What are the various statistics and characteristics of events held in 2022 such as:

- the number of events,
- average number of guests,
- total number of guests,
- minimum and maximum number of guests,
- most and least popular food choices,
- most and least popular alcoholic drinks,
- most and least popular soft drinks,
- average duration of events,
- total cost of disc jockeys and venues, and
- the names of all venues with good acoustics?
- How do these statistics and characteristics vary across different years and function types?

```
db.Function.aggregate([
 {
  $match: {
   Year_Held: 2022
  }
 },
 {
  $group: {
   _id: {
    year: "$Year_Held",
    functionType: "$Function_Type"
   },
   numEvents: { $sum: 1 },
   avgGuests: { $avg: "$Num_Guests" },
   totalGuests: { $sum: "$Num_Guests" },
   minGuests: { $min: "$Num_Guests" },
```

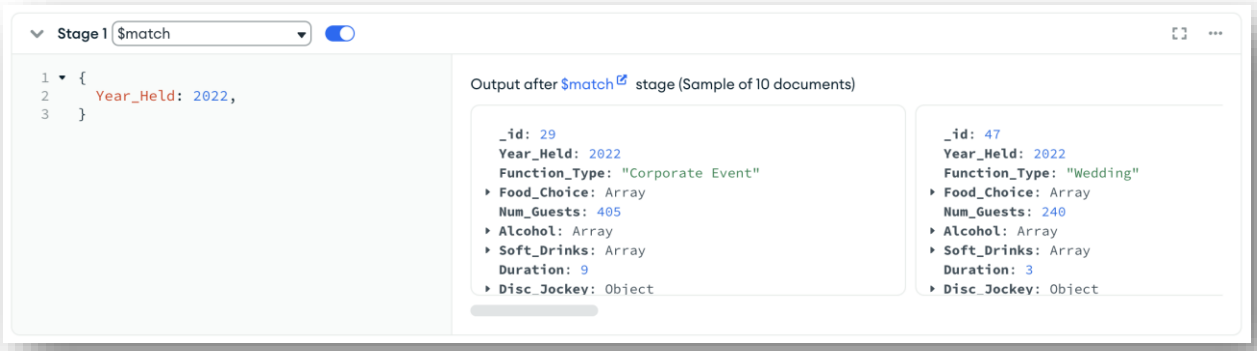
```
      maxGuests: { $max: "$Num_Guests" },
      mostPopularFood: { $max: "$Food_Choice" },
      leastPopularFood: { $min: "$Food_Choice" },
      mostPopularAlcohol: { $max: "$Alcohol" },
      leastPopularAlcohol: { $min: "$Alcohol" },
      mostPopularSoftDrink: { $max: "$Soft_Drinks" },
      leastPopularSoftDrink: { $min: "$Soft_Drinks" },
      avgDuration: { $avg: "$Duration" },
      totalCost: {
       $sum: {
        $add: ["$Disc_Jockey.Fee", 1000, { $cond: [{ $eq: ["$Venue.Type",
              "Sports Venue"] }, 2000, 0] }]
       }
      },
      goodAcousticsVenues: { $addToSet: { $cond: [{ $eq:
                ["$Venue.Acoustics", "Good"] }, "$Venue.Name", null] } }
     }
    },
    {
     $sort: {
      "_id.year": -1,
      "_id.functionType": 1
     }
    }
])
```

1. Paste the above complex aggregate query within the **db.Function.aggregate([ ])** on the text editor in **Aggregations** section of MongoDB Compass User Interface.

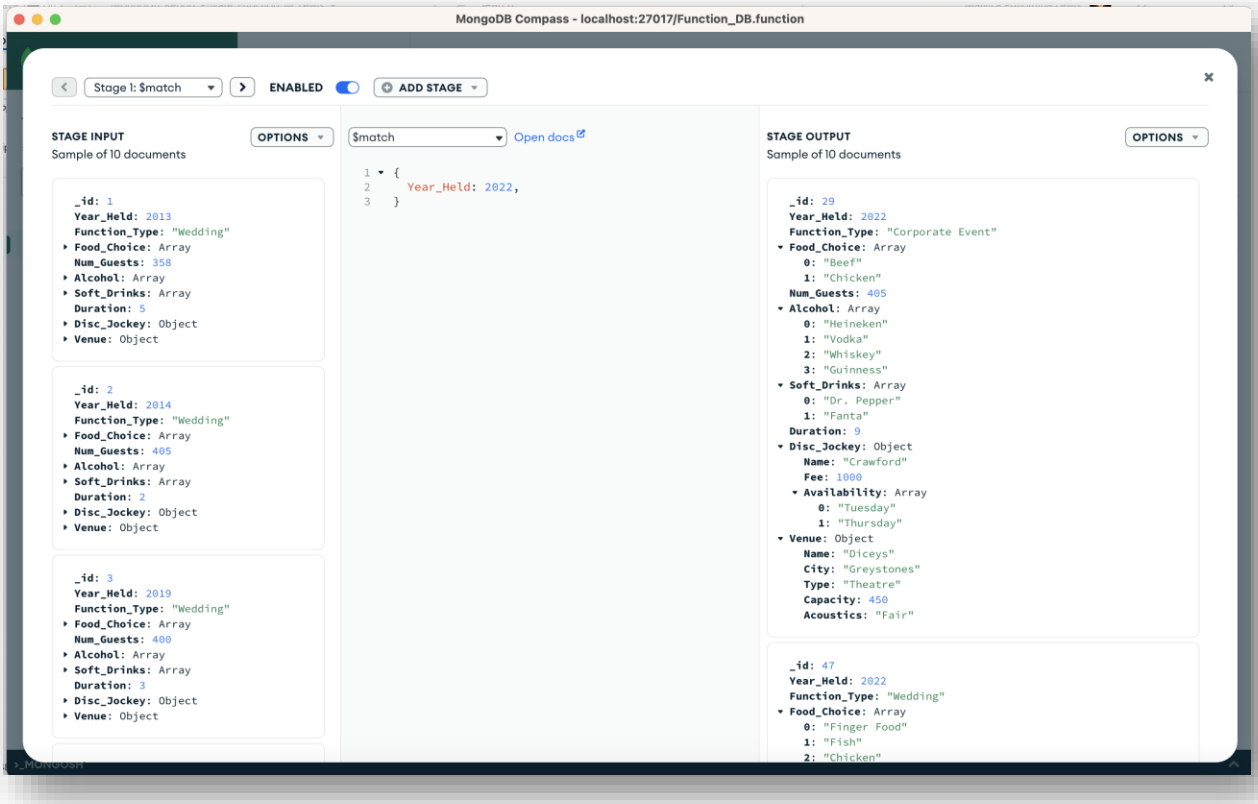


2. Click on the **{} STAGES** button to view the various stages of the pipeline and their individual outputs.

Stage 1: $match



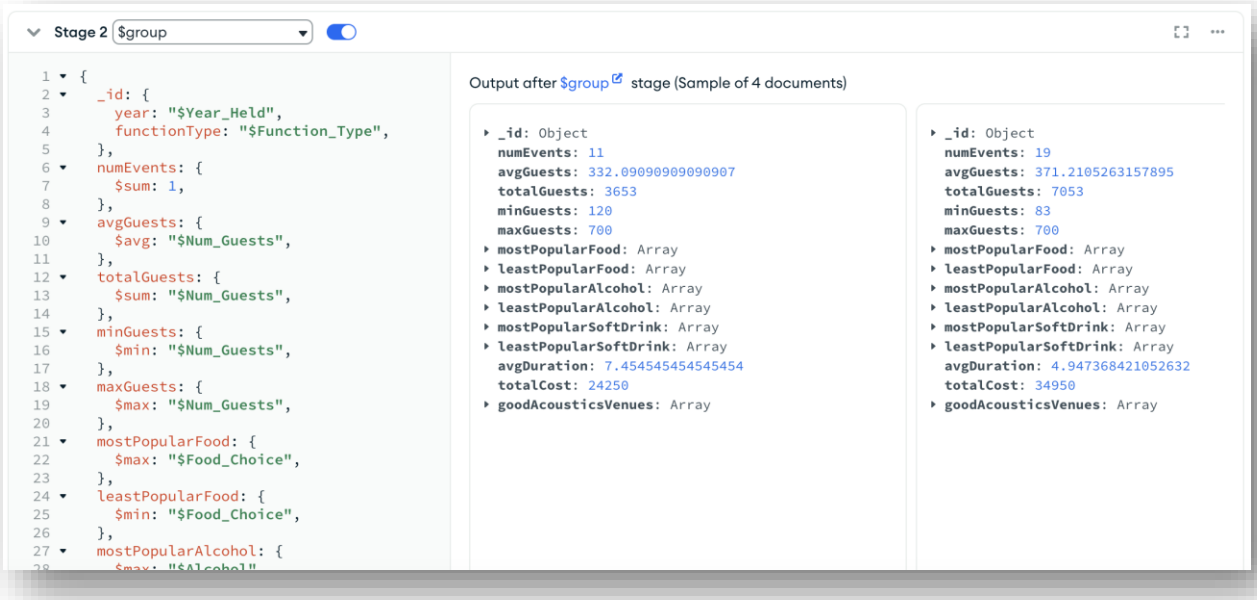
$match: This stage filters the records to only include those with a **Year_Held** field value of **2022**.



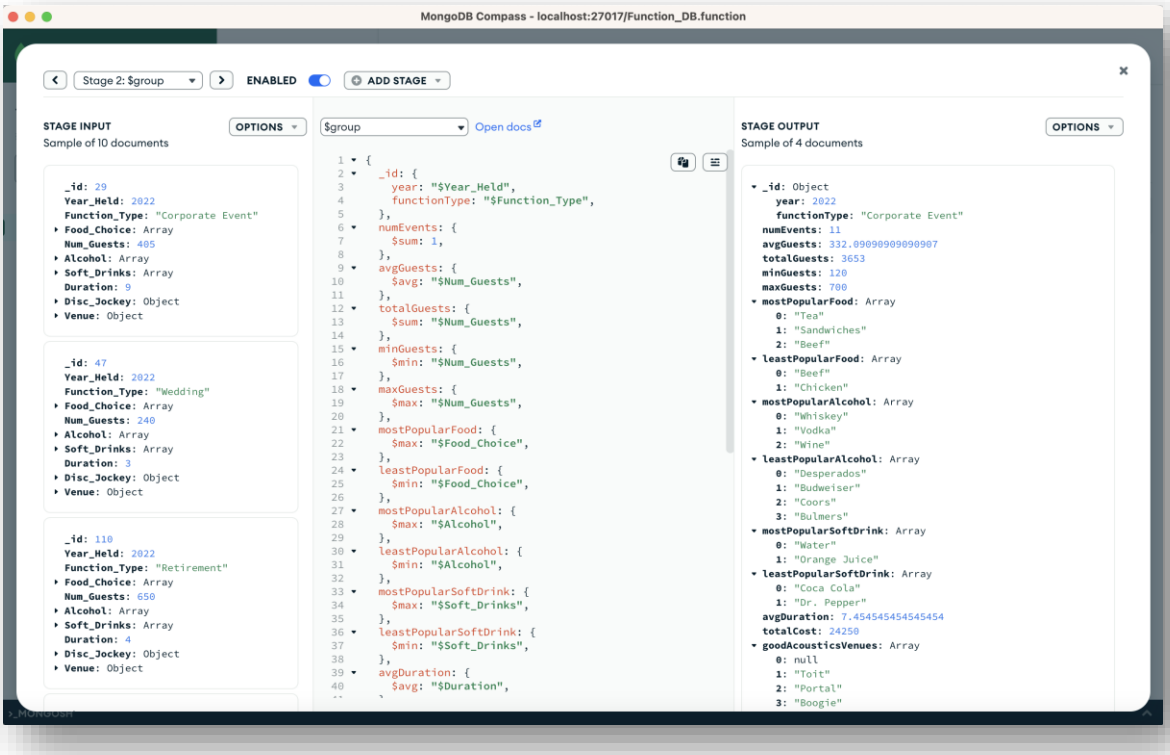
Stage input: The input to this stage is the entire **Function** collection.

Stage output: The output is a subset of the documents where the **Year_Held** field is equal to **2022**.

Stage 2: $group



$group: This stage groups the filtered records by **Year_Held** and **Function_Type** fields, and calculates various statistics for each group.



Stage input: The input to this stage takes the documents from the previous **$match** stage as input and groups them based on the specified criteria. The **_id** field specifies the grouping criteria, and, in this case, the documents are grouped by the **Year_Held** field and **Function_Type** field. The stage output contains a set of documents with unique combinations of **Year_Held** and **Function_Type** fields.

For each group, the $group stage calculates the following aggregation expressions:

- **numEvents**: the number of events in the group, which is calculated using the $sum accumulator.
- **avgGuests**: the average number of guests per event, which is calculated using the **$avg** accumulator.
- **totalGuests**: the total number of guests for all events in the group, which is calculated using the **$sum** accumulator.
- **minGuests**: the minimum number of guests among all events in the group, which is calculated using the **$min** accumulator.
- **maxGuests**: the maximum number of guests among all events in the group, which is calculated using the **$max** accumulator.
- **mostPopularFood**: the most popular food choice among all events in the group, which is calculated using the **$max** accumulator.
- **leastPopularFood**: the least popular food choice among all events in the group, which is calculated using the **$min** accumulator.
- **mostPopularAlcohol**: the most popular alcoholic drink among all events in the group, which is calculated using the **$max** accumulator.
- **leastPopularAlcohol**: the least popular alcoholic drink among all events in the group, which is calculated using the **$min** accumulator.
- **mostPopularSoftDrink**: the most popular soft drink among all events in the group, which is calculated using the **$max** accumulator.
- **leastPopularSoftDrink**: the least popular soft drink among all events in the group, which is calculated using the **$min** accumulator.
- **avgDuration**: the average duration of events in the group, which is calculated using the **$avg** accumulator.
- **totalCost**: the total cost of disc jockeys and venues for all events in the group, which is calculated using the **$sum** accumulator with some conditional logic.
- **goodAcousticsVenues**: a list of names of all venues with good acoustics in the group, which is created using the **$addToSet** accumulator with some conditional logic.
    - The **$addToSet** operator is used in the **$group** stage of the above query to create an array of unique values of a specific field. In this case, the field being added to the set is the name of venues with good acoustics. The **$cond**

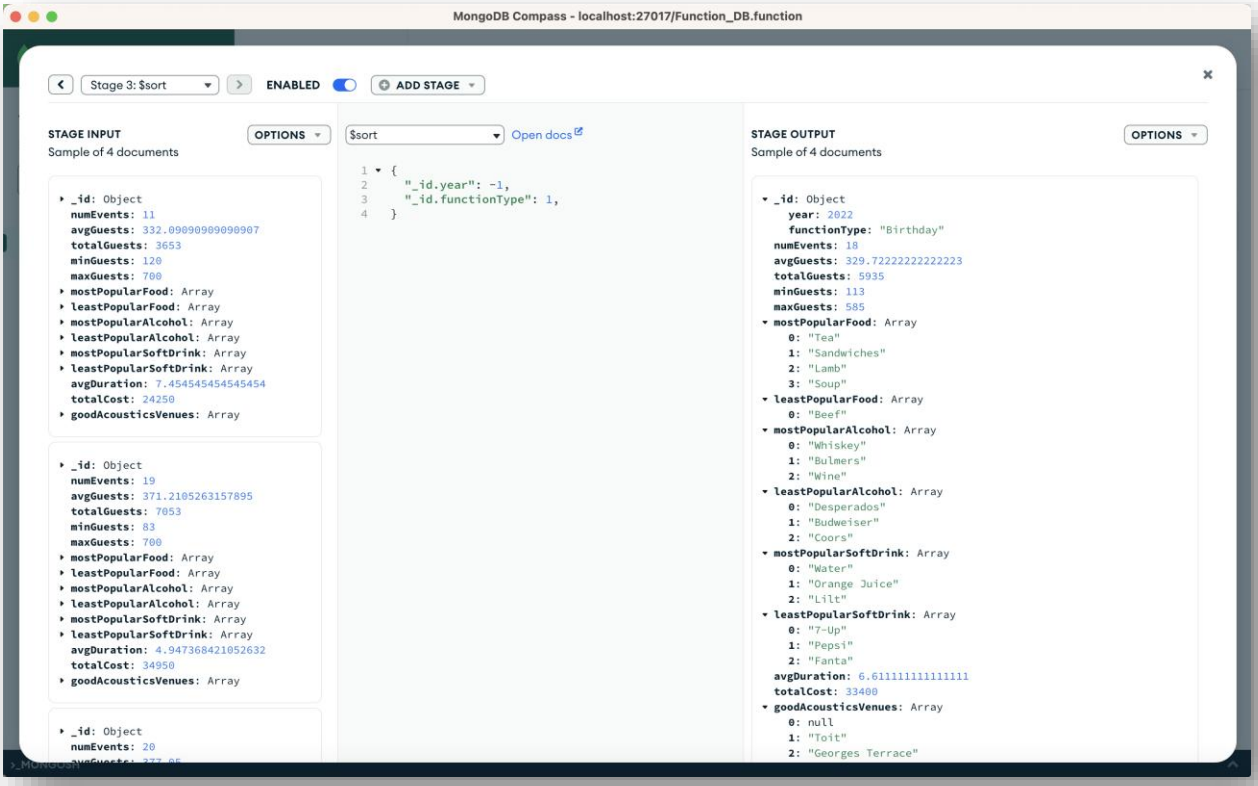
operator is used to evaluate a conditional expression that checks if the value of the "**Acoustics**" field in the "**Venue**" subdocument is "**Good**". If the condition is true, then the name of the venue is added to the set. If the condition is false, then null is added to the set.

- Using **$addToSet** ensures that there are no duplicate values in the resulting array. If the **$push** operator was used instead, the array could contain duplicates, which is not desirable in this case.

Stage output: The output is a set of documents with the grouped **_id** field and the aggregation expressions for each group as new fields.

Stage 3: $sort

$sort: This stage sorts the output of the previous stage by **Year_Held** in descending order and **Function_Type** in ascending order.



Stage input: The input to this stage is the output of the previous **$group** stage and sorts the resulting documents based on the specified sorting criteria.

The stage input for **$sort** is the output of the **$group** stage, which is a set of documents with fields **_id**, **numEvents**, **avgGuests**, **totalGuests**, **minGuests**, **maxGuests**, **mostPopularFood**, **leastPopularFood**, **mostPopularAlcohol**, **leastPopularAlcohol**, **mostPopularSoftDrink**, **leastPopularSoftDrink**, **avgDuration**, **totalCost**, and **goodAcousticsVenues**.

Stage output: The output is the same set of documents, but in a sorted order according to the specified criteria. In this case, the documents are first sorted by the **_id.year** field in descending order, then by the **_id.functionType** field in ascending order. The output of this stage is the final result of the aggregation pipeline.

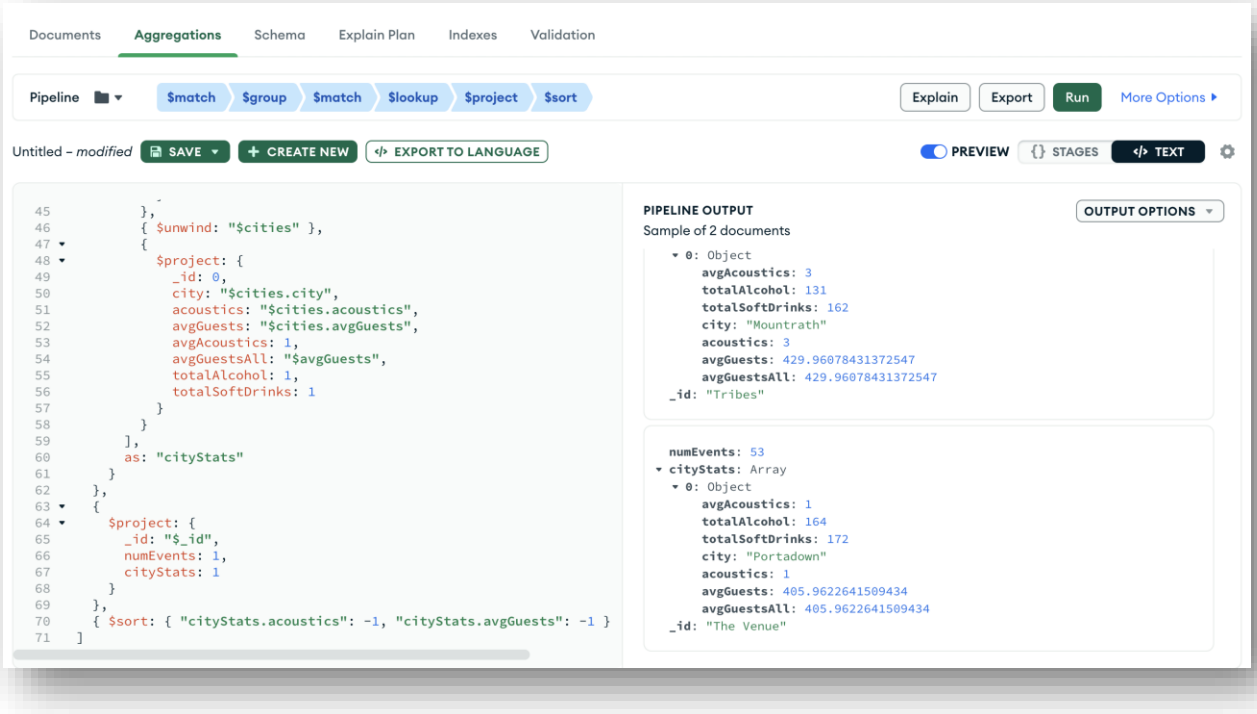
Query 2:

What are the venues with good acoustics that have hosted at least 50 events, and what are the average number of guests and acoustic ratings for each city in which they are located? Also, how much alcohol and soft drinks have been served at these events in each city?

```
db.Function.aggregate([
 { $match: { "Venue.Name": { $exists: true } } },
 { $group: { _id: "$Venue.Name", numEvents: { $sum: 1 } } },
 { $match: { numEvents: { $gte: 50 } } },
 {
  $lookup: {
   from: "Function",
   let: { venueName: "$_id" },
   pipeline: [
    { $match: { $expr: { $eq: [ "$Venue.Name", "$$venueName" ] } } },
    {
     $group: {
      _id: "$Venue.City",
      acoustics: {
       $avg: {
        $switch: {
         branches: [
          { case: { $eq: [ "$Venue.Acoustics", "Poor" ] }, then: 1 },
          { case: { $eq: [ "$Venue.Acoustics", "Fair" ] }, then: 2 },
          { case: { $eq: [ "$Venue.Acoustics", "Good" ] }, then: 3 }
         ],
         default: 0
        }
       }
      },
      avgGuests: { $avg: "$Num_Guests" },
      totalAlcohol: { $sum: { $size: "$Alcohol" } },
      totalSoftDrinks: { $sum: { $size: "$Soft_Drinks" } }
     }
    },
    {
     $group: {
      _id: null,
      cities: {
       $push: {
```

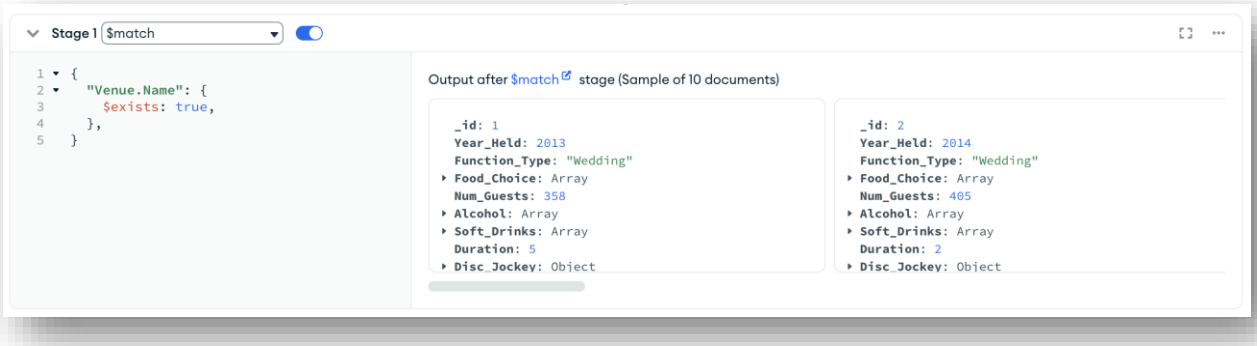
```
            city: "$_id",
            acoustics: "$acoustics",
            avgGuests: "$avgGuests"
          }
        },
        avgAcoustics: { $avg: "$acoustics" },
        avgGuests: { $avg: "$avgGuests" },
        totalAlcohol: { $sum: "$totalAlcohol" },
        totalSoftDrinks: { $sum: "$totalSoftDrinks" }
      }
    },
    { $unwind: "$cities" },
    {
     $project: {
      _id: 0,
      city: "$cities.city",
      acoustics: "$cities.acoustics",
      avgGuests: "$cities.avgGuests",
      avgAcoustics: 1,
      avgGuestsAll: "$avgGuests",
      totalAlcohol: 1,
      totalSoftDrinks: 1
     }
    }
   ],
   as: "cityStats"
  }
 },
 {
  $project: {
   _id: "$_id",
   numEvents: 1,
   cityStats: 1
   }
 },
 { $sort: { "cityStats.acoustics": -1, "cityStats.avgGuests": -1 } }
])
```

1. Paste the above complex aggregate query within the **db.Function.aggregate([ ])** on the text editor in **Aggregations** section of MongoDB Compass User Interface.

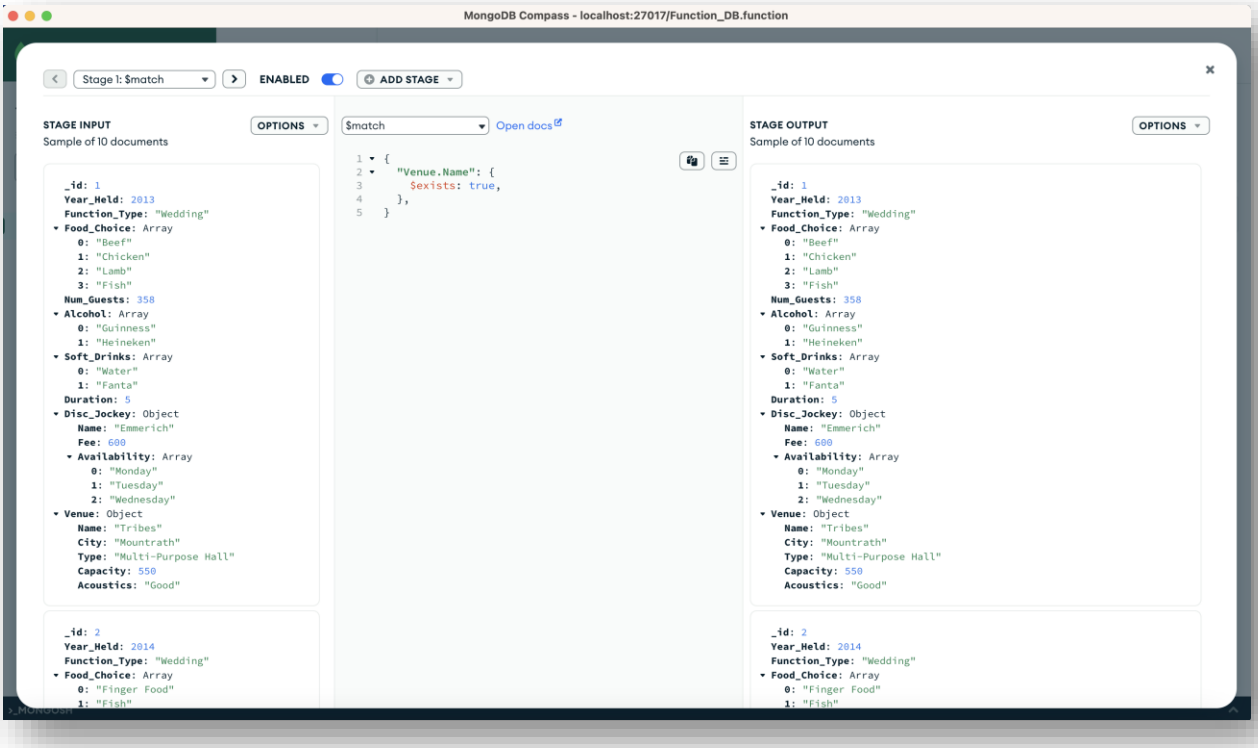


2. Click on the **{} STAGES** button to view the various stages of the pipeline and their individual outputs.

Stage 1: $match



$match: This filters the documents in the collection where the **Venue.Name** field exists.

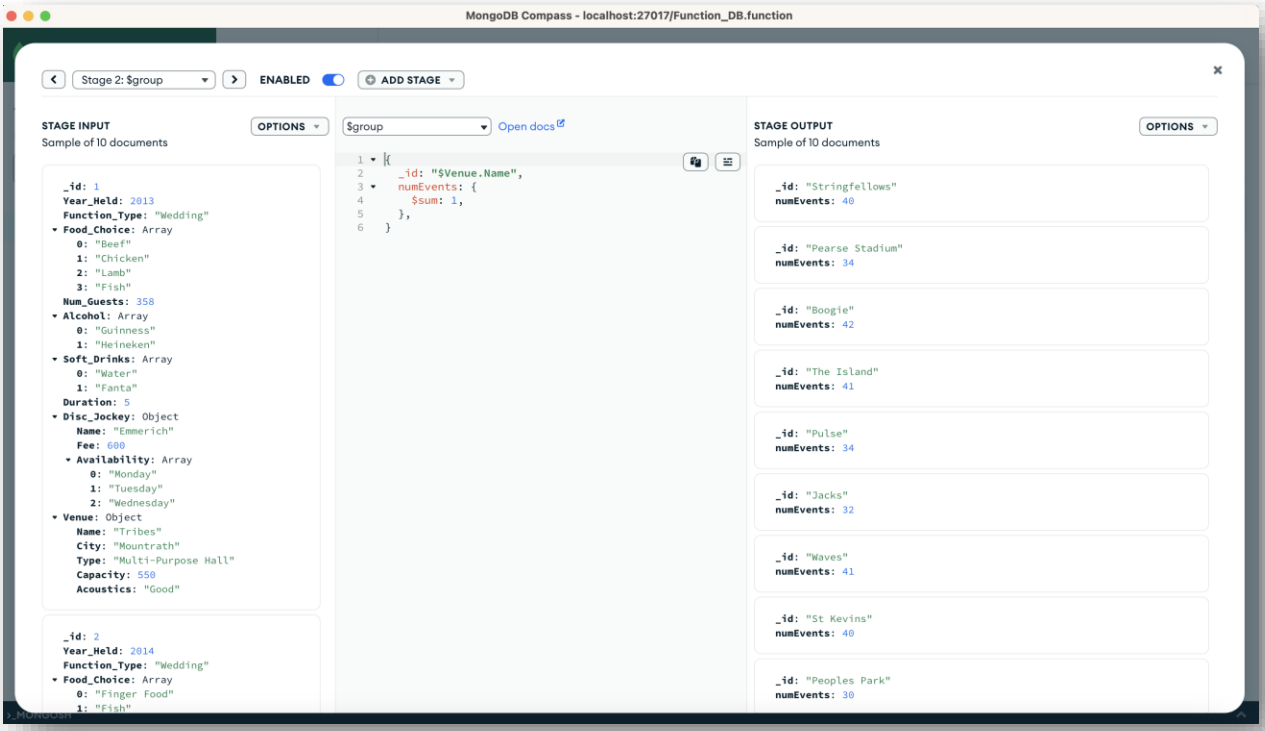


Stage input: This stage filters the documents in the input collection by matching only the documents where the **Venue.Name** field exists. This means that only the documents that have a non-null **Venue.Name** field will pass through this stage to the next stage of the pipeline.

Stage output: The output retains the same document structure as the input, with the documents that did not match the filter condition being excluded.

Stage 2: $group

$group: This groups the documents by the **Venue.Name** field and calculates the total number of events held at each venue using the **$sum** operator. The **_id** field in the resulting documents is set to the **Venue.Name**.

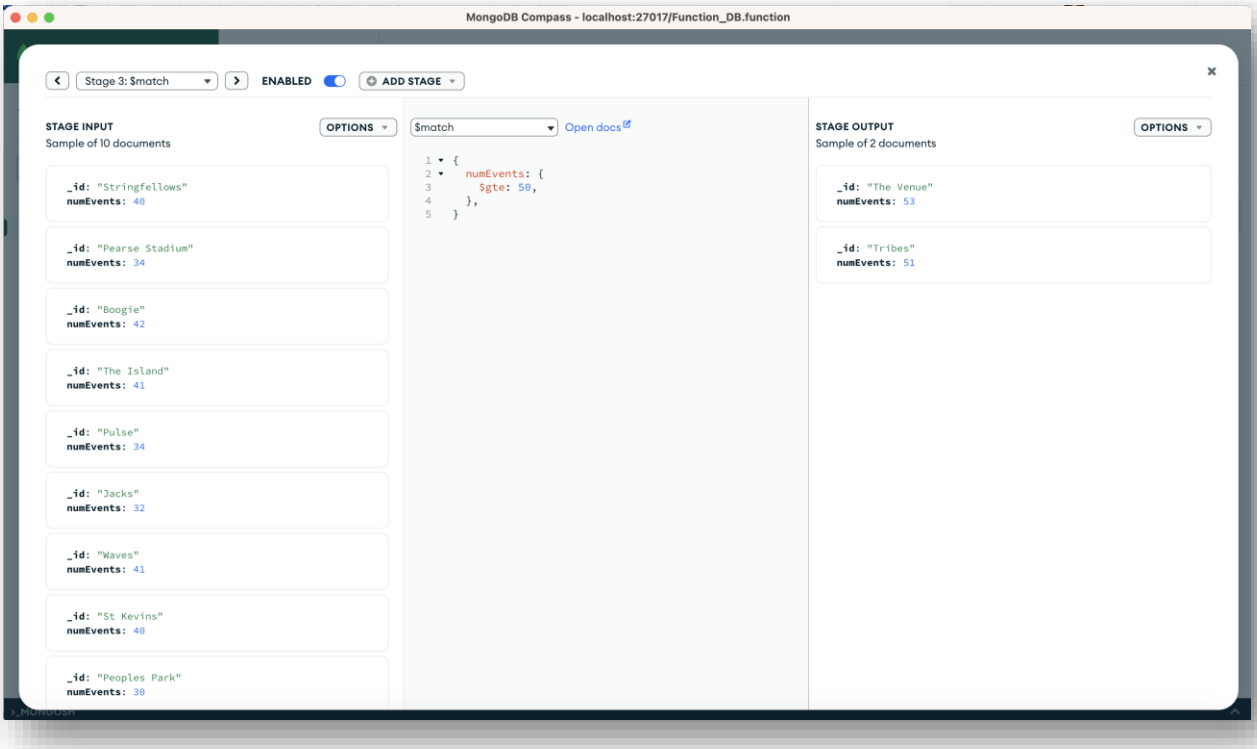


Stage input: The input to the **$group** stage is the output from the previous **$match** stage, which is a set of documents that contain the **Venue.Name** field.

Stage output: The output of the **$group** stage is a set of documents that contain two fields: **_id** and **numEvents**. The **_id** field contains the name of the venue, while the **numEvents** field contains the number of events that occurred at that venue.

Stage 3: $match

$match: This filters out the documents where the **numEvents** field is less than **50**.

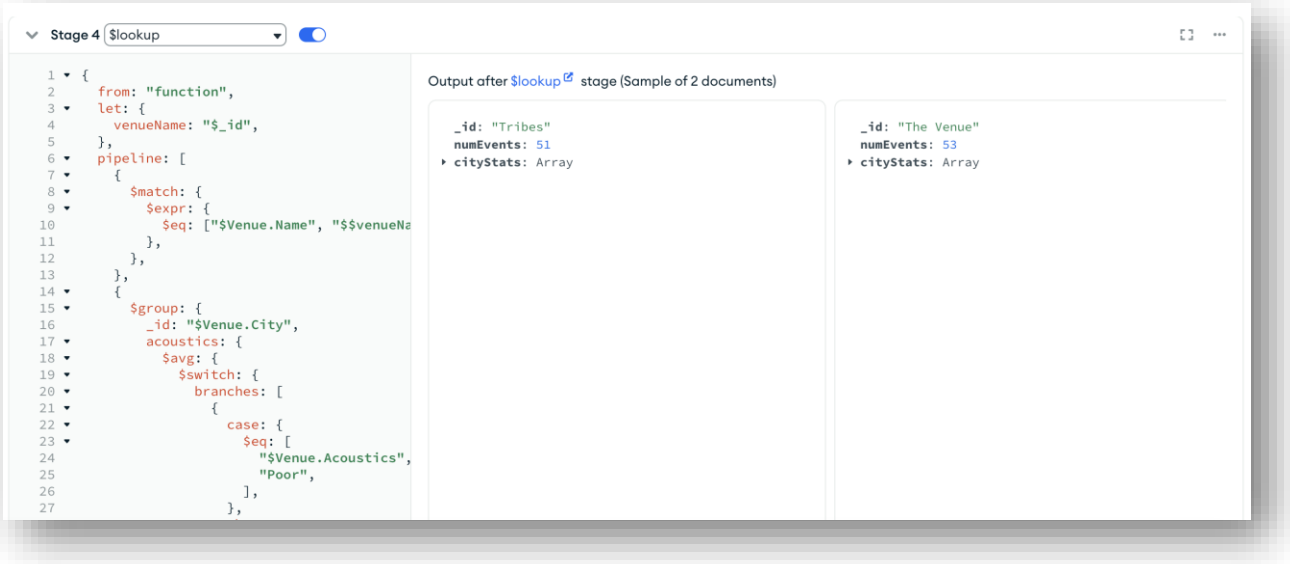


Stage input: The input to the third stage **$match** is the output from the second stage **$group**. The second stage **$group** returns documents that contain the **_id** field with the unique venue names and the **numEvents** field with the total number of events held at each venue. The third stage **$match** filters this output and retains only the documents where the **numEvents** field is greater than or equal to **50**.

Therefore, the input to the third stage **$match** is a set of documents containing **_id** field with unique venue names and **numEvents** field with the total number of events held at each venue.

Stage output: The output of the third stage **$match** is a set of documents that satisfy the filter criteria, i.e., **numEvents** field greater than or equal to **50**.

Stage 4: $lookup

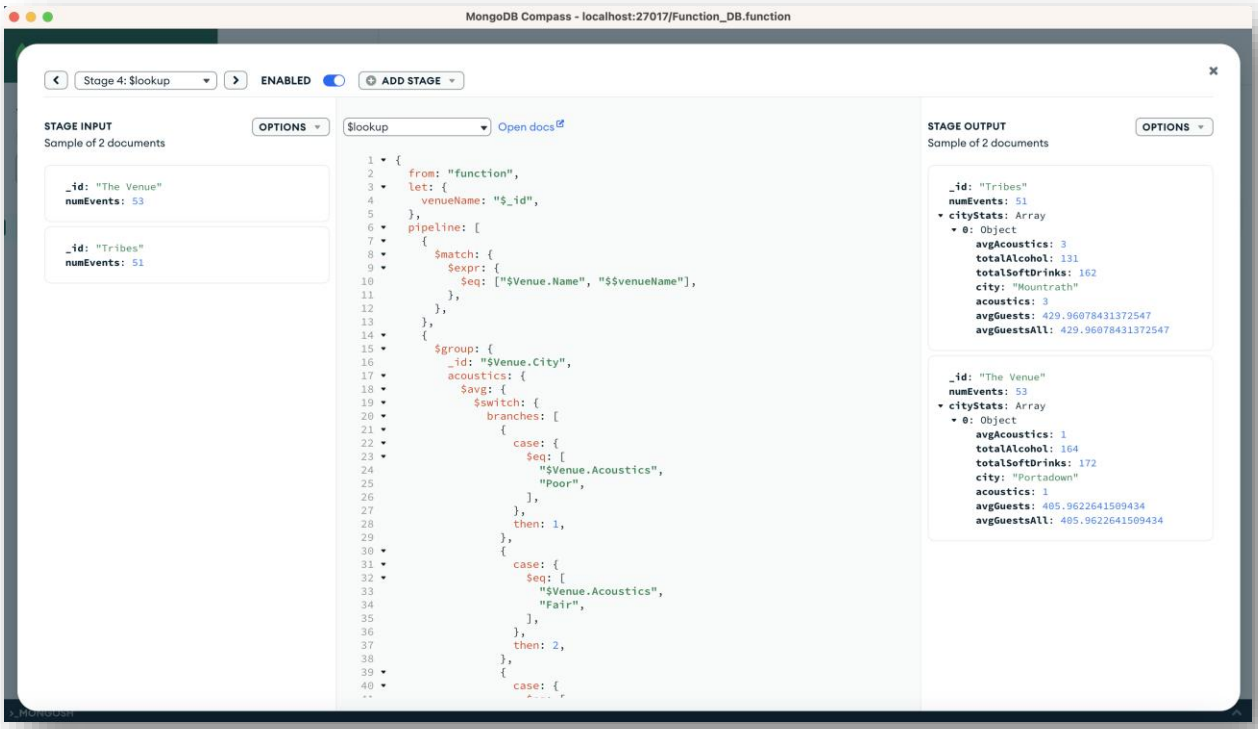


$lookup: This performs a left outer join with the same collection **Function**. The pipeline parameter specifies another set of stages to be performed on the matching documents.

Within the $lookup stage consist of the following functions:

- **$match**: This filters the matching documents by matching the **Venue.Name** field with the value of the **venueName** variable.
- **$group**: This groups the matching documents by the **Venue.City** field and calculates the average acoustics rating of the venue, average number of guests, total alcohol served, and total soft drinks served.
- **$group**: This groups the documents again to calculate the overall average acoustics rating, average number of guests, total alcohol served, and total soft drinks served for all cities. It also creates an array of documents that contains the city name, acoustics rating, and average number of guests.
- **$unwind**: This flattens the array of documents created in the previous stage.

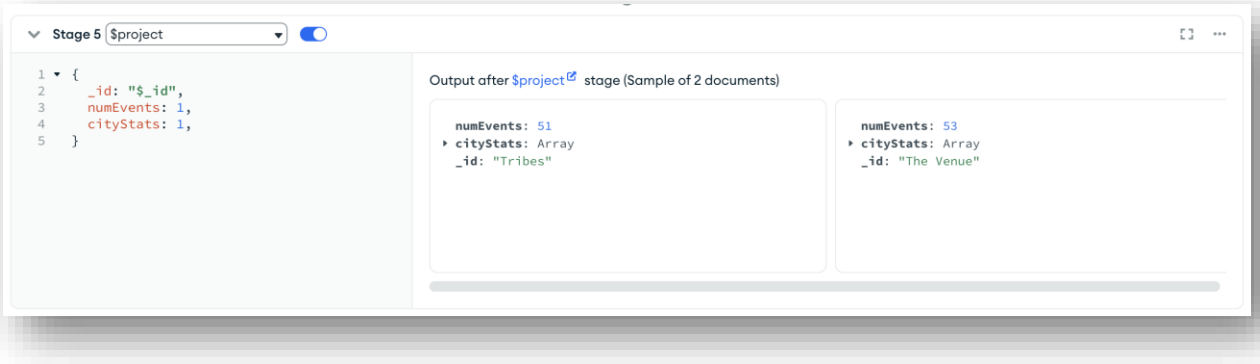
- **$project**: This includes/excludes the specified fields in the documents and also renames the fields. The resulting documents will contain fields such as city, acoustics, **avgGuests**, **avgAcoustics**, **avgGuestsAll**, **totalAlcohol**, and **totalSoftDrinks**.



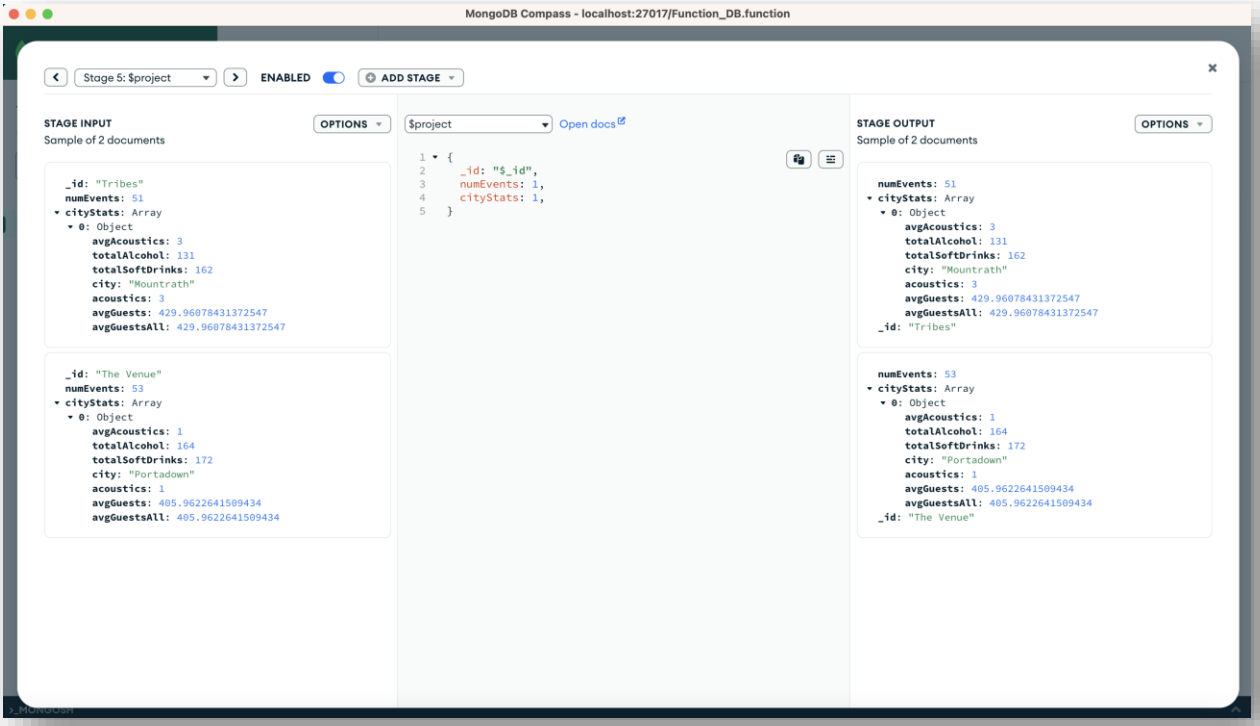
Stage input: The output of the previous pipeline stage is used as input for the **$lookup** stage. It consists of documents that have been grouped by **Venue.Name** and filtered to include only those documents that have a **numEvents** value greater than or equal to **50**.

Stage output: The output of the **$lookup** stage is a new document that combines information from the original document with information from the **Function** collection using the **Venue.Name** field. Each document in the output contains an array called **cityStats** that contains information about the **Venue.City** field in the **Function** collection. This stage adds the **cityStats** field to each document in the input.

Stage 5: $project



$project: This includes/excludes the specified fields in the documents and renames the fields. The resulting documents will contain fields such as **_id**, **numEvents**, and **cityStats**.



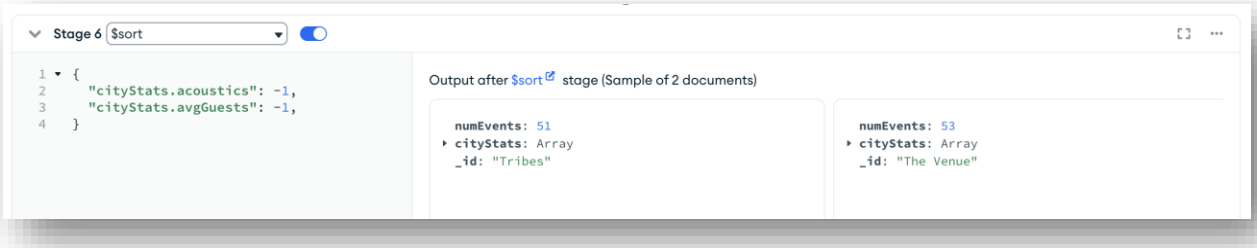
Stage input: The input to this stage is the output of the previous stage, which includes the following fields for each document: **_id**, **numEvents**, **cityStats**.

Stage output: The output of this stage is a modified document with the following fields:

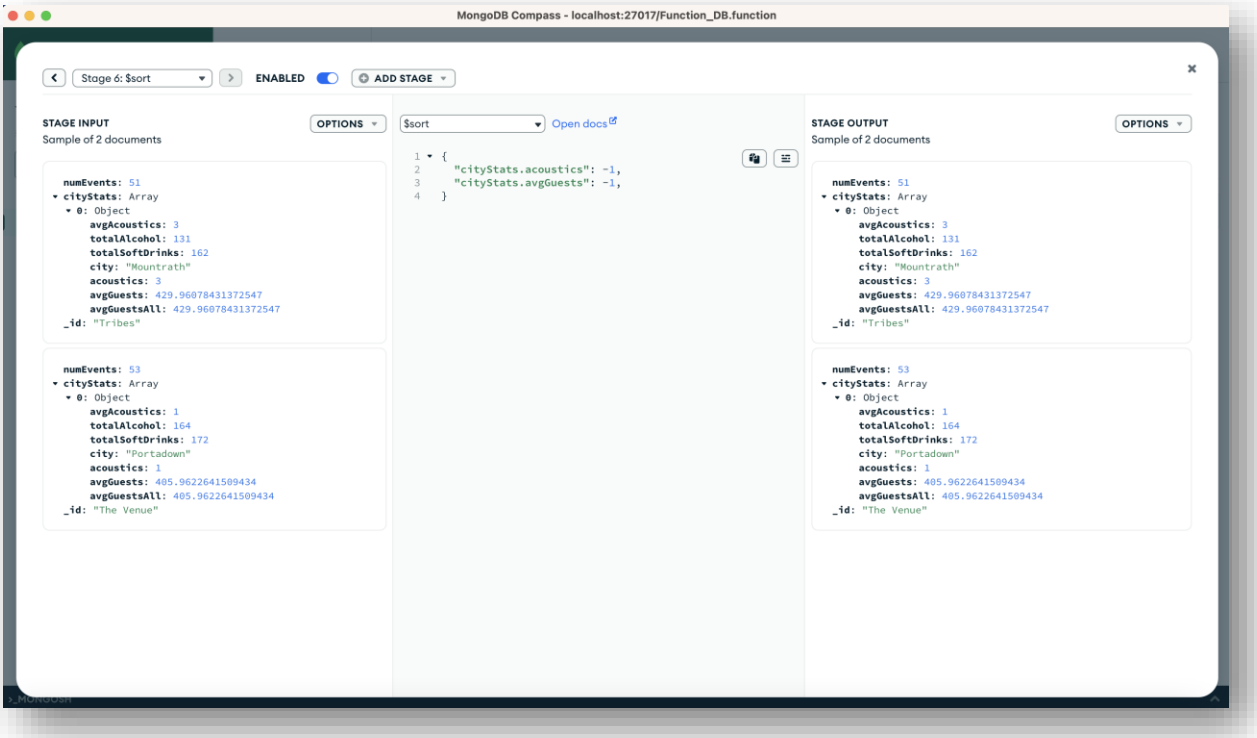
- **_id:** The **_id** field is set to the original **_id** value.
- **numEvents:** The **numEvents** field is set to the original **numEvents** value.
- **cityStats:** The **cityStats** field is set to the original **cityStats** value.

Since the **$project** stage does not add or remove any fields, the input and output documents are identical. The purpose of this stage is simply to rename the **_id** field to match the original document and to explicitly include the fields that we want in the output.

Stage 6: $sort



$sort: This sorts the resulting documents in descending order of the acoustics and **avgGuests** fields in the **cityStats** array.



Stage input: The input to the last **$sort** stage is the output of the previous **$project** stage, which includes the following fields:

- **_id**: the venue name
- **numEvents**: the total number of events held at the venue
- **cityStats**: an array of objects with statistics for each city where the venue is located, including the following fields:
- **city**: the name of the city
- **acoustics**: the average acoustics rating for events held at the venue in that city
- **avgGuests**: the average number of guests at events held at the venue in that city
- **avgAcoustics**: the overall average acoustics rating for events held at the venue in all cities
- **avgGuestsAll**: the overall average number of guests at events held at the venue in all cities
- **totalAlcohol**: the total number of alcohol servings at events held at the venue in that city
- **totalSoftDrinks**: the total number of soft drink servings at events held at the venue in that city

Stage output: The output of this stage is the same as the input, but with the **cityStats** array sorted in descending order by acoustics and then by **avgGuests**. This will result in a list of venues sorted by the cities with the best acoustics and the highest average number of guests.

YouTube Link

Running an aggregate query in Mongosh using MongoDB

https://youtu.be/NrbVi8fTVpU

Using MongoDB Compass to run a complex aggregate query in MongoDB

https://youtu.be/GBNCGTnEV0Q

References

Official MongoDB Documentation:

https://docs.mongodb.com/

MongoDB University:

https://university.mongodb.com/

MongoDB Compass Documentation:

https://docs.mongodb.com/compass/

MongoDB Blog:

https://www.mongodb.com/blog

MongoDB GitHub Repository:

https://github.com/mongodb/mongo