Mongo query exercises for shell

Install MongoDB local database, shell or Compass

Use ‘mongosh’ tool from MongoDB, install the entire community edition (which includes the ‘mongosh’ tool) or the Compass app

1. Windows installation
    a. Download the ‘mongosh’ tool from MongoDB or the MongoDB local community edition database or the Compass application
    b. If you just installed the shell tool, you need to add ‘mongosh.exe’ to the Path in the system environment variables
2. Set up a database called ‘marvel-db’ on Atlas
3. Connect to the cluster and database

Seed the database

```
db.characters.insertMany([
  {
    name: "Captain America",
    real_name: "Steve Rogers",
    abilities: ["Super strength", "Enhanced agility", "Vibranium shield"],
    team: "Avengers",
    power_rating: 8,
    status: "Active"
  },
  {
    name: "Iron Man",
    real_name: "Tony Stark",
    abilities: ["Genius-level intellect", "Powered armor suit", "Flight"],
    team: "Avengers",
    power_rating: 9,
    status: "Inactive"
  },
  {
    name: "Thor",
    real_name: "Thor Odinson",
    abilities: ["Thunder god", "Superhuman strength", "Flight"],
    team: "Avengers",
```

```
    power_rating: 10,
    status: "Active"
  },
  {
    name: "Spider-Man",
    real_name: "Peter Parker",
    abilities: ["Wall-crawling", "Spider sense", "Superhuman agility"],
    team: "Avengers",
    power_rating: 7,
    status: "Active"
  },
  {
    name: "Jessica Jones",
    real_name: "Jessica Campbell Jones",
    abilities: ["Superhuman strength", "Flight", "Accelerated healing"],
    team: "Defenders",
    power_rating: 6,
    status: "Active"
  },
  {
    name: "Moon Knight",
    real_name: "Marc Spector",
    abilities: ["Expert combatant", "High pain tolerance", "Multiple personalities"],
    team: "West Coast Avengers",
    power_rating: 6,
    status: "Active"
  }
])
```

The exercise

1. Insert a new character into the characters collection:

```
db.characters.insertOne({
  name: "Captain America",
  real_name: "Steve Rogers",
  abilities: ["Super strength", "Enhanced agility",
"Vibranium shield"],
  team: "Avengers"
```

```
})
```

2. Insert multiple characters at once:

```
db.characters.insertMany([
  {
    name: "Jessica Jones",
    real_name: "Jessica Campbell Jones",
    abilities: ["Superhuman strength", "Flight",
"Accelerated healing"],
     team: "Defenders",
     power_rating: 3
  },
  {
    name: "Moon Knight",
    real_name: "Marc Spector",
    abilities: ["Expert combatant", "High pain
tolerance", "Multiple personalities"],
     team: "West Coast Avengers",
     power_rating: 3
  }
])
```

3. Find all characters in the collection:

```
db.characters.find()
```

4. Find a specific character by name:

```
db.characters.findOne({ name: "Spider-Man" })
```

5. Find all Avengers:

```
db.characters.find({ team: "Avengers" })
```

The $set operator is one of the most commonly used update operators in MongoDB. It's used to set the value of a field in a document. If the field doesn't exist, $set will create the field with the specified value.
Example:

```
db.collection.updateOne(
{ filter },
{ $set: { field1: value1, field2: value2, ... }})
```

6. Update multiple characters' team:

```
db.characters.updateMany(
  { team: "Avengers" },
  { $set: { status: "Active" } }
)
```

7. Update multiple characters, add "power_rating" field to all characters and set it to 5:
   Use: updateMany, $set

   The $inc operator is another commonly used update operator in MongoDB. It's used to increment the value of a numeric field in a document. If the field doesn't exist, $inc will create the field and set its value to the specified increment amount.
   Example:
   db.characters.updateOne(
   { name: "Hulk" },
   { $inc: { power_rating: 2 }})

8. Update multiple characters, increase the power_rating of every Avengers by 2:
   Use: updateMany, $inc

   The $push operator is a commonly used update operator in MongoDB. It's used to add an element to an array field in a document. If the field doesn't exist, $push will create the field as an array with the specified element as its only member.
   Example:
   db.collection.updateOne(

```
{ filter },
{ $push: { field: value } })
```

9. Update one character's abilities, and push a new value to the abilities array:

```
db.characters.updateOne(
  { name: "Iron Man" },
  { $push: { abilities: "Nanotech suit" } }
)
```

The $each operator is a modifier used in conjunction with array update operators like $push, $addToSet, and $pull in MongoDB. It allows you to add multiple elements to an array in a single operation.
Example:

```
db.collection.updateOne(
{ filter },
{ $push: { field: { $each: [value1, value2, ...] }
}})
```

10. Update “Spider-Man” and add multiple abilities to his abilities array:

Use: updateOne, $push, $each

The $pull operator is an update operator in MongoDB used to remove elements from an array field in a document. It removes all instances of a specified value or values that match a specified condition from an existing array.
Example:

```
db.collection.updateOne(
{ filter },
{ $pull: { field: value_or_condition }})
```

11. Update “Thor” by Removing an ability from his abilities array:

Use: updateOne, pull

The $addToSet operator is an update operator in MongoDB used to add elements to an array field in a document, but only if the elements don't already exist in the array. It ensures that an array contains only unique values.
Example:

```
db.collection.updateOne(
{ filter },
{ $addToSet: { field: value }})
```

12. Add a unique ability to Black Widow's abilities array (only if it doesn't already exist):

    Use: updateOne, $addToSet

13. Delete one character:

    db.characters.deleteOne({ name: "Wolverine" })

14. Delete a character by their real name:

    Use: deleteOne

15. Delete a character by its ID:
    Use: deleteOne, _id: ObjectId("60b98...")

The $size operator is a query operator in MongoDB used to select documents based on the number of elements in an array field. It allows you to match documents where an array field has a specific number of elements.
Example:
db.collection.find({ field: { $size: number } })

The $gt operator is a comparison query operator in MongoDB used to select documents where the value of a field is greater than (>) a specified value. It's part of MongoDB's set of comparison operators.
Example:
db.collection.find({ field: { $gt: value } })

The $expr operator in MongoDB is a tool that lets you create more flexible queries. It's especially useful when you want to compare different fields within the same document or use special calculation operators in your search.
Example:
db.characters.find({ $expr: { $gt: ["$power_rating", "$age"] } })

16. Delete the first character found with more than 3 abilities:

    { $expr: { $gt: [{ $size: "$array_name" }, 3] } }
    Use: deleteOne, $expr, $gt, $size

17. Delete multiple characters:

    db.characters.deleteMany({ team: "X-Men" })

The $ne operator in MongoDB stands for "not equal." It's used to find documents where a field's value is not equal to a specified value.
Example:
db.collection.find({ field: { $ne: value } })

18. Delete all inactive characters:
Use deleteMany, $ne

The /^value/ pattern is a regular expression used in MongoDB queries. It's specifically used to match strings that start with a certain value.

19. Delete all characters whose names start with "Captain":

Use deleteMany, name: /^Captain/

20. Find characters with super strength:

db.characters.find({ abilities: "Super strength" })

The $in operator is used in MongoDB queries to select documents where the value of a field equals any value in a specified array
Example:
{ field: { $in: [value1, value2, ...] } }

21. Find characters with specific abilities using $in operator:

```
db.characters.find({ abilities: { $in: ["Healing factor", "Expert spy"] } })
```

22. Find characters who are members of "Avengers", "X-Men", "Fantastic Four":

Use: find, $in

23. Find characters with specific power_rating:

Use: find, $in

The $nin operator is used in MongoDB queries to select documents where the value of a field is not in a specified array or where the field does not exist.
Example:
```
{ field: { $nin: [value1, value2, ... ] } }
```

24. Find characters who are not members of "Avengers", "X-Men":

Use: find, $nin

25. Find characters with power_rating not in a specific range:

    Use: find, $nin

    The $all operator in MongoDB is used to find documents where an array field contains all the specified elements, regardless of their order or whether other elements are present.
    Example:
    db.collection.find({ arrayField: { $all: [value1, value2, ...] } })

26. Find characters with multiple specific abilities:

    db.characters.find({ abilities: { $all: ["Super strength", "Enhanced agility"] } })

27. Find characters with both “Flight” and “Energy projection” abilities:

    Use: find, $all

The $and operator is used in MongoDB queries to combine multiple conditions, all of which must be true for a document to be included in the result set.
Example:
{ $and: [{ condition1 }, { condition2 }, ...] }

db.characters.find({ $and: [{ team: "Avengers" }, { powerLevel: { $gt: 7 } }] })

28. Find characters with both "Super strength" and "Enhanced agility":

db.characters.find({ $and: [{ abilities: "Super strength" }, { abilities: "Enhanced agility" }] })

29. Find Avengers with a power_rating greater than 8:

Use: find, $and

30. Find characters who have both “Flight” and “Energy projection” abilities, but are not on the Avengers team:

Use: find, $and, $ne

31. Find characters who teamed with Avengers and has more than 3 abilities:

Use: find, $and, $expr, $gt, $size

The $or operator in MongoDB is used to perform a logical OR operation on an array of two or more expressions and select the documents that satisfy at least one of the expressions.

```
db.collection.find({ $or: [
{ condition1 },
{ condition2 }, ...
]})
```

32. Find Avengers with either “Super strength” or “Enhanced durability”:

Use: find, $or

The $elemMatch operator is used to query arrays in MongoDB. It allows you to find documents where an array field contains at least one element that matches all the specified criteria.

Example:

```
db.collection.find({
arrayField: {
$elemMatch: {
condition1,
condition2, ...
}}})
```

33. Find characters with exactly three abilities, one of which must be "Genius-level intellect":

```
db.characters.find({ abilities: {
$size: 3,
$elemMatch: { $eq: "Genius-level intellect" }
}})
```

34. Find characters with at least four abilities, including "Superhuman strength" and "Invulnerability":

Use: find, $size, $gte, $all

35. Find characters with exactly two abilities, neither of which is "Flight":

Use: find, $size, $not, $elemMatch, $eq

The notation abilities.0 is used to refer to the first element of an array field named "abilities" in a MongoDB document. This is part of MongoDB's dot notation for accessing array elements and embedded document fields.

Example:

```
db.collection.find({ "abilities.0": value })
```

36. Find characters with abilities that include "Energy manipulation" as their first or second ability:

    Use: find, $or, abilities.0 abilities.1

37. Find characters whose name starts with "Iron":

    db.characters.find({ name: /^Iron/ })

38. Find characters with names containing "man" (case-insensitive):

    db.characters.find({ name: /man/i })

39. Find characters whose names start with "Spider" (case-insensitive):

    Use: /^value/i

40. Find characters whose names end with "woman" (case-insensitive):

    Use: /value$/i

The countDocuments() method is used to count the number of documents in a collection that match a specific query criteria.
Example:
db.collection.countDocuments(query, options)

41. Count the number of Avengers:

db.characters.countDocuments({ team: "Avengers" })

42. Count the number of characters with super strength:

Use: countDocument

43. Count the number of characters with a power_rating greater than 8:

Use: countDocument, $gte

44. Count the number of active characters who are not on the Avengers team:

Use: countDocument, $ne

The sort() method is used to order the results of a query in ascending or descending order based on one or more fields.
Example:
db.characters.find().sort({ team: 1, power_rating: -1 })

45. Find characters and sort them by name:

db.characters.find().sort({ name: 1 })

46. Find characters and sort them by power_rating in descending order:

Use: sort

47. Find Avengers and sort them by the number of missions completed:

Use: find, sort

48. Find characters and sort them first by team alphabetically, then by name within each team:

Use: find, sort

The limit() method is used to restrict the number of documents returned by a query.
Example:
db.collection.find().limit(number)

49. Find characters and limit the results to 3:

db.characters.find().limit(3)

50. Find the top 5 characters with the highest power_rating:

Use: find, sort (-1), limit

51. Find the first 3 Avengers alphabetically by name:

Use: find, sort, limit

The skip() method is used to skip over a specified number of documents in a query result.
Example:
db.collection.find().skip(number)

52. Find characters and skip the first 2 results:

db.characters.find().skip(2)

53. Find Avengers, skip the first 5, and return the next 3:

Use: find, skip, limit

54. Find characters sorted by power_rating, skipping the top 3:

Use: find, sort, skip

55. Find characters with more than 2 abilities:

```
db.characters.find({ $expr: { $gt: [{ $size: "$abilities" }, 2] } })
```

Projection in MongoDB is a feature that allows you to specify which fields to include or exclude from the documents returned by a query. It's a way to control the shape of the data you get back from the database.
Example:
db.collection.find(query, projection)
db.characters.find({}, { name: 1, team: 1 })

56. Find characters and only return their names:

```
db.characters.find({}, { name: 1, _id: 0 })
```

57. Find characters with a specific ability and only return their names:

```
db.characters.find({ abilities: "Expert spy" }, { name: 1, _id: 0 })
```

58. Find Avengers and return only their names and power_rating:

```
Use: find, { key: 1 }
```

59. Find characters with super strength and return their names and teams:

```
Use: find, { key: 1 }
```

The findOneAndUpdate() method is used to update a single document in a collection based on a specified filter and return either the original document or the updated document.
returnDocument: Specifies whether to return the document before or after the update. Values are "before" (default) or "after".
upsert: If true, creates a new document when no document matches the query criteria. Default is false.
Example:

```
db.characters.findOneAndUpdate(
  { name: "Iron Man" },
  { $set: { powerLevel: 9 } },
  { returnDocument: "after" }
)
```

60. Update a character's team and return the updated document:

```
db.characters.findOneAndUpdate(
  { name: "Spider-Man" },
  { $set: { team: "Fantastic Four" } },
  { returnDocument: "after" }
)
```

61. Increase a character's power_rating and add a new ability:

    Use: findOneAndUpdate, $inc, $push, returnDocument

62. Update a character's status and lastMission date:

    Use: findOneAndUpdate, returnDocument

63. Change a character's team, creating the document if it doesn't exist:

    Use: findOneAndUpdate, returnDocument, upsert

The aggregate() method is used to perform advanced data processing and analysis operations on the documents in a collection. It allows you to perform complex operations that transform and combine documents in various ways.

Example:

```
db.collection.aggregate([{ stage1 }, { stage2 },
... ])
```

How it works:

It takes an array of stages as its argument.

Each stage in the pipeline transforms the documents as they pass through.

The documents that are output from a stage are passed to the next stage.

- $match: Filters the documents (similar to find())
- $group: Groups documents by a specified expression
- $sort: Sorts the documents
- $project: Reshapes documents (select/rename fields)
- $limit: Limits the number of documents
- $unwind: Deconstructs an array field

64. Find all Avengers and sort them by their real name in ascending order:

```
db.characters.aggregate([
{ $match: { team: 'Avengers' } },
{ $sort: { real_name: 1 } }
])
```

65. Get the first 2 characters when sorted by power_rating descending order:

    Use: $sort, $limit

66. Find characters with 'Super strength' ability and only return their name and team:

    Use: $match, $project

67. Count the total number of characters:

    Use: $count

68. Find characters with more than 2 abilities and sort them by name:

    Use: $match, $expr, $gt, $size, $sort

The $group stage is a powerful tool in MongoDB's aggregation framework used to group documents by a specified expression and can perform various accumulations on grouped data.

Example:

```
{ $group: {
    _id: <expression>, // Group key
    <field1>: { <accumulator1> : <expression1> },
... } }
```

How it works:

The _id field specifies the group key.

Documents with the same group key are grouped together.

Other fields use accumulators to perform calculations on the grouped documents.

Common accumulators:

$sum: Sums up values

$avg: Calculates the average

$first: Gets the first document in each group

$last: Gets the last document in each group

$max: Gets the maximum value

$min: Gets the minimum value

$push: Creates an array of all values

Example:

```
db.characters.aggregate([
{ $group: {
_id: "$team",
memberCount: { $sum: 1 },
averagePower: { $avg: "$power_rating" },
allAbilities: { $push: "$abilities" } }}])
```

69. Find characters with abilities containing the word "super" (case-insensitive) and group them by team:

```
db.characters.aggregate([
  { $match: { abilities: /super/i } },
  { $group: { _id: "$team", characters: { $push: "$name" } } }
])
```

70. Find characters with power rating above 3, group them by team, and calculate average power_rating:

Use: $match, $gt, $group, $avg, $push

71. Group characters by the first letter of their name and count them:

Use: $group, $substr, $push, $sort

72. Find characters with "strength" in their abilities (case-insensitive), group by team, and list their real names:

User: $match, $group, $push

73. Find the average power_rating for each team and sort by highest average:

```
db.characters.aggregate([
  { $group: {
      _id: "$team",
      avg_power_rating: { $avg: "$powers.power_rating" },
      members: { $push: { name: "$name", power_rating: "$powers.power_rating" } }
    }
  },
  { $sort: { avg_power_rating: -1 } }
])
```