CRUD I: QUERYING ON ARRAY FIELDS

Querying for An Entire Array

At this point, we should be familiar with querying in MongoDB using the `.find()` method. Let's take things a step further by learning how we can use this same method to filter documents based on array fields.

Consider a collection called `books` where each document shares a similar structure to the following:

```
{
  _id: ObjectId(...),
  title: "Alice in Wonderland",
  author: "Lewis Carroll",
  year_published: 1865,
  genres: ["childrens", "fantasy", "literary nonsense"]
}
```

Imagine we are looking for a new book to dive into – specifically, one that spans the young adult, fantasy, and adventure genres. We can query the collection for an array containing these exact values by using the `.find()` method and passing in a query argument that includes the field and array we want to match:

```
db.books.find({ genres: ["young adult", "fantasy", "adventure"] })
```

This query would return documents where the `genres` field is an array containing exactly three values, `"young adult"`, `"fantasy"`, and `"adventure"`. For example, we would get a result that might look like this:

```
{
  _id: ObjectId(...),
  title: "Harry Potter and The Sorcerer's Stone",
  author: "JK Rowling",
  year_published: 1997,
  genres: ["young adult", "fantasy", "adventure"]
},
{
  _id: ObjectId(...),
  title: "The Gilded Ones",
  author: "Namina Forna",
  year_published: 2021,
  genres: ["young adult", "fantasy", "adventure"]
}
```

Note that this query would only return documents where the array field contains precisely the values included in the query in the specified order. The following document contains the same values as mentioned in our query, but it wouldn't be matched by our search because these values are in a different order:

```
{
  _id: ObjectId(...),
  title: "Children of Blood and Bone",
  author: "Tomi Adeyemi",
  year_published: 2018,
  genres: ["fantasy", "young adult", "adventure"]
}
```

The following document would also not be matched because it contains an additional value not specified by our query:

```
{
  _id: ObjectId(...),
  title: "Eragon",
  author: "Christopher Paolini,
  year_published: 2002,
  genres: ["young adult", "fantasy", "adventure", "science fiction"]
}
```

Before moving on, let's practice querying array fields for exact matches!

Instructions

1.

We recently upgraded our database. We assigned some restaurants a new field named `michelin_stars` which contains an array of years (e.g., 2019) that they received a Michelin star for their outstanding cuisine.

Connect to the `restaurants` database. Then, query the `listingsAndReviews` collection for all restaurants that earned exactly two `michelin_stars` in the years 2019 and 2020.

Hint

To query a collection for an array containing values for a specific field, you can use the following syntax:

```
db.<collection>.find({ <field>: [<value1>, <value2>, ....] });
```

```
restaurants> db.listingsAndReviews.find({michelin_stars:[2019, 2020]});
[
  {
    _id: ObjectId("62f69e8aa82d46a4c3a255d0"),
    address: { building: '284', street: '3rd Ave', zipcode: '11215' },
    borough: 'Brooklyn',
    cuisine: 'Mexican',
    grades: [
      {
        date: ISODate("2014-12-29T00:00:00.000Z"),
        grade: 'A',
        score: 11
      },
      {
        date: ISODate("2014-06-26T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
```

Matching Multiple Array Elements with $all

So far, we've learned to query an array for exact matches, or individual elements. These are two extremes: searching for a specific ordering of elements, or only matching a single element. MongoDB offers us a middle ground. We can use the `$all` operator to match documents for an array field that includes all the specified elements, without regard for the order of the elements or additional elements in the array.

For example, let's say we've finished our young adult novel and are ready to move on to something that spans the science fiction and adventure genres. We could use the `$all` operator to perform this query, like so:

```
db.books.find({ genres: { $all: [ "science fiction", "adventure" ] } })
```

This query might return the following documents:

```
{
  _id: ObjectId(...),
  title: "Jurassic Park",
  author: "Michael Crichton",
  year_published: 1990,
  genres: ["science fiction", "adventure", "fantasy", "thriller"]
},
{
  _id: ObjectId(...),
  title: "A Wrinkle in Time",
  author: "Madeleine L'Engle",
  year_published: 1962,
  genres: ["young adult", "fantasy", "science fiction", "adventure"]
},
{
  _id: ObjectId(...),
  title: "Dune",
```

```
  author: "Frank Herbert",
  year_published: 1965,
  genres: ["science fiction", "fantasy", "adventure"]
},
…
```

Notice that using the `$all` operator will match documents where the given array field contains all the specified elements in *any* order, not necessarily the order provided in the query. Furthermore, our search returns documents where the `genres` array includes other elements, in addition to the ones specified in our query.

Let's practice writing queries with the `$all` operator!

Instructions

1.
Connect to the `restaurants` database. Then, search the `listingsAndReviews` collection for any restaurants where the `michelin_stars` field has at least two award years: 2018 and 2019.

Hint

Be sure to run the command `use restaurants` to connect to the `restaurants` collection first. To match multiple array elements in a query, you can use the following syntax:

```
db.<collection>.find({ <field>: { $all: [ <value1>, <value2>, … ] } })
```

```
test> use restaurants
switched to db restaurants
restaurants> db.listingsAndReviews.find({ michelin_stars: { $all: [2018, 2019]}})
[
  {
    _id: ObjectId("62f69e7aa82d46a4c3a255cb"),
    address: { building: '89', street: 'E 42nd Street', zipcode: '10017' },
    borough: 'Manhattan',
    cuisine: 'Scandinavian',
    grades: [
      {
        date: ISODate("2014-12-29T00:00:00.000Z"),
        grade: 'A',
        score: 7
      },
      {
        date: ISODate("2014-06-26T00:00:00.000Z"),
        grade: 'A',
        score: 12
```

Querying on Compound Filter Conditions

In addition to querying array fields for exact matches and individual elements, we can use comparison operators to search for documents where elements in an array field meet some condition or range of conditions.

For example, imagine we have a collection of tennis athletes, called `tennis_players`, with each document having a similar structure:

```
{
  _id: ObjectId(...),
  name: "Serena Williams",
  country: "United States",
  wimbledon_singles_wins: [2002, 2003, 2009, 2010, 2012, 2015, 2016]
}
```

If we wanted to search this collection for all athletes who have been Wimbledon Singles Champions from the year 2000 onward, we could run the following query:

```
db.tennis_players.find({ wimbledon_singles_wins: { $gte: 2000 } });
```

This would return all documents where the `wimbledon_singles_wins` array has at least one element with a value of `2000` or greater. Our query result might look something like this:

```
{
  _id: ObjectId(...),
  name: "Serena Williams",
  country: "United States",
  wimbledon_singles_wins: [2002, 2003, 2009, 2010, 2012, 2015, 2016]
},
{
  _id: ObjectId(...),
  name: "Venus Williams",
  country: "United States",
  wimbledon_singles_wins: [2000, 2001, 2005, 2007, 2008]
},
{
  _id: ObjectId(...),
  name: "Roger Federer",
  country: "Switzerland",
  wimbledon_singles_wins: [2003, 2004, 2005, 2006, 2007, 2009, 2012,
2017]
},
```

We can also query based on compound conditions. Let's consider that we want to search our `tennis_players` collection to find all athletes who won a Wimbledon Singles Championship either before 1935, in the first 50 years of the championship, or after 1971, in the 50 most recent years of the tournament. We could achieve this with the following query:

```
db.tennis_players.find({ wimbledon_singles_wins: { $gt: 1971, $lt: 1935 } })
```

This might return the following set of documents:

```
{
  _id: ObjectId(...),
  name: "Suzanna Lenglen",
  country: "United States",
  wimbledon_singles_wins: [1919, 1920, 1921, 1922, 1923, 1925]
},
{
  _id: ObjectId(...),
  name: "Roger Federer",
  country: "Switzerland",
  wimbledon_singles_wins: [2003, 2004, 2005, 2006, 2007, 2009, 2012, 2017]
},
…
```

Note that this query would match documents where the array contains elements that satisfy the query conditions in some combination. One element could satisfy the greater than `1971` condition, while another could satisfy the less than `1935` condition. And if the ranges overlapped, a single element could satisfy both conditions. However, using this syntax, it is not necessary that a single element satisfies all conditions.

In the next exercise we'll learn how to filter our queries such that the matching documents have at least one array element that satisfies *all* the specified criteria.

Before moving on, let's practice querying with comparison operators on compound filter conditions!

Instructions

1.

Connect to the `restaurants` database. Then, search the `listingsAndReviews` collection for restaurants that received a Michelin star after the year 2010.

Hint

Be sure to run the command `use restaurants` to connect to the `restaurants` database. To query on an array field based on a single condition, you can use the following syntax:

```
db.<collection>.find({ <field>: { <operator>: <value> } })
```

2.

In the same `listingsAndReviews` collection, search for restaurants that have a `michelin_stars` array field containing values that are greater than `2015` or less than `2010`.

Hint

To query on an array field based on multiple conditions, you can use the following syntax:

```
db.<collection>.find({ <field>: { <operator>: <value>, <operator2>: <value2>, … } })
```

```
test> use restaurants
switched to db restaurants
restaurants> db.listingsAndRevies.find({michelin_stars: { $gt: 2010 }});

restaurants> db.listingsAndReviews.find({michelin_stars: { $gt: 2010 }});
[
  {
    _id: ObjectId("62f69e7aa82d46a4c3a255cb"),
    address: { building: '89', street: 'E 42nd Street', zipcode: '10017' },
    borough: 'Manhattan',
    cuisine: 'Scandinavian',
    grades: [
      {
        date: ISODate("2014-12-29T00:00:00.000Z"),
        grade: 'A',
        score: 7
      },
      {
        date: ISODate("2014-06-26T00:00:00.000Z"),
```

Querying for all conditions with $elemMatch

More often than not, when we specify multiple query conditions for an array field, we'll want to match at least one array element that meets *all* the filter criteria. We can accomplish this by using another operator, `$elemMatch`.

The `$elemMatch` operator is used in queries to specify multiple criteria on the elements of an array field, such that any returned documents have at least one array element that satisfies all the specified criteria.

Let's reconsider our previous example about professional tennis players to see `$elemMatch` in action. Recall that documents from this collection have the following structure:

```
{
  _id: ObjectId(...),
  name: "Serena Williams",
  country: "United States",
```

```
  wimbledon_singles_wins: [2002, 2003, 2009, 2010, 2012, 2015, 2016]
}
```

Imagine that we want to search our collection again, this time, for any athletes who have won the Wimbledon Singles Championship in the current millennium, between the years 2000 and 2019.

Our query would look something like this:

```
db.tennis_players.find({
 wimbledon_singles_wins: { $elemMatch: { $gte: 2000, $lt: 2020 } }
})
```

This would only return documents whose `wimbledon_singles_wins` field is an array containing at least one element that is both greater than or equal to `2000` *and* less than `2020`. Our resulting cursor might contain the following documents:

```
{
  _id: ObjectId(...),
  name: "Pete Sampras",
  country: "United States",
  wimbledon_singles_wins: [1993, 1994, 1995, 1997, 1998, 1999, 2000]
},
{
  _id: ObjectId(...),
  name: "Serena Williams",
  country: "United States",
  wimbledon_singles_wins: [2002, 2003, 2009, 2010, 2012, 2015, 2016]
},
{
  _id: ObjectId(...),
  name: "Roger Federer",
  country: "Switzerland",
  wimbledon_singles_wins: [2003, 2004, 2005, 2006, 2007, 2009, 2012, 2017]
}
```

Note that while any matching documents *must* contain at least one value in the `wimbledon_singles_wins` array that is between the range of `2000` and `2020`, this array can also include values that fall outside that range.

Let's practice querying arrays using the `$elemMatch` operator!

Instructions

1.

Connect to the `restaurants` database. Then, search the `listingsAndReviews` collection for any restaurants that were awarded a Michelin star between the years `2005` and `2010`, inclusive.

Hint

To match at least one array element that meets *all* the filter criteria in your query, you can use the following syntax:

```
restaurants> db.listingsAndReviews.find({ michelin_stars: { $elemMatch: { $gte: 2005, $lte: 2010}}});
[
  {
    _id: ObjectId("62f69e7aa82d46a4c3a255ce"),
    address: { building: '155', street: 'W 51st St', zipcode: '10019' },
    borough: 'Manhattan',
    cuisine: 'French',
    grades: [
      {
        date: ISODate("2014-12-29T00:00:00.000Z"),
        grade: 'A',
        score: 11
      },
      {
        date: ISODate("2014-06-26T00:00:00.000Z"),
        grade: 'A',
        score: 12
      },
```

Querying an Array of Embedded Documents

Now that we have a grasp on the basics of querying array fields, we can tackle one more common scenario - querying embedded documents. It's common for a collection to have an array of documents rather than individual values. For instance, take our `tennis_players` collection again, but now with a slightly different structure:

```
{
  _id: ObjectId(...),
  name: "Miyu Kato",
  country: "Japan",
  wimbledon_doubles_placements:
  [{
    year: 2016,
    place: 2
  },
  {
    year: 2017,
    place: 1
  },
  {
    year: 2018,
    place: 1
  },
  {
    year: 2019,
    place: 1
  }]
}
```

In the above example, we have an array field named `wimbledon_doubles_placements` that contains documents inside of it. There are two primary ways we can query the above collection: a match on an entire embedded document or a match based on a single field.

First, let's see how we can do an exact match on the *entire* embedded document. For example, if we wanted to query our `tennis_players` collection for players who placed 2nd in 2019:

```
db.tennis_players.find( { "wimbledon_doubles_placements": { year: 2019, place: 2 } } )
```

In the above query, the field order must be exactly the order we are looking for, with the exact field values. This query would match the below document because the order and values are exactly the same as the one in the query:

```
{
  _id: ObjectId(...),
  name: "Gabriela Dabrowski",
  country: "Canada",
  wimbledon_doubles_placements:
  [{
    year: 2019,
    place: 2
  }]
},
{
  _id: ObjectId(...),
  name: "Yifan Xu",
  country: “China”,
  wimbledon_doubles_placements:
  [{
    year: 2019,
    place: 2
  }]
}…
```

However, a query like this:

```
db.tennis_players.find( { "wimbledon_doubles_placements": {place: 2,  year: 2019 } } )
```

Would not return any results since there would be no documents with that specific ordering.

We can also query based on a single field. For example, if we just wanted to query for any Wimbledon doubles winners in the year 2016, we can do the following:

```
db.tennis_players.find( { "wimbledon_doubles_placements.year": 2016 )
```

Notice that the syntax is exactly the same as when we were querying for non-array fields. The embedded document field and parent document field must be wrapped in quotation marks (single or double) and use the dot (`.`) notation. This query would return results like the following:

```
{
  _id: ObjectId(...),
  name: "Tímea Babos",
  country: "Hungary",
  wimbledon_doubles_placements:
  [{
    year: 2015,
    place: 4
   },
   {
    year: 2016,
    place: 2
  }]
{
  _id: ObjectId(...),
  name: "Yaroslava Shvedova",
  country: "Kazakhstan",
  wimbledon_doubles_placements:
  [{
    year: 2010,
    place: 1
   },
   {
    year: 2016,
    place: 2
  }]
}…
```

Here, our query result has all the documents that have an embedded document with a `year` field with the value of `2016`.

It's important to note we can even combine these queries with query operators and even do multiple query conditions using `$elemMatch`. For more examples, take a look at the official MongoDB documentation on querying embedded documents in arrays.

Before we wrap up, let's practice querying the embedded documents in our `restaurants` collection!

Instructions

1.

Connect to the `restaurants` database. Then, search the listingsAndReviews collection for a restaurant that has the following embedded document inside of the `grades` field:

```
{
  date: ISODate("2014-07-11T00:00:00.000Z"),
  grade: 'A',
  score: 8
}
```

Hint

To query for an exact match on an embedded document, you can use the following syntax:

```
db.<collection>.find({ <parent field>: { <embedded document field>: <value>, <embedded document field2>: <value2>, ... } } )
```

Hint

To query this collection, you can call the `.find()` method on the `listingsAndReviews` collection. To query for a specific embedded document in an array field, query for the exact order for the fields and values you are looking for. In this case, we are looking for an embedded document that looks like this:

```
{
  date: ISODate("2014-07-11T00:00:00.000Z"),
  grade: 'A',
  score: 8
}
```

Once you've put it together, your query command should look like this:

```
db.listingsAndReviews.find({ grades: {date: ISODate("2014-07-11T00:00:00.000Z"), grade: 'A', score: 8} })
```

2.

Now, let's search the `listingsAndReviews` collection for a restaurant based on a single embedded field. Query the collection and find all the restaurants that have embedded documents in the `grades` field with a `grade` of `"C"`.

Hint

The `grade` field is inside embedded documents inside of the `grades` parent field. Recall that to query embedded documents, we must use dot notation (`.`) and wrap the fields in quotation marks.

Hint

Your query should look as follows:

```
db.listingsAndReviews.find({ "grades.grade": "C" })
```

```
restaurants> db.listingsAndReviews.find({ grades: {date: ISODate("2014-07-11T00:00:00.000Z"), grade: 'A', score: 8}
})
[
  {
    _id: ObjectId("5eb3d669b31de5d588f46fb3"),
    address: {
      building: '22423',
      coord: [ -73.7458771, 40.7362064 ],
      street: 'Union Turnpike',
      zipcode: '11364'
    },
    borough: 'Queens',
    cuisine: 'Japanese',
    grades: [
      {
        date: ISODate("2014-12-31T00:00:00.000Z"),
        grade: 'A',
        score: 10
      },
```

```
restaurants> db.listingsAndReviews.find({ "grades.grade": "C"});
[
  {
    _id: ObjectId("5eb3d668b31de5d588f4366a"),
    address: {
      building: '1009',
      coord: [ -73.957135, 40.6470531 ],
      street: 'Flatbush Avenue',
      zipcode: '11226'
    },
    borough: 'Brooklyn',
    cuisine: 'Juice, Smoothies, Fruit Salads',
    grades: [
      {
        date: ISODate("2014-07-10T00:00:00.000Z"),
        grade: 'A',
        score: 10
      },
      {
```

Review
Well done! In this lesson, we continued our exploration of querying in MongoDB by learning how to query array fields. Let's recap some important takeaways from this lesson:

- We can query documents for exact array matches by using the `.find()` method and passing in a query document containing the field name, and its array as the value.
- We can match a single array element by using the `.find()` method and passing in a query document containing the field name, and the element we want to match as its value.
- To match multiple elements in an array, we can apply the `$all` operator to the `.find()` method.
- The `$all` operator will match any document where the given array field contains all the specified values, in any order and regardless of other elements in the array.
- We can use the `.find()` method with comparison operators to match documents where the array contains one or more elements that satisfy the query conditions in some combination.
- To match documents where the array contains one or more elements that satisfy *all* the query conditions, we can apply the `$elemMatch` operator to the `.find()` method.
- We can query embedded documents in an array field by querying for either an exact match (with the exact order) or by querying for a single field.

In addition to the syntax we've covered in this lesson, MongoDB provides us with even more operators that can be useful to us when querying on array fields:

- The `$size` operator is used with `.find()` to match any array with the specified number of elements.
- The `$in` operator can be included in queries to match documents where the field is an array that contains at least one element in the specified array.
- The `$nin` operator can be included in queries to match documents where the field is an array that contains no elements mentioned in the given array.

Instructions

We have provided you with the `listingsAndReviews` collection. Before moving on, spend some time experimenting with writing queries, using the syntax you learned throughout this lesson. If you are up for a challenge, try any of the following tasks listed below. Remember to first connect to the `restaurants` database to access the `listingsAndReviews` collection. Good luck, and click **Up Next** when you are ready to move on!

Optional Tasks:

- Find all the restaurants that have received a Michelin star exactly 14 times.
- Find all the restaurants that have received a Michelin star every year from 2006 to 2016.
- Find all the restaurants that have received only grades above a `"B"`.