MongoDB Queries I

CS 341 Database Systems

MongoDB Query API

- The way you will interact with your data.
- Used in 2 ways
  - CRUD Operations
  - Aggregation Pipelines

MongoDB Query API Uses

You can use the MongoDB Query API to perform:

- Adhoc queries with mongosh, Compass, VS Code, or a MongoDB driver for the programming language you use.
- Data transformations using aggregation pipelines.
- Document join support to combine data from different collections.
- Graph and geospatial queries.
- Full-text search.
- Indexing to improve MongoDB query performance.
- Time series analysis.

Install MongoDB

Create a **MongoDB Atlas** Account

MongoDB Mongosh (Shell)

- **db**
  - Shows the database you are using
- **show dbs**
  - Shows all available databases
- **use database_name**
  - Change or create a database
  - Example: use iba_db
  - Remember: In MongoDB, a database is not actually created until it gets content!

```
Atlas atlas-9egitm-shard-0 [primary] test> use iba_db
switched to db iba_db
Atlas atlas-9egitm-shard-0 [primary] iba_db> show dbs
mydb    48.00 KiB
test    72.00 KiB
admin  288.00 KiB
local   42.56 GiB
```

- Once a collection is created, the database gets some content and is now created.

```
Atlas atlas-9egitm-shard-0 [primary] iba_db> db.createCollection("restaurants")
{ ok: 1 }
Atlas atlas-9egitm-shard-0 [primary] iba_db> show dbs
iba_db    8.00 KiB
mydb     48.00 KiB
test     72.00 KiB
admin   288.00 KiB
local    42.56 GiB
Atlas atlas-9egitm-shard-0 [primary] iba_db>
```

Create a Collection

- **db.createCollection(“nameofCollection")**
  - Creates a new collection
- **db.students.insertOne(object)**
  - Collection during the insert process

Insert Documents

insertOne()

```
db.posts.insertOne({ title: "Post Title 1", body: "Body of post.", category: "News", likes: 1, tags: ["news", "events"], date: Date() })
```

insertMany()

```
db.posts.insertMany([ {
title: "Post Title 2",
body: "Body of post.",
category: "Event", likes: 2, tags: ["news", "events"], date: Date() },
{ title: "Post Title 3",
body: "Body of post.",
category: "Technology",
likes: 3, tags: ["news", "events"], date: Date() }
])
```

Data Retrieval

- **db.collection.find()**
  - Retrieves data
  - Fetches all documents if left empty
- **db.collection.findOne()**

db.collection.find()

- Enclose in brackets
- First {} contains the query criteria
- Second {} specifies the projected columns (either 1s or 0s, only exception is _id which acts as the PK for the object.
- Within the query criteria, the comparator operators are to be used with **:**

SQL VS Mongo Query API

SQL	Mongo Query
Select * from Students	db.students.find()
Select student_name from Students	db.students.find({}, {“student_name”:1})
Select student_name from Students Where major = ‘CS’	db.students.find({“major”: “CS”}, {“student_name”:1})

Projection

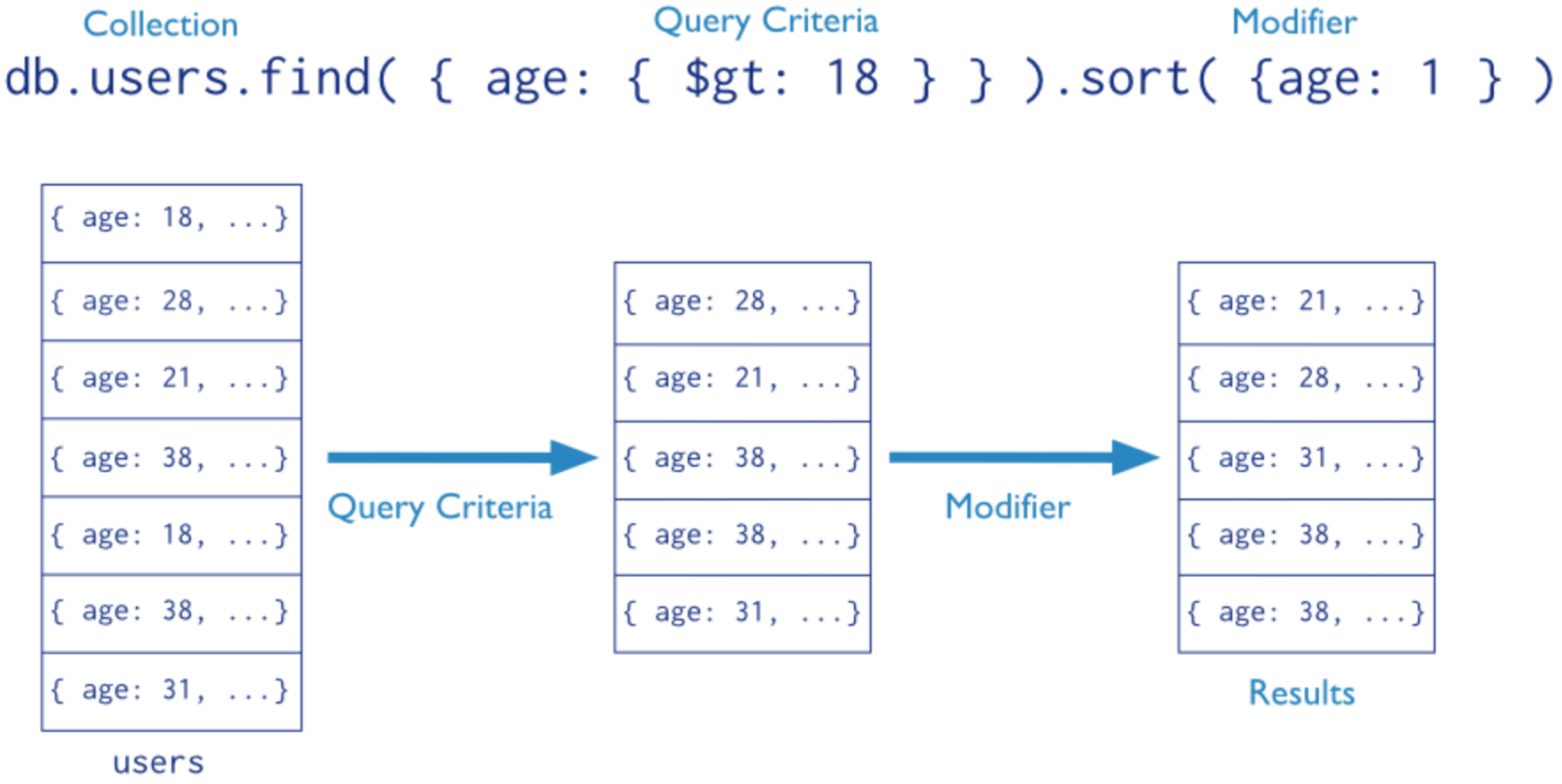
Select student_name from Students Where major = 'CS'	db.students.find({"major": "CS"}, {"student_name":1, "_id":0})

- Note: The _id field is also included. This field is always included unless specifically excluded.
- We use a 1 to include a field and 0 to exclude a field.
- You cannot use both 0 and 1 together except in case of "_id"
- Either specify the fields to include or to exclude.

Comparison Operators

- **$eq** Values are equal
- **$ne** Values are not equal
- **$gt** Value is greater than another value
- **$gte** Value is greater than or equal to another value
- **$lt** Value is less than another value
- **$lte** Value is less than or equal to another value
- **$in** Value is matched within an array

Find the users of age greater than 18 and sort by age.



Logical Operators

- **$and** Returns documents where both queries match
- **$or** Returns documents where either query matches
- **$nor** Returns documents where both queries fail to match
- **$not** Returns documents where the query does not match

Note: In MongoDB, when you provide multiple conditions in the find method, it implicitly interprets them as an AND operation. So, the use of $and is often optional but recommended.

Logical Operators

- db.students.find({gender: “m”, nationality: “Pakistan”})

Is equivalent to:

- db.students.find({$and: [{ gender: "m" },{ nationality: "Pakistan" }]})

- db.musicians.find({$or: [{ instrument: "Drums" }, { born: 1945 }] })

- db.musicians.find({ "instrument": { $in: ["Keyboards", "Bass"] } })

Evaluation Operators

- **$regex** Allows the use of regular expressions when evaluating field values
- **$text** Performs a text search
- **$where** Uses a JavaScript expression to match documents

Regex – contains

- All products that have t in them

```
db.purchase_orders.find( { product: { $regex: "t"} } )
```

- All products that have t in them (case insensitive)

```
db.purchase_orders.find( { product: { $regex: /t/i} } )
```

Regex – starts with

- Product starting with t:

```
db.purchase_orders.find( { product: { $regex: "^t"} } )
```

- Product starting with T (case insensitive)

```
db.purchase_orders.find( { product: { $regex: /^t/i } } )
```

Regex – ends with

Product ending with h:

```
db.purchase_orders.find( { product: { $regex: "h$"} } )
```

Case Insensitive:

```
db.purchase_orders.find( { product: { $regex: /h$/i} } )
```

Regex – start and end

```
db.purchase_orders.find({product:{$regex: /^M.*e$/}})
```

.* → One or Many

. : Matches any single character (except for a newline character).
* : Matches zero or more occurrences of the preceding character or group.

Sort

- Ascending **1**
- Descending -**1**

```
db.students.find({gender: 'm'}).sort({nationality: -1});
```

```
db.students.find({gender: ‘m’}).sort({nationality: -1,
firstName: 1});
```

Limit

- Limit records to just 2

```
db.students.find({gender: ‘f', $or: [{nationality: ‘pakistani'}, {nationality: 'american'}]}).limit(2);
```

- Retrieve the 3rd and 4th record by skipping over first 2.

```
db.students.find({gender: ‘f', $or: [{nationality: ‘pakistani'}, {nationality: 'american'}]}).limit(2).skip(2);
```

Count

Counting number of batteries sold

```
db.purchase_orders.countDocuments({product: “battery"})
```

Count number of documents found

```
db.purchase_orders.find({}).count()
```

Distinct

- List all the distinct products

```
db.purchase_orders.distinct("product")
```

ElemMatch

Element Matching

ElemMatch for Querying Arrays

- The $elemMatch operator matches documents that contain an array field with at **least one element that matches all the specified query criteria.**
- This is primarily used when you want to *match documents based on conditions within an array field.*
- It is useful when you want **all the specified conditions** to be met by **a single element within an array.**
- You don't need to use the $elemMatch operator when specifying a single search condition on an array field. You may directly access the field through dot notation.

ElemMatch in Scores collection

```
{ _id: 1, results: [ 82, 85, 88 ] }
{ _id: 2, results: [ 75, 88, 89 ] }
```

The following query matches only those documents where the results array contains at least one element that is **both** greater than or equal to 80 and is less than 85:

```
db.scores.find(
{ results: { $elemMatch: { $gte: 80, $lt: 85 } } }
)
```

Dot Notation

- Dot notation is used to access fields within nested documents or arrays.
- It helps you navigate through the structure of a document.
- Example *Results.subject.score*

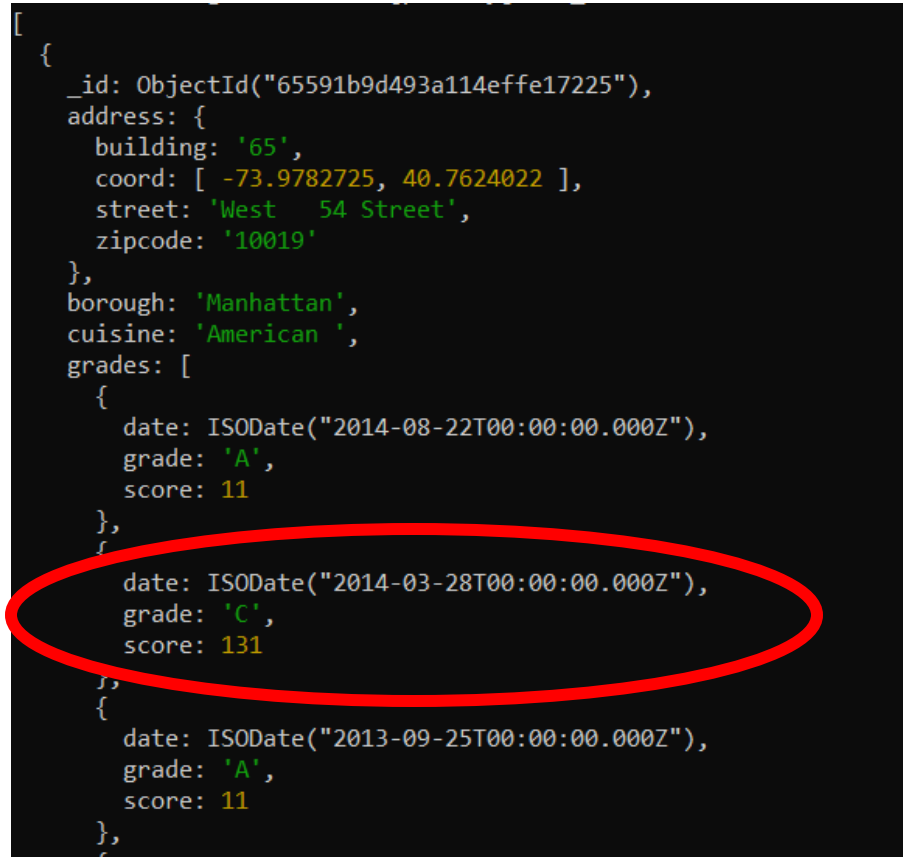
Restaurants Collection

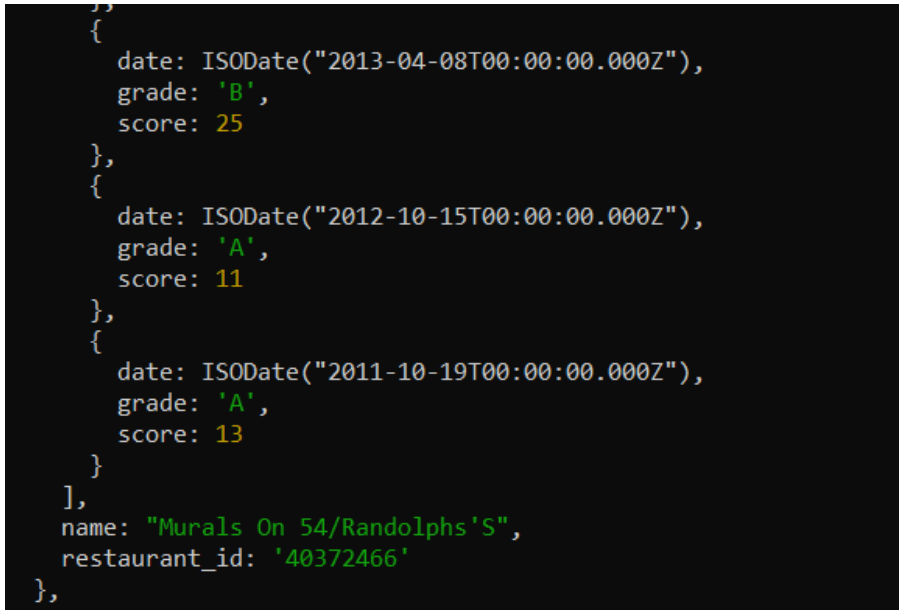

```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "score": 2 },
     { "date": { "$date": 1378857600000 }, "grade": "A", "score": 6 },
     { "date": { "$date": 1358985600000 }, "grade": "A", "score": 10 },
     { "date": { "$date": 1322006400000 }, "grade": "A", "score": 9 },
     { "date": { "$date": 1299715200000 }, "grade": "B", "score": 14 }
  ],
  "name": "Morris Park Bake Shop",
  "restaurant_id": "30075445"
}
```


Display restaurant names with score more than 80 but less than 100.

date: ISODate("2014-06-27T00:00:00.000Z")

db.restaurants.find({"grades.score": { $gt: 80, $lt :100}});





```
Atlas atlas-9egitm-shard-0 [primary] iba_db> db.restaurants.find({ "grades": { $elemMatch: { "score": { $gt: 80, $lt: 100 } } } }, { "name": 1 }).count()
3
Atlas atlas-9egitm-shard-0 [primary] iba_db> db.restaurants.find({ "grades.score": { $gt: 80, $lt: 100 } }, { "name": 1 }).count()
4
Atlas atlas-9egitm-shard-0 [primary] iba_db>
```

- **db.restaurants.find({grades : { $elemMatch:{"score":{$gt : 80 , $lt :100}}}});**

```
{
  _id: ObjectId("673ceb13d4e1d7875c6d112e"),
  address: {
    building: '',
    coord: [ -74.0163793, 40.7167671 ],
    street: 'Hudson River',
    zipcode: '10282'
  },
  borough: 'Manhattan',
  cuisine: 'American ',
  grades: [
    {
      date: ISODate("2014-06-27T00:00:00.000Z"),
      grade: 'C',
      score: 89
    },
    {
      date: ISODate("2013-06-06T00:00:00.000Z"),
      grade: 'A',
      score: 6
    },
    {
      date: ISODate("2012-06-19T00:00:00.000Z"),
      grade: 'A',
      score: 13
    }
  ],
  name: 'West 79Th Street Boat Basin Cafe',
  restaurant_id: '40756344'
}
```

elemMatch is more Specific

- **Directly on "grades.score":**

```
db.restaurants.find({ "grades.score": { $gt: 80, $lt: 100 } }, { "name": 1 })
```

- This query will return documents where there is at least one grade within the "grades" array with a score in range 80 and less than 100. However, it doesn't ensure that both conditions are satisfied by the same grade element within the array.
- **Using $elemMatch:**

```
db.restaurants.find({ "grades": { $elemMatch: { "score": { $gt: 80, $lt: 100 } } } }, { "name": 1 })
```

- This query will return documents where there is at least one grade within the "grades" array that has a score greater than 80 and less than 100. It ensures that a single grade within the array satisfies both conditions simultaneously.

$all operator

- $all is used to match array fields where the array contains all specified elements. It can also be combined with $elemMatch to apply specific conditions to elements in the array.
- *Q. Find documents where there is at least one grade with score 10 and at least one grade with grade 'A'*
- Notice in our dataset, a score of 10 would imply a grade A hence the condition would appear to be correct in a single array element. Let's consider a smaller and different version of the restaurants collection

```
{
  "name": "Restaurant A",
  "grades": [
    { "score": 10, "grade": "B" },
    { "score": 8, "grade": "A" },
    { "score": 5, "grade": "C" }
  ]
},
{
  "name": "Restaurant B",
  "grades": [
    { "score": 10, "grade": "A" },
    { "score": 6, "grade": "B" }
  ]
},
{
  "name": "Restaurant C",
  "grades": [
    { "score": 7, "grade": "B" },
    { "score": 8, "grade": "C" }
  ]
}
```

```
db.restaurants.find({
  "grades": {
    $all: [
      { $elemMatch: { "score": 10 } },
      { $elemMatch: { "grade": "A" } }
    ]
  }
})
```

- When using $all with $elemMatch, we are specifying that the array must contain elements such that both conditions are satisfied, but the conditions do not need to be met by the same element.

```
[
        { "name": "Restaurant A" },
        { "name": "Restaurant B" }
]
```