MongoDB In-Class Queries

//Finding Documents (where)

**** Find comments made by “Lauren Carr”**

- db.comments.find()
- db.comments.find({"name" : "Lauren Carr"})
- db.comments.find({"name" : "Lauren Carr"}).pretty()
- db.comments.findOne()

//Cursor

- var comments = db.comments.find({"name" : "Lauren Carr"})
  comments.next()

//Projection (select)

**** Find comments made by “Lauren Carr”, display “date”**

```
- db.comments.find(
    {"name" : "Lauren Carr"},
    {"name" : 1, "date": 1})
- db.comments.find(
    {"name" : "Lauren Carr"},
    {"name" : 1, "date": 1, "_id" : 0})
```

//Distinct & Count

**** How many documents by “rated”, with “year=1994”**

- db.movies.distinct("rated")
- db.movies.distinct("rated", {"year" : 1994})
- db.movies.count()
- db.movies.count({"name" : "Lauren Carr"})

//Conditional operators:
//Equals ($eq)
**** Find movies with 5 comments**

- db.movies.find({"num_mflix_comments" : 5})
- db.movies.find({ "num_mflix_comments" : {$eq : 5 }})

//Not Equal To ($ne)
**** Find movies with more or less than 5 comments**

```
db.movies.find(
  { "num_mflix_comments" :
    {$ne : 5 }
```

```
                    }
                )
```

//Greater Than ($gt) and Greater Than or Equal To ($gte)

```
                db.movies.find(
                    {year : {$gt : 2015}}).count()
                db.movies.find(
                    {year : {$gte : 2015}}).count()
                db.movies.find(
                    {"released" :
                        {$gte: new Date('2000-01-01')}
                    }).count()
```

//Less Than ($lt) and Less Than or Equal To ($lte)

**** Find movies with less than 2 comments, less than or equal to 2 comments, before "2000-01-01"**

```
                db.movies.find(
                    {"num_mflix_comments" :
                        {$lt : 2}
                    }).count()
                db.movies.find(
                    {"num_mflix_comments" :
                        {$lte : 2}
                    }).count()
                db.movies.find(
                    {"released" :
                        {$lt : new Date('2000-01-01')}
                    }).count()
```

//In ($in) and Not In ($nin)

**** Find movies rated with G, PG, or PG-13, not with G, PG, or PG-13**

```
                db.movies.find(
                    {"rated" :
                        {$in : ["G", "PG", "PG-13"]}
                    }
                )
                db.movies.find(
                    {"rated" :
                        {$nin : ["G", "PG", "PG-13"]}
                    }
                )
```

//To see what happens when you use $nin with a non-existent field

```
                db.movies.countDocuments({})
                db.movies.countDocuments(
                    {"nef" :
                        {$nin : ["a value", "another value"]}
                    }
                )
```

```
        db.movies.countDocuments(
          {"nef" :
            {$nin : ["a value", "another value", null ]}
          }
        )
```

//Logical Operators

//$and operator

* *

```
        db.movies.countDocuments (
          {$and :
            [{"rated" : "UNRATED"}, {"year" : 2008}]
          }
        )
        db.movies.countDocuments (
          {"rated": "UNRATED", "year" : 2008}
        )
```

//$or Operator

```
        db.movies.find(
          { $or : [
            {"rated" : "G"},
            {"rated" : "PG"},
            {"rated" : "PG-13"}
          ]}
        )
        db.movies.find(
          {$or:[
            {"rated" : "G"},
            {"year" : 2005},
            {"num_mflix_comments" : {$gte : 5}}
         ]}
        )
```

//$nor Operator

```
        db.movies.find(
          {$nor:[
            {"rated" : "G"},
            {"year" : 2005},
            {"num_mflix_comments" : {$gte : 5}}
          ]}
        )
```

//$not Operator

```
        db.movies.find(
          {"num_mflix_comments" :
            {$gte : 5}
          }
        )
        db.movies.find(
          {"num_mflix_comments" :
                {$not : {$gte : 5} }
            }
        )
```

//Regular Expressions

```
// find all the movies whose titles contain this character pattern
db.movies.find(
   {"title" : {$regex :"Opera"}}
)

//Using the caret (^) operator // start with the given regular expression
        db.movies.find(
            {"title" : {$regex :"^Opera"}}
        )
//Using the dollar ($) operator // end with the given regular expression
        db.movies.find(
            {"title" : {$regex :"Opera$"}}
        )
//Case-Insensitive Search
        db.movies.find(
            {"title" : {"$regex" : "the"}}
        )

//case-insensitive
        db.movies.find(
            {"title" :
                {"$regex" : "the", $options: "i"}
            }
        )
```

//Query Arrays and Nested Documents

```
//Finding an Array by an Element
        db.movies.find({"cast" : "Charles Chaplin"})
        db.movies.find(
            {$and :[
                {"cast" : "Charles Chaplin"},
                {"cast": "Edna Purviance"}
            ]}
        )
//Finding an Array by an Array
        db.movies.find(
            {"languages" : ["English", "German"]}
        )
        db.movies.find(
            {"languages" : ["German", "English"]}
        )
// Find movies languages by [ "English", "French", "Cantonese", "German"]
        db.movies.find(
            {"languages": [ "English", "French", "Cantonese", "German"]}
        )
```
// Find movies languages by ["English", "French", "Cantonese"]
```
                db.movies.find(
                    {"languages": ["English", "French", "Cantonese"]}
                )
```
//Searching an Array with the $all Operator // irrespective of their order or size
```
                db.movies.find(
                    {"languages":{
                        "$all" :[ "English", "French", "Cantonese"]
                    }}
                )
```

//Projecting Array Elements
//Projecting Matching Elements Using ($) // use projection to exclude all but the first matching element of the array, display only the matched “Syriac” in “languages”
```
                db.movies.find(
                    {"languages" : "Syriac"},
                    {"languages" :1}
                )
                db.movies.find(
                    {"languages" : "Syriac"},
                    {"languages.$" :1}
                )
```
//Projecting Matching Elements by their Index Position ($slice)
** print only the first three elements of “languages”
```
                db.movies.find(
                    {"title" : "Youth Without Youth"},
                    {"languages" : {$slice : 3}}
                ).pretty()

                // You can try more…
                {"languages" : {$slice : -2}}
                {"languages" : {$slice : [2, 4]}}
                {"languages" : {$slice : [-5, 4]}}
```

//Querying Nested Objects
**** Find movies with 1 “wins”**
```
                db.movies.find(
                    {"awards":
                        {"wins": 1, "nominations": 0, "text": "1 win."}
                    }
                )
```
//order matters!
```
                db.movies.find(
                    {"awards":
                        {"nominations": 0, "wins": 1, "text": "1 win."}
                    }
                    )
```
//Querying Nested Object Fields
// The nested field search is performed independently on the given fields, irrespective of the order of the elements.
```
                    db.movies.find(
                       {"awards.wins" : 4}
                    )
                    db.movies.find(
                       {
                          "awards.wins" : {$gte : 5},
                          "awards.nominations" : 6
                       }
                    )
```

//Limiting, Skipping, and Sorting Documents
//Listing 3 titles by "Charles Chaplin"
```
                    db.movies.find(
                       {"cast" : "Charles Chaplin"},
                       {"title": 1, "_id" :0}
                    ).limit(3)
```

//No difference if negative number, but with batch, it matters
```
                    db.movies.find(
                       {"cast" : "Charles Chaplin"},
                       {"title": 1, "_id" :0}
                    ).limit(-2)

                    db.movies.find(
                       {"cast" : "Charles Chaplin"},
                       {"title": 1, "_id" :0}
                    ).batchSize(5)

                    db.movies.find(
                       {"cast" : "Charles Chaplin"},
                       {"title": 1, "_id" :0}
                    ).limit(7).batchSize(5)

                    db.movies.find(
                       {"cast" : "Charles Chaplin"},
                       {"title": 1, "_id" :0}
                    ).limit(-7).batchSize(5)
```

//Skipping Documents
// the first two documents will be excluded from the output, does not allow negative numbers
```
                    db.movies.find(
                       {"cast" : "Charles Chaplin"},
                   {"title": 1, "_id" :0}
               ).skip(2)
```

//Sorting Documents

```
               db.movies.find(
                   {"cast" : "Charles Chaplin"},
                   {"title" : 1, "_id" :0}
               ).sort({"title" : 1})
               db.movies.find(
                   {"cast" : "Charles Chaplin"},
                   {"title" : 1, "_id" :0}
               ).sort({"title" : -1})
               db.movies.find()
                   .limit(50)
                   .sort({"imdb.rating": -1, "year" : 1})
```

Data Manipulation

//Inserting Documents

```
               db.new_movies.insert({"_id" : 1, "title" : "Dunkirk"})
               db.new_movies.find({"_id" : 1})
               show collections
```

//Inserting Multiple Documents

```
               db.new_movies.insertMany([
                   {"_id" : 2, "title": "Baby Driver"},
                   {"_id" : 3, "title": "Logan"},
                   {"_id" : 4, "title": "John Wick: Chapter 2"},
                   {"_id" : 5, "title": "A Ghost Story"}
               ])
               db.new_movies.insertMany([
                   {"_id" : 9, "title" : "movie_1"},
                   {"_id" : 10, "title" : "movie_2"},
                   {"title" : "movie_3"}, # automatically generate a primary key
                   {"_id" : 8, "title" : "movie_4"},
               ])
```

//Inserting Duplicate Keys: causing errors
//Without _id

```
          > db.new_movies.find({"title" : "Thelma"})
```

//Deleting Documents

```
               db.new_movies.deleteOne({"_id": 2})
```

//Deleting Multiple Documents Using deleteMany()

```
               db.new_movies.deleteMany({"title" : {"$regex": "^movie"}})
```

```
db.new_movies.deleteOne({})
db.new_movies.deleteMany({})
```

//The deleteOne() function will delete the document that is found first. However, the deleteMany() function will delete all the documents in the collection.

//.findOneAndDelete

```
db.new_movies.findOneAndDelete({"_id": 3})
{ "_id" : 3, "title" : "Logan" }
```

- It finds one document and deletes it.
- If more than one document is found, only the first one will be deleted.
- Once deleted, it returns the deleted document as a response.
- In the case of multiple document matches, the `sort` option can be used to influence which document gets deleted.
- Projection can be used to include or exclude fields from the document in response.

```
db.new_movies.insertMany([
  { "_id" : 11, "title" : "movie_11" },
  { "_id" : 12, "title" : "movie_12" },
  { "_id" : 13, "title" : "movie_13" },
  { "_id" : 14, "title" : "movie_14" },
  { "_id" : 15, "title" : "series_15" }
])
```

//**sort** matters

```
db.new_movies.findOneAndDelete(
     {"title" : {"$regex" : "^movie"}},
     {sort : {"_id" : -1}}
  )
```

//Using projection

```
> db.new_movies.findOneAndDelete(
    {"title" : {"$regex" : "^movie"}},
    {sort : {"_id" : -1}, projection : {"_id" : 0, "title" : 1}}
)
```