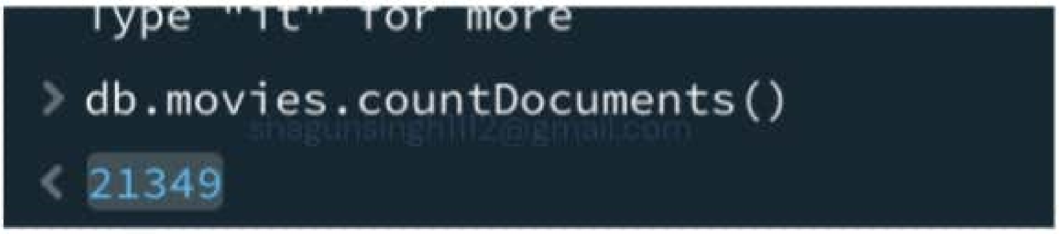
Mongo Queries

Count all document

To count all the documents we can use the `countDocuments` query.

```
db.movies.countDocuments()
```



```
  Type "it" for more
> db.movies.countDocuments()
< 21349
```

Listing all the documents of a collections

To list all the documents of a collection we can use the `find` function on the collection

```
db.movies.find()
```

This query will give you all the records of movie collection in the current database. `find` function returns an array of documents.

To only fetch the first document of the collection we can use `findOne` instead of `find`.

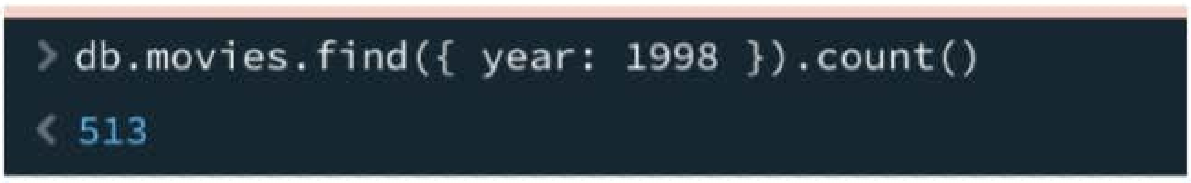
Filtering data using `find` function

Find function takes a parameter which looks mostly like a JS object, which acts as a filtering criteria for the the find query.
For example:
If we want to find all the movies release in the year 1998, then we can use the following query

```
db.movies.find({ year: 1998 })
```

This will return all the movies with `year` property equals to `1998`. If we want to count how many records are actually filtered instead of printing the records we can club it with a count function.

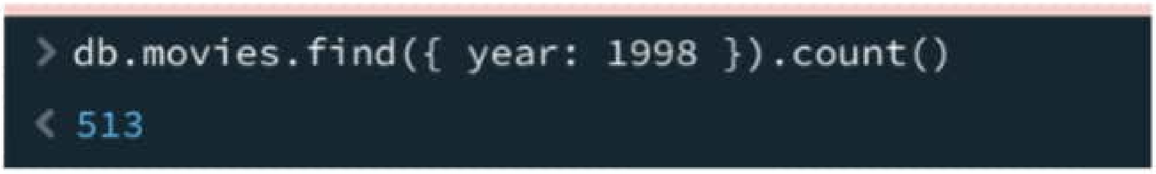
```
db.movies.find({ year: 1998 }).count()
```



```
> db.movies.find({ year: 1998 }).count()
< 513
```

We can modify and make the query a bit more advanced by introducing mongodb operators

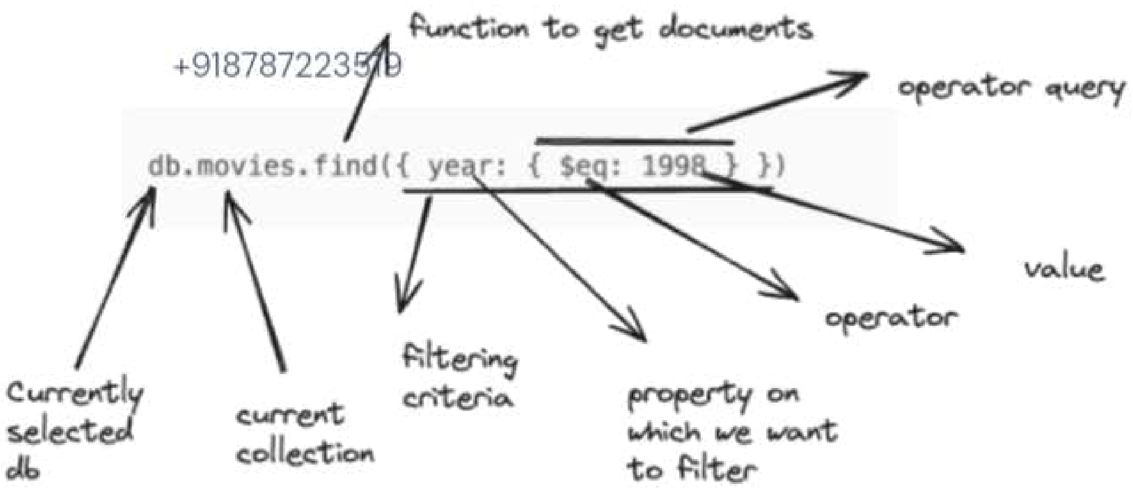
```
db.movies.find({ year: 1998 }).count()
```



```
> db.movies.find({ year: 1998 }).count()
< 513
```

We can modify and make the query a bit more advanced by introducing mongodb operators. These operators helps us to make more alternate comparisons.

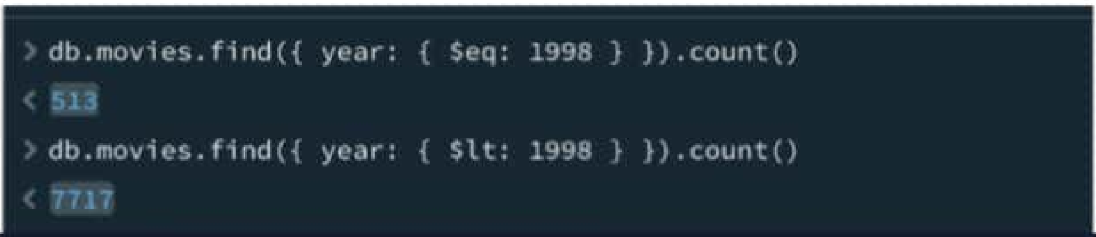
```
db.movies.find({ year: { $eq: 1998 } })
```



```
db.movies.find({ year: { $eq: 1998 } })
```

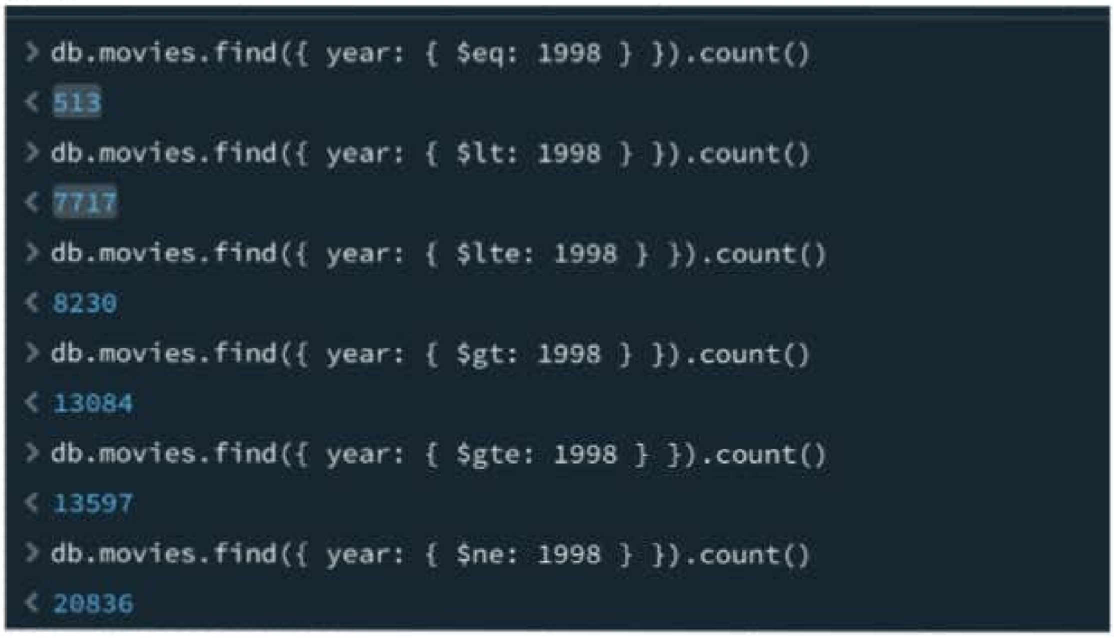
We can replace `1998` with a custom operator query `{ $eq: 1998}` which will help us to introduce custom operators.
More operators are:

- $eq: equals to
- $ne: not equals to
- $lt: less than
- $gt: greater than
- $gte: greater than or equals to
- $ltr: less than or equals to
- $in: checks if the data is In the array
- $nin: checks if the data is not in the array
- $and: This operator is used to join multiple conditions with logical AND.
- $or: This operator is used to join multiple conditions with logical OR.
- $not: This operator is used to join multiple conditions with logical NOT.
- $nor: This operator is used to join multiple conditions with logical nor.
- etc ...



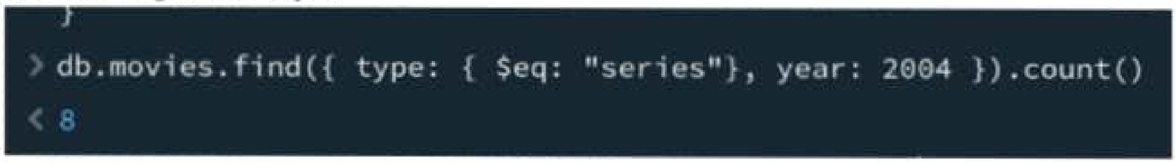
```
> db.movies.find({ year: { $eq: 1998 } }).count()
< 513
> db.movies.find({ year: { $lt: 1998 } }).count()
< 7717
```

- $not: This operator is used to join multiple conditions with logical NOT.
- $nor: This operator is used to join multiple conditions with logical nor.
- etc ...



```
> db.movies.find({ year: { $eq: 1998 } }).count()
< 513
> db.movies.find({ year: { $lt: 1998 } }).count()
< 7717
> db.movies.find({ year: { $lte: 1998 } }).count()
< 8230
> db.movies.find({ year: { $gt: 1998 } }).count()
< 13084
> db.movies.find({ year: { $gte: 1998 } }).count()
< 13597
> db.movies.find({ year: { $ne: 1998 } }).count()
< 20836
```

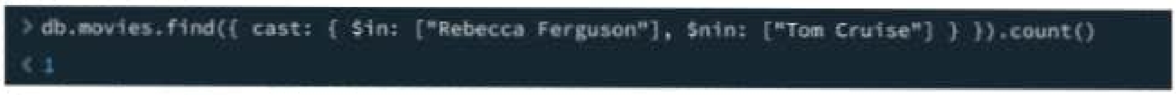
To query on more than one properties together we can club then in a comma separated fashion in the filtering criteria object



```
  }
> db.movies.find({ type: { $eq: "series"}, year: 2004 }).count()
< 8
```

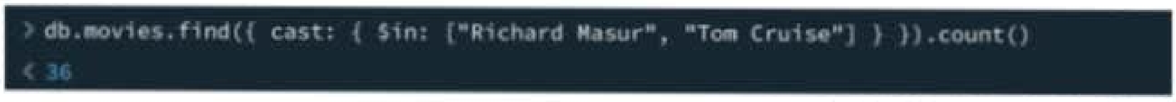
Here we are trying to find all the movies with type as series and year as 2004

We can also add multiple conditions in the operator query.



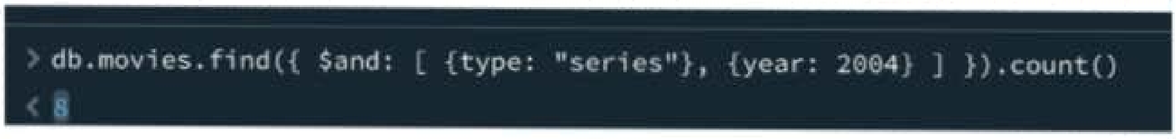
```
> db.movies.find({ cast: { $in: ["Rebecca Ferguson"], $nin: ["Tom Cruise"] } }).count()
< 1
```

Here we are trying to fetch all the movies in which the cast array has "Rebecca Ferguson" but not "Tom Cruise"



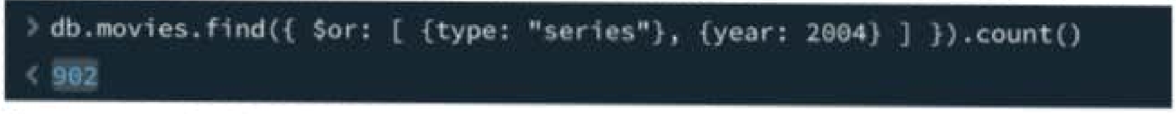
```
> db.movies.find({ cast: { $in: ["Richard Masur", "Tom Cruise"] } }).count()
< 36
```

In this query we are trying to find all the movies which has the both "Richard Masur" or "Tom Cruise" in the cast array.



```
> db.movies.find({ $and: [ {type: "series"}, {year: 2004} ] }).count()
< 8
```

Here also, we are check all the movie count which has type as series AND year as 2004



```
> db.movies.find({ $or: [ {type: "series"}, {year: 2004} ] }).count()
< 902
```

Here, we are checking all the movie count which has either the type as series or the year as 2004.

```
> db.movies.find({ $and: [ {type: "series"}, {year: 2004} ] }).count()
< 8
```

Here also, we are check all the movie count which has type as series AND year as 2004

```
> db.movies.find({ $or: [ {type: "series"}, {year: 2004} ] }).count()
< 902
```

Here, we are checking all the movie count which has either the type as series or the year as 2004.

By default if we put multiple values in the array for `$in` or `$nin` then it is kind of an OR query. If we want to change it to AND, then we should explicitly use `$and` .

```
> db.movies.find({ $and: [ { cast: { $in: ["Tom Cruise"]} }, { cast: { $in: ["Rebecca Ferguson"]}} ] }).count()
< 0
```

We are here trying to find count of all movies which have got both "Tom Cruise" and "Rebecca Ferguson" together.

```
> db.movies.find({ cast: { $all: ["Tom Cruise", "Richard Masur"] } }).count()
< 1
```

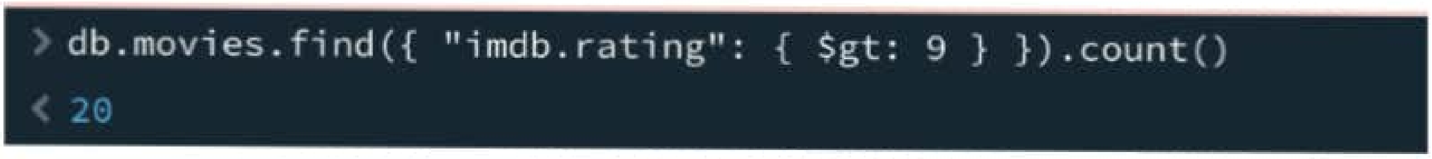
An alternative way of putting AND condition in an array can be by using `$all` operator. Here `$all` ensures we only count those movies which have both "Tom Cruise" and "Richard Masur".

```
> db.movies.find({
    $or: [
      { cast: { $in: ["Rebecca Ferguson"] } },
      { $and: [
            { cast: { $in: ["Tom Cruise"] }},
            { year: { $eq: 2004}}
          ] }
    ]
  }).count()
< 2
```

In the above query we are trying to find count of all movies which has got either "Rebecca Ferguson" in the case OR "Tom Cruise" in the cast along with 2004 year.

Sometimes we might be having nested structures for query properties, like a keys has object and inside that object we want to filter on a property.

For example:

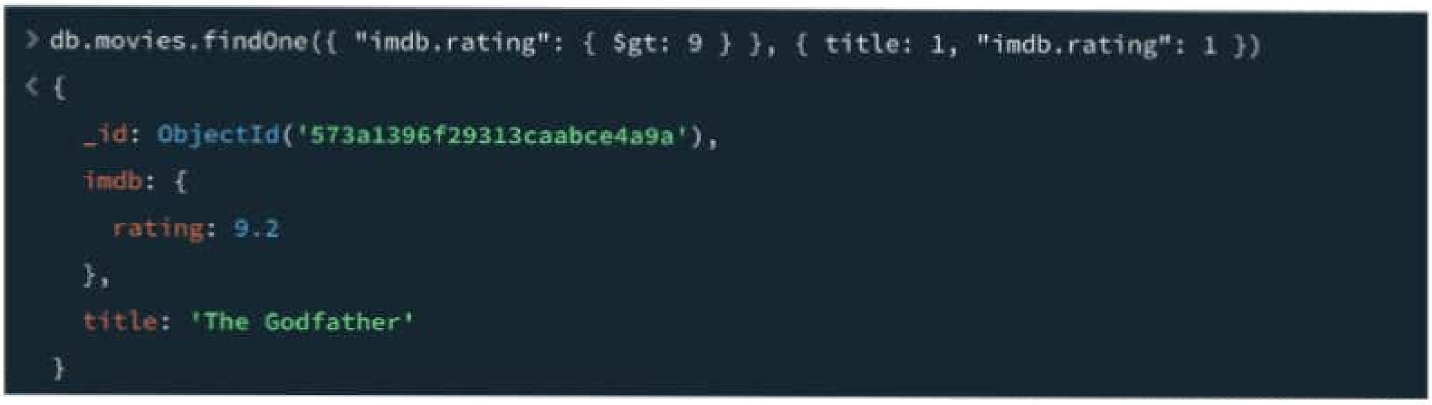


```
> db.movies.find({ "imdb.rating": { $gt: 9 } }).count()
< 20
```

Here, we are trying to find all the movie count where imdb rating is more than 9.

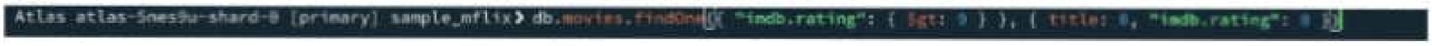
Projection in mongoDb

Projection is a mechanism using which we actually try to include/exclude a few properties from our final result. The first parameter to find and findOne is the filtering criteria, but the second parameter is also an object which tells the projection criteria. Here also we can put keys, and we can give it a 0 value to exclude it from the final output or non zero value to include it.



```
> db.movies.findOne({ "imdb.rating": { $gt: 9 } }, { title: 1, "imdb.rating": 1 })
< {
    _id: ObjectId('573a1396f29313caabce4a9a'),
    imdb: {
      rating: 9.2
    },
    title: 'The Godfather'
  }
```

Here we wanted the first movie with rating more than 9 and we wanted to print only title and imdb rating. +918787223519



```
Atlas atlas-5nes9u-shard-0 [primary] sample_mflix> db.movies.findOne({ "imdb.rating": { $gt: 9 } }, { title: 0, "imdb.rating": 0 })
```

Whereas in this query we want everything excluding title and imdb rating.

Pagination in Mongo

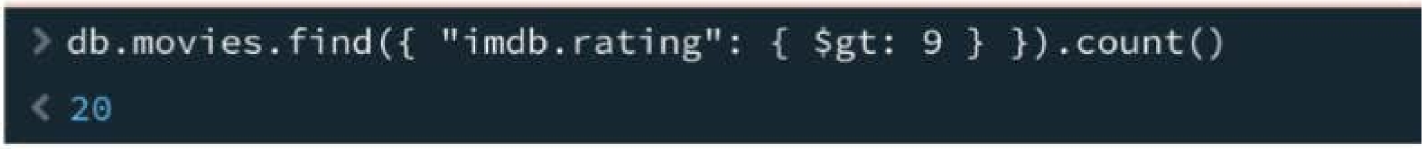
If we want to fetch records in a paginated way such that we only fetch few records page by page, then we can use the `limit` and `skip` function.
Limit function tells how many records to fetch and skip tell from what record number we should start fetch.

```
db.movies.find({}, {title: 1}).limit(3).skip(3)
```

This will fetch 3 movies starting from the 3rd movie (document count starts from 0).

Updating documents in mongodb

For example:

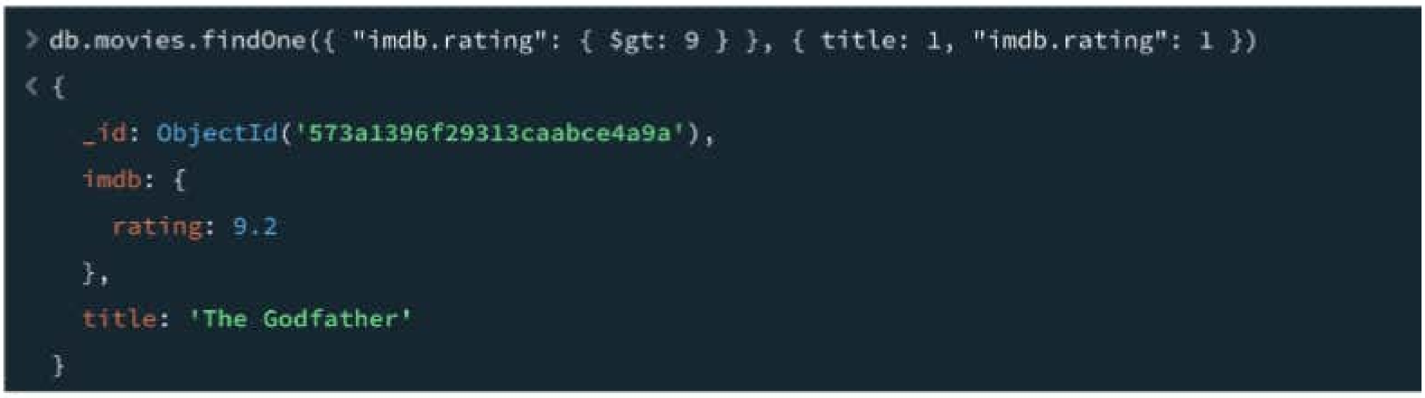


```
> db.movies.find({ "imdb.rating": { $gt: 9 } }).count()
< 20
```

Here, we are trying to find all the movie count where imdb rating is more than 9.

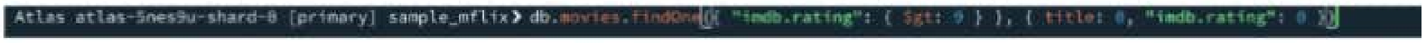
Projection in mongoDb

Projection is a mechanism using which we actually try to include/exclude a few properties from our final result. The first parameter to find and findOne is the filtering criteria, but the second parameter is also an object which tells the projection criteria. Here also we can put keys, and we can give it a 0 value to exclude it from the final output or non zero value to include it.



```
> db.movies.findOne({ "imdb.rating": { $gt: 9 } }, { title: 1, "imdb.rating": 1 })
< {
    _id: ObjectId('573a1396f29313caabce4a9a'),
    imdb: {
      rating: 9.2
    },
    title: 'The Godfather'
  }
```

Here we wanted the first movie with rating more than 9 and we wanted to print only title and imdb rating.



```
Atlas atlas-5nes9u-shard-0 [primary] sample_mflix> db.movies.findOne({ "imdb.rating": { $gt: 9 } }, { title: 0, "imdb.rating": 0 })
```

Whereas in this query we want everything excluding title and imdb rating.

Pagination in Mongo

If we want to fetch records in a paginated way such that we only fetch few records page by page, then we can use the `limit` and `skip` function.
Limit function tells how many records to fetch and skip tell from what record number we should start fetch.

```
db.movies.find({}, {title: 1}).limit(3).skip(3)
```

This will fetch 3 movies starting from the 3rd movie (document count starts from 0).

Updating documents in mongodb

Whereas in this query we want everything excluding title and imdb rating.

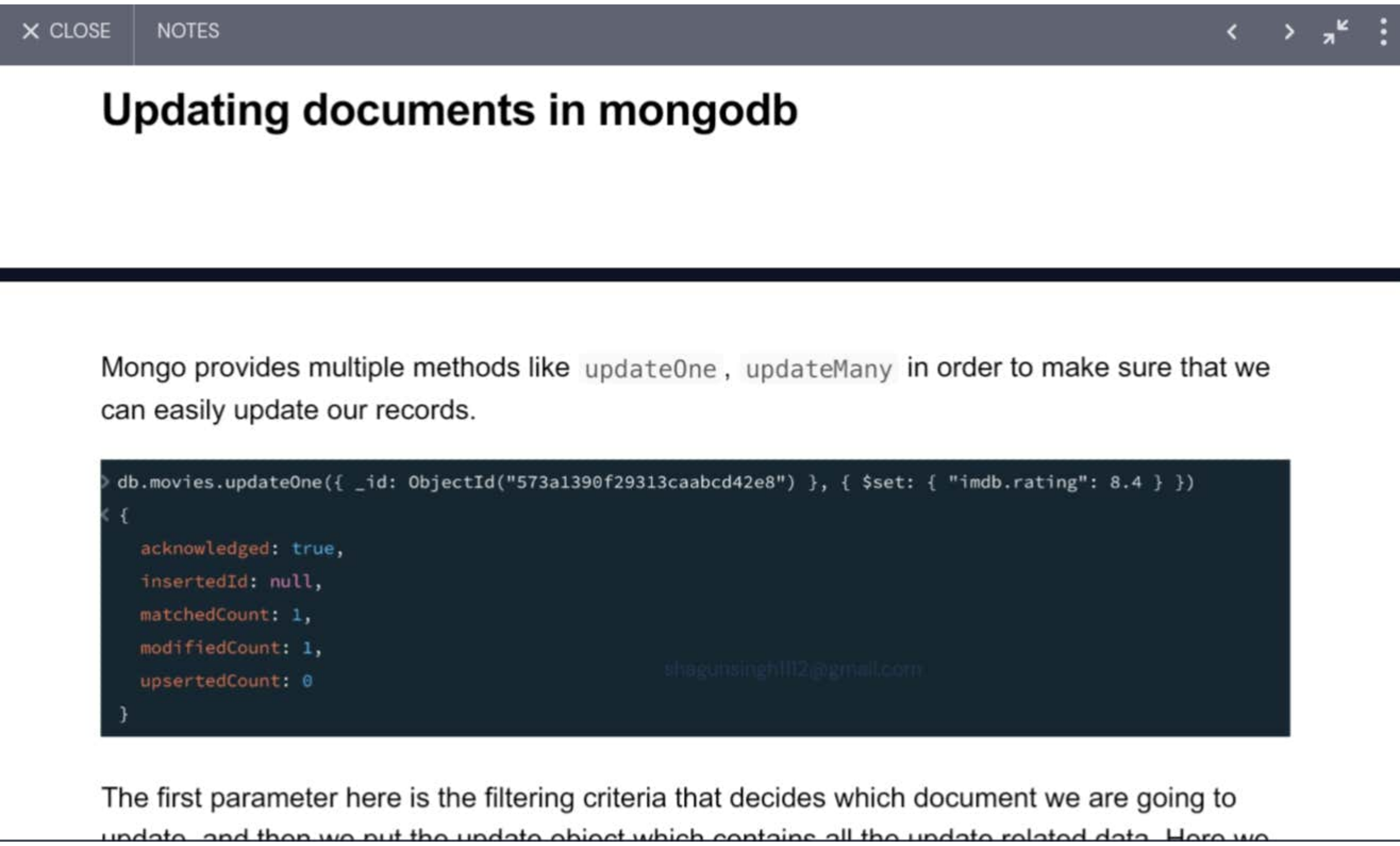
Pagination in Mongo

If we want to fetch records in a paginated way such that we only fetch few records page by page, then we can use the `limit` and `skip` function.
Limit function tells how many records to fetch and skip tell from what record number we should start fetch.

```
db.movies.find({}, {title: 1}).limit(3).skip(3)
```

This will fetch 3 movies starting from the 3rd movie (document count starts from 0).

Updating documents in mongodb



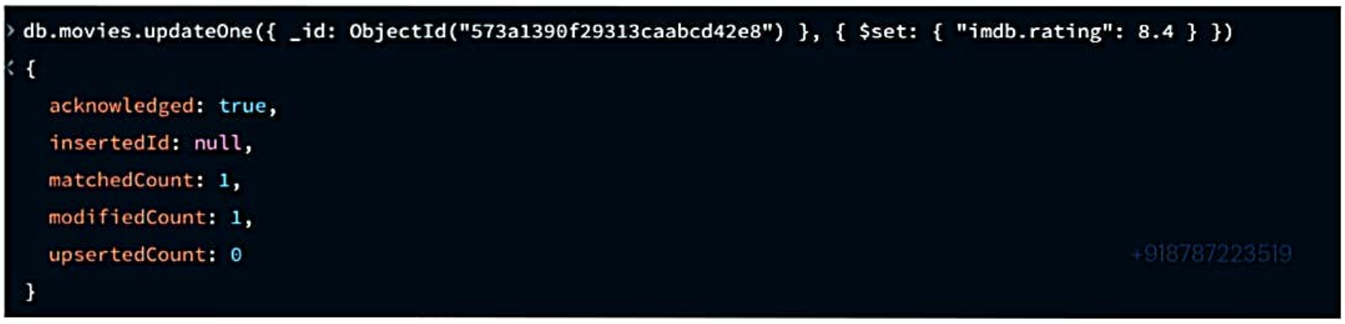
Updating documents in mongodb

Mongo provides multiple methods like `updateOne`, `updateMany` in order to make sure that we can easily update our records.

```
> db.movies.updateOne({ _id: ObjectId("573a1390f29313caabcd42e8") }, { $set: { "imdb.rating": 8.4 } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

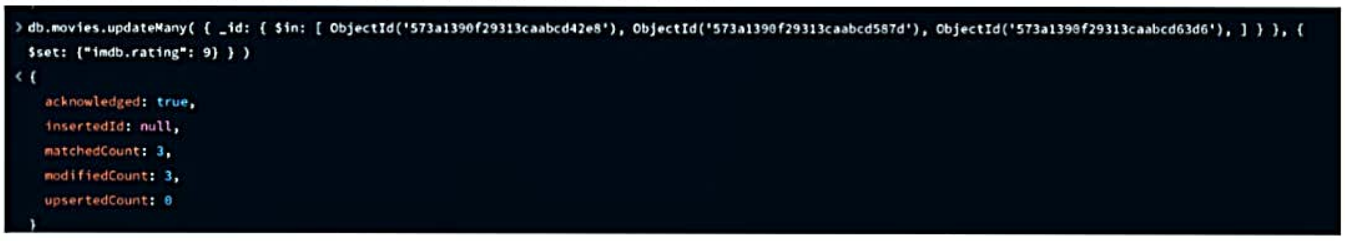
The first parameter here is the filtering criteria that decides which document we are going to update, and then we put the update object which contains all the update related data. Here we

Mongo provides multiple methods like `updateOne`, `updateMany` in order to make sure that we can easily update our records.



```
> db.movies.updateOne({ _id: ObjectId("573a1390f29313caabcd42e8") }, { $set: { "imdb.rating": 8.4 } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

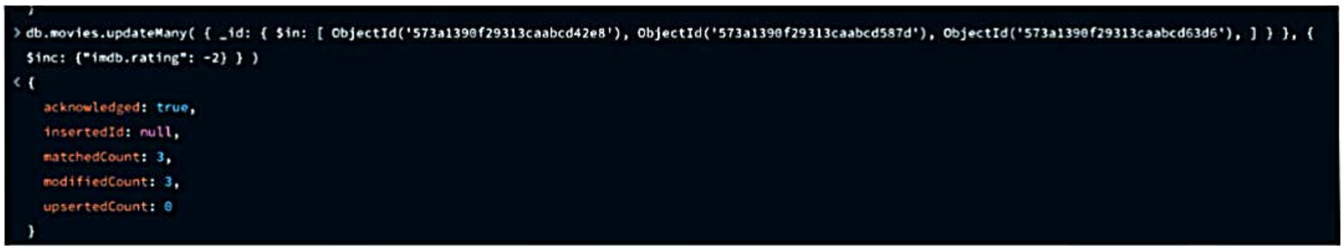
The first parameter here is the filtering criteria that decides which document we are going to update, and then we put the update object which contains all the update related data. Here we use a `$set` operator to set a value in the document.



```
> db.movies.updateMany( { _id: { $in: [ ObjectId('573a1390f29313caabcd42e8'), ObjectId('573a1390f29313caabcd587d'), ObjectId('573a1390f29313caabcd63d6'), ] } }, {
  $set: {"imdb.rating": 9} } )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 3,
    modifiedCount: 3,
    upsertedCount: 0
  }
```

In updateMany, the filtering criteria can select multiple documents and update all of them together.

Here `$set` operator is being used to set the values. There can be more operator which can be used for update queries.



```
> db.movies.updateMany( { _id: { $in: [ ObjectId('573a1390f29313caabcd42e8'), ObjectId('573a1390f29313caabcd587d'), ObjectId('573a1390f29313caabcd63d6'), ] } }, {
  $inc: {"imdb.rating": -2} } )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 3,
    modifiedCount: 3,
    upsertedCount: 0
  }
```

Here `$inc` is being used for incrementing the values instead of directly setting them.

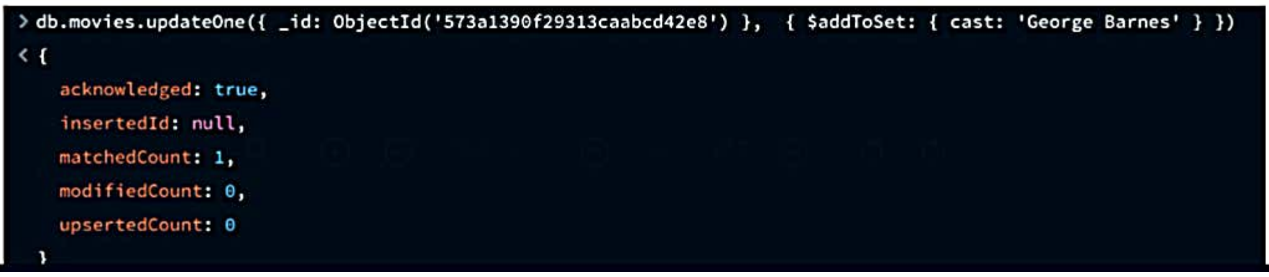
There are more operators:

- $pull: It can help you to remove a single value from an array property
- $pullAll: It can help you to remove multiple values from an array property
- $push: it can help you to push a value inside an array property
- $pop: It can help you to remove either the last or first value from an array

here are more operators:

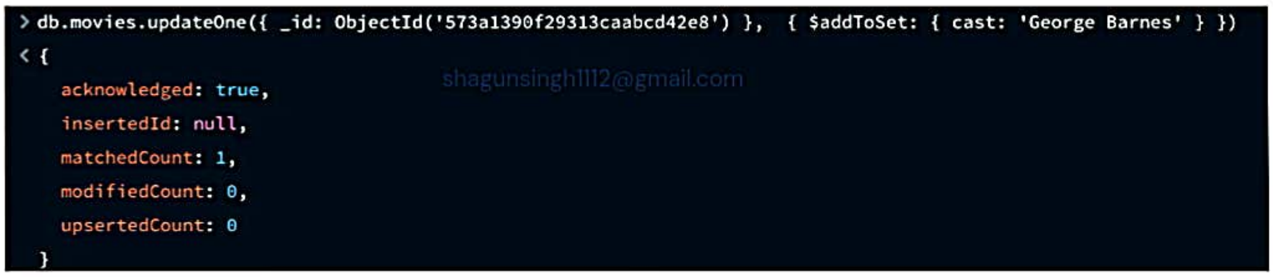
- $pull: It can help you to remove a single value from an array property
- $pullAll: It can help you to remove multiple values from an array property
- $push: it can help you to push a value inside an array property
- $pop: It can help you to remove either the last or first value from an array
  - `db.movies.updateOne({ ... }, { $pop: { propertyName: 1 } })` -> 1 denotes removing last value of the array and -1 would have denoted removal of first value
- $unset: It can help you to remove a field

- $addToSet: It is similar to push but it will not add duplicate values



```
> db.movies.updateOne({ _id: ObjectId('573a1390f29313caabcd42e8') },  { $addToSet: { cast: 'George Barnes' } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 0,
    upsertedCount: 0
  }
```

- $addToSet: It is similar to push but it will not add duplicate values



```
> db.movies.updateOne({ _id: ObjectId('573a1390f29313caabcd42e8') },  { $addToSet: { cast: 'George Barnes' } })
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 0,
    upsertedCount: 0
  }
```

  Here `addToSet` is inserting as if it would have inserted a value in a set.

- $