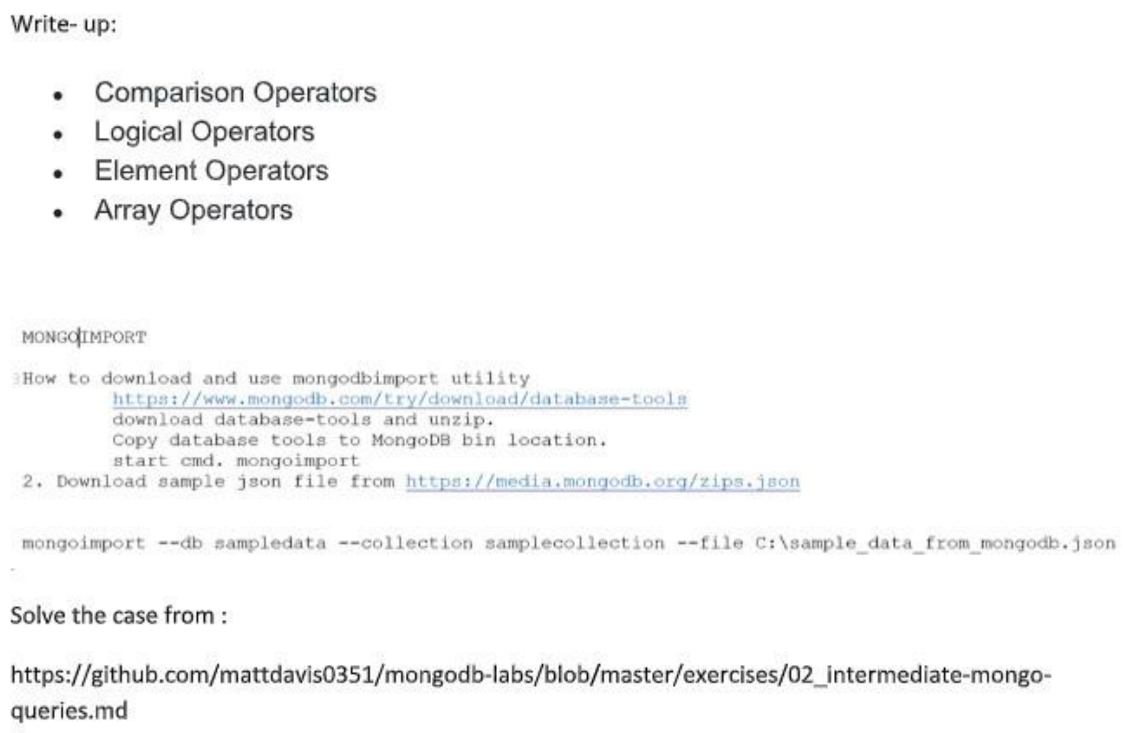
PRACTICAL N0 – 5

Aim: Aggregation using Mongodb



Write- up:

- Comparison Operators
- Logical Operators
- Element Operators
- Array Operators

```
MONGOIMPORT

How to download and use mongodbimport utility
        https://www.mongodb.com/try/download/database-tools
        download database-tools and unzip.
        Copy database tools to MongoDB bin location.
        start cmd. mongoimport
2. Download sample json file from https://media.mongodb.org/zips.json

mongoimport --db sampledata --collection samplecollection --file C:\sample_data_from_mongodb.json
```

Solve the case from :

https://github.com/mattdavis0351/mongodb-labs/blob/master/exercises/02_intermediate-mongo-queries.md

Step1: Download json file from

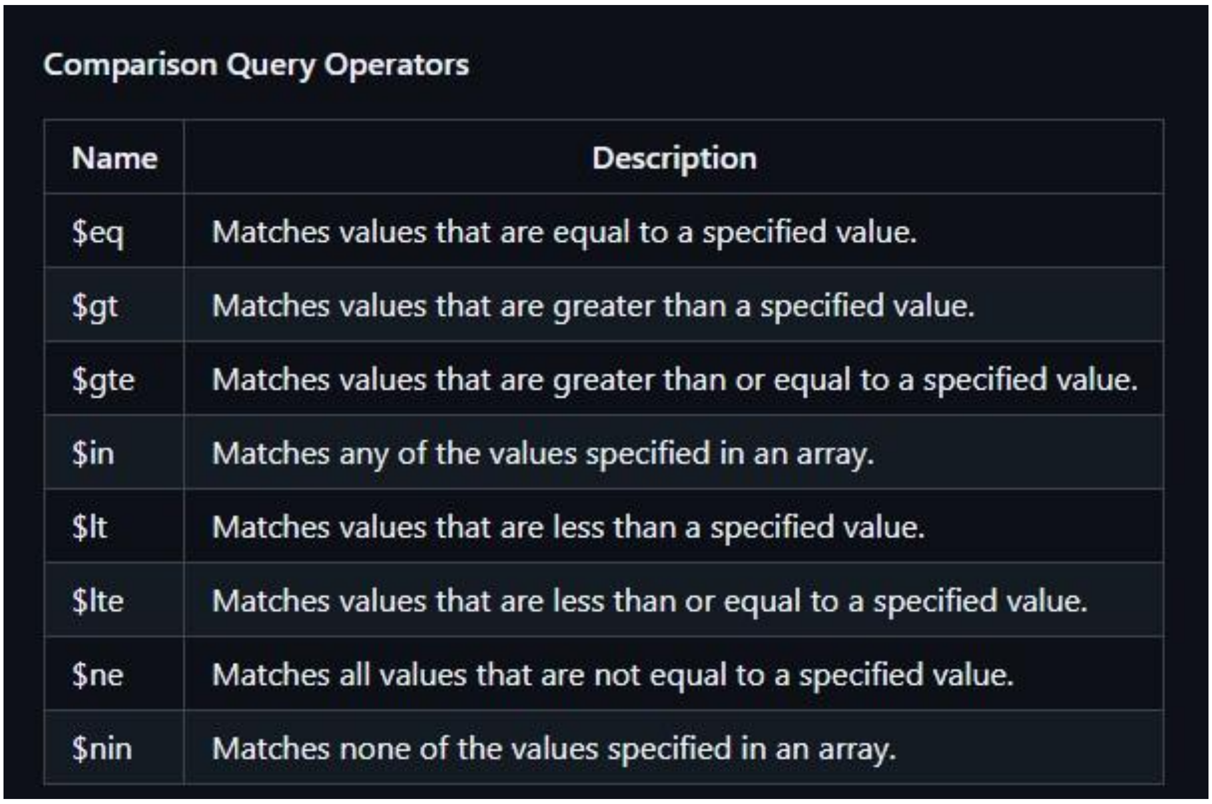
https://media.mongodb.org/zips.json **Step2:**

Go to the cmd prompt and type :

mongoimport --db admin –collection movieDetails –file C:\Users\Admin\Downloads\zips.json

Step3: Go to mongodb compass and type queries

Step4: Perform the following queries on the dataset



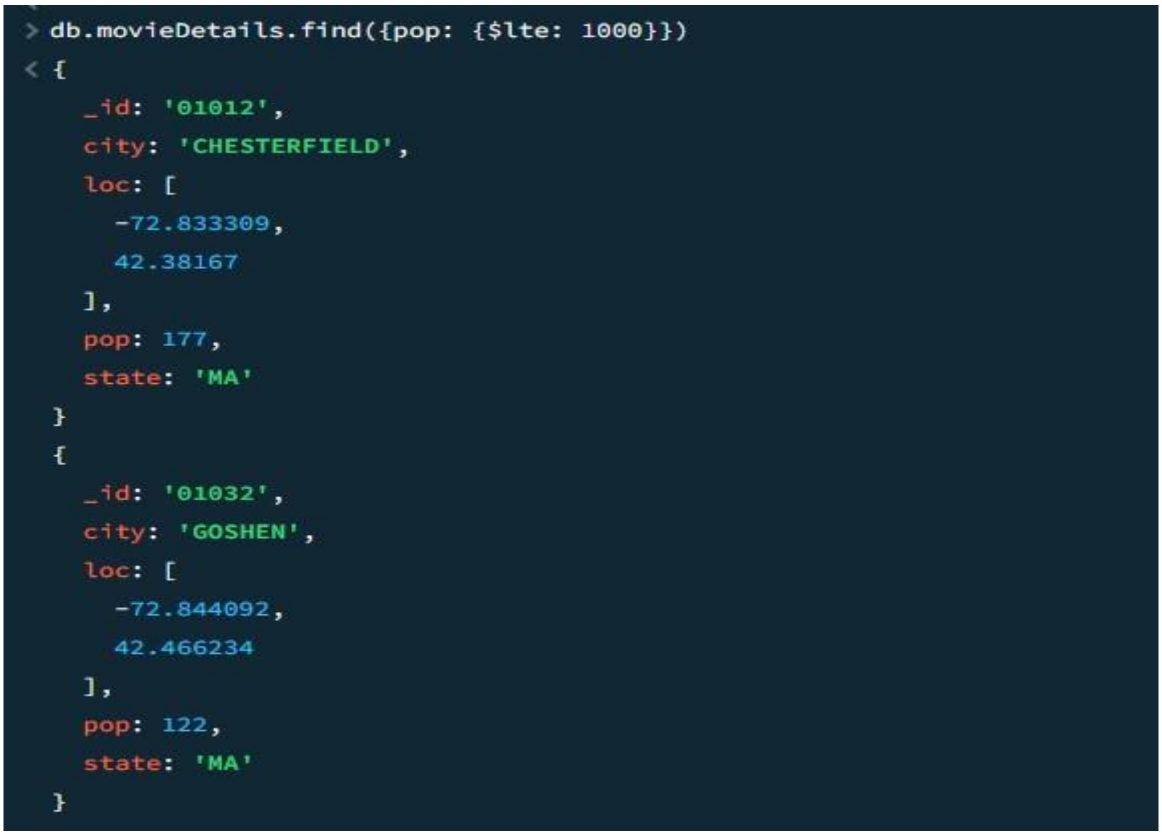
Comparison Query Operators

Name	Description
$eq	Matches values that are equal to a specified value.
$gt	Matches values that are greater than a specified value.
$gte	Matches values that are greater than or equal to a specified value.
$in	Matches any of the values specified in an array.
$lt	Matches values that are less than a specified value.
$lte	Matches values that are less than or equal to a specified value.
$ne	Matches all values that are not equal to a specified value.
$nin	Matches none of the values specified in an array.

Code:

```
db.movieDetails.find({pop: {$lte: 1000}})
```

Output:

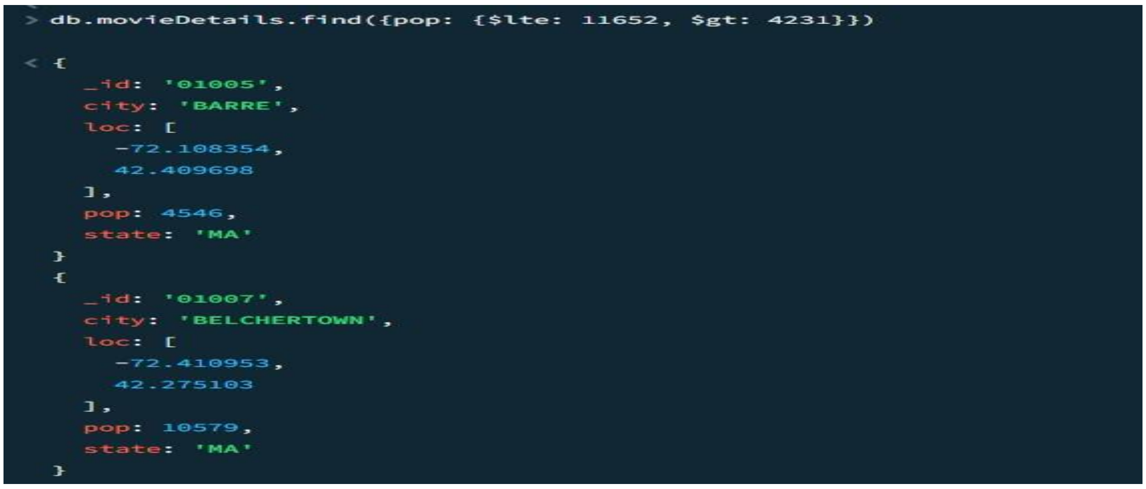


```
> db.movieDetails.find({pop: {$lte: 1000}})
< {
    _id: '01012',
    city: 'CHESTERFIELD',
    loc: [
      -72.833309,
      42.38167
    ],
    pop: 177,
    state: 'MA'
  }
  {
    _id: '01032',
    city: 'GOSHEN',
    loc: [
      -72.844092,
      42.466234
    ],
    pop: 122,
    state: 'MA'
  }
```

Code:

```
db.movieDetails.find({pop: {$lte: 11652, $gt: 4231}})
```

Output:



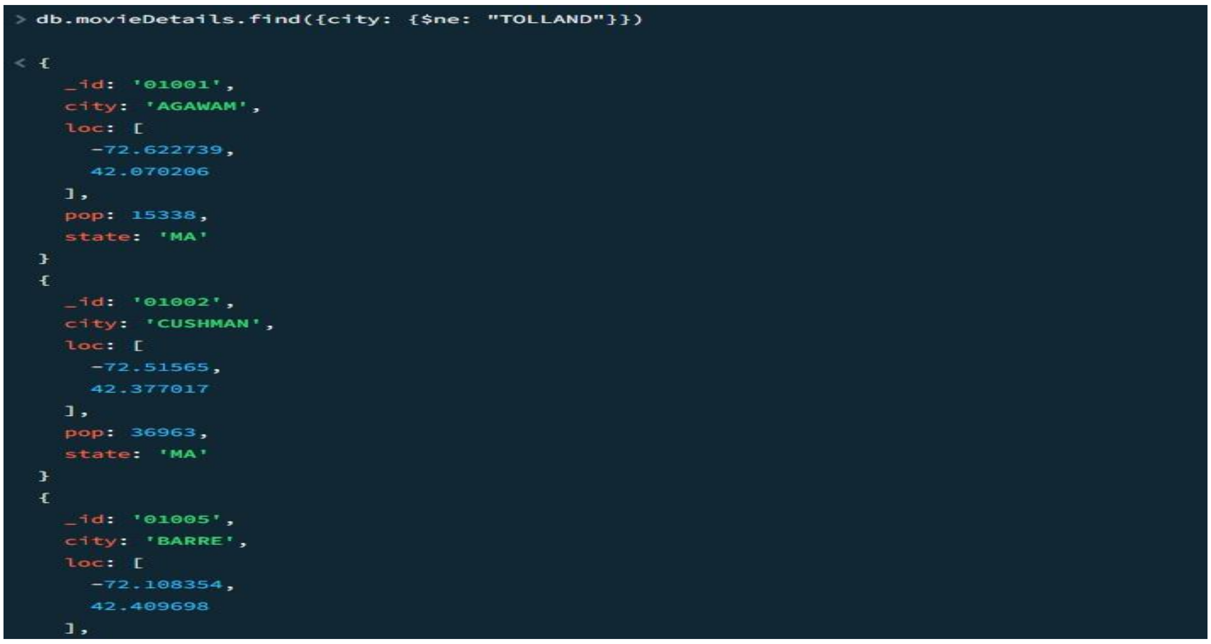
```
> db.movieDetails.find({pop: {$lte: 11652, $gt: 4231}})

< {
    _id: '01005',
    city: 'BARRE',
    loc: [
      -72.108354,
      42.409698
    ],
    pop: 4546,
    state: 'MA'
  }
  {
    _id: '01007',
    city: 'BELCHERTOWN',
    loc: [
      -72.410953,
      42.275103
    ],
    pop: 10579,
    state: 'MA'
  }
```

Code:

```
db.movieDetails.find({city: {$ne: "TOLLAND"}})
```

Output:



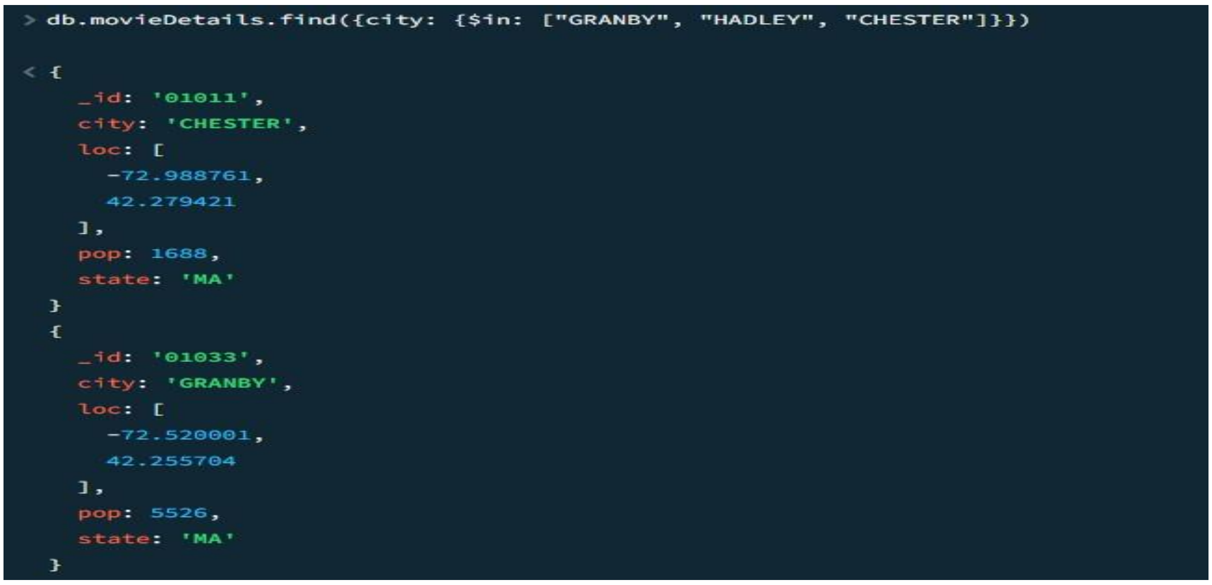
```
> db.movieDetails.find({city: {$ne: "TOLLAND"}})

< {
    _id: '01001',
    city: 'AGAWAM',
    loc: [
      -72.622739,
      42.070206
    ],
    pop: 15338,
    state: 'MA'
  }
  {
    _id: '01002',
    city: 'CUSHMAN',
    loc: [
      -72.51565,
      42.377017
    ],
    pop: 36963,
    state: 'MA'
  }
  {
    _id: '01005',
    city: 'BARRE',
    loc: [
      -72.108354,
      42.409698
    ],
```

Code:

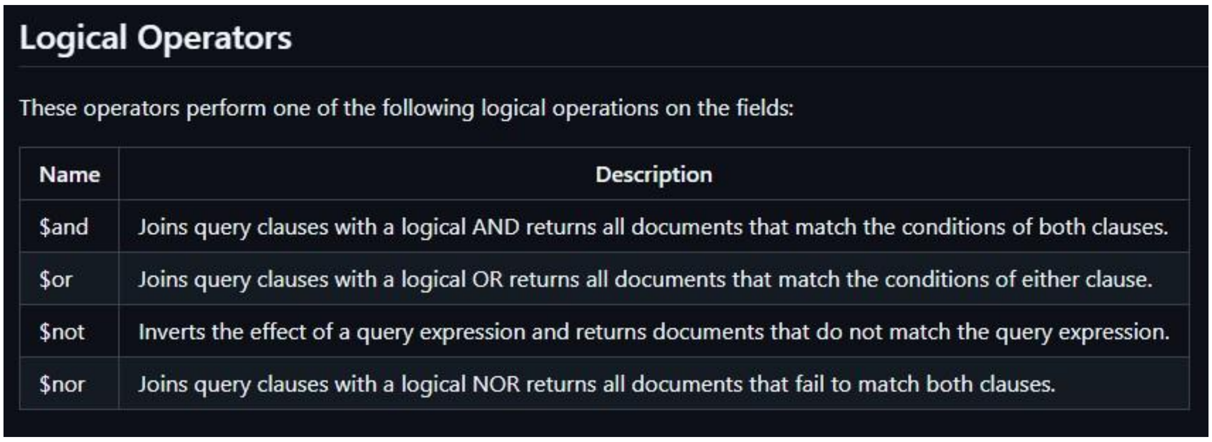
```
db.movieDetails.find({city: {$in: ["GRANBY", "HADLEY", "CHESTER"]}})
```

Output:



```
> db.movieDetails.find({city: {$in: ["GRANBY", "HADLEY", "CHESTER"]}})

< {
    _id: '01011',
    city: 'CHESTER',
    loc: [
      -72.988761,
      42.279421
    ],
    pop: 1688,
    state: 'MA'
  }
  {
    _id: '01033',
    city: 'GRANBY',
    loc: [
      -72.520001,
      42.255704
    ],
    pop: 5526,
    state: 'MA'
  }
```



Logical Operators

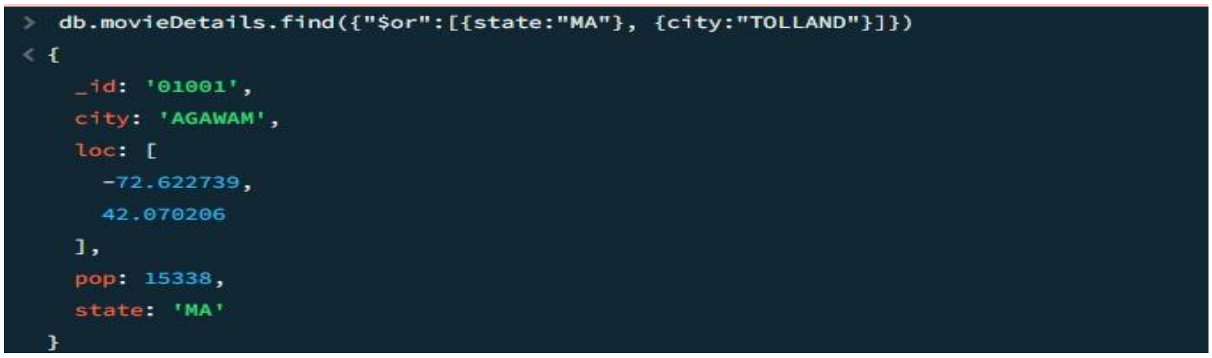
These operators perform one of the following logical operations on the fields:

Name	Description
$and	Joins query clauses with a logical AND returns all documents that match the conditions of both clauses.
$or	Joins query clauses with a logical OR returns all documents that match the conditions of either clause.
$not	Inverts the effect of a query expression and returns documents that do not match the query expression.
$nor	Joins query clauses with a logical NOR returns all documents that fail to match both clauses.

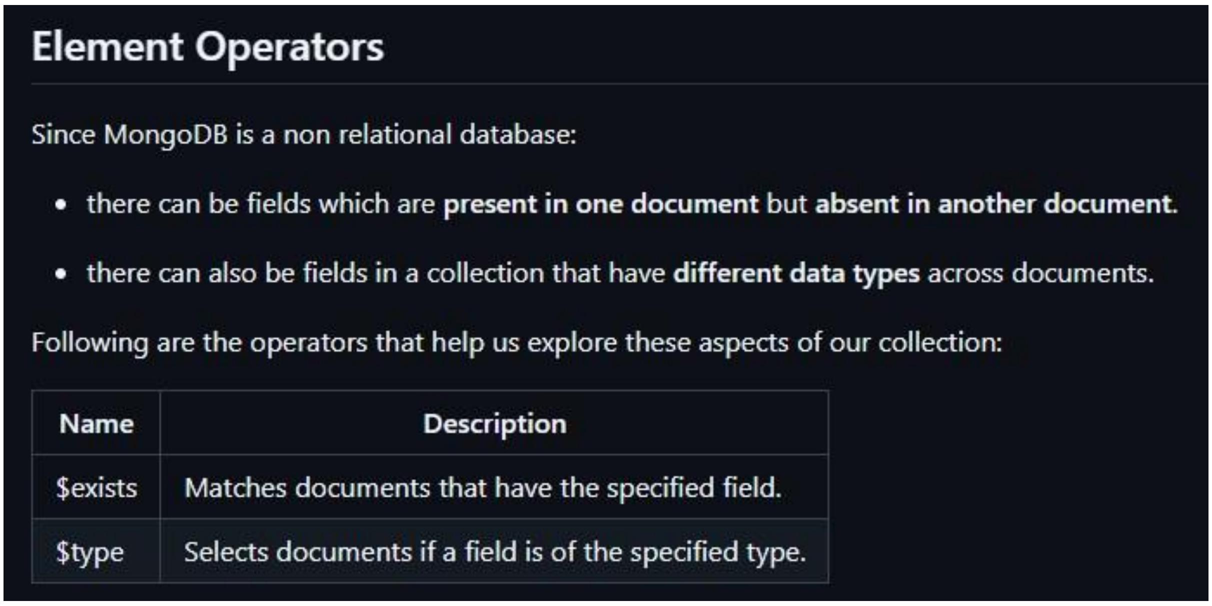
Code:

```
db.movieDetails.find({"$or":[{state:"MA"}, {city:"TOLLAND"}]})
```

Output:



```
> db.movieDetails.find({"$or":[{state:"MA"}, {city:"TOLLAND"}]})
< {
    _id: '01001',
    city: 'AGAWAM',
    loc: [
      -72.622739,
      42.070206
    ],
    pop: 15338,
    state: 'MA'
  }
```



Element Operators

Since MongoDB is a non relational database:

- there can be fields which are **present in one document** but **absent in another document.**
- there can also be fields in a collection that have **different data types** across documents.

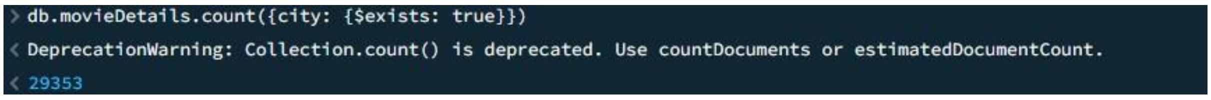
Following are the operators that help us explore these aspects of our collection:

Name	Description
$exists	Matches documents that have the specified field.
$type	Selects documents if a field is of the specified type.

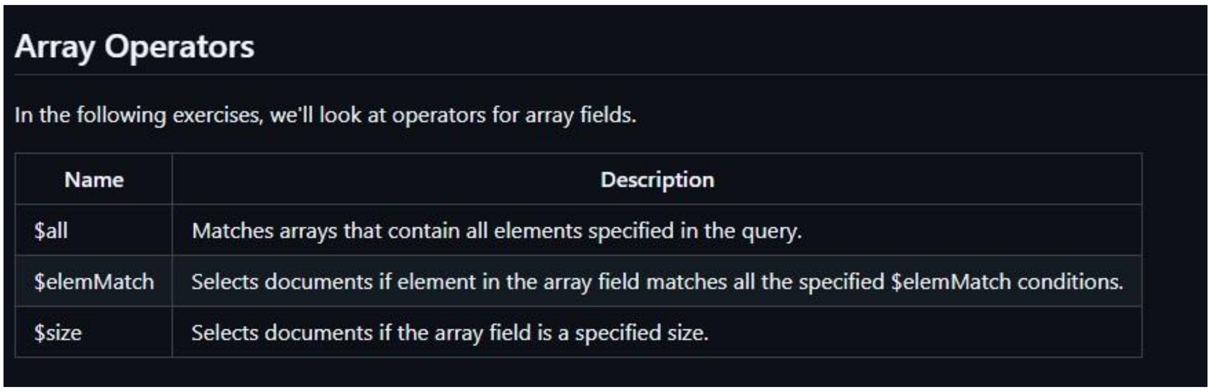
Code:

```
db.movieDetails.count({city: {$exists: true}})
```

Output:



```
> db.movieDetails.count({city: {$exists: true}})
< DeprecationWarning: Collection.count() is deprecated. Use countDocuments or estimatedDocumentCount.
< 29353
```



Array Operators

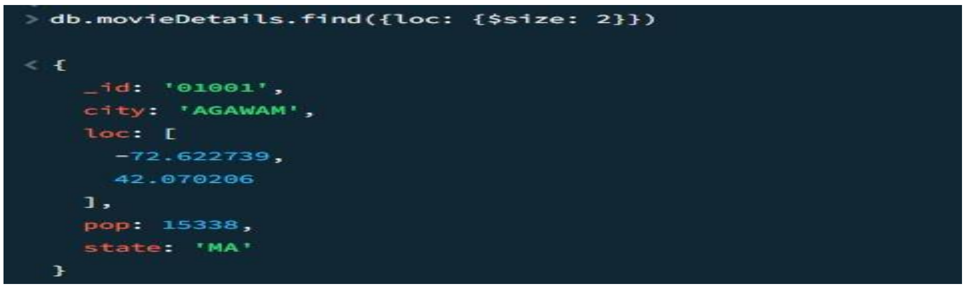
In the following exercises, we'll look at operators for array fields.

Name	Description
$all	Matches arrays that contain all elements specified in the query.
$elemMatch	Selects documents if element in the array field matches all the specified $elemMatch conditions.
$size	Selects documents if the array field is a specified size.

Code:

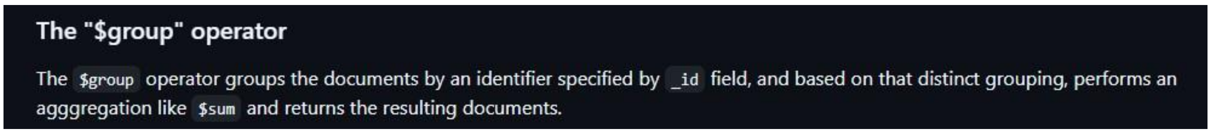
```
db.movieDetails.find({loc: {$size: 2}})
```

Output:



```
> db.movieDetails.find({loc: {$size: 2}})
< {
    _id: '01001',
    city: 'AGAWAM',
    loc: [
      -72.622739,
      42.070206
    ],
    pop: 15338,
    state: 'MA'
  }
```

Advanced Queries



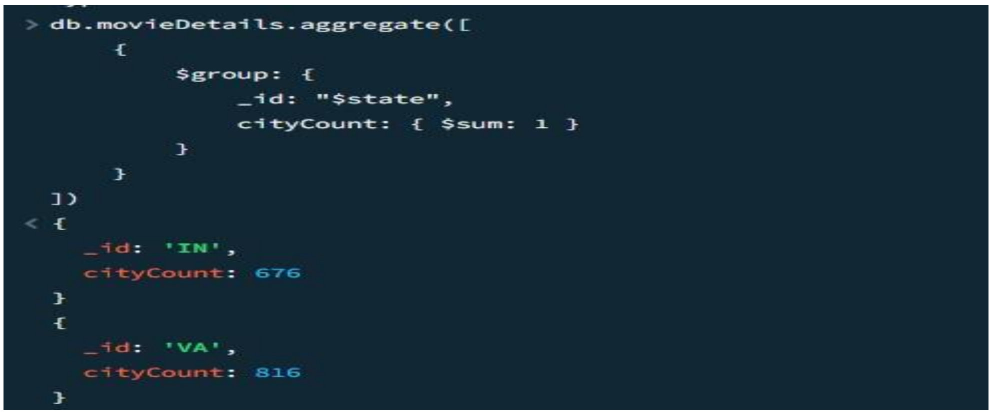
The "$group" operator

The `$group` operator groups the documents by an identifier specified by `_id` field, and based on that distinct grouping, performs an agggregation like `$sum` and returns the resulting documents.

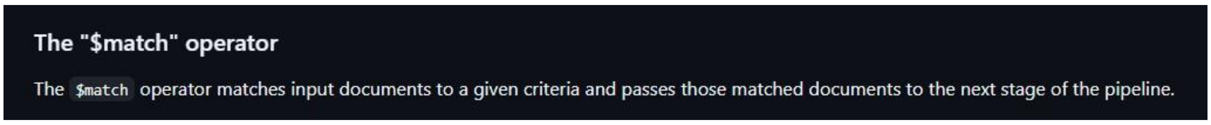
Code:

```
db.movieDetails.aggregate([
   {
      $group: {
         _id: "$state",
         cityCount: { $sum: 1 }
      }
   } ])
```

Output:



```
> db.movieDetails.aggregate([
        {
                $group: {
                        _id: "$state",
                        cityCount: { $sum: 1 }
                }
        }
  ])
< {
    _id: 'IN',
    cityCount: 676
  }
  {
    _id: 'VA',
    cityCount: 816
  }
```



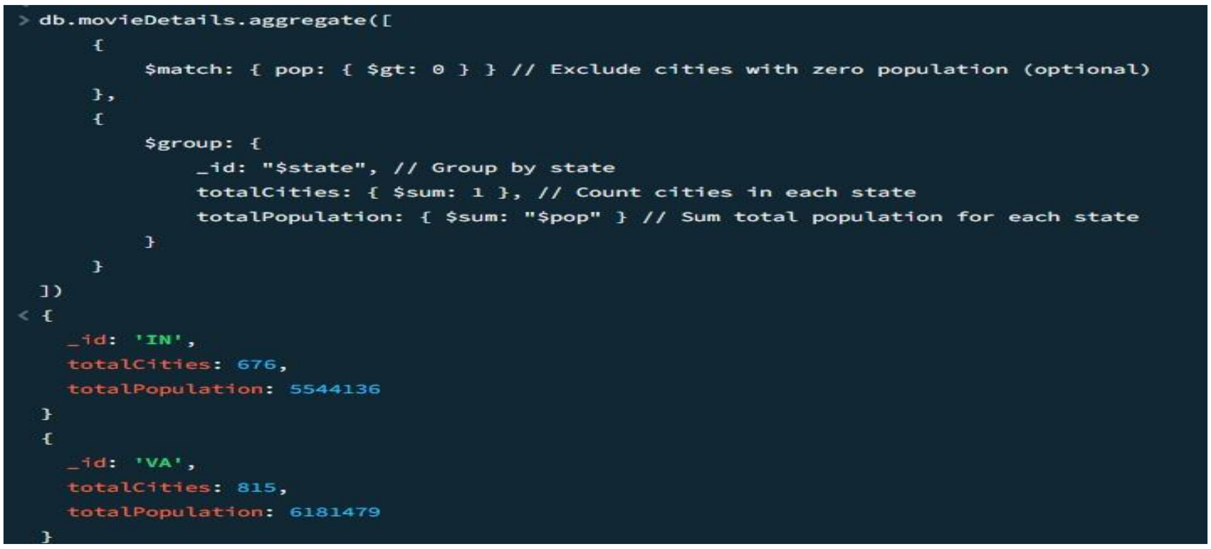
The "$match" operator

The `$match` operator matches input documents to a given criteria and passes those matched documents to the next stage of the pipeline.

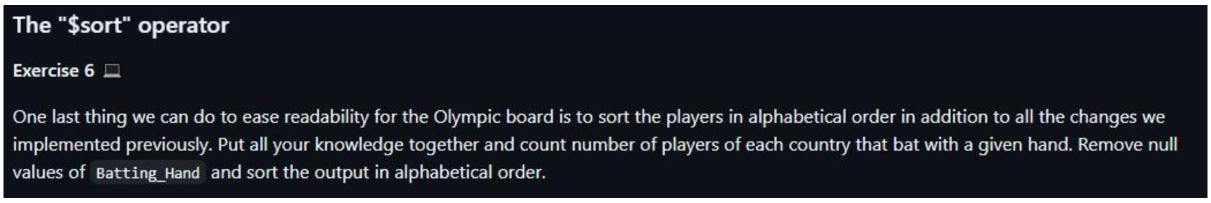
Code:

```
db.movieDetails.aggregate([
   {
      $match: { pop: { $gt: 0 } } // Exclude cities with zero population (optional)
   },
   {
      $group: {
         _id: "$state", // Group by state
         totalCities: { $sum: 1 }, // Count cities in each state
         totalPopulation: { $sum: "$pop" } // Sum total population for each state
      }
   } ])
```

Output:



```
> db.movieDetails.aggregate([
      {
          $match: { pop: { $gt: 0 } } // Exclude cities with zero population (optional)
      },
      {
          $group: {
              _id: "$state", // Group by state
              totalCities: { $sum: 1 }, // Count cities in each state
              totalPopulation: { $sum: "$pop" } // Sum total population for each state
          }
      }
  ])
< {
    _id: 'IN',
    totalCities: 676,
    totalPopulation: 5544136
  }
  {
    _id: 'VA',
    totalCities: 815,
    totalPopulation: 6181479
  }
```



The "$sort" operator

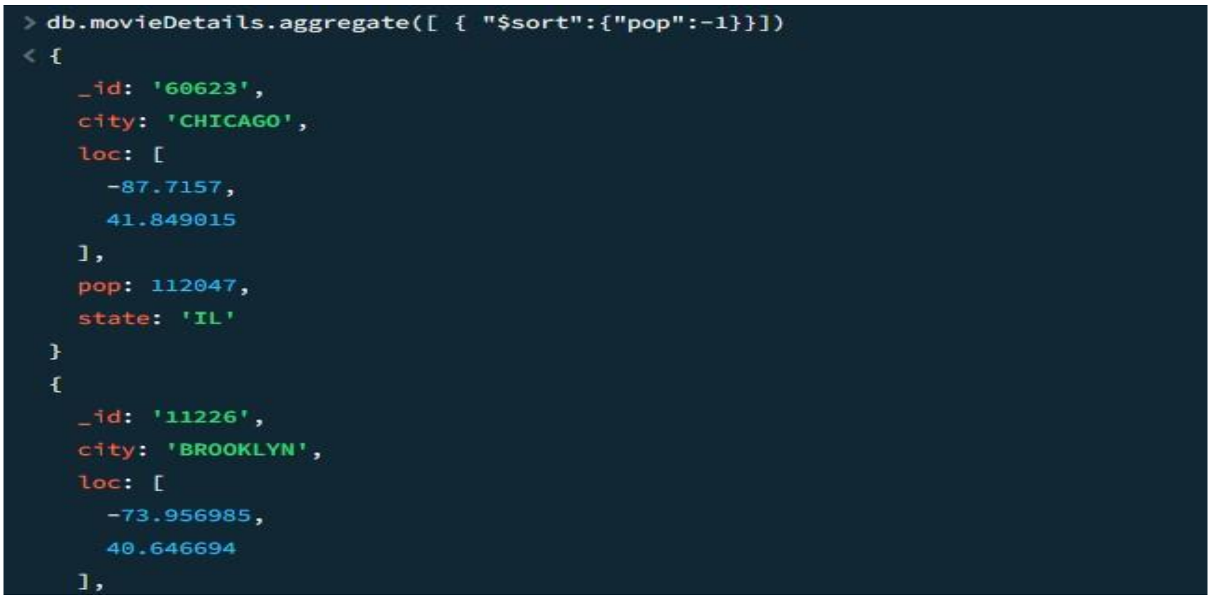
Exercise 6

One last thing we can do to ease readability for the Olympic board is to sort the players in alphabetical order in addition to all the changes we implemented previously. Put all your knowledge together and count number of players of each country that bat with a given hand. Remove null values of `Batting_Hand` and sort the output in alphabetical order.

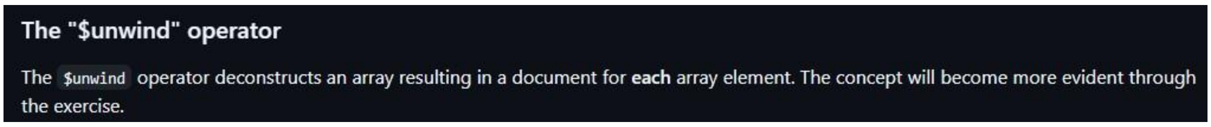
Code:

```
db.movieDetails.aggregate([ { "$sort":{"pop":-1}}])
```

Output:



```
> db.movieDetails.aggregate([ { "$sort":{"pop":-1}}])
< {
    _id: '60623',
    city: 'CHICAGO',
    loc: [
      -87.7157,
      41.849015
    ],
    pop: 112047,
    state: 'IL'
  }
  {
    _id: '11226',
    city: 'BROOKLYN',
    loc: [
      -73.956985,
      40.646694
    ],
```



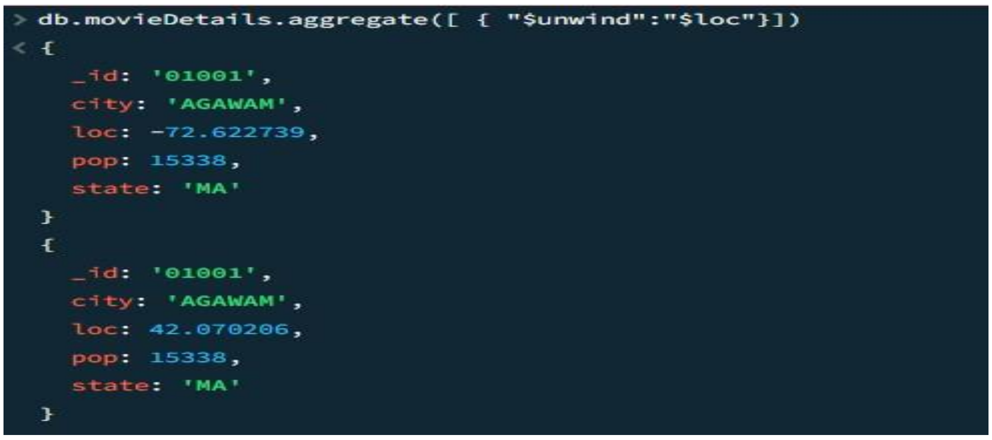
The "$unwind" operator

The `$unwind` operator deconstructs an array resulting in a document for **each** array element. The concept will become more evident through the exercise.

Code:

```
db.movieDetails.aggregate([ { "$unwind":"$loc"}])
```

Output:



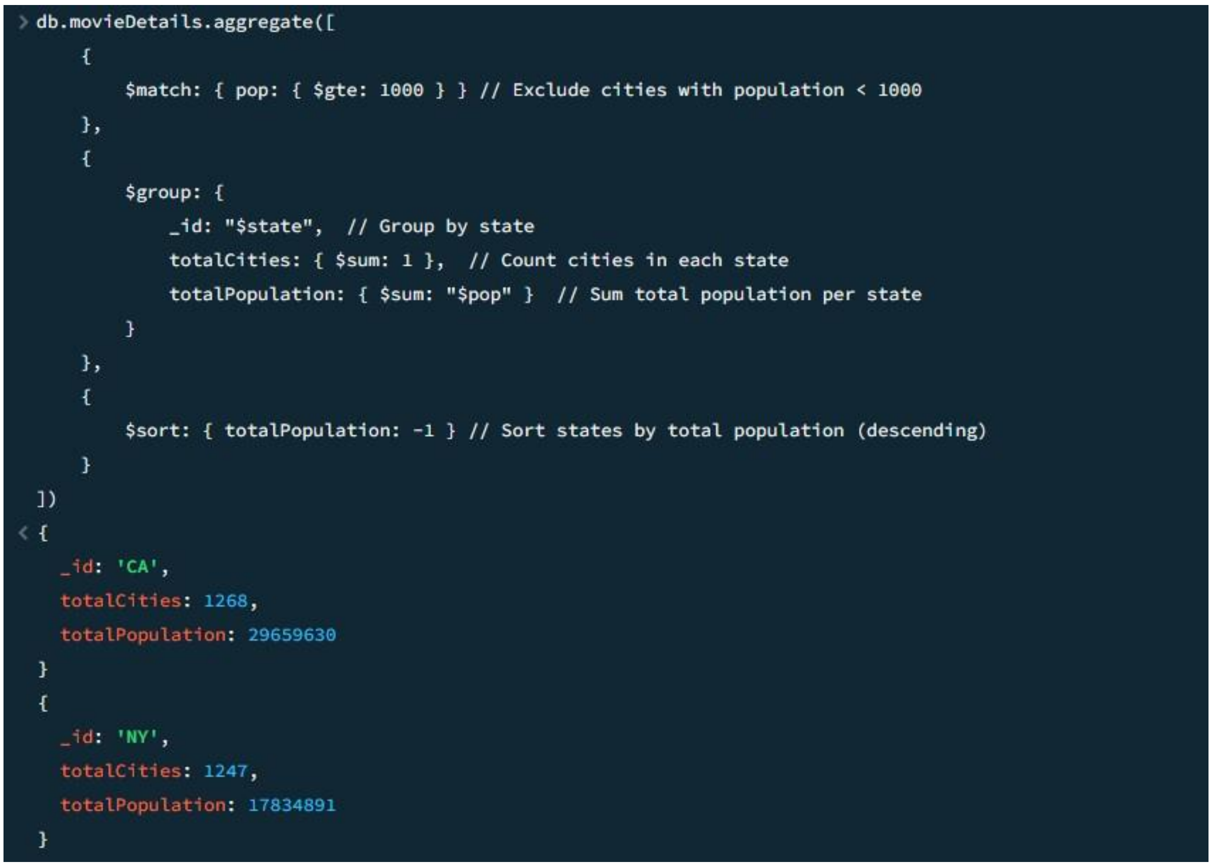
```
> db.movieDetails.aggregate([ { "$unwind":"$loc"}])
< {
    _id: '01001',
    city: 'AGAWAM',
    loc: -72.622739,
    pop: 15338,
    state: 'MA'
  }
  {
    _id: '01001',
    city: 'AGAWAM',
    loc: 42.070206,
    pop: 15338,
    state: 'MA'
  }
```

Combining Operators

Code:

```
db.movieDetails.aggregate([
   {
      $match: { pop: { $gte: 1000 } } // Exclude cities with population < 1000
   },
   {
      $group: {
         _id: "$state",  // Group by state
         totalCities: { $sum: 1 },  // Count cities in each state
         totalPopulation: { $sum: "$pop" }  // Sum total population per state
      }
   },    {
      $sort: { totalPopulation: -1 } // Sort states by total population (descending)
   } ])
```

Output:



```
> db.movieDetails.aggregate([
    {
        $match: { pop: { $gte: 1000 } } // Exclude cities with population < 1000
    },
    {
        $group: {
            _id: "$state",  // Group by state
            totalCities: { $sum: 1 },  // Count cities in each state
            totalPopulation: { $sum: "$pop" }  // Sum total population per state
        }
    },
    {
        $sort: { totalPopulation: -1 } // Sort states by total population (descending)
    }
])
< {
    _id: 'CA',
    totalCities: 1268,
    totalPopulation: 29659630
  }
  {
    _id: 'NY',
    totalCities: 1247,
    totalPopulation: 17834891
  }
```