



به نام خدا
دانشگاه تهران
دانشکده مهندسی برق و کامپیوتر



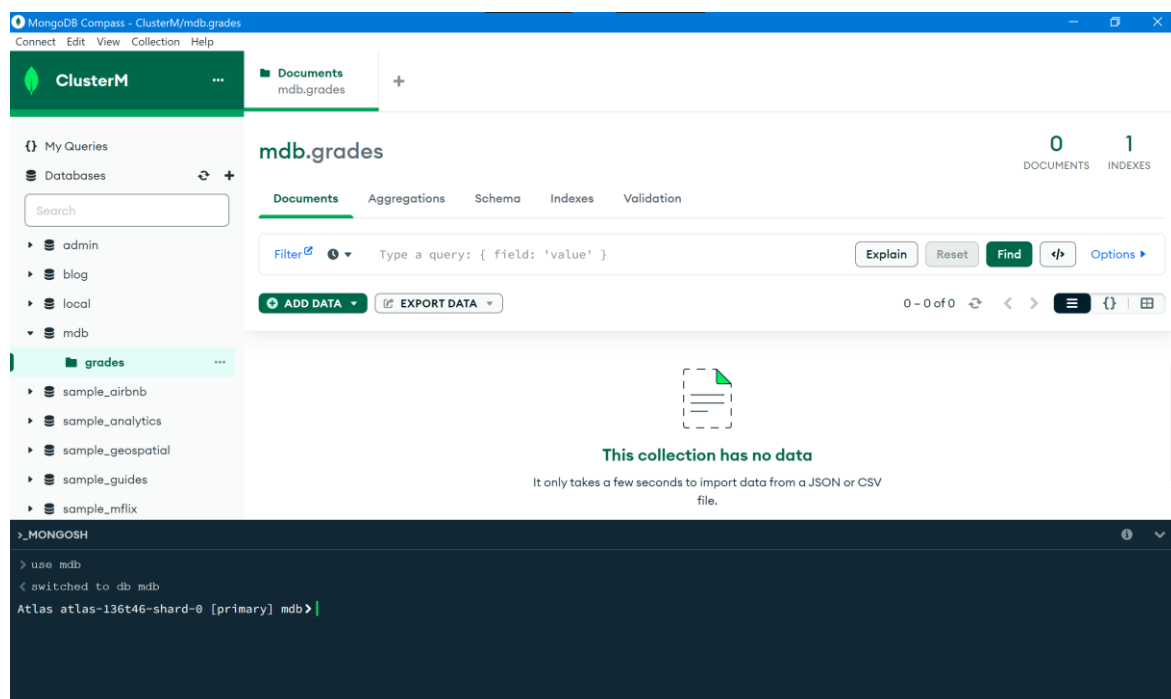
درس آزمایشگاه پایگاه داده پیش گزارش پنجم

نام و نام خانوادگی	میثاق محقق
شماره دانشجویی	810199484
تاریخ ارسال گزارش	1402.08.23

فهرست

2	پاسخ 1. آشنایی با MongoDB
2	1-1 Insert
5	2-1 Find
9	3-1 Replace
10	4-1 Update
12	6-1 Delete
12	7-1 Cursor
13	8-1 Projection
13	9-1 Aggregation
18	پاسخ 2. پیاده‌سازی retrogames
18	1-2 ساخت و تست پایگاه داده

پاسخ 1. آشنایی با MongoDB



ابتدا در MongoDB Atlas اکانتی ساخته شده و با محیط آن آشنا شدیم.

از برنامه MongoDB Compass برای اتصال به Atlas که در آن sample database لود شده است استفاده می‌کنیم. در پایین این برنامه MongoDB Shell هم قرار دارد.

دیتابیس mdb ساخته شده را انتخاب می‌کنیم. داخل mdb، یک collection به نام grades وجود دارد که هیچ document-ای داخل آن قرار ندارد.

1-1 Insert

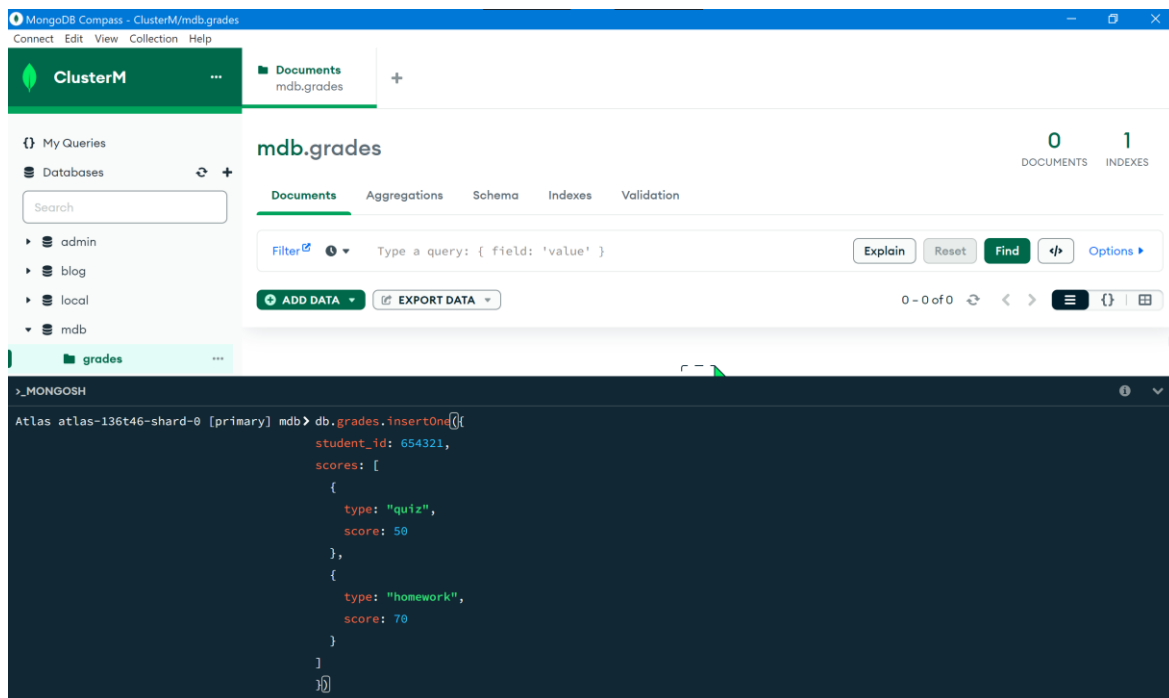
برای insert کردن می‌توان از دو روش استفاده کرد.

در صورتی که فقط یک چیز را می‌خواهیم وارد کنیم، تابع insertOne و برای اینسرت کردن چند آیتم به طور همزمان، از insertMany استفاده می‌کنیم.

با استفاده از کلیدواژه db، کالکشن مدنظر را انتخاب کرده و به آن اینسرت می‌کنیم.

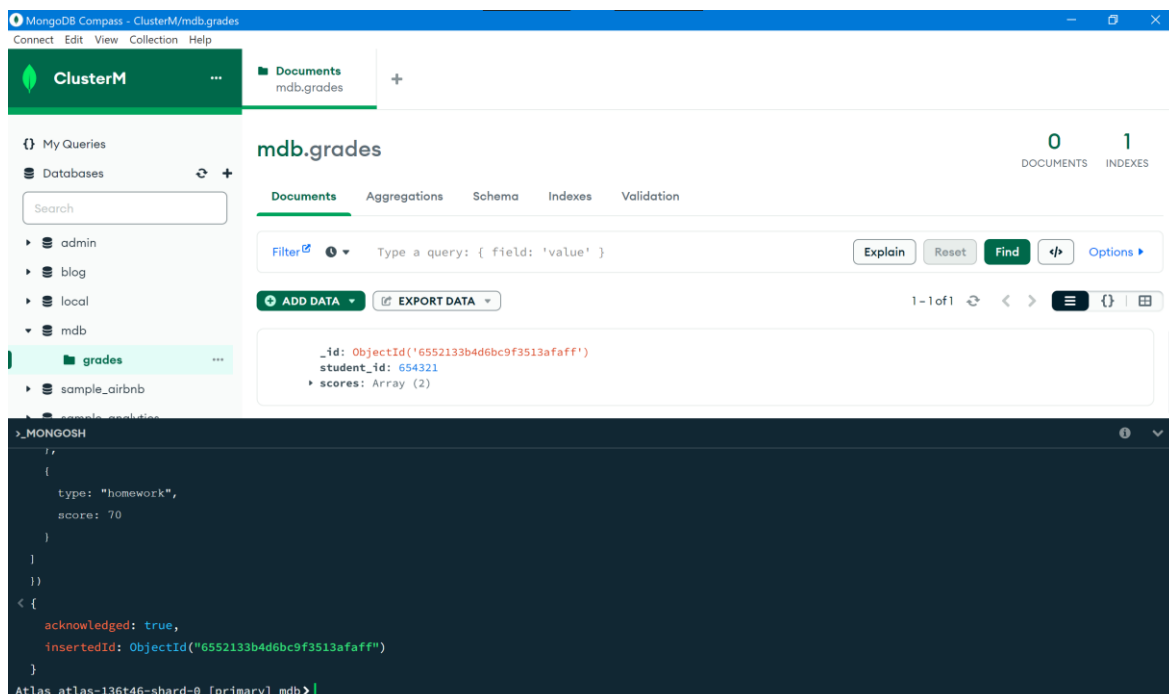
```
db.collection.insertOne({ name: "test" })
```

```
db.collection.insertMany([{ name: "test" }, { }, ...])
```

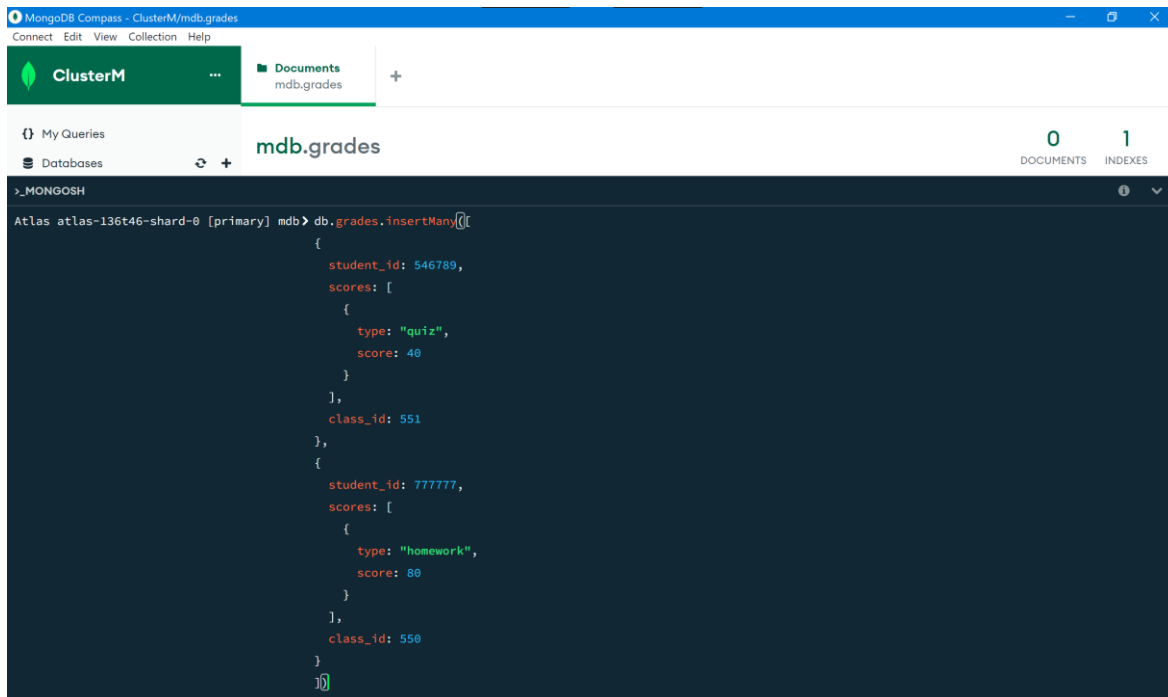


در اینجا یک document (به شکل فرمت JSON نوشته شده و به طور BSON ذخیره می‌شود) ساخته شده و به کالکشن grades اینسرت می‌شود.

نتیجه اجرای کامند:



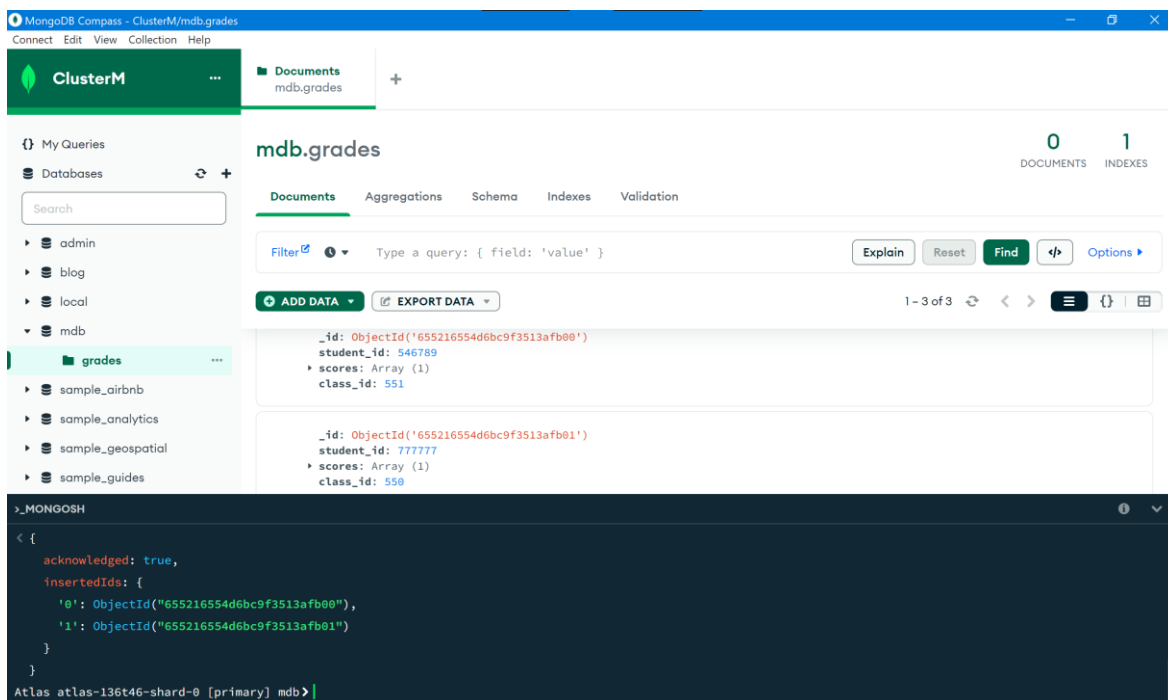
کنون از insertMany استفاده می‌کنیم:



The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'ClusterM' and the database 'mdb.grades'. The left sidebar shows the 'Databases' list with 'mdb' selected. The main panel displays the MongoDB shell with the following command and JSON documents:

```
>_MONGOSH
Atlas atlas-136t46-shard-0 [primary] mdb> db.grades.insertMany([
  {
    student_id: 546789,
    scores: [
      {
        type: "quiz",
        score: 40
      }
    ],
    class_id: 551
  },
  {
    student_id: 777777,
    scores: [
      {
        type: "homework",
        score: 89
      }
    ],
    class_id: 550
  }
])
```

نتیجه اجرا:



The screenshot shows the MongoDB Compass interface after the insertMany operation. The top bar indicates the connection to 'ClusterM' and the database 'mdb.grades'. The left sidebar shows the 'Databases' list with 'mdb' selected. The main panel displays the 'Documents' tab for the 'grades' collection. The documents are shown in a table format:

Document
{ "_id": ObjectId("655216554d6bc9f3513afb00"), "student_id": 546789, "scores": Array (1), "class_id": 551 }
{ "_id": ObjectId("655216554d6bc9f3513afb01"), "student_id": 777777, "scores": Array (1), "class_id": 550 }

The bottom panel shows the MongoDB shell with the following command and JSON document:

```
>_MONGOSH
Atlas atlas-136t46-shard-0 [primary] mdb> {
  acknowledged: true,
  insertedIds: {
    '0': ObjectId("655216554d6bc9f3513afb00"),
    '1': ObjectId("655216554d6bc9f3513afb01")
  }
}
```

Find 2-1

برای یافتن داده‌ها (کوئری زدن) از توابع زیر می‌توان استفاده کرد:

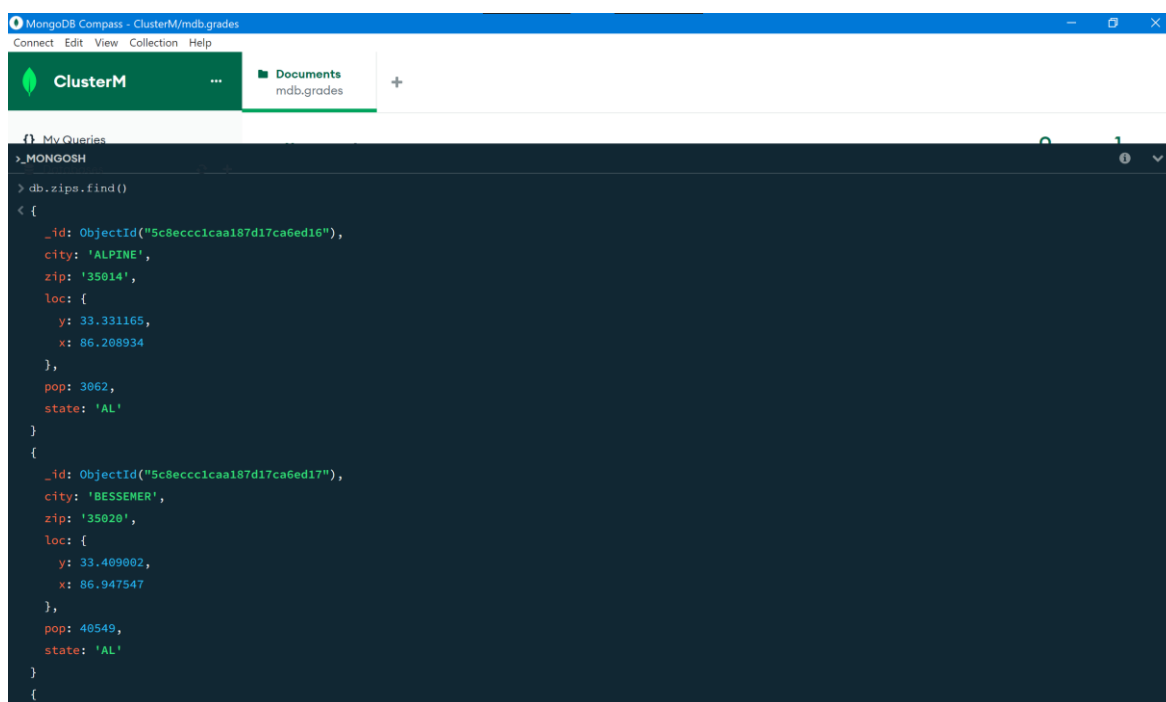
`db.collection.find()`

`db.collection.findOne()`

`db.collection.countDocuments()`

تابع اول همه داده‌هایی که می‌کنند را ریترن می‌کند (در صورت خالی بودن پارامتر، تمامی داکيومنت‌های کالکشن را ریترن می‌کند).

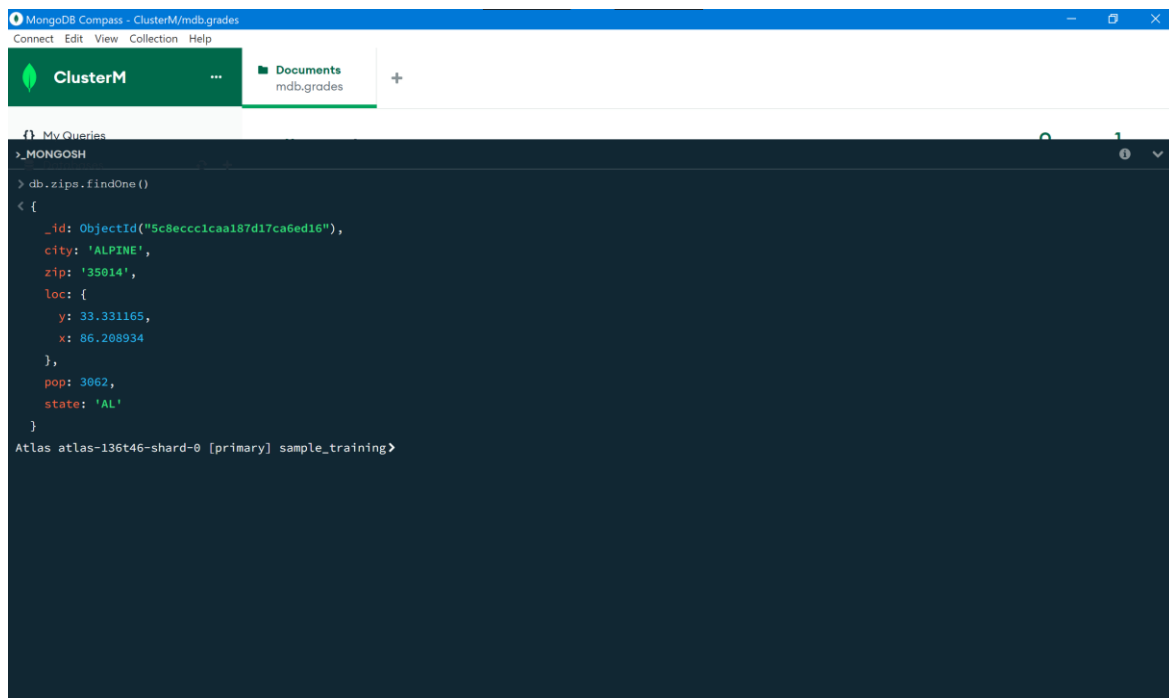
تابع دوم اولین نتیجه را داده و تابع سوم تعداد نتیجه‌ها را می‌گوید.



```
ClusterM ... Documents mdb.grades +
My Queries
>_MONGOSH
> db.zipcodes.find()
< {
  "_id": ObjectId("5c8eccc1caa187d17ca6ed16"),
  "city": 'ALPINE',
  "zip": '35014',
  "loc": {
    "y": 33.331165,
    "x": 86.208934
  },
  "pop": 3062,
  "state": 'AL'
}
{
  "_id": ObjectId("5c8eccc1caa187d17ca6ed17"),
  "city": 'BESSEMER',
  "zip": '35020',
  "loc": {
    "y": 33.409002,
    "x": 86.947547
  },
  "pop": 40549,
  "state": 'AL'
}
}
```

تمامی داکيومنت‌های کالکشن grades را می‌توانیم به طور بالا به دست بیاوریم.

اولین نتیجه فایند با استفاده از `findOne`:



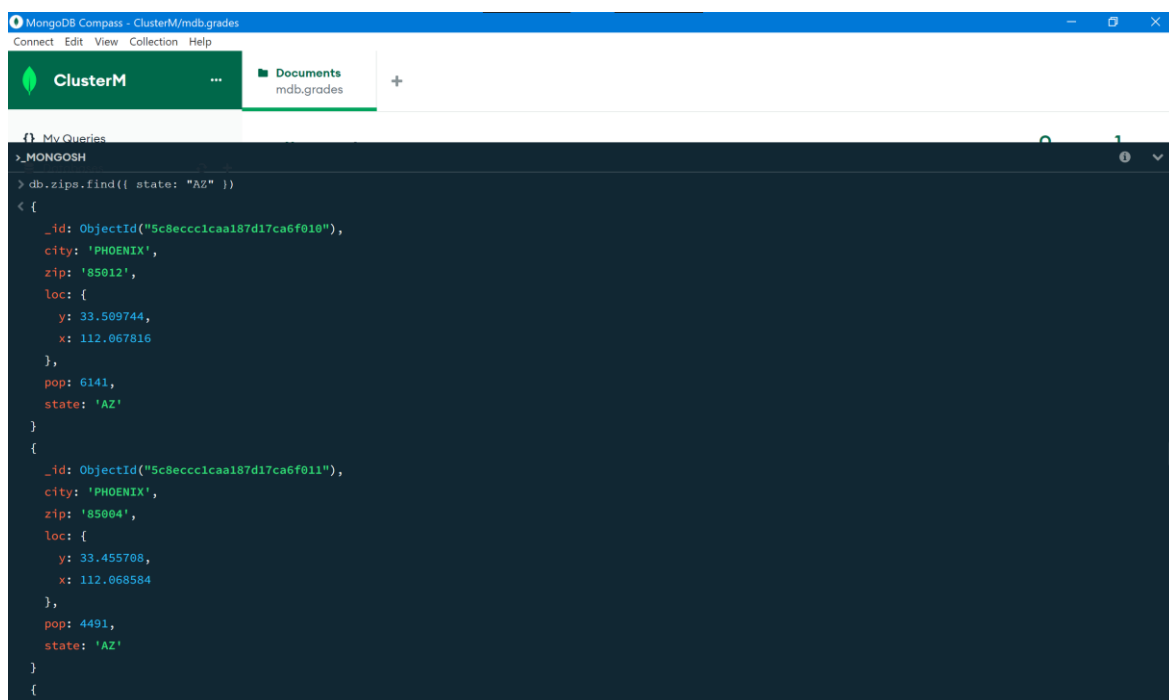
```
ClusterM ... Documents mdb.grades +
My Queries
>_MONGOSH
> db.zips.findOne()
< {
  _id: ObjectId("5c8eccc1caa187d17ca6ed16"),
  city: 'ALPINE',
  zip: '35014',
  loc: {
    y: 33.331165,
    x: 86.208934
  },
  pop: 3062,
  state: 'AL'
}
Atlas atlas-136t46-shard-0 [primary] sample_training>
```

حال می‌توان در کوئری شرطهایی هم اضافه کرد که ساده‌ترین آن \$eq است:

```
db.collection.find({ name: { $eq: "test" } })
```

در این حالت می‌توان \$eq را حذف کرد و به طور معادل داریم:

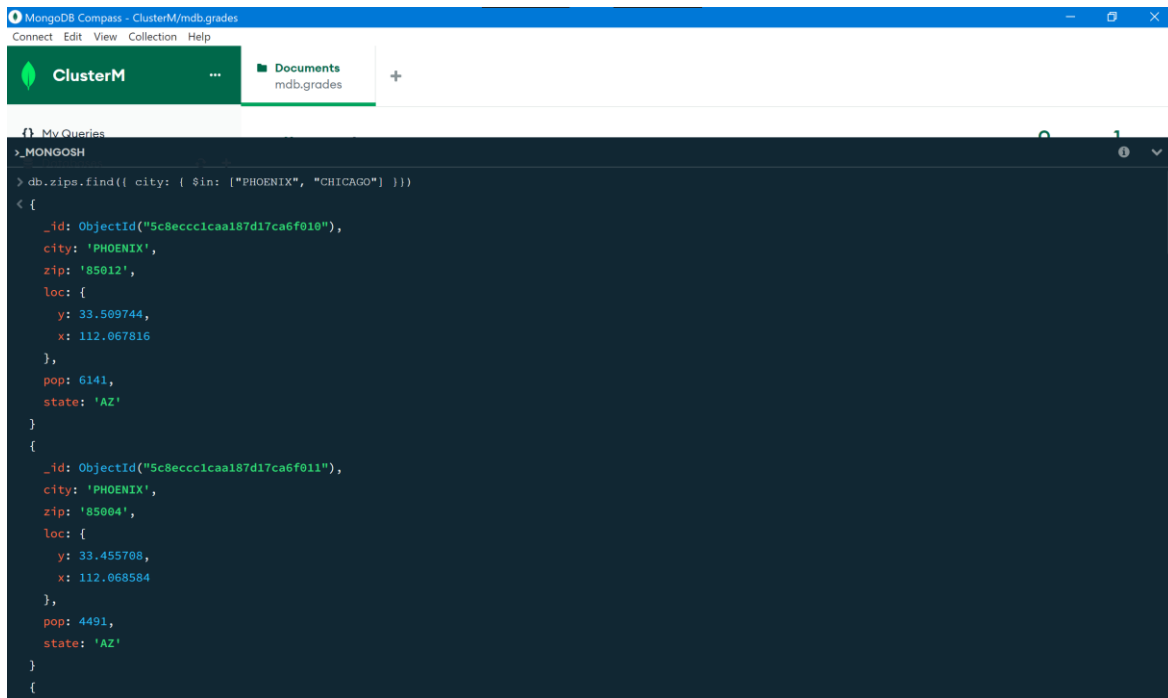
```
db.collection.find({ name: "test" })
```



```
ClusterM ... Documents mdb.grades +
My Queries
>_MONGOSH
> db.zips.find({ state: "AZ" })
< [
  {
    _id: ObjectId("5c8eccc1caa187d17ca6f010"),
    city: 'PHOENIX',
    zip: '85012',
    loc: {
      y: 33.509744,
      x: 112.067816
    },
    pop: 6141,
    state: 'AZ'
  },
  {
    _id: ObjectId("5c8eccc1caa187d17ca6f011"),
    city: 'PHOENIX',
    zip: '85004',
    loc: {
      y: 33.455708,
      x: 112.068584
    },
    pop: 4491,
    state: 'AZ'
  }
]
```

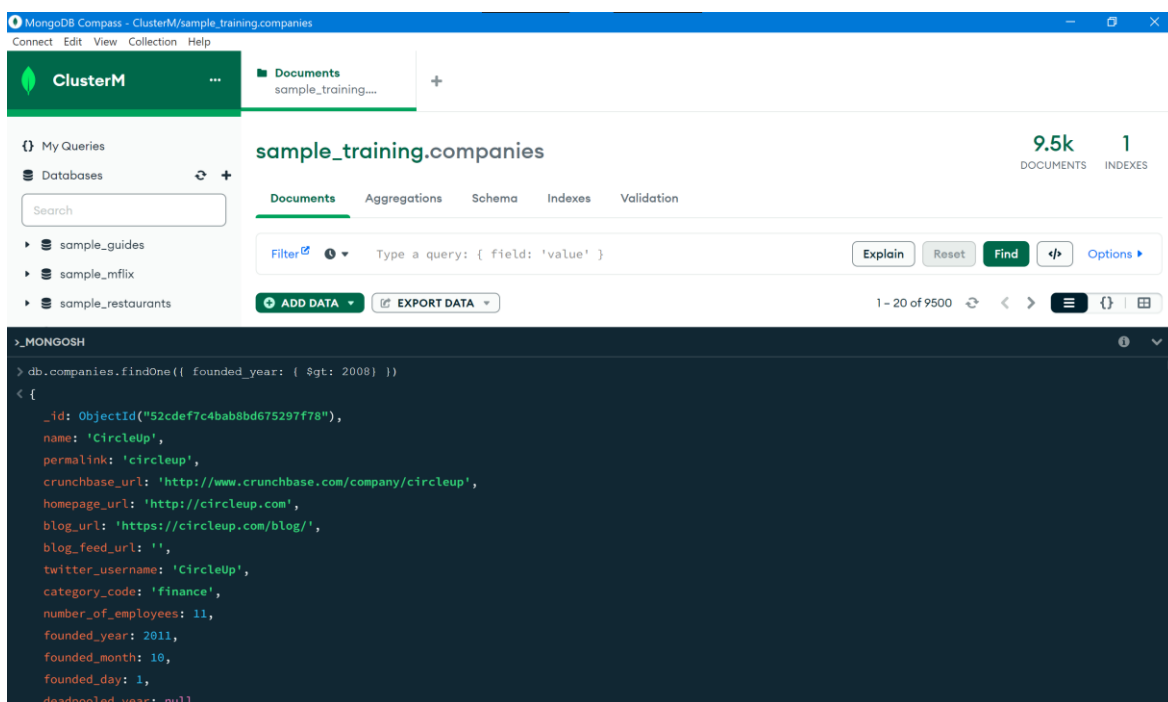
با استفاده از \$in می‌توانیم بودن مقدار در میان عناصر آرایه داده شده را بررسی کنیم:

```
db.collection.find({ name: { $in: ["test", "no"]} })
```



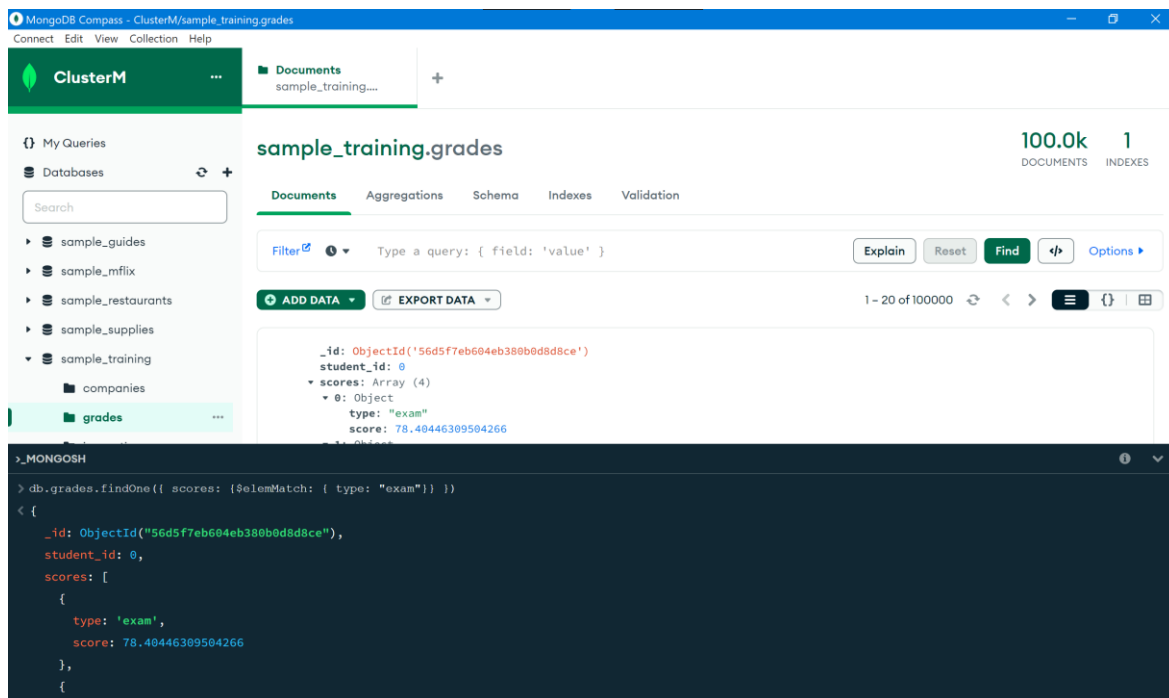
```
>_MONGOSH
> db.zipcodes.find({ city: { $in: ["PHOENIX", "CHICAGO"] }})
< {
  _id: ObjectId("5c8ecccc1caa187d17ca6f010"),
  city: 'PHOENIX',
  zip: '85012',
  loc: {
    y: 33.509744,
    x: 112.067816
  },
  pop: 6141,
  state: 'AZ'
}
{
  _id: ObjectId("5c8ecccc1caa187d17ca6f011"),
  city: 'PHOENIX',
  zip: '85004',
  loc: {
    y: 33.455708,
    x: 112.068584
  },
  pop: 4491,
  state: 'AZ'
}
{
```

مشابه \$eq، می‌توان از \$gt \$gte \$lt \$lte \$ne هم استفاده کرد:

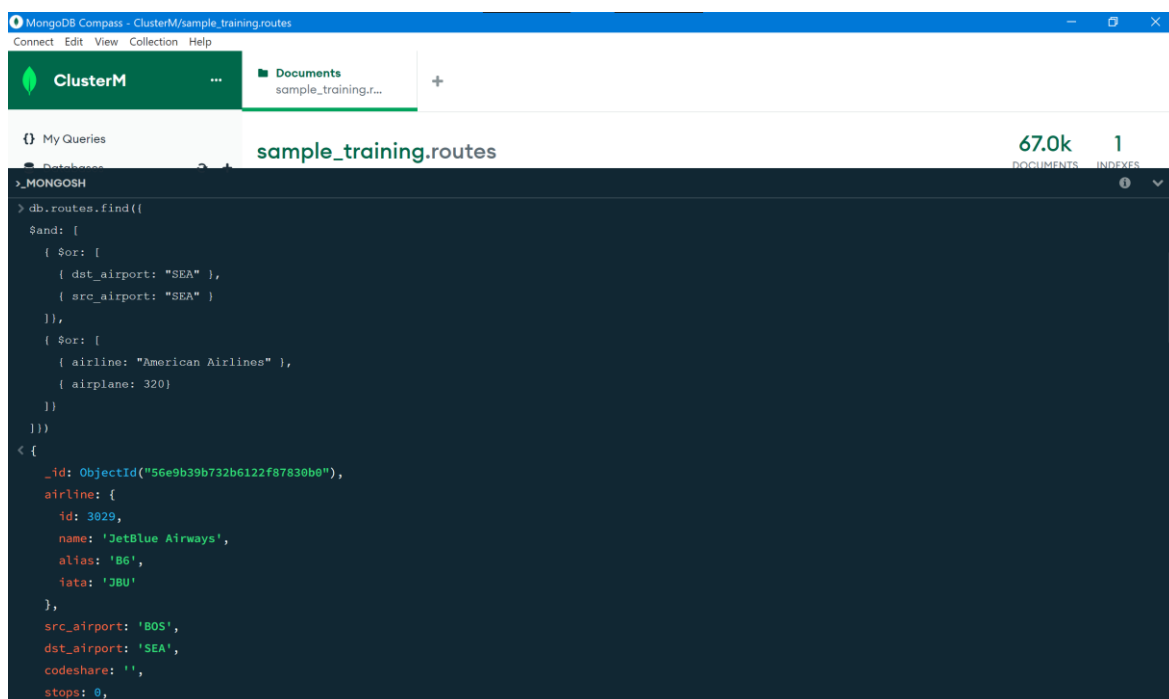


```
>_MONGOSH
> db.companies.findOne({ founded_year: { $gt: 2008 } })
< {
  _id: ObjectId("52cdef7c4bab8bd675297f78"),
  name: 'CircleUp',
  permalink: 'circleup',
  crunchbase_url: 'http://www.crunchbase.com/company/circleup',
  homepage_url: 'http://circleup.com',
  blog_url: 'https://circleup.com/blog/',
  blog_feed_url: '',
  twitter_username: 'CircleUp',
  category_code: 'finance',
  number_of_employees: 11,
  founded_year: 2011,
  founded_month: 10,
  founded_day: 1,
  deadpooled_year: null,
}
```

با استفاده از \$elemMatch می‌توانیم بگوییم که شرط ما روی المان یک آرایه است:



در نهایت، می‌توانیم از logical operator-ها نیز استفاده کنیم که شامل \$and و \$or هستند. استفاده از کاما در اکثر حالات به همان معنی \$and است ولی در برخی حالات (مانند زیر) باید ذکر شود:



3-1 Replace

با استفاده از `replaceOne` می‌توانیم کل یک داکيومنت را با داکيومنتی جدید عوض کنیم.

`db.collection.replaceOne(filter, replacement_document, options)`

برای مثال یک داکيومنت را با استفاده از `_id` آن پیدا می‌کنیم و با چیزی دیگر جایگزین می‌کنیم:

The screenshot shows the MongoDB Compass interface. On the left, the 'Databases' sidebar is open, showing the 'sample_restaurants' database and the 'restaurants' collection. The main panel displays the 'sample_restaurants.restaurants' collection with 25.4k documents and 1 index. The 'Documents' tab is selected, showing a list of documents. The first document is expanded, showing its fields: `_id`, `address`, `borough`, `cuisine`, `grades`, `name`, and `restaurant_id`. Below the document list, the 'MONGODB' shell is open, showing the `replaceOne` command being executed:

```
>_MONGOSH
Atlas atlas-136t46-shard-0 [primary] sample_restaurants> db.restaurants.replaceOne(
  { _id: ObjectId("5eb3d668b31de5d588f4292a") },
  {
    borough: "Brooklyn",
    cuisine: "Italian",
    name: "Rivi Restaurant",
    restaurant_id: "40356018"
  }
)
```

نتیجه اجرا:

The screenshot shows the MongoDB Compass interface after the `replaceOne` command has been executed. The 'MONGODB' shell now displays the result of the operation:

```
>_MONGOSH
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
Atlas atlas-136t46-shard-0 [primary] sample_restaurants>
```

Update 4-1

می‌توان با تابع زیر فیلدهای یک داکيومنت را آپدیت کرد:

`db.collection.updateOne(filter, update_document, options)`

در فیلتر داکيومنت مد نظر را انتخاب می‌کنیم و در `update_document` از دستورات `$set` یا `$push` استفاده می‌کنیم. `$set` به فیلد مقدار جدید می‌دهد و `$push` به آرایه اضافه می‌کند.

در اینجا نام رستوران را آپدیت می‌کنیم:

The screenshot shows the MongoDB Compass interface for the `sample_restaurants.restaurants` collection. The document count is 25.4k and there is 1 index. The document being updated has the following fields: `_id`, `borough` (Brooklyn), `cuisine` (Italian), `name` (Ravi Rant), and `restaurant_id` (40356018). Below the Compass interface, the MONGODB shell shows the execution of the `updateOne` command, which successfully updates the `name` field to "Ravi Rant".

```
> db.restaurants.updateOne(
  { _id: ObjectId("5eb3d668b31de5d588f4292a") },
  { $set: { name: "Ravi Rant" } }
)
{
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}
```

یا با `$push` به یک رستوران `grade` ای اضافه می‌کنیم. قبل از `$push`:

The screenshot shows a MongoDB document with the following fields: `_id`, `address`, `borough` (Brooklyn), `cuisine` (Delicatessen), `grades` (an array of 6 objects), `date` (2011-10-14T00:00:00.000+00:00), `grade` (A), `score` (9), `name` (Wilken's Fine Food), and `restaurant_id` (40356483). The `grades` array contains 6 objects, each with `score` and `date` fields.

```
1  _id: ObjectId('5eb3d668b31de5d588f4292b')
2  address: Object
3  borough: "Brooklyn"
4  cuisine: "Delicatessen"
5  grades: Array (6)
6    0: Object
7      score: 9
8      date: 2011-10-14T00:00:00.000+00:00
9    1: Object
10   2: Object
11   3: Object
12   4: Object
13   5: Object
14     date: 2011-10-14T00:00:00.000+00:00
15     grade: "A"
16     score: 9
17   name: "Wilken's Fine Food"
18   restaurant_id: "40356483"
```

اضافه کردن `grade` جدید:

```

> MONGODB
> db.restaurants.updateOne(
  { _id: ObjectId("5eb3d668b31de5d588f4292b") },
  { $push: {
    grades: { date: "none", grade: "B", score: 5 }
  }
}
)
< {
  acknowledged: true,
  insertedId: null,
  matchedCount: 1,
  modifiedCount: 1,
  upsertedCount: 0
}

```

```

  _id: ObjectId('5eb3d668b31de5d588f4292b')
  address: Object
  borough: "Brooklyn"
  cuisine: "Delicatessen"
  grades: Array (7)
    0: Object
    1: Object
    2: Object
    3: Object
    4: Object
    5: Object
    6: Object
      date: "none"
      grade: "B"
      score: 5
  name: "Wilken'S Fine Food"
  restaurant_id: "40356483"

```

با استفاده از آپشن اختیاری upsert می‌توانیم در صورت مچ نشدن داکيومنتی در ورودی updateOne، به طور خودکار یک داکيومنت جدید ساخته که فیلدهای آپدیت را دارد.

در صورتی که می‌خواهیم پس از آپدیت یک داکيومنت، خود آن را ریترن کنیم، بهتر است به جای اینکه یک بار updateOne و سپس findOne بزنیم، از findAndModify استفاده کنیم:

`db.collection.findAndModify({ query: {...}, update: {...}, new: true/false })`

در صورتی که `new: true` باشد، داکيومنت آپدیت شده را ریترن می‌کند.

برای مثال تعداد کارمندان یک رستوران را یکی اضافه کرده و آن را ریترن می‌کنیم:

- neighborhoods
- restaurants
- sample_supplies
- sample_training
- companies**
- grades

```

1  _id: ObjectId('52cdef7c4bab8bd675297d8a')
2  name: "Wetpaint"
3  permalink: "abc2"
4  crunchbase_url: "http://www.crunchbase.com/company/wetpaint"
5  homepage_url: "http://wetpaint-inc.com"
6  blog_url: "http://digitalquarters.net/"
7  blog_feed_url: "http://digitalquarters.net/feed/"
8  twitter_username: "BachelorWetpaint"
9  category_code: "web"
10 number_of_employees: 47

```

```

> MONGODB
> db.companies.findAndModify(
  query: { _id: ObjectId("52cdef7c4bab8bd675297d8a") },
  update: { $inc: { number_of_employees: 1 } },
  new: true
)
< {
  _id: ObjectId("52cdef7c4bab8bd675297d8a"),
  name: "Wetpaint",
  category_code: 'web',
  number_of_employees: 48,
  founded_year: 2005
}

```

از 47 به 48 افزایش یافت.

به طور مشابه تابع updateMany هم وجود دارد که همه میچها را آپدیت می کند.

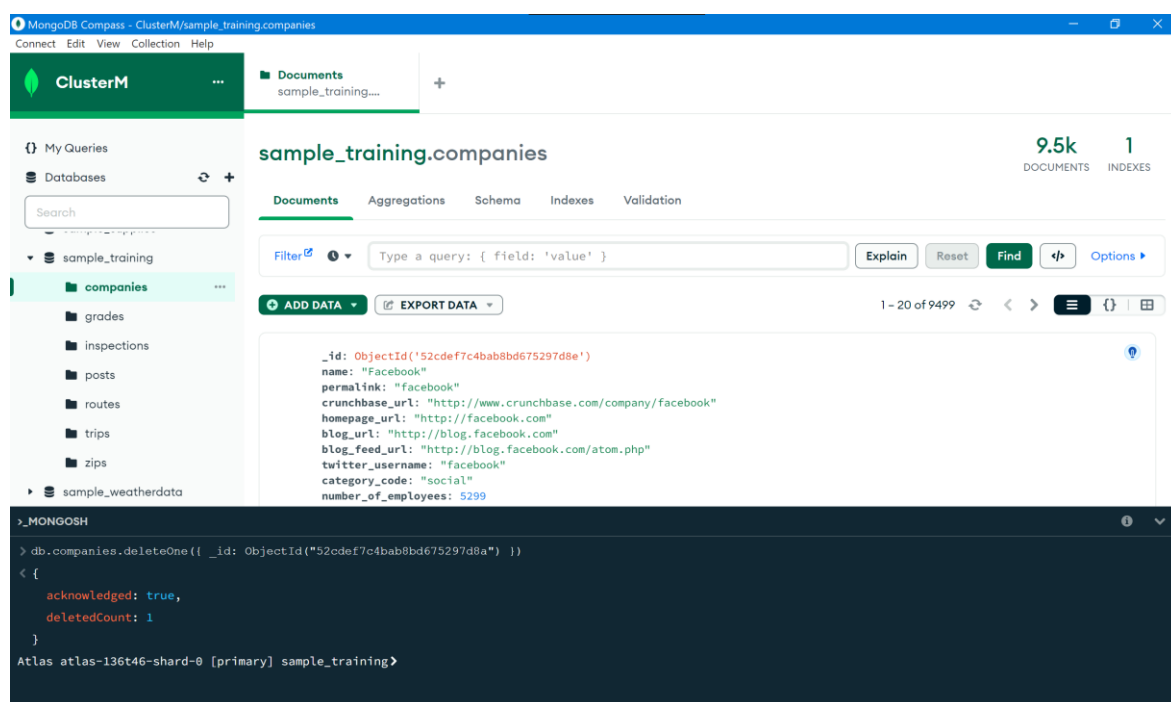
6-1 Delete

برای حذف داکيومنتها، از توابع زیر استفاده می کنیم:

`db.collection.deleteOne(filter)`

`db.collection.deleteMany(filter)`

به طور مثال یک داکيومنت از companies را حذف می کنیم:



7-1 Cursor

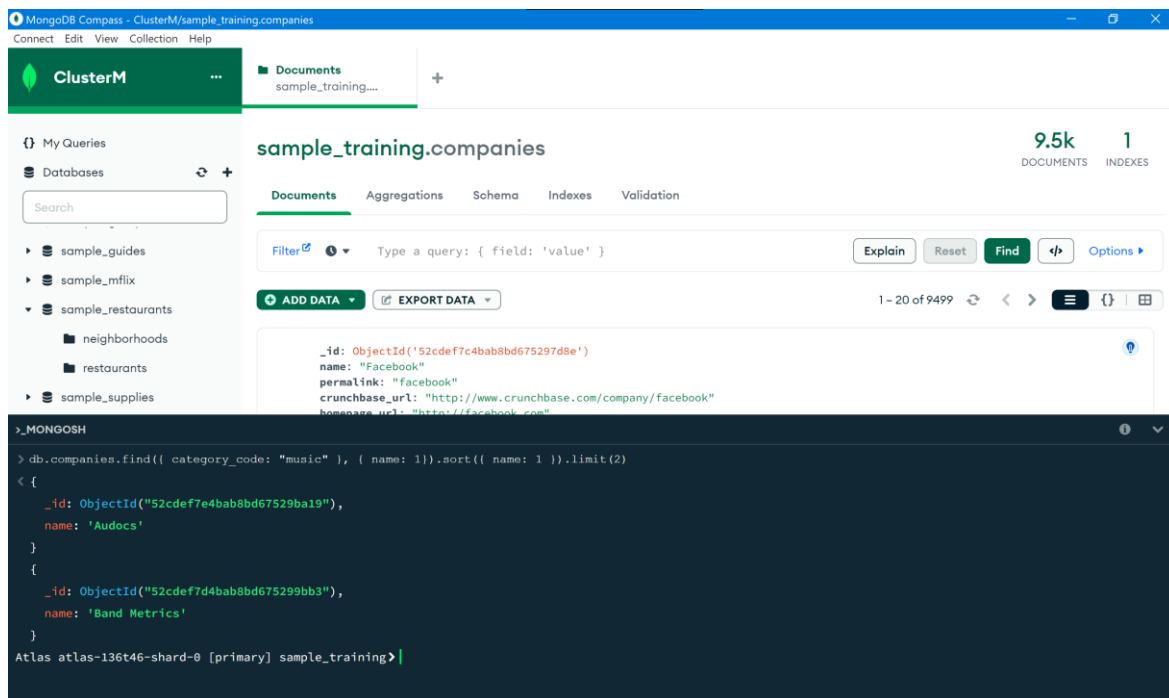
خروجی find، یک cursor است که به مجموعه پاسخهای کوئری اشاره می کند. cursor-ها متدهای مختلفی دارند که از جمله آنها sort و limit است.

`db.collection.find(...).sort({ field: 1/-1 })`

در این حالت خروجی را بر حسب فیلد سورت کرده که اگر 1 بگذاریم به طور ascending و با -1 به طور descending سورت می شود. می توان بر حسب چند فیلد هم سورت کرد.

`db.collection.find(...).limit(n)`

در این حالت تعداد نتایج را محدود می کنیم.



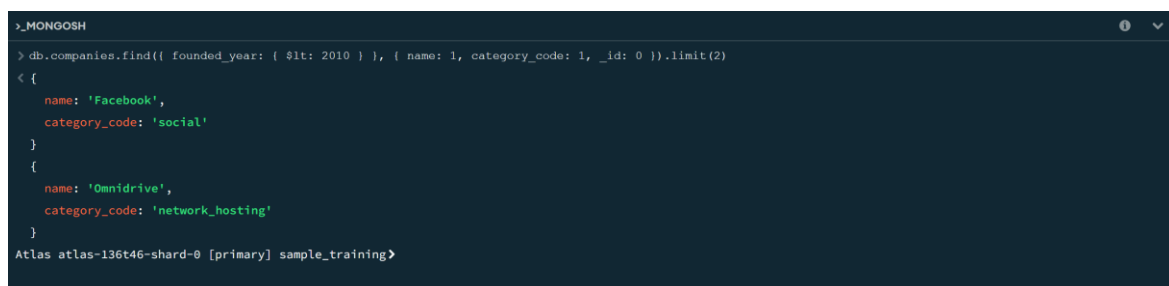
Projection 8-1

می‌توانیم با استفاده از projection فیلدهای داکيومنت‌های خروجی را محدود کنیم.

برای این کار از آرگومان دوم find استفاده می‌کنیم:

```
db.collection.find(filter, { field: 1, field2: 1 })
```

در این حالت فقط فیلد و فیلد2 در پاسخ include می‌شوند. جهت exclude کردن از 0 استفاده می‌کنیم. همه projection-ها یا include باید باشند و یا exclude. تنها استثنا فیلد _id است که می‌تواند در هر نوع projection بیاید.



Aggregation 9-1

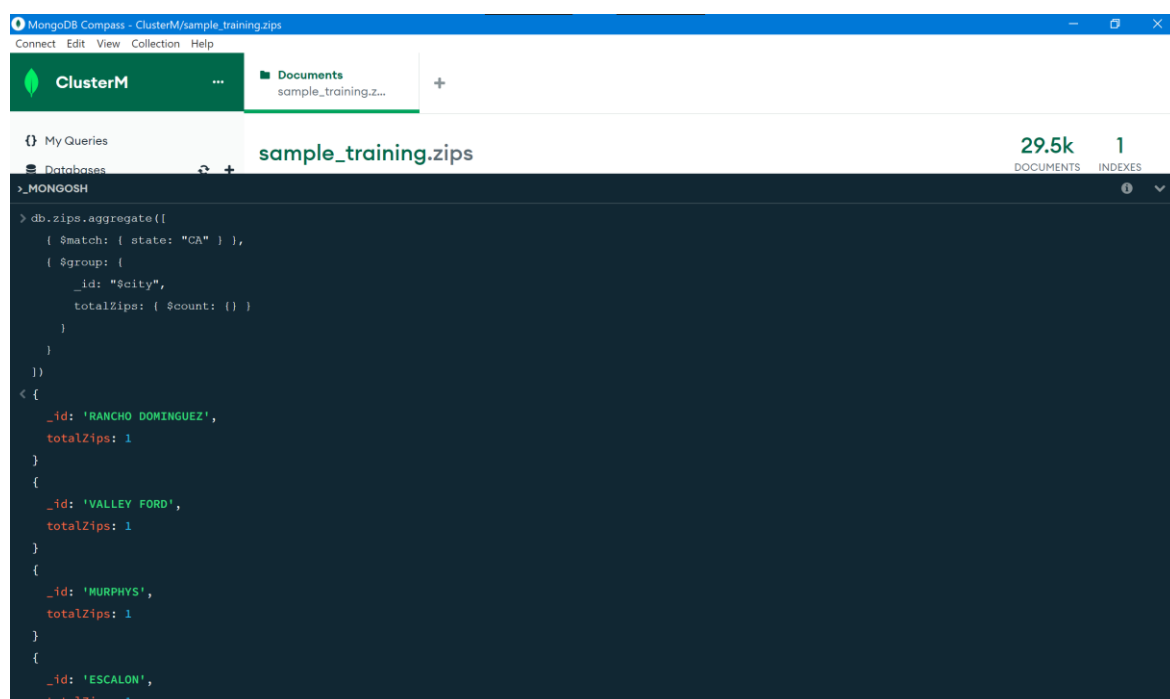
از aggregation برای جمع‌آوری داده و رپورت کردن چیزهایی از جمله تعداد و sum و غیره استفاده می‌شود.

یک پایپ‌لاین aggregation، از چند stage تشکیل شده است که در هر کدام روی داده محاسباتی می‌شود که آن را تغییر نمی‌دهد.

```
db.collection.aggregate([ {stage},... ])
```

از جمله stage‌های یک aggregation، استیج \$match و \$group است. \$group مانند group by در SQL و \$match مانند where می‌باشد.

به عنوان مثال، در میان همه zip‌هایی که در state: CA هستند، آنها را بر حسب city گروه می‌کنیم و در هر کدام تعدادشان را با نام totalZips ریترن می‌کنیم:



The screenshot shows the MongoDB Compass interface. The top bar indicates the connection to 'ClusterM/sample_training.zips'. The left sidebar shows the 'Databases' and 'Collections' list. The main panel displays the aggregation query and its results. The query is:

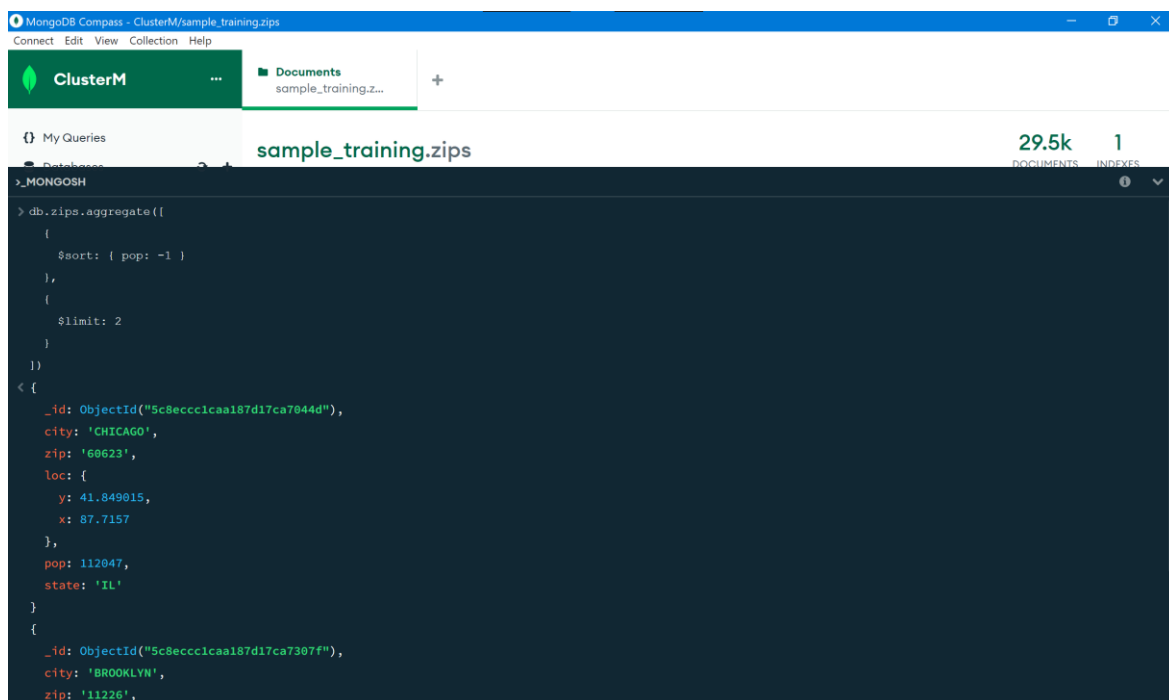
```
db.zips.aggregate([
  { $match: { state: "CA" } },
  { $group: {
    _id: "$city",
    totalZips: { $count: {} }
  } }
])
```

The results show four documents, each representing a city and its total number of zip codes:

```
{
  "_id": "RANCHO DOMINGUEZ",
  "totalZips": 1
},
{
  "_id": "VALLEY FORD",
  "totalZips": 1
},
{
  "_id": "MURPHYS",
  "totalZips": 1
},
{
  "_id": "ESCALON",
  "totalZips": 1
}
```

در این پایپ‌لاین از دو stage پشت سر هم استفاده شده است و در نهایت از استیج \$count استفاده شده است که تعداد را می‌شمارد.

دو stage دیگر، \$sort و \$limit می‌باشند. به طور مثال:

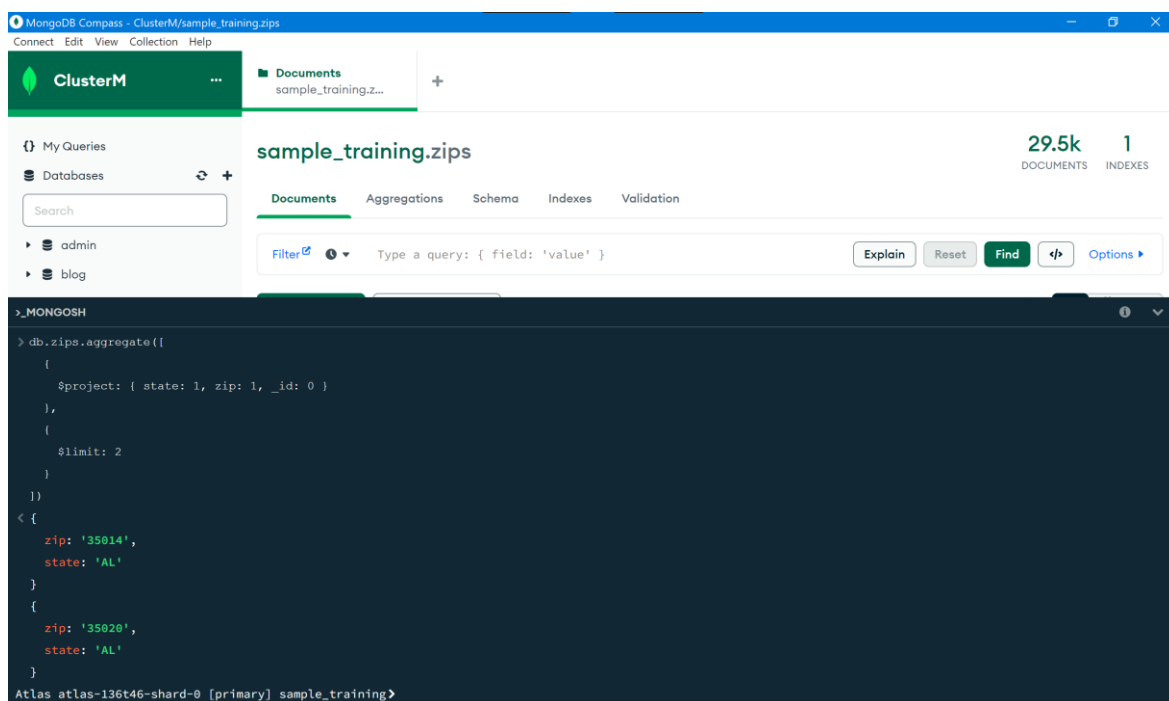


```
> db.zips.aggregate([
  {
    $sort: { pop: -1 }
  },
  {
    $limit: 2
  }
])
< {
  _id: ObjectId("5c8ecc1caa187d17ca7844d"),
  city: 'CHICAGO',
  zip: '60623',
  loc: {
    y: 41.849815,
    x: 87.7157
  },
  pop: 112847,
  state: 'IL'
}
{
  _id: ObjectId("5c8ecc1caa187d17ca7387f"),
  city: 'BROOKLYN',
  zip: '11226',
```

در اینجا ابتدا بر حسب population به طور descending سورت شده و سپس 2 تای اول آن ریترن می شود. دقت می کنیم که ترتیب stage-ها در پایپ لاین مهم است.

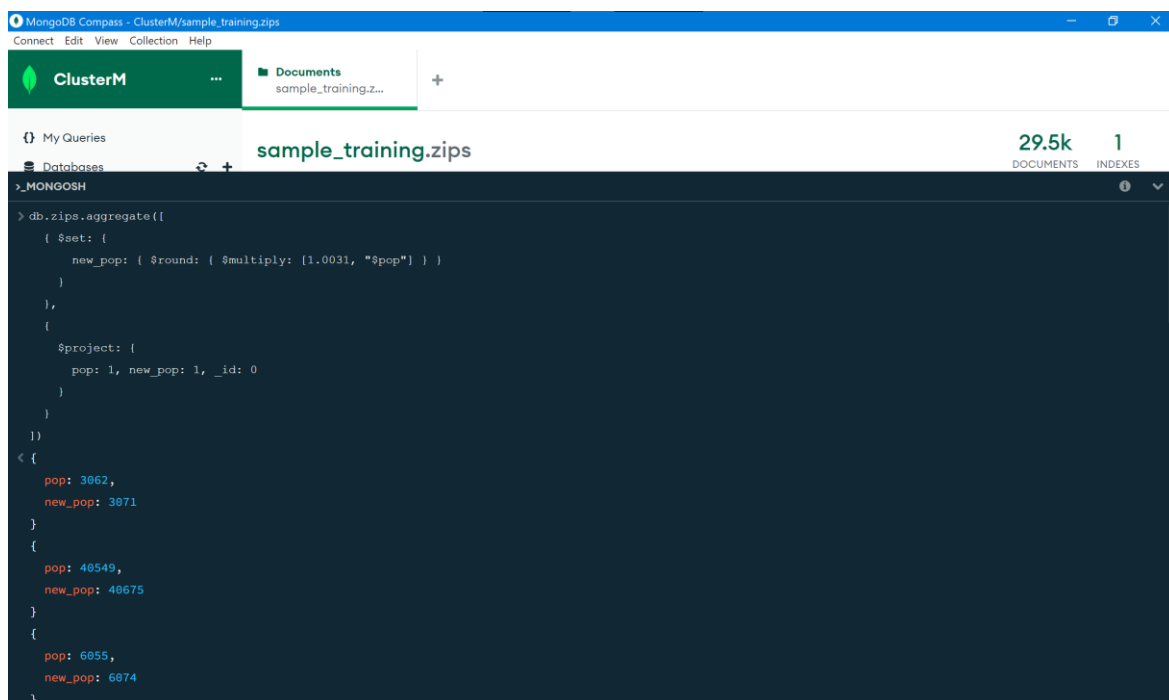
دو stage دیگر، \$project و \$set است.

استیج \$project مانند projection در find است و فیلدهای خاصی را می توانیم نگه داریم یا از خروجی حذف کنیم:



```
> db.zips.aggregate([
  {
    $project: { state: 1, zip: 1, _id: 0 }
  },
  {
    $limit: 2
  }
])
< {
  zip: '35014',
  state: 'AL'
}
{
  zip: '35020',
  state: 'AL'
}
Atlas atlas-136t46-shard-0 [primary] sample_training>
```


استیج \$set مانند استفاده آن در update، فیلدها را آپدیت کرده یا فیلد جدید می‌سازد.
در مثال زیر، جمعیت آیتم‌ها را 0.31٪ بیشتر می‌کنیم و در فیلدی جدید ذخیره می‌کنیم:

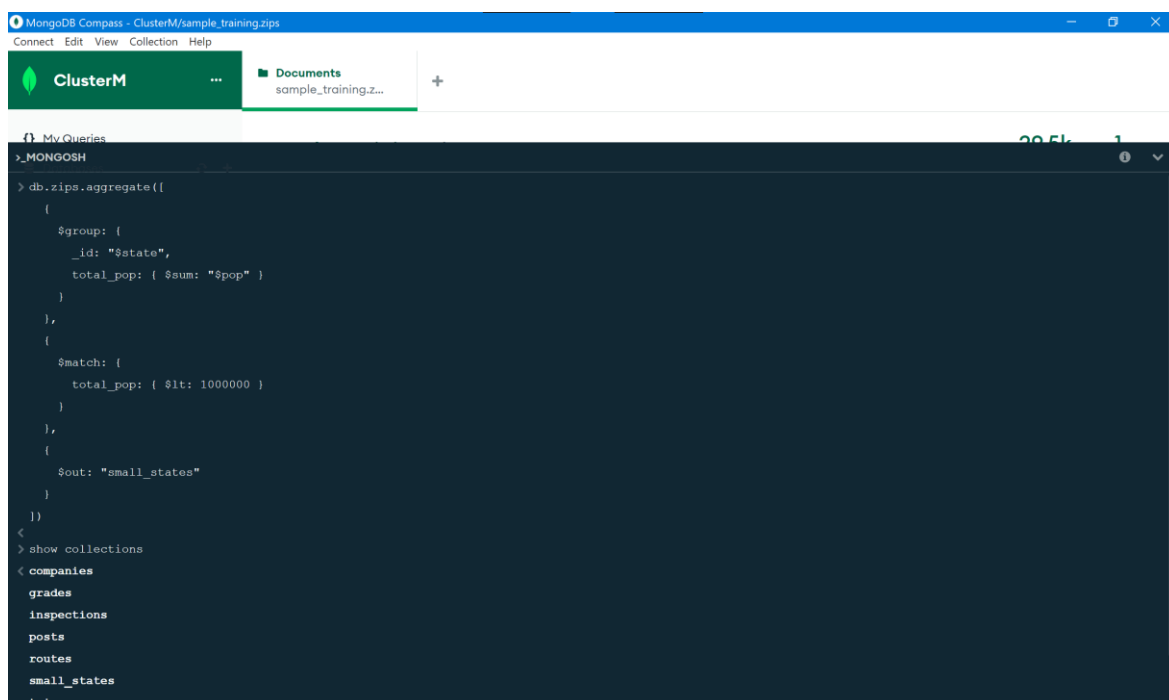


```

> db.zips.aggregate([
  {
    $set: {
      new_pop: { $round: ( { $multiply: [1.0031, "$pop"] } ) }
    }
  },
  {
    $project: {
      pop: 1, new_pop: 1, _id: 0
    }
  }
])
< {
  pop: 3062,
  new_pop: 3071
}
{
  pop: 40549,
  new_pop: 40675
}
{
  pop: 6055,
  new_pop: 6074
}

```

در نهایت استیج \$out را داریم که در صورت استفاده، باید آخرین stage پایپ‌لاین باشد. این استیج، خروجی پایپ‌لاین را به یک collection جدید می‌ریزد.



```

> db.zips.aggregate([
  {
    $group: {
      _id: "$state",
      total_pop: { $sum: "$pop" }
    }
  },
  {
    $match: {
      total_pop: { $lt: 1000000 }
    }
  },
  {
    $out: "small_states"
  }
])
<
> show collections
< companies
  grades
  inspections
  posts
  routes
  small_states
  trips

```

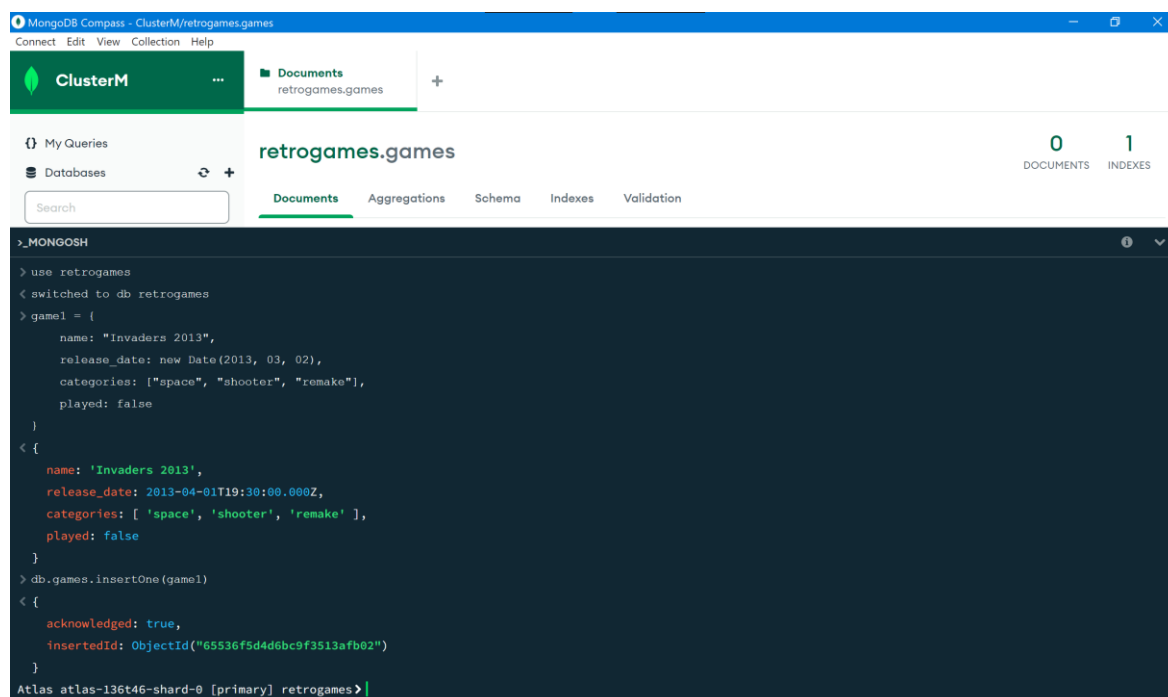
در اینجا، ابتدا بر حسب state گروه شدند و sum فیلد pop را با نام total_pop می‌گیریم. سپس آنهایی که جمع جمعیت‌شان کمتر از 1000000 است را فیلتر کرده و نتیجه را در collection جدیدی به نام small_states می‌ریزیم.

وجود این کالکشن جدید با کامند show collections نشان داده شده است.

پاسخ 2. پیاده‌سازی retrogames

1-2. ساخت و تست پایگاه داده

ابتدا دیتابیس retrogames را ساخته و در آن دو collection به نام‌های games و players می‌سازیم.
حال یک بازی به games اضافه می‌کنیم:



```
> use retrogames
< switched to db retrogames
> game1 = {
  name: "Invaders 2013",
  release_date: new Date(2013, 03, 02),
  categories: ["space", "shooter", "remake"],
  played: false
}
< {
  name: 'Invaders 2013',
  release_date: 2013-04-01T19:30:00.000Z,
  categories: [ 'space', 'shooter', 'remake' ],
  played: false
}
> db.games.insertOne(game1)
< {
  acknowledged: true,
  insertedId: ObjectId("65536f5d4d6bc9f3513afb02")
}
Atlas atlas-136t46-shard-0 [primary] retrogames>
```

```
▶ _id: ObjectId('65536f5d4d6bc9f3513afb02')
  name: "Invaders 2013"
  release_date: 2013-04-01T19:30:00.000+00:00
  ▶ categories: Array (3)
  played: false
```



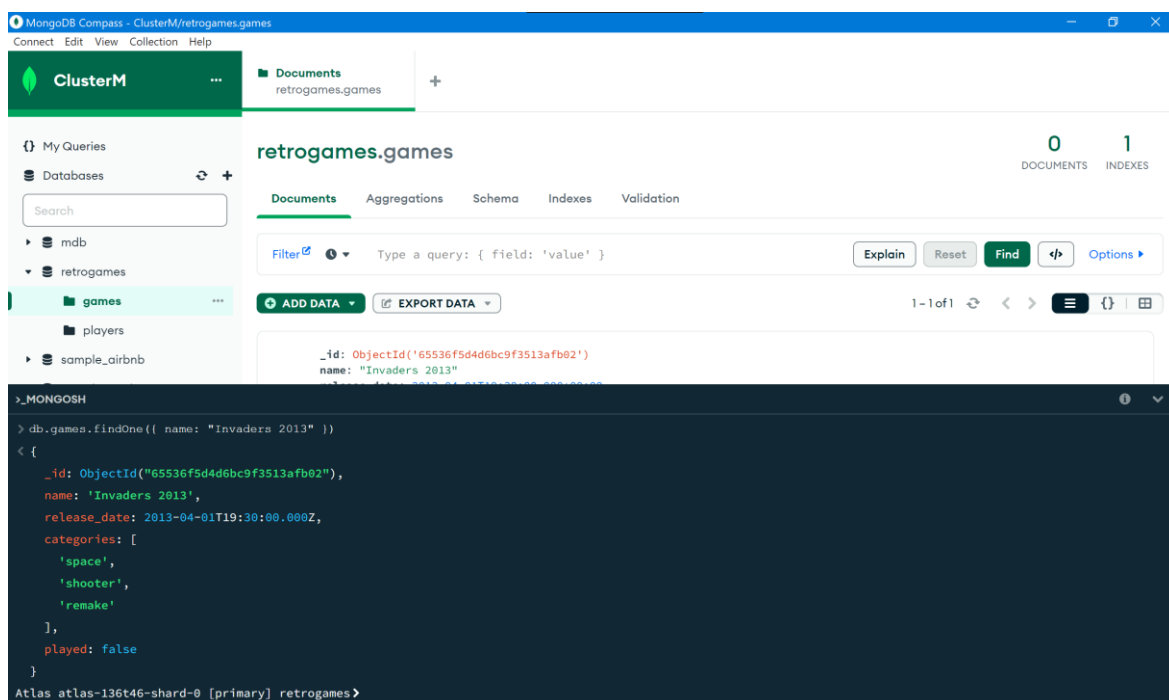
```
> _MONGOSH
> db.games.find()
< {
  _id: ObjectId("65536f5d4d6bc9f3513afb02"),
  name: 'Invaders 2013',
  release_date: 2013-04-01T19:30:00.000Z,
  categories: [
    'space',
    'shooter',
    'remake'
  ],
  played: false
}
Atlas atlas-136t46-shard-0 [primary] retrogames>
```

همانطور که می‌بینیم یک بازی بیشتر وجود ندارد.

با استفاده از دستور زیر نیز همین خروجی را می‌گیریم:

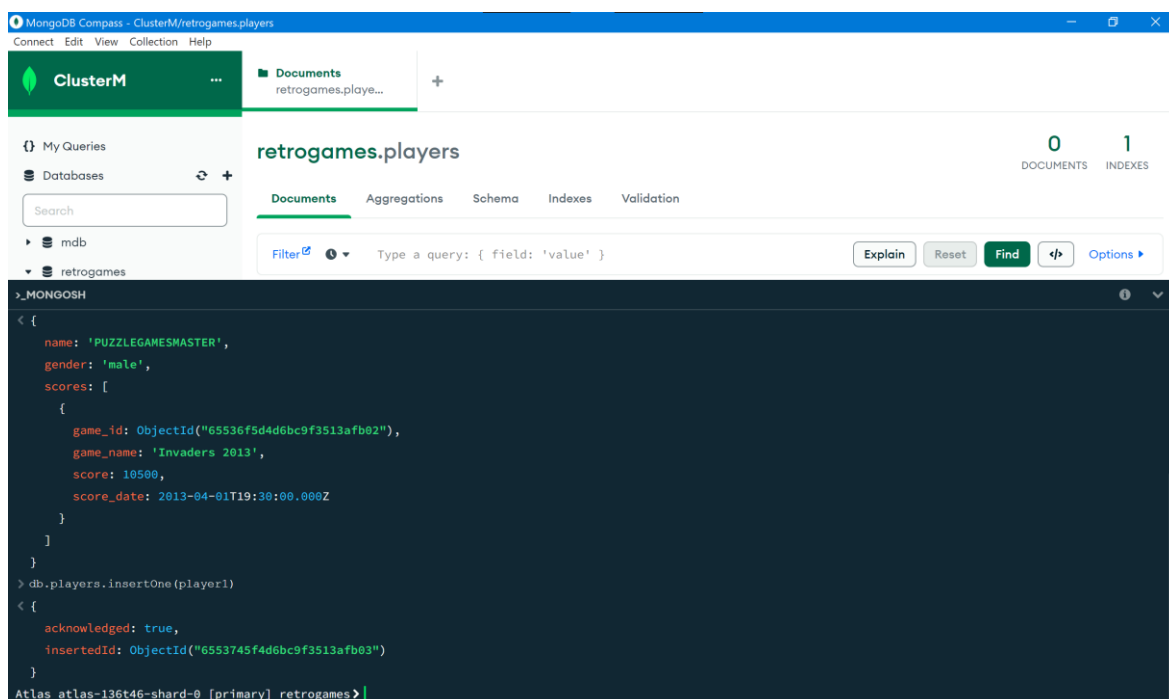
```
db.games.find().limit(100)
```

با استفاده از دستور findOne می‌توانیم بازی اینسرت شده را پیدا کنیم:



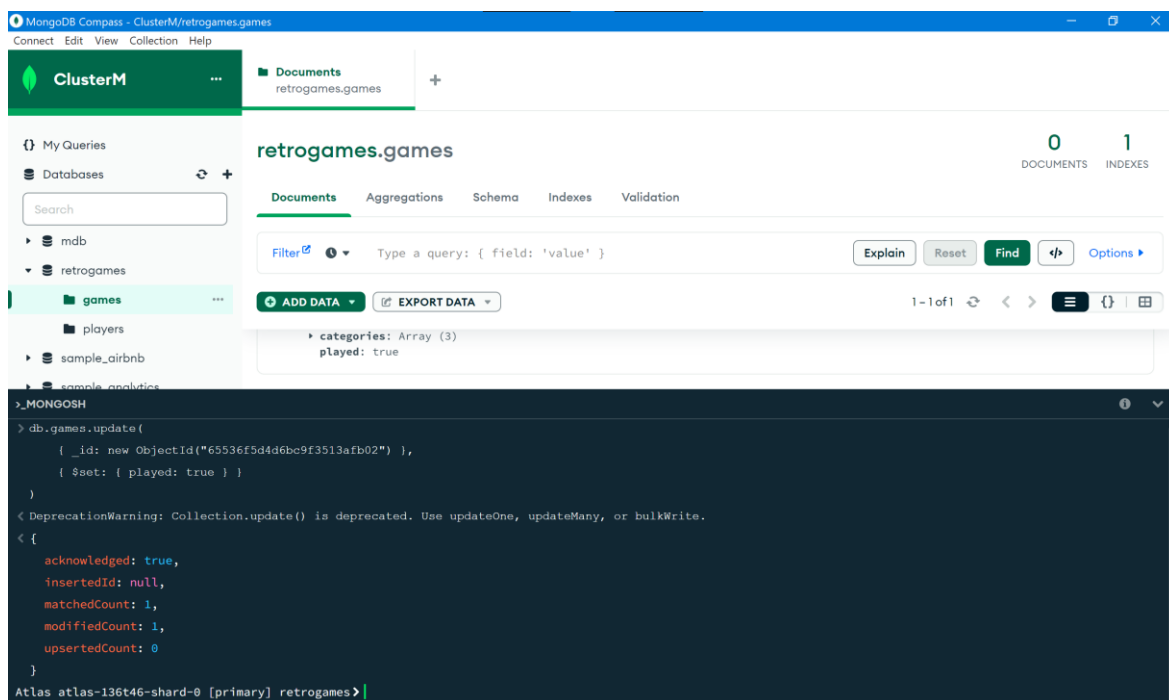
```
> db.games.findOne({ name: "Invaders 2013" })
< {
  _id: ObjectId("65536f5d4d6bc9f3513afb02"),
  name: 'Invaders 2013',
  release_date: 2013-04-01T19:30:00.000Z,
  categories: [
    'space',
    'shooter',
    'remake'
  ],
  played: false
}
```

اکنون یک player اضافه می‌کنیم که در آن بازی ساخته شده شرکت کرده است:

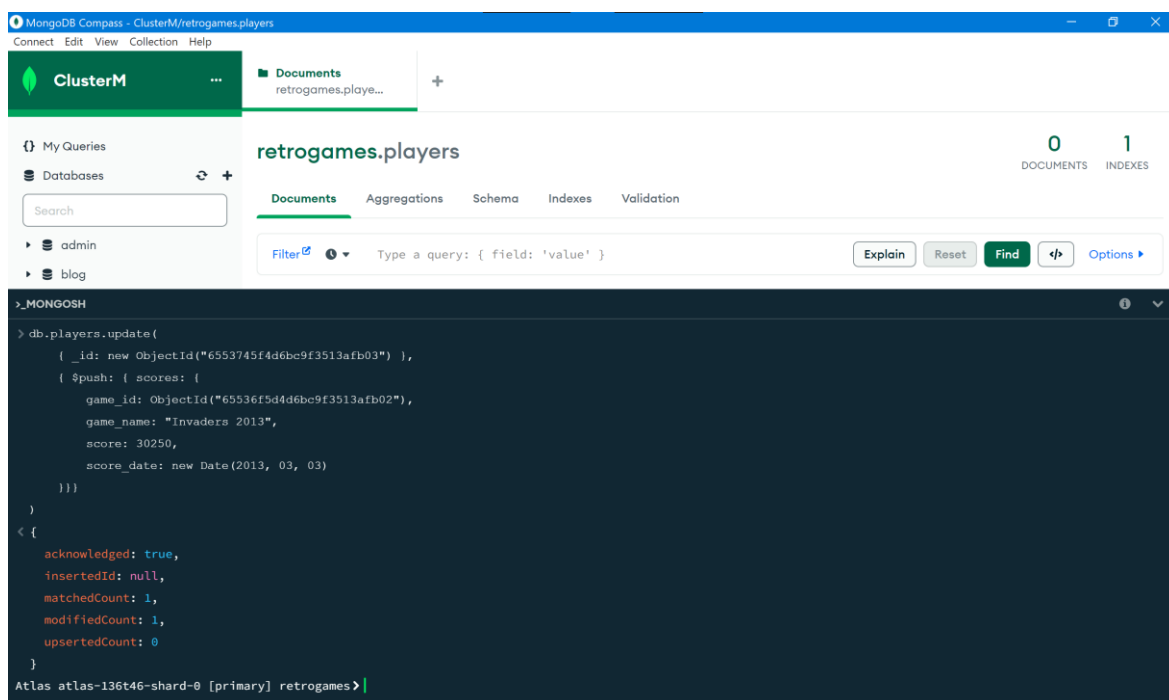


```
> db.players.insertOne(player1)
< {
  acknowledged: true,
  insertedId: ObjectId("6553745f4d6bc9f3513afb03")
}
```

همانطور که می‌بینیم علاوه بر id بازی ساخته شده قبلی، نام آن هم ذخیره کردیم که redundancy است. حال از آنجا که یکی بازی کرده است، فیلد played بازی را به true آپدیت می‌کنیم:



حال یک score جدید برای player برای همین بازی اضافه می‌کنیم. از آنجا که فیلد scores یک آرایه است از \$push استفاده می‌کنیم:



می‌توانستیم به جای \$push از \$addToSet استفاده کنیم که در این صورت، هنگام اضافه کردن به آرایه، تکراری بودن المان را نیز بررسی می‌کند و اجازه پوش کردن عضو تکراری را نمی‌دهد.

در نهایت، player یک داکيومنت به شکل زیر خواهد داشت:

```
_id: ObjectId('6553745f4d6bc9f3513afb03')
name: "PUZZLEGAMESMASTER"
gender: "male"
▼ scores: Array (2)
  ▼ 0: Object
    game_id: ObjectId('65536f5d4d6bc9f3513afb02')
    game_name: "Invaders 2013"
    score: 10500
    score_date: 2013-04-01T19:30:00.000+00:00
  ▼ 1: Object
    game_id: ObjectId('65536f5d4d6bc9f3513afb02')
    game_name: "Invaders 2013"
    score: 30250
    score_date: 2013-04-02T19:30:00.000+00:00
```