Lecture 8: More MQL

Contents

DSCI 513

Databases & Data Retrieval

Lecture outline

- Operators in MongoDB
- Query conditionals
- Querying arrays and sub-documents

```
from pymongo import MongoClient
import json

with open('data/credentials_mongodb.json') as f:
    login = json.load(f)

client = MongoClient(**login)
```

Skip to main content

`$` and operators in MongoDB

In MongoDB, operators are denoted with a dollar sign `$`. For example, the `$sum` operator which is used in aggregation pipelines, or comparison operators such as `$gte` which is equivalent to `>=` in SQL or Python.

Comparison operators

`$gt`, `$gte`, `$lt`, `$lte`

These operators have the same meaning as `>`, `>=`, `<`, and `<=` in SQL or Python. For example, while a filter `{'runtime': 200}` would return documents whose runtime is exactly 200 minutes, `{'runtime': {'$gte': 200}}` would return documents whose runtime is greater that 200 minutes.

Example: Return the title, runtime, and production year of 5 movies with a runtime of 200 minutes or greater.

```
list(
    client['sample_mflix']['movies'].find(
        filter={'runtime': {'$gte': 200}},
        projection={'_id': 0, 'title': 1, 'runtime': 1, 'year': 1},
        limit=5
    )
```

Skip to main content

```
[{'runtime': 399, 'title': 'Les vampires', 'year': 1915},
 {'runtime': 240, 'title': 'Napoleon', 'year': 1927},
 {'runtime': 281, 'title': 'Les Misèrables', 'year': 1934},
 {'runtime': 245, 'title': 'Flash Gordon', 'year': 1936},
 {'runtime': 238, 'title': 'Gone with the Wind', 'year': 1939}]
```

Example: How many movies are there with a runtime of 200 minutes or greater?

```
client['sample_mflix']['movies'].count_documents(filter={'runtime': {'$gte': 2
```

```
227
```

$ne

The `$ne` (not equal) operator has the same meaning as `<>` in SQL or `!=` in Python.

Example: Find the title and the type of 5 documents in the `movies` collection that are not of type `movie`.

```
list(
    client['sample_mflix']['movies'].find(
        filter={'type': {'$ne': 'movie'}},
        projection={'_id': 0, 'title': 1, 'type': 1},
        limit=5
    )
)
```

Skip to main content

```
[{'title': 'The Forsyte Saga', 'type': 'series'},
 {'title': 'Scenes from a Marriage', 'type': 'series'},
 {'title': 'Ironiya sudby, ili S legkim parom!', 'type': 'series'},
 {'title': 'I, Claudius', 'type': 'series'},
 {'title': 'Sybil', 'type': 'series'}]
```

Note that the `type` field of none of the returned documents is `movie` (all of them are `series` because that's the only other type that exists in the documents of the `movies` collection).

`$in`, `$nin`

These two operators are equivalent to `IN` and `NOT IN` in SQL, or `in` and `not in` in Python. They are used to check if the value of a field is equal (or not equal) to any value in a given list. What these operators do can also be imitated with `$or` or `$nor`, but these are more concise.

Example: Return the title, production year, and the cast of these movies: The Sixth Sense, Imitation Game, The Red Violin, Match Point, Forrest Gump.

```
list(
    client["sample_mflix"]["movies"].find(
        filter={
            "title": {
                "$in": [
                    "The Sixth Sense",
                    "Imitation Game",
                    "The Red Violin",
                    "Match Point",
                    "Forrest Gump",
                ],
            },
        },
        projection={"_id": 0, "title": 1, "cast": 1, "year": 1},
        limit=5,
    )
)
```

Skip to main content

```
[{'year': 1994,
  'title': 'Forrest Gump',
  'cast': ['Tom Hanks',
   'Rebecca Williams',
   'Sally Field',
   'Michael Conner Humphreys']},
 {'cast': ['Carlo Cecchi',
   'Irene Grazioli',
   'Anita Laurenzi',
   'Tommaso Puntelli'],
  'title': 'The Red Violin',
  'year': 1998},
 {'year': 1999,
  'title': 'The Sixth Sense',
  'cast': ['Bruce Willis',
   'Haley Joel Osment',
   'Toni Collette',
   'Olivia Williams']},
 {'year': 2005,
  'title': 'Match Point',
  'cast': ['Jonathan Rhys Meyers',
   'Alexander Armstrong',
   'Paul Kaye',
   'Matthew Goode']}]
```

Example: Find the number of movies that are not available in any of these languages: English, French, Italian or German.

```
client["sample_mflix"]["movies"].count_documents(
    filter={'languages': {'$nin': ['English', 'French', 'German', 'Italian']}}
)
```

```
4888
```

Skip to main content

Logical operators

Implicit AND, `$and`

In MongoDB, specifying multiple conditions on a field or multiple fields implies an implicit logical AND and we don't need to explicitly need to use `$and`. For example, in these filters

```
{'year': {'$gte': 2000, '$lte': 2010}}
```

or

```
{'year': 2010, 'directors': 'Woody Allen'}
```

a logical AND is implied between the multiple given conditions.

Example: Using the `sample_supplies` database, return 2 documents associated with sales that are made in Seattle and used a coupon. Exclude the `items` field from the results.

```
list(
    client["sample_supplies"]["sales"].find(
        filter={
            'storeLocation': 'Seattle',
            'couponUsed': True
        },
        projection={"_id": 0, "items": 0},
        limit=2,
    )
)
```

Skip to main content

```
  'customer': {'gender': 'F',
   'age': 26,
   'email': 'kupeen@gareha.ne',
   'satisfaction': 5},
  'couponUsed': True,
  'purchaseMethod': 'In store'},
 {'saleDate': datetime.datetime(2016, 10, 22, 18, 50, 44, 216000),
  'storeLocation': 'Seattle',
  'customer': {'gender': 'F',
   'age': 52,
   'email': 'nupzig@apubu.hu',
   'satisfaction': 3},
  'couponUsed': True,
  'purchaseMethod': 'In store'}]
```

The above query is equivalent to:

```
list(
    client["sample_supplies"]["sales"].find(
        filter={
            '$and': [
                {'storeLocation': 'Seattle'},
                {'couponUsed': True}
            ]
        },
        projection={"_id": 0, "items": 0},
        limit=2,
    )
)
```

```
[{'saleDate': datetime.datetime(2016, 5, 16, 3, 43, 15, 808000),
  'storeLocation': 'Seattle',
  'customer': {'gender': 'F',
   'age': 26,
   'email': 'kupeen@gareha.ne',
   'satisfaction': 5},
  'couponUsed': True,
  'purchaseMethod': 'In store'},
 {'saleDate': datetime.datetime(2016, 10, 22, 18, 50, 44, 216000),
  'storeLocation': 'Seattle',
  'customer': {'gender': 'F',
   'age': 52,
   'email': 'nupzig@apubu.hu',
   'satisfaction': 3},
  'couponUsed': True,
  'purchaseMethod': 'In store'}]
```

Skip to main content

Example: Using the `sample_supplies` database, find the number of sales made between October 1, 2014 and December 1, 2014.

```
import datetime

client["sample_supplies"]["sales"].count_documents(
        filter={
            'saleDate': {
                '$gte': datetime.datetime(2014, 10, 1),
                '$lte': datetime.datetime(2014, 12, 1)
            }
        }
)
```

```
153
```

> **Note:** When specifying multiple conditions for a field, all conditions should be put into a single query document for that field (or an `$and` operator should be used). For example, the following query only checks the **last condition on the field** and returns unexpected results:

```
import datetime

client["sample_supplies"]["sales"].count_documents(
        filter={
            'saleDate': {'$gte': datetime.datetime(2014, 10, 1)},
            'saleDate': {'$lte': datetime.datetime(2014, 12, 1)}
    }
)
```

```
1861
```

Skip to main content

`$or`, `$nor`

These are general logical OR/NOR operators. The difference with `$in`/`$nin` is that `$in`/`$nin` can be used for an equality condition on a single field, but `$or`/`$nor` are general and can be used with any boolean expression.

Example: Using the `sample_supplies` database, find 5 sales that were made in New York and were either paid by phone or didn't use a coupon.

```
list(
    client["sample_supplies"]["sales"].find(
        filter={
            'storeLocation': 'New York',
            '$or': [
                {'purchaseMethod': 'Phone'},
                {'couponUsed': False}
            ]
        },
        projection={"_id": 0, "items": 0},
        limit=5,
    )
)
```

```
[{'saleDate': datetime.datetime(2017, 3, 21, 1, 54, 26, 657000),
  'storeLocation': 'New York',
  'customer': {'gender': 'M',
   'age': 26,
   'email': 'rapifoozi@viupoen.bb',
   'satisfaction': 5},
  'couponUsed': True,
  'purchaseMethod': 'In store'},
 {'saleDate': datetime.datetime(2013, 8, 21, 9, 36, 7, 188000),
  'storeLocation': 'New York',
  'customer': {'gender': 'F',
   'age': 53,
   'email': 'se@nacwev.an',
   'satisfaction': 4},
  'couponUsed': False,
```

Skip to main content

```
  'storeLocation': 'New York',
  'customer': {'gender': 'M',
   'age': 34,
   'email': 'bubbecgu@odidecned.tf',
   'satisfaction': 3},
  'couponUsed': False,
  'purchaseMethod': 'Phone'},
 {'saleDate': datetime.datetime(2015, 3, 6, 19, 13, 35, 155000),
  'storeLocation': 'New York',
  'customer': {'gender': 'F',
   'age': 42,
   'email': 'jecosab@copfatma.af',
   'satisfaction': 3},
  'couponUsed': False,
  'purchaseMethod': 'Phone'},
 {'saleDate': datetime.datetime(2015, 10, 14, 23, 25, 30, 610000),
  'storeLocation': 'New York',
  'customer': {'gender': 'F',
   'age': 39,
   'email': 'fip@ruc.cg',
   'satisfaction': 5},
  'couponUsed': False,
  'purchaseMethod': 'Phone'}]
```

`$not`

This is similar to `NOT` in SQL or `not` in Python, and is used to negate a boolean expression. This could be useful, for example, when you want to match field values that **do not** follow a particular regex pattern.

Field existence with `$exists`

One of the hallmarks of document-based NoSQL databases is that **their schema is flexible**, meaning that not all documents need to have the same fields. This is why sometimes we need

Skip to main content

to check for the existence of a field before trying to use it. This is done using the `$exists` operator in MongoDB.

In the following example, I want to select documents for movies that are not in English. However, I should make sure that `languages` field actually exists, because otherwise I might filter out English movies that simply don't have the `languages` field:

```
list(
    client["sample_mflix"]["movies"].find(
        filter={
            'languages': {
                '$ne': 'English',
                '$exists': True  # comment this line to see the difference
            }
        },
        projection={'_id': 0, 'languages': 1, 'title': 1},
        limit=10,
    )
)
```

```
[{'title': 'Les vampires', 'languages': ['French']},
 {'title': 'Nosferatu', 'languages': ['German']},
 {'title': 'Battleship Potemkin', 'languages': ['Russian']},
 {'title': 'Metropolis', 'languages': ['German']},
 {'title': "Pandora's Box", 'languages': ['German']},
 {'title': 'The Passion of Joan of Arc', 'languages': ['French']},
 {'title': 'Storm Over Asia', 'languages': ['Russian']},
 {'title': 'Asphalt', 'languages': ['German']},
 {'title': 'David Golder', 'languages': ['French']},
 {'title': 'The Blood of a Poet', 'languages': ['French']}]
```

Querying sub-documents

```
client['mds']['instructors'].drop()
```

Let's first create a toy database called `mds`:

```
if client['mds']['instructors'].count_documents({}) == 0:

    client['mds']['instructors'].insert_many([
        {
```

Skip to main content

```
                'name': 'Computer Science',
                'campus': 'Vancouver'
            },
            'courses': [
                {
                    'name': 'Algorithms & Data Structures',
                    'code': 512
                },
                {
                    'name': 'Descriptive Statistics and Probability',
                    'code': 551
                },
            ]
        },
        {
            'name': 'Rodolfo',
            'department': {
                'name': 'Statistics',
                'campus': 'Vancouver'
            },
            'courses': [
                {
                    'name': 'Programming for Data Manipulation',
                    'code': 523
                },
                {
                    'name': 'Data Science Workflows',
                    'code': 522
                },
                {
                    'name': 'Collaborative Software Development',
                    'code': 524
                },
            ]
        },
        {
            'name': 'Alexi',
            'department': {
                'name': 'Statistics',
                'campus': 'Vancouver'
            },
            'courses': [
                {
                    'name': 'Statistical Inference and Computation',
                    'code': 552
                },
                {
                    'name': 'Regression II',
                    'code': 562
                },
            ]
        }
    ])
```

Skip to main content

Sub-documents can be queried for either for an **exact match** or for **particular sub-fields**.

The following query wouldn't work because it looks for a sub-document `'department': {'name': 'Statistics'}` (exact match, no more no less) which does not exist:

```
list(
    client['mds']['instructors'].find(
        filter={
            'department': {'name': 'Statistics'}
        }
    )
)
```

```
[]
```

But if we supply the full `department` sub-document:

```
list(
    client['mds']['instructors'].find(
        filter={
            'department': {'name': 'Statistics', 'campus': 'Vancouver'}
        }
    )
)
```

```
[{'_id': ObjectId('6391cf14210a99d148d3180e'),
  'name': 'Rodolfo',
  'department': {'name': 'Statistics', 'campus': 'Vancouver'},
  'courses': [{'name': 'Programming for Data Manipulation', 'code': 523},
   {'name': 'Data Science Workflows', 'code': 522},
   {'name': 'Collaborative Software Development', 'code': 524}]},
 {'_id': ObjectId('6391cf14210a99d148d3180f'),
  'name': 'Alexi',
  'department': {'name': 'Statistics', 'campus': 'Vancouver'},
  'courses': [{'name': 'Statistical Inference and Computation', 'code': 552},
   {'name': 'Regression II', 'code': 562}]}]
```

However, we can look for particular values for sub-fields directly using **the dot notation**.

Skip to main content

For example, if we want to find instructors from the Statistics department, we can use the following query:

```
list(
    client['mds']['instructors'].find(
        filter={
            'department.name': 'Statistics'
        }
    )
)
```

```
[{'_id': ObjectId('6391cf14210a99d148d3180e'),
  'name': 'Rodolfo',
  'department': {'name': 'Statistics', 'campus': 'Vancouver'},
  'courses': [{'name': 'Programming for Data Manipulation', 'code': 523},
   {'name': 'Data Science Workflows', 'code': 522},
   {'name': 'Collaborative Software Development', 'code': 524}]},
 {'_id': ObjectId('6391cf14210a99d148d3180f'),
  'name': 'Alexi',
  'department': {'name': 'Statistics', 'campus': 'Vancouver'},
  'courses': [{'name': 'Statistical Inference and Computation', 'code': 552},
   {'name': 'Regression II', 'code': 562}]}]
```

In this way, if over time other fields are added to the `department` field, our query will still be robust and run without problems.

Querying arrays

Similar to sub-documents, arrays can either be matched exactly or partially.

If we're looking for a **single value** inside an array, we can write our query **as if the array field was a regular simple field**.

For example, the following query returns movies that are available in French, among other languages:

Skip to main content

```
        filter={'languages': 'French'},
        projection={'_id': 0, 'languages': 1, 'title': 1},
        limit=10,
    )
)
```

```
[{'title': 'Les vampires', 'languages': ['French']},
 {'title': 'The Passion of Joan of Arc', 'languages': ['French']},
 {'title': 'All Quiet on the Western Front',
  'languages': ['English', 'French', 'German', 'Latin']},
 {'title': 'David Golder', 'languages': ['French']},
 {'title': 'The Divorcee', 'languages': ['English', 'French']},
 {'title': 'Morocco',
  'languages': ['English', 'French', 'Spanish', 'Arabic', 'Italian']},
 {'title': 'The Blood of a Poet', 'languages': ['French']},
 {'title': 'Under the Roofs of Paris', 'languages': ['French', 'Romanian']},
 {'title': 'Cimarron', 'languages': ['English', 'French']},
 {'title': 'Comradeship', 'languages': ['French', 'German']}]
```

Note that the `languages` field in all of the above documents contains French, while other languages are also occasionally found in the array.

Things are different if we're looking for **more than one element**. If the desired elements are specified inside `[]`, then MongoDB looks for an exact match.

For example, with the following query:

```
list(
    client["sample_mflix"]["movies"].find(
        filter={
            'languages': ['English', 'French']  # order matters here
            # 'languages': ['French', 'English']
        },
        projection={'_id': 0, 'languages': 1, 'title': 1},
        limit=10,
    )
)
```

```
[{'title': 'The Divorcee', 'languages': ['English', 'French']},
 {'title': 'Cimarron', 'languages': ['English', 'French']},
```

Skip to main content

```
 {'title': 'Baby Face', 'languages': ['English', 'French']},
 {'title': 'Footlight Parade', 'languages': ['English', 'French']},
 {'title': 'Going Hollywood', 'languages': ['English', 'French']},
 {'title': 'Death Takes a Holiday', 'languages': ['English', 'French']},
 {'title': 'Becky Sharp', 'languages': ['English', 'French']},
 {'title': 'Folies Bergère de Paris', 'languages': ['English', 'French']}]
```

only those documents will be returned that **only and only** have `['English', 'French']` in their `'languages'`, **in the exact same order**. You can switch the order of the languages to see that you get different results.

We can also query an array by the **index** of its elements, just like in Python.

Here, I'm looking for movies which have Italian listed as the first language in their `languages` field:

```
list(
    client["sample_mflix"]["movies"].find(
        filter={
            'languages.0': 'Italian'
        },
        projection={'_id': 0, 'languages': 1, 'title': 1},
        limit=10,
    )
)
```

```
[{'title': "Everybody's Woman", 'languages': ['Italian']},
 {'title': 'The Siege of the Alcazar', 'languages': ['Italian']},
 {'title': 'The White Ship', 'languages': ['Italian']},
 {'title': 'Ossessione', 'languages': ['Italian']},
 {'title': 'The Testimony', 'languages': ['Italian']},
 {'title': 'The Bandit', 'languages': ['Italian', 'German', 'English']},
 {'title': 'La porta del cielo', 'languages': ['Italian']},
 {'title': 'Paisan',
  'languages': ['Italian', 'English', 'German', 'Sicilian']},
 {'title': 'Rome, Open City', 'languages': ['Italian', 'German', 'Latin']},
 {'title': 'Shoeshine', 'languages': ['Italian', 'English']}]
```

Skip to main content

`$size`

The `$size` operator checks for the number of elements inside an array.

For example, suppose that we want to return all movies that are have 2 directors:

```
list(
    client["sample_mflix"]["movies"].find(
        filter={
            'directors': {'$size': 2}
        },
        projection={'_id': 0, 'directors': 1, 'title': 1},
        limit=10,
    )
)
```

```
[{'title': 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Mov
  'directors': ['Winsor McCay', 'J. Stuart Blackton']},
 {'title': 'The Perils of Pauline',
  'directors': ['Louis J. Gasnier', 'Donald MacKenzie']},
 {'title': 'Where Are My Children?',
  'directors': ['Phillips Smalley', 'Lois Weber']},
 {'title': 'From Hand to Mouth',
  'directors': ['Alfred J. Goulding', 'Hal Roach']},
 {'title': 'The Last of the Mohicans',
  'directors': ['Clarence Brown', 'Maurice Tourneur']},
 {'title': 'One Week', 'directors': ['Edward F. Cline', 'Buster Keaton']},
 {'title': 'The Saphead', 'directors': ['Herbert Blachè', 'Winchell Smith']},
 {'title': 'Now or Never', 'directors': ['Fred C. Newmeyer', 'Hal Roach']},
 {'title': 'Cops', 'directors': ['Edward F. Cline', 'Buster Keaton']},
 {'title': 'Salomè', 'directors': ['Charles Bryant', 'Alla Nazimova']}]
```

`$all`

Now suppose that we want to find movies that are available in both English and French,

Skip to main content

situation, we can use the `$all` operator to match arrays that contain all listed values among other values, in no particular order:

```
list(
    client["sample_mflix"]["movies"].find(
        filter={
            'languages': {'$all': ['English', 'French']}
        },
        projection={'_id': 0, 'languages': 1, 'title': 1},
        limit=10,
    )
)
```

```
[{'title': 'All Quiet on the Western Front',
  'languages': ['English', 'French', 'German', 'Latin']},
 {'title': 'The Divorcee', 'languages': ['English', 'French']},
 {'title': 'Morocco',
  'languages': ['English', 'French', 'Spanish', 'Arabic', 'Italian']},
 {'title': 'Cimarron', 'languages': ['English', 'French']},
 {'title': 'The Sin of Madelon Claudet', 'languages': ['English', 'French']},
 {'title': 'Forbidden', 'languages': ['English', 'French']},
 {'title': 'Freaks', 'languages': ['English', 'German', 'French']},
 {'title': 'The Mummy', 'languages': ['English', 'Arabic', 'French']},
 {'title': 'Shanghai Express',
  'languages': ['English', 'French', 'Cantonese', 'German']},
 {'title': 'Baby Face', 'languages': ['English', 'French']}]
```

Note that the `languages` field of all of the above documents contains English and French **in no particular order**, in addition to other languages that might happen to exist in the same array.

`$elemMatch`

We can best see the use case of this operator with an example. Consider these documents in the `students` collection of the `mds` database:

```
client['mds']['students'].drop()
```

Skip to main content

```
if client['mds']['students'].count_documents({}) == 0:

    client['mds']['students'].insert_many([
        {
            'name': 'Quan',
            'grades': [79, 87, 97]
        },
        {
            'name': 'Varada',
            'grades': [75, 82, 90]
        },
        {
            'name': 'Florencia',
            'grades': [92, 93, 77]
        }
    ])
```

Suppose that we want to retrieve the document for students who have a grade in the `[80, 85]` range. In our collection, only the student named “Varada” has a grade in that range (i.e. 82). Let’s see if the following query works:

```
list(
    client['mds']['students'].find(
        filter={'grades': {'$gte': 80, '$lte': 85}}
    )
)
```

```
[{'_id': ObjectId('6391cf15210a99d148d31810'),
  'name': 'Quan',
  'grades': [79, 87, 97]},
 {'_id': ObjectId('6391cf15210a99d148d31811'),
  'name': 'Varada',
  'grades': [75, 82, 90]},
 {'_id': ObjectId('6391cf15210a99d148d31812'),
  'name': 'Florencia',
  'grades': [92, 93, 77]}]
```

Oops, it returned all documents!

The reason is that when multiple conditions are defined on an array, MongoDB checks the array **as a whole**. If there is at least one match for each condition in the whole array, then that document will be returned.

In order to force MongoDB to check the conditions on **each** individual element, we need to

Skip to main content

```
list(
    client['mds']['students'].find(
        filter={'grades': {'$elemMatch': {'$gte': 80, '$lte': 85}}}
    )
)
```

```
[{'_id': ObjectId('6391cf15210a99d148d31811'),
  'name': 'Varada',
  'grades': [75, 82, 90]}]
```

Arrays of sub-documents

Querying sub-documents nested inside an array works a lot like querying arrays with simple elements. Let's take a look at the `instructors` collection of the `mds` database we created earlier:

```
list(
    client['mds']['instructors'].find({})
)
```

```
[{'_id': ObjectId('6391cf14210a99d148d3180d'),
  'name': 'Arman',
  'department': {'name': 'Computer Science', 'campus': 'Vancouver'},
  'courses': [{'name': 'Algorithms & Data Structures', 'code': 512},
   {'name': 'Descriptive Statistics and Probability', 'code': 551}]},
 {'_id': ObjectId('6391cf14210a99d148d3180e'),
  'name': 'Rodolfo',
  'department': {'name': 'Statistics', 'campus': 'Vancouver'},
  'courses': [{'name': 'Programming for Data Manipulation', 'code': 523},
   {'name': 'Data Science Workflows', 'code': 522},
   {'name': 'Collaborative Software Development', 'code': 524}]},
 {'_id': ObjectId('6391cf14210a99d148d3180f'),
  'name': 'Alexi',
  'department': {'name': 'Statistics', 'campus': 'Vancouver'},
  'courses': [{'name': 'Statistical Inference and Computation', 'code': 552},
   {'name': 'Regression II', 'code': 562}]}]
```

Skip to main content

Let's find instructor(s) who teach DSCI 512:

```
list(
    client['mds']['instructors'].find(
        filter={'courses': {'name': 'Algorithms & Data Structures', 'code': 51
    )
)
```

```
[{'_id': ObjectId('6391cf14210a99d148d3180d'),
  'name': 'Arman',
  'department': {'name': 'Computer Science', 'campus': 'Vancouver'},
  'courses': [{'name': 'Algorithms & Data Structures', 'code': 512},
   {'name': 'Descriptive Statistics and Probability', 'code': 551}]}]
```

Note that the following query fails:

```
list(
    client['mds']['instructors'].find(
        filter={'courses': {'code': 512}}
    )
)
```

```
[]
```

because just like arrays, a condition like `{'code': 512}` has to match **an entire sub-document**.

The easier way is to use the dot notation for reaching into the sub-documents in the array and place a condition on a particular sub-field

```
list(
```

Skip to main content

```
        filter={'courses.code': 512}
    )
)
```

```
[{'_id': ObjectId('6391cf14210a99d148d3180d'),
  'name': 'Arman',
  'department': {'name': 'Computer Science', 'campus': 'Vancouver'},
  'courses': [{'name': 'Algorithms & Data Structures', 'code': 512},
   {'name': 'Descriptive Statistics and Probability', 'code': 551}]}]
```

The End