Lecture 7: Introduction to non-relational databases

Contents

DSCI 513
Databases & Data Retrieval

Lecture outline

- Relational vs non-relational databases
- MongoDB: a NoSQL database management system
- Basic queries in MongoDB

We covered relational databases in 6 lectures, and are dedicating only 2 lectures to NoSQL databases. A few reasons:

Skip to main content

- Many concepts are shared between relational and non-relational databases (this extends even to the SQL syntax itself in some cases, e.g. SQL syntax used in Hive and Couchbase)
- The theory behind SQL is a little bit more involved and takes more time to feel comfortable with
- Relational databases are more common, and all of them share the same query language

Relational databases

Remember we talked about why everything is not stored in a single table?

The relational model is designed to enable the database to enforce referential integrity between tables in the database, normalized to reduce the redundancy, and generally optimized for storage.

Importance of normalization

Normalization is the process of designing tables and relationships in a database such that there are no unnecessary duplication of data and update anomalies are minimized.

Relational databases are usually normalized to ensure that

- Redundancies are eliminated or minimized
- Data integrity can be maintained with ease

Here is a great blog post that works through an example of normalizing a database.

- Relational databases are best suited to applications where **strong consistency/integrity**

- **But there is a price to pay**: relational databases require significant upfront design and preparation.

Distribution models

Why do we need to scale our DBMS resources?

- To increase availability
  - Replication: the same data is copied over multiple nodes

- To store big data
  - Sharding (or partitioning): different data chunks are put on different nodes

Downsides of relational databases

- Relational databases have fixed, inflexible data models, and are therefore not suitable for rapid evolution and development.
- Limited to one powerful servers. Usually scaling up is the only option.

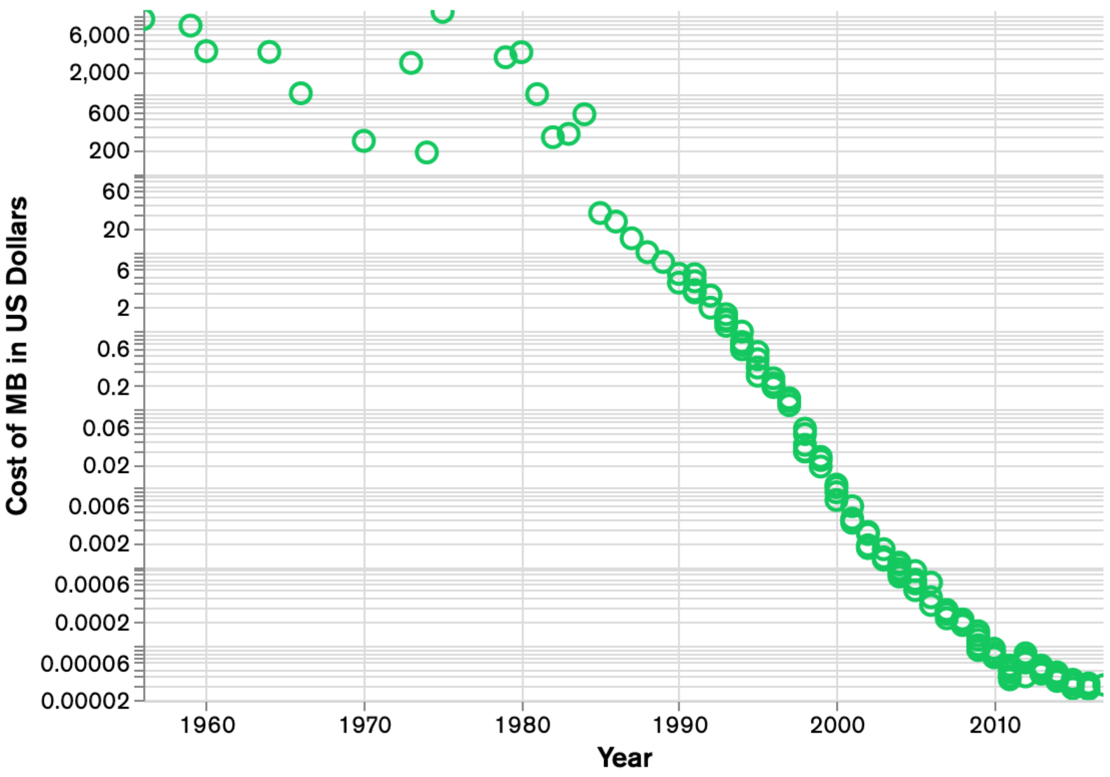
Non-relational databases

Non-relational or NoSQL databases store data in a different format than relational databases.

Skip to main content

- The data model can evolve as the requirements change. In other words, the schema is flexible
- Horizontal scaling is built in
- Queries are fast due to the data model (at the expense of redundancy). Relevant information is kept together rather than doing joins, which enhances the performance. Data models in NoSQL databases are typically optimized for queries and not for reducing data duplication.

There was eventually a time that storage costs were not as significant anymore. Also, internet and the age of data appeared. The dynamic nature of many of today's applications called for new database systems that were able to handle large amounts of data together with flexibility in their structure.



(image source)

It's important to note that:

- NoSQL databases are generally capable of storing relationship data, but they do it

Skip to main content

- NoSQL databases can support ACID too (e.g. MongoDB supports ACID).

One important downside of NoSQL databases is that their query language is not standardized like SQL. Therefore, each NoSQL DBMS has its own query language. Some NoSQL DBMSs (e.g. Couchbase, Hive) support a SQL-like query language.

Different kinds of NoSQL DBMSs

> "All happy families are alike; each unhappy family is unhappy in its own way."
> — Leo Tolstoy in *Anna Karenina* (1877)

While relational databases are all more or less alike, each NoSQL database is non-relational in its own way:

- Key-value
  - Consists of only key-value pairs
  - Good for querying only by key
  - Good if all data of each key is usually needed
  - Not good for querying by data
  - Example: Amazon DynamoDB

- Document-based
  - Consists of documents and nested sub-documents (e.g. JSON, XML)
  - Good for querying by keys and data
  - Good for hierarchical tree structures
  - Example: MongoDB, Couchbase

- Graph
  - Focuses on relationships between data

  - Optimized for searching connections, avoids joins
  - Few applications, run alongside other DBMSs
  - Example: Amazon Neptune, Neo4j, OrientDB

- Column-oriented
  - Optimized for columnar access to data (as opposed to row-wise access in RDBMSs)
  - Example: Google BigTable, Cassandra

MongoDB

MongoDB is a document-based DBMS:

- Released in 2009
- Written in C++
- Open source
- Cross platform
- Super fast

MongoDB is based on **JSON-like documents** for data storage. It offers:

- Native replication and sharding
- Automatic scaling and load balancing

Skip to main content

- Powerful query language

Who uses MongoDB

Google, ebay, Craigslist, Toyota, Forbes, Electronic Arts, Adobe, AstraZeneca, and the list goes on.

(https://www.mongodb.com/who-uses-mongodb)

JSON

- JSON is short for *Java Script Object Notation*.
- JSON documents are simple containers, where a string key is mapped to a value (e.g. a number, string, function, another object).

```
{
  "_id": 1,
  "name" : { "first" : "John", "last" : "Backus" },
  "contribs" : [ "Fortran", "ALGOL", "Backus-Naur Form", "FP" ],
  "awards" : [
    {
      "award" : "W.W. McDowell Award",
      "year" : 1967,
      "by" : "IEEE Computer Society"
    }, {
      "award" : "Draper Prize",
      "year" : 1993,
      "by" : "National Academy of Engineering"
    }
  ]
}
```

Skip to main content

- APIs
- Configuration files
- Log messages
- Database storage

BSON

Although the JSON document may look great for storing data ***as is***, but it has a number of drawbacks:

- JSON is text, and text parsing is very slow
- JSON's format is readable but not space-efficient (a database concern)
- JSON's support of various data types is not great

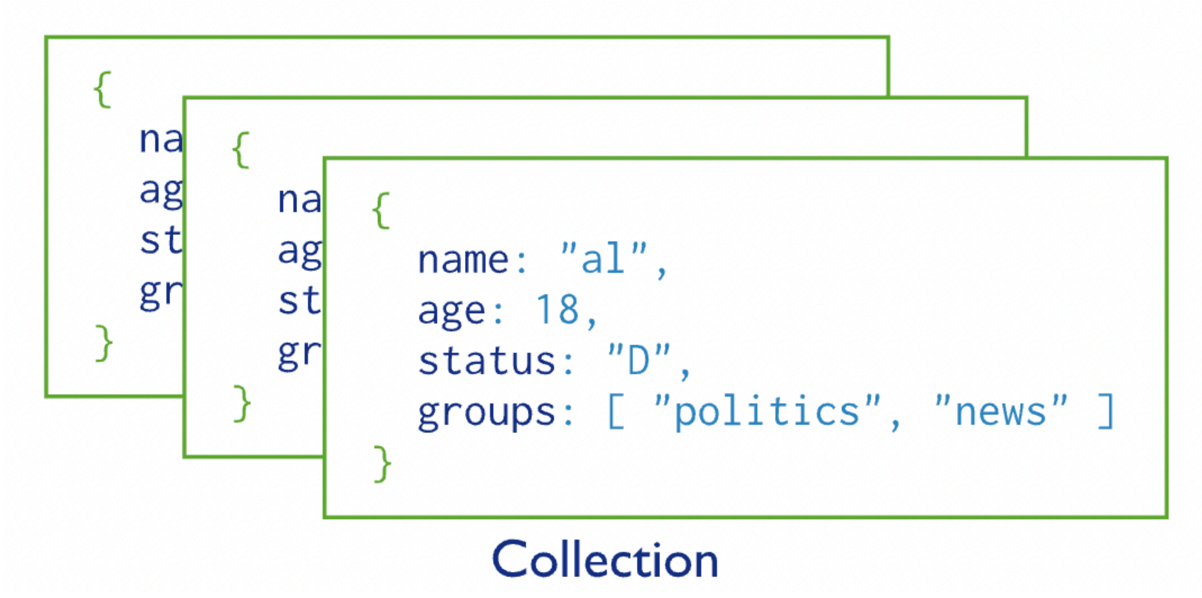
It's because of the above reasons that MongoDB stores data in BSON (Binary JSON) files, which address all of the above issues but still look like JSON when we work with them in MongoDB.

For a an overview, see here.

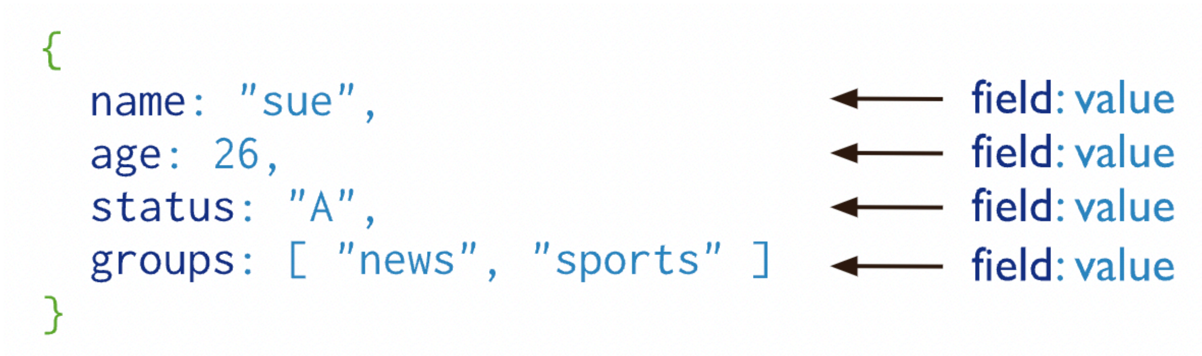
Collections

In MongoDB, a database consists of one or more **collections**, each containing multiple **documents**.

Skip to main content



Documents



- Each document contains field-value pairs
- The field name `_id` acts as the primary key of each document, and should therefore be unique in a collection
- MongoDB automatically assigns an `_id` value if not specified at the time of inserting a document
- MongoDB creates an index on the `_id` field by default
- The maximum size of a BSON document is about 16MB

Skip to main content

MongoDB Atlas

MongoDB Atlas is a fully managed cloud database service, that automates the whole process of configuring, administration and maintaining of a database server for you. Basically, you specify what kind of server (CPU, RAM, number of nodes, location, etc.) you need, and MongoDB Atlas sets it up for you. They've partnered with Amazon Web Services, Google Cloud Platform, Microsoft Azure to host their database instances.

The majority of these services are paid, however, they also offer a basic database service that is **free** and is best suited for learning and exploring. We'll use the free MongoDB Atlas clusters for our course. You can set up your own cluster here.

MongoDB interfaces

MongoDB shell (`mongosh`)

This is command line interface for interacting with a MongoDB database, similar to `psql` that we've used for Postgres. `mongosh` is based on the JavaScript language. We will not use `mongosh` much in this course.

MongoDB Compass

Compass is a versatile graphical user interface for using MongoDB databases. This is a similar application to pgAdmin that we've used for Postgres.

MongoDB's Python driver (`pymongo`)

And finally, `pymongo` is the official Python driver for MongoDB. If your using the course `conda` environment, this package is installed and ready to use in Jupyter Lab. You can take a look at `pymongo`s documentation here.

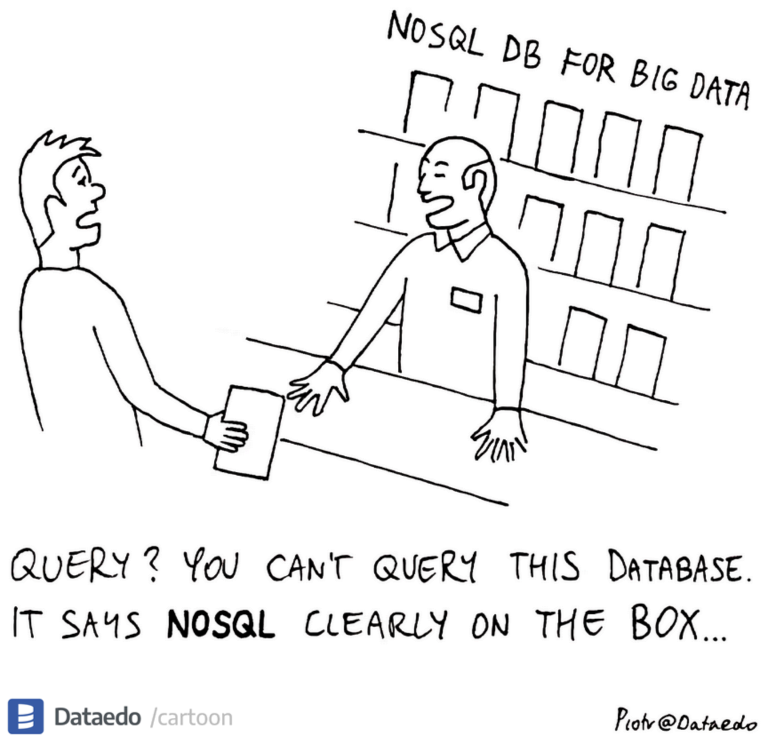
Skip to main content

```python
from pymongo import MongoClient
import json

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

```python
client = MongoClient(**login)  # for remote DB instance
# client = MongoClient(host='localhost', port=27017)  # for local DB instance
```

MongoDB query language (MQL)


NOSQL DB FOR BIG DATA
QUERY? YOU CAN'T QUERY THIS DATABASE. IT SAYS NOSQL CLEARLY ON THE BOX...
Dataedo /cartoon
Piotr@Dataedo

(image source)

As mentioned earlier, there is no standard query language among NoSQL DBMSs. This is because each NoSQL DBMS supports a different data model and obviously no one language can suit all data models.

Skip to main content

MongoDB has its own query language known as MongoDB Query Language or MQL. I will walk you through the usage of MQL in the remainder of this lecture.

Accessing databases and collections

Here is how we can access databases through different interfaces.

Compass:

It's just point and click. I'll demo this in class.

`mongosh`:

```
show dbs
use my_db
```

`pymongo`:

```
my_db = client['my_db']
my_db
```

```
Database(MongoClient(host=['ac-sbhdkey-shard-00-00.jkqvsli.mongodb.net:27017',
```

Running the above cell just gives you some information about our connection to the server. We'll learn how to run queries on this connection in a bit. For now, let's see what databases we have:

```
client.list_database_names()
```

```
['sample_airbnb',
 'sample_analytics',
```

Skip to main content

```
 'sample_mflix',
 'sample_restaurants',
 'sample_supplies',
 'sample_training',
 'sample_weatherdata',
 'admin',
 'local']
```

To access collections withing each database, use the following syntax:

`mongosh`:

```
db.my_collection.method()
```

`pymongo`:

```
my_collection = my_db['my_collection']
my_collection
```

```
Collection(Database(MongoClient(host=['ac-sbhdkey-shard-00-00.jkqvsli.mongodb.
```

Again, some information that we don't need. We will never use the database or collection objects simple like this. For now, let's take a look at the collections inside the `sample_mflix` database:

```
client['sample_mflix'].list_collection_names()
```

```
['comments', 'movies', 'sessions', 'users', 'theaters']
```

Or alternatively:

```
client.sample_mflix.list_collection_names()
```

```
['comments', 'movies', 'sessions', 'users', 'theaters']
```

Skip to main content

A very important thing to know before using MQL is that

> **Everything in MongoDB is a JSON-like document**

even queries themselves!

`find`

The main method used for querying documents is the `.find()` method. Here is an example of a query in MongoDB:

`mongosh`:

```
db.movies.find( {title: 'Titanic'} )
```

`pymongo`:

```
client['sample_mflix']['movies'].find( filter={'title': 'Titanic'} )
```

```
<pymongo.cursor.Cursor at 0x110b9d7d0>
```

Using `filter=` is optional in the argument list, but if you remember from Python's Zen advice, "*explicit is better than implicit*".

Well, the above code doesn't do anything because it returns a cursor object which is basically a Python generator. Let's return the first element of this generator:

Skip to main content

```
next(client['sample_mflix']['movies'].find( {'title': 'Titanic'} ))
```

```
{'_id': ObjectId('573a1394f29313caabcdf639'),
 'plot': 'An unhappy married couple deal with their problems on board the ill-
 'genres': ['Drama', 'History', 'Romance'],
 'runtime': 98,
 'rated': 'NOT RATED',
 'cast': ['Clifton Webb',
  'Barbara Stanwyck',
  'Robert Wagner',
  'Audrey Dalton'],
 'num_mflix_comments': 0,
 'poster': 'https://m.media-amazon.com/images/M/MV5BMTU3NTUyMTc3Nl5BMl5BanBnXkE
 'title': 'Titanic',
 'fullplot': 'Unhappily married and uncomfortable with life among the British u
 'languages': ['English', 'Basque', 'French', 'Spanish'],
 'released': datetime.datetime(1953, 7, 13, 0, 0),
 'directors': ['Jean Negulesco'],
 'writers': ['Charles Brackett', 'Walter Reisch', 'Richard L. Breen'],
 'awards': {'wins': 0,
  'nominations': 3,
  'text': 'Won 1 Oscar. Another 2 nominations.'},
 'lastupdated': '2015-09-16 00:00:16.593000000',
 'year': 1953,
 'imdb': {'rating': 7.3, 'votes': 4677, 'id': 46435},
 'countries': ['USA'],
 'type': 'movie',
 'tomatoes': {'viewer': {'rating': 3.6, 'numReviews': 86400, 'meter': 65},
  'dvd': datetime.datetime(2003, 9, 2, 0, 0),
  'critic': {'rating': 6.7, 'numReviews': 9, 'meter': 89},
  'lastUpdated': datetime.datetime(2015, 9, 10, 19, 15, 34),
  'rotten': 1,
  'production': '20th Century Fox',
  'fresh': 8}}
```

Or we can pass it to `list()` to materialize the generator entirely:

```
list(
    client['sample_mflix']['movies'].find( {'title': 'Titanic'} )
)
```

```
[{'_id': ObjectId('573a1394f29313caabcdf639'),
  'plot': 'An unhappy married couple deal with their problems on board the ill-
  'genres': ['Drama', 'History', 'Romance'],
  'runtime': 98,
  'rated': 'NOT RATED',
```

Skip to main content

```
    'Robert Wagner',
    'Audrey Dalton'],
   'num_mflix_comments': 0,
   'poster': 'https://m.media-amazon.com/images/M/MV5BMTU3NTUyMTc3Nl5BMl5BanBnXk
   'title': 'Titanic',
   'fullplot': 'Unhappily married and uncomfortable with life among the British
   'languages': ['English', 'Basque', 'French', 'Spanish'],
   'released': datetime.datetime(1953, 7, 13, 0, 0),
   'directors': ['Jean Negulesco'],
   'writers': ['Charles Brackett', 'Walter Reisch', 'Richard L. Breen'],
   'awards': {'wins': 0,
    'nominations': 3,
    'text': 'Won 1 Oscar. Another 2 nominations.'},
   'lastupdated': '2015-09-16 00:00:16.593000000',
   'year': 1953,
   'imdb': {'rating': 7.3, 'votes': 4677, 'id': 46435},
   'countries': ['USA'],
   'type': 'movie',
   'tomatoes': {'viewer': {'rating': 3.6, 'numReviews': 86400, 'meter': 65},
    'dvd': datetime.datetime(2003, 9, 2, 0, 0),
    'critic': {'rating': 6.7, 'numReviews': 9, 'meter': 89},
    'lastUpdated': datetime.datetime(2015, 9, 10, 19, 15, 34),
    'rotten': 1,
    'production': '20th Century Fox',
    'fresh': 8}},
  {'_id': ObjectId('573a139af29313caabcefb1d'),
   'plot': 'The story of the 1912 sinking of the largest luxury liner ever buil
   'genres': ['Action', 'Drama', 'History'],
   'runtime': 173,
   'cast': ['Peter Gallagher',
    'George C. Scott',
    'Catherine Zeta-Jones',
    'Eva Marie Saint'],
   'poster': 'https://m.media-amazon.com/images/M/MV5BYWM0MDE3OWMtMzlhZC00YzMyLT
   'title': 'Titanic',
   'fullplot': "The plot focuses on the romances of two couples upon the doomed
   'languages': ['English'],
   'released': datetime.datetime(1996, 11, 17, 0, 0),
   'rated': 'PG-13',
   'awards': {'wins': 0,
    'nominations': 9,
    'text': 'Won 1 Primetime Emmy. Another 8 nominations.'},
   'lastupdated': '2015-08-30 00:47:02.163000000',
   'year': 1996,
   'imdb': {'rating': 5.9, 'votes': 3435, 'id': 115392},
   'countries': ['Canada', 'USA'],
   'type': 'series',
   'tomatoes': {'viewer': {'rating': 3.8, 'numReviews': 30909, 'meter': 71},
    'dvd': datetime.datetime(1999, 9, 7, 0, 0),
    'production': 'Hallmark Entertainment',
    'lastUpdated': datetime.datetime(2015, 8, 15, 18, 12, 51)},
   'num_mflix_comments': 0},
  {'_id': ObjectId('573a139af29313caabcf0d74'),
   'fullplot': '84 years later, a 101-year-old woman named Rose DeWitt Bukater
```

Skip to main content

```
    'plot': 'A seventeen-year-old aristocrat falls in love with a kind, but poor
    'genres': ['Drama', 'Romance'],
    'rated': 'PG-13',
    'metacritic': 74,
    'title': 'Titanic',
    'lastupdated': '2015-09-13 00:41:42.117000000',
    'languages': ['English',
     'French',
     'German',
     'Swedish',
     'Italian',
     'Russian'],
    'writers': ['James Cameron'],
    'type': 'movie',
    'tomatoes': {'website': 'http://www.titanicmovie.com/',
     'viewer': {'rating': 3.3, 'numReviews': 35792304, 'meter': 69},
     'dvd': datetime.datetime(2012, 9, 10, 0, 0),
     'critic': {'rating': 8.0, 'numReviews': 178, 'meter': 88},
     'boxOffice': '$57.9M',
     'consensus': 'A mostly unqualified triumph for James Cameron, who offers a 
     'rotten': 21,
     'production': 'Paramount Pictures',
     'lastUpdated': datetime.datetime(2015, 9, 13, 17, 5, 18),
     'fresh': 157},
    'poster': 'https://m.media-amazon.com/images/M/MV5BMDdmZGU3NDQtY2E5My00ZTliL
    'num_mflix_comments': 128,
    'released': datetime.datetime(1997, 12, 19, 0, 0),
    'awards': {'wins': 127,
     'nominations': 63,
     'text': 'Won 11 Oscars. Another 116 wins & 63 nominations.'},
    'countries': ['USA'],
    'cast': ['Leonardo DiCaprio', 'Kate Winslet', 'Billy Zane', 'Kathy Bates'],
    'directors': ['James Cameron'],
    'runtime': 194}]
```

> **Note:** `.find( filter={} )` or `.find()` returns every document in the collection.

Note that there is another method `.findOne()` in `mongosh` and `.find_one()` in `pymongo`. This method returns only one document regardless of how many there are, according to the order in which documents are stored on the physical disk.

Skip to main content

`projection`

Remember what projection meant in SQL? Returning a particular set of columns among all that exist in a table was called projection (of the results onto particular columns).

Projection has a similar meaning in NoSQL: it means explicitly choosing the fields that we are interested in, instead of all fields that are returned by default. This is done by feeding a list of fields to the `projection=` argument, as well as a truthy of falsy value that indicates whether or not that field should be included.

For example, here I return the `title` and `year` fields only from the document in the result:

`mongosh`:

```
db.movies.find( {title: 'Titanic'}, {'title': 1, 'year': 1} )
```

`pymongo`:

```
list(
    client['sample_mflix']['movies'].find(
        filter={'title': 'Titanic'},
        projection={'title': 1, 'year': 1}
    )
)
```

```
[{'_id': ObjectId('573a1394f29313caabcdf639'),
  'title': 'Titanic',
  'year': 1953},
 {'_id': ObjectId('573a139af29313caabcefb1d'),
  'title': 'Titanic',
  'year': 1996},
 {'_id': ObjectId('573a139af29313caabcf0d74'),
  'year': 1997,
  'title': 'Titanic'}]
```

> **Note:** In `pymongo`, you can use `True` instead of `1` and `False` instead of `0`.

> **Note:** In `pymongo`, we need to enclose all field names in single or double quotes (e.g. `'title'` not `title`), otherwise Python would complain because it doesn't recognize

Skip to main content

In the above returned documents, note that the primary key field, namely, the `_id` field is always returned by default unless you explicitly exclude it using `{'_id': 0}` or `{'_id': False}`. **This is the only scenario where we might mix up `1`s and `0`s (or `True`s and `False`s) in the projection field.**

```
list(
    client['sample_mflix']['movies'].find(
        filter={'title': 'Titanic'},
        projection={'_id': 0, 'title': 1, 'year': 1}
    )
)
```

```
[{'title': 'Titanic', 'year': 1953},
 {'title': 'Titanic', 'year': 1996},
 {'year': 1997, 'title': 'Titanic'}]
```

`sort`

`mongosh`:

```
db.movies.find(<filter>, <projection>).sort( {runtime: 1, year:-1} )
```

`pymongo`:

```
list(
    client['sample_mflix']['movies'].find(
        filter={'title': 'Titanic'},
        projection={'_id': 0, 'title': 1, 'year': 1, 'runtime': 1},
        sort=[('runtime', 1), ('year', -1)]
    )
)
```

```
[{'runtime': 98, 'title': 'Titanic', 'year': 1953},
```

Skip to main content

```
 {'year': 1997, 'title': 'Titanic', 'runtime': 194}]
```

`limit`

`mongosh`:

```
db.movies.find({}, {title: 1, _id: 0}).limit(5)
```

`pymongo`:

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

```
[{'title': 'Blacksmith Scene'},
 {'title': 'The Great Train Robbery'},
 {'title': 'The Land Beyond the Sunset'},
 {'title': 'A Corner in Wheat'},
 {'title': 'Winsor McCay, the Famous Cartoonist of the N.Y. Herald and His Mov
```

`count_documents`

`mongosh`:

```
db.movies.countDocuments({year:2000})
```

Skip to main content

`pymongo`:

```
client['sample_mflix']['movies'].count_documents(filter={'year': 2000})
```

```
618
```

`skip`

`mongosh`:

```
db.movies.find( filter={title: 'Titanic'}, projection={'title': 1, 'year': 1}
```

`pymongo`:

```
list(
    client['sample_mflix']['movies'].find(
        filter={'title': 'Titanic'},
        projection={'title': 1, 'year': 1},
        skip=2
    )
)
```

```
[{'_id': ObjectId('573a139af29313caabcf0d74'),
  'year': 1997,
  'title': 'Titanic'}]
```

`distinct`

`mongosh`:

```
db.movies.distinct( 'title', {title: 'Titanic'} )
```

`pymongo`:

```
list(
    client['sample_mflix']['movies'].find(
        filter={'title': 'Titanic'},
        # projection={'title': 1, 'year': 1}
    )
    .distinct('title')
)
```

```
['Titanic']
```

The `distinct` method here only returns unique **values**, not entire documents.