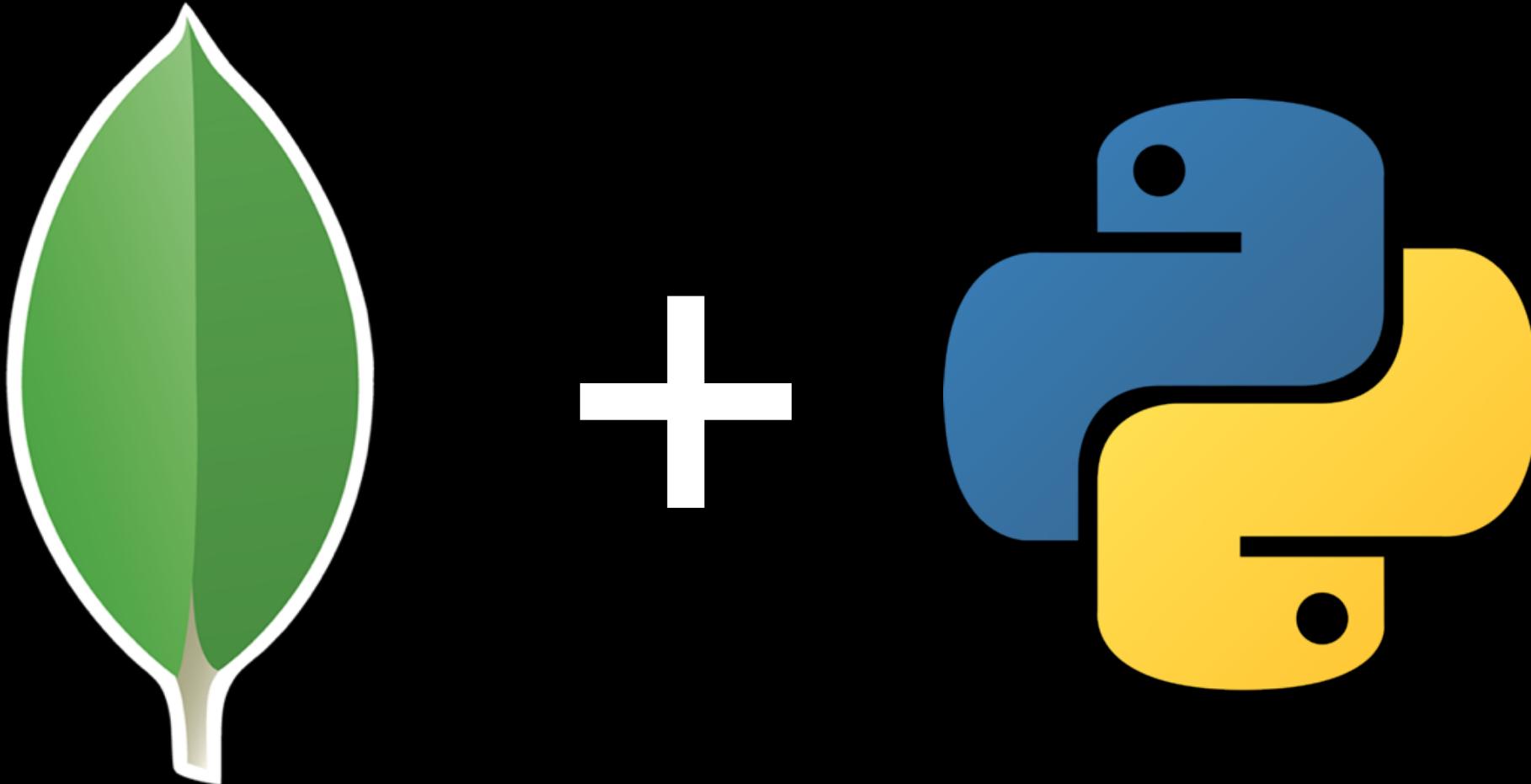


MongoDB + Python with PyCharm



Topics

- Tools
- Why MongoDB?
- Building something fun!
- Concepts
- Questions

Take the code with you

The screenshot shows a GitHub repository page for 'mikeckennedy/jetbrains-webcast-build-with-mongodb'. The repository has 1 commit, 1 branch, 0 releases, 1 contributor, and is licensed under MIT. The latest commit was made by mikeckennedy on Jan 29, 2018. The repository contains files like .gitignore, LICENSE, README.md, and README.md.

mikeckennedy/jetbrains-webcast-build-with-mongodb

Code and handouts for my JetBrains webcast recorded January 30, 2018

Add topics

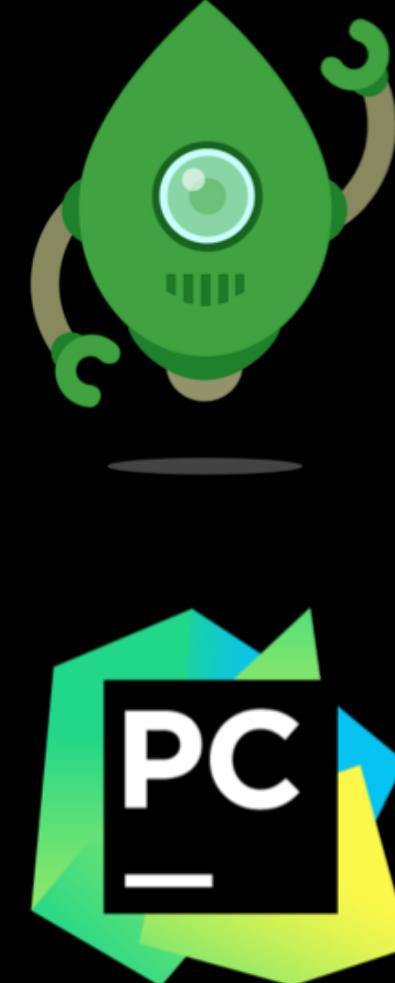
1 commit | 1 branch | 0 releases | 1 contributor | MIT

Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

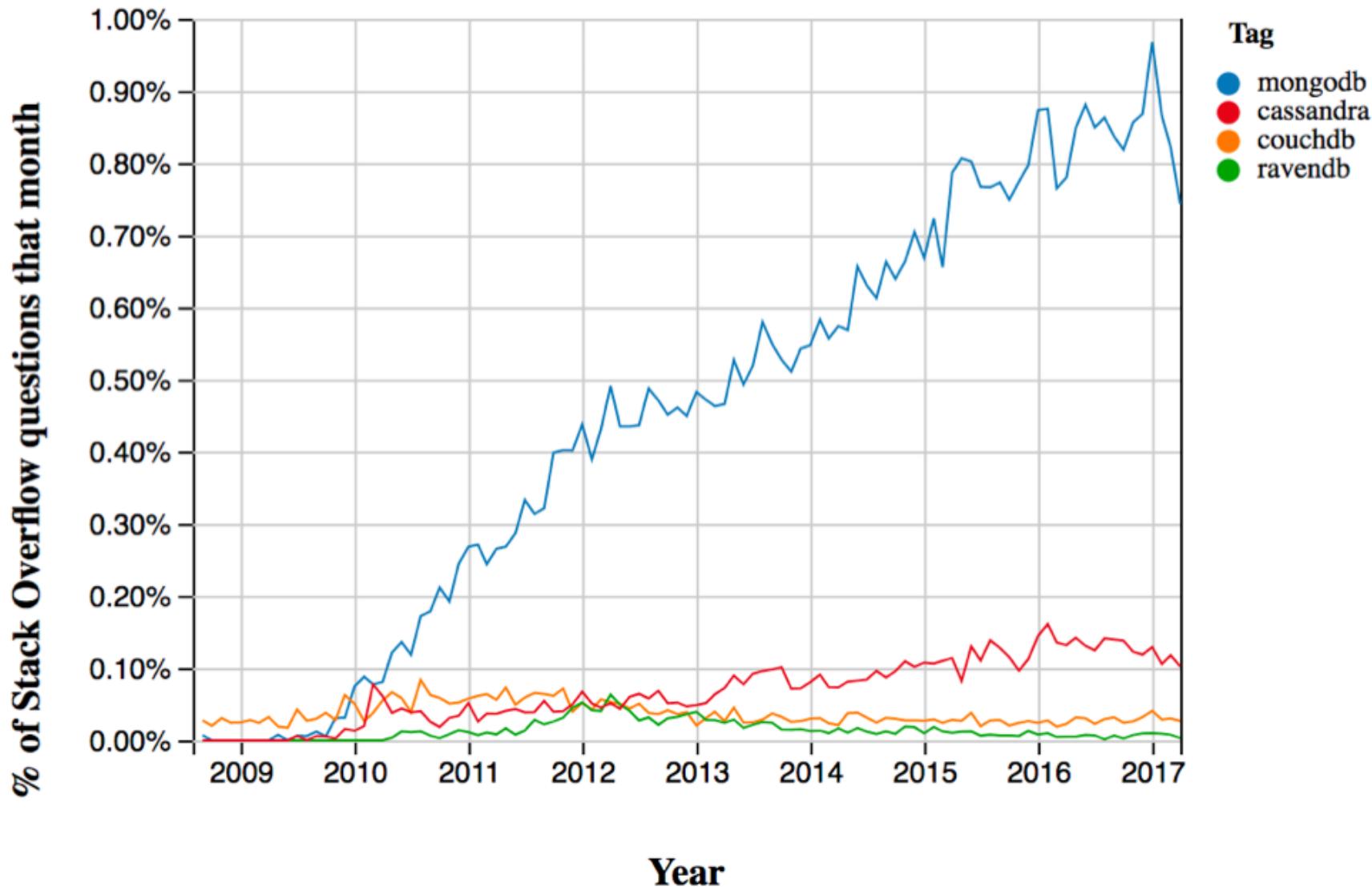
File	Commit	Time
.gitignore	Initial commit	3 minutes ago
LICENSE	Initial commit	3 minutes ago
README.md	Initial commit	3 minutes ago
README.md		

github.com/mikeckennedy/jetbrains-webcast-build-with-mongodb

Tools



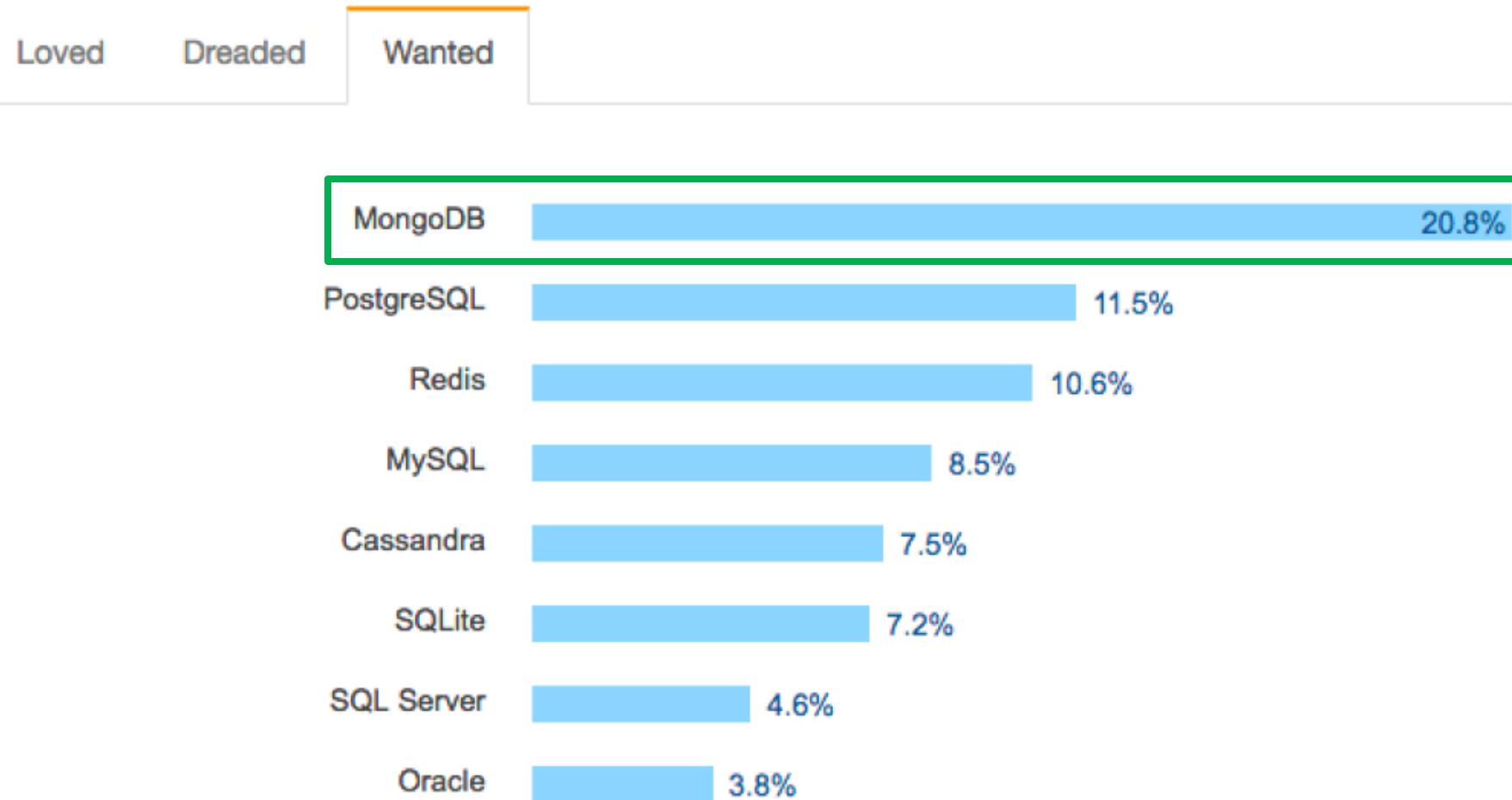
MongoDB is popular



<https://insights.stackoverflow.com/trends?tags=mongodb%2Ccouchdb%2Cravendb%2Ccassandra>

MongoDB is wanted

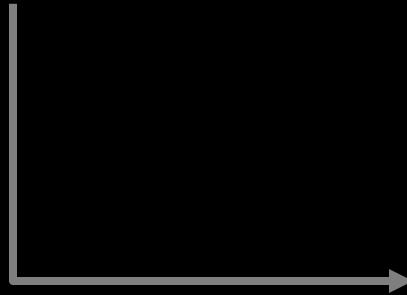
Most Loved, Dreaded, and Wanted Databases



% of developers who are not developing with the language or technology but have expressed interest in developing with it

How do document databases work?

Consider lectures to be a pre-computed **join**.



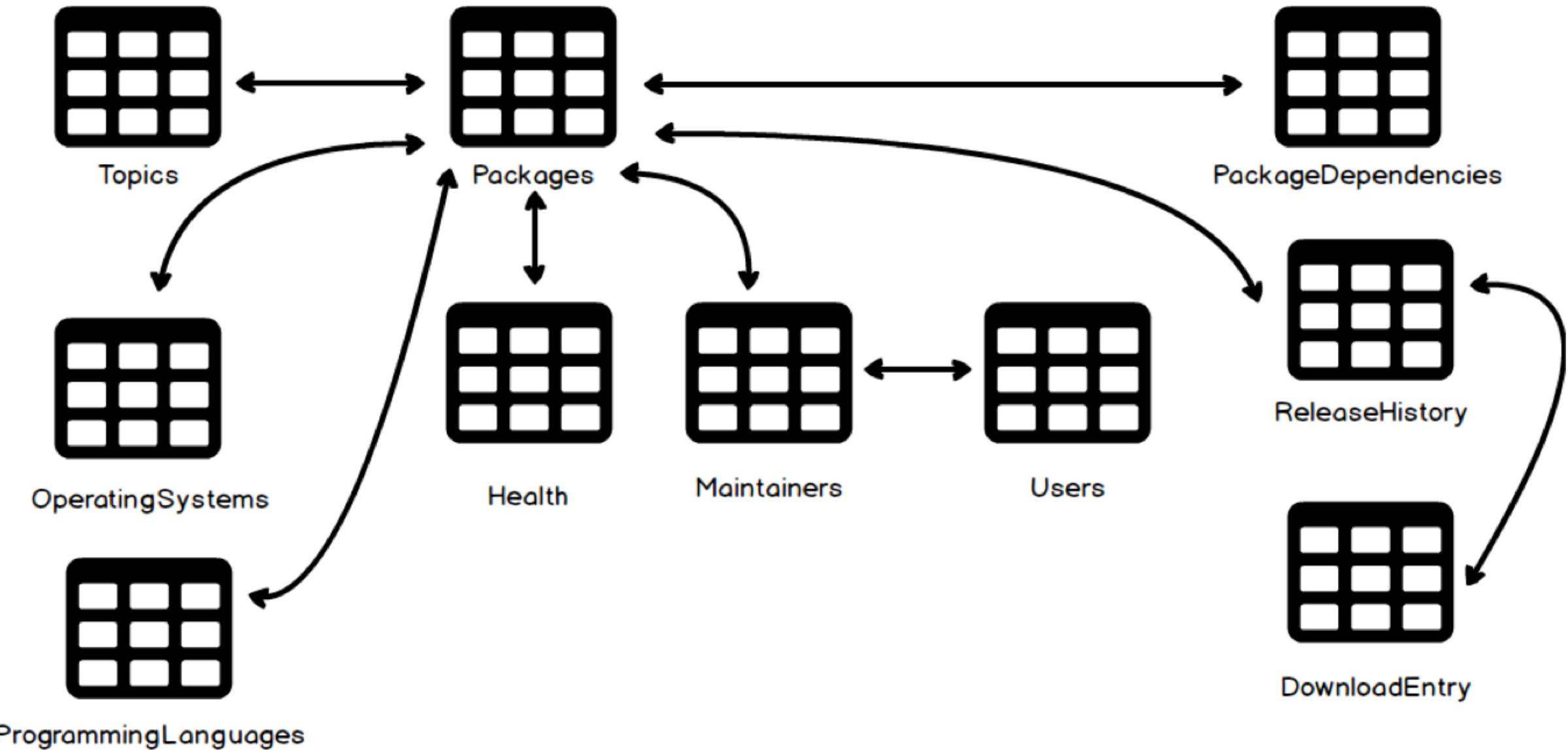
Question: Will I be able to find lecture 10106 directly?

```
// Chapter 1001 from Python Jumpstart by Building 10 Apps
{
    "_id" : 1001,
    "title" : "Welcome to the course",
    "course_id" : 1,
    "duration_in_sec" : 1707,
    "lectures" : [
        {
            "id" : 10101,
            "title" : "Welcome and thanks for coming",
            "video_url" : "https://s3.amazonaws.com/.../00-00-welcome.mp4",
            "duration_in_sec" : 380
        },
        // ...
        {
            "id" : 10106,
            "title" : "Linux: Installing Python and PyCharm",
            "video_url" : "https://s3.amazonaws.com/.../00-05-linux-setup.mp4",
            "duration_in_sec" : 307
        }
    ]
}
```

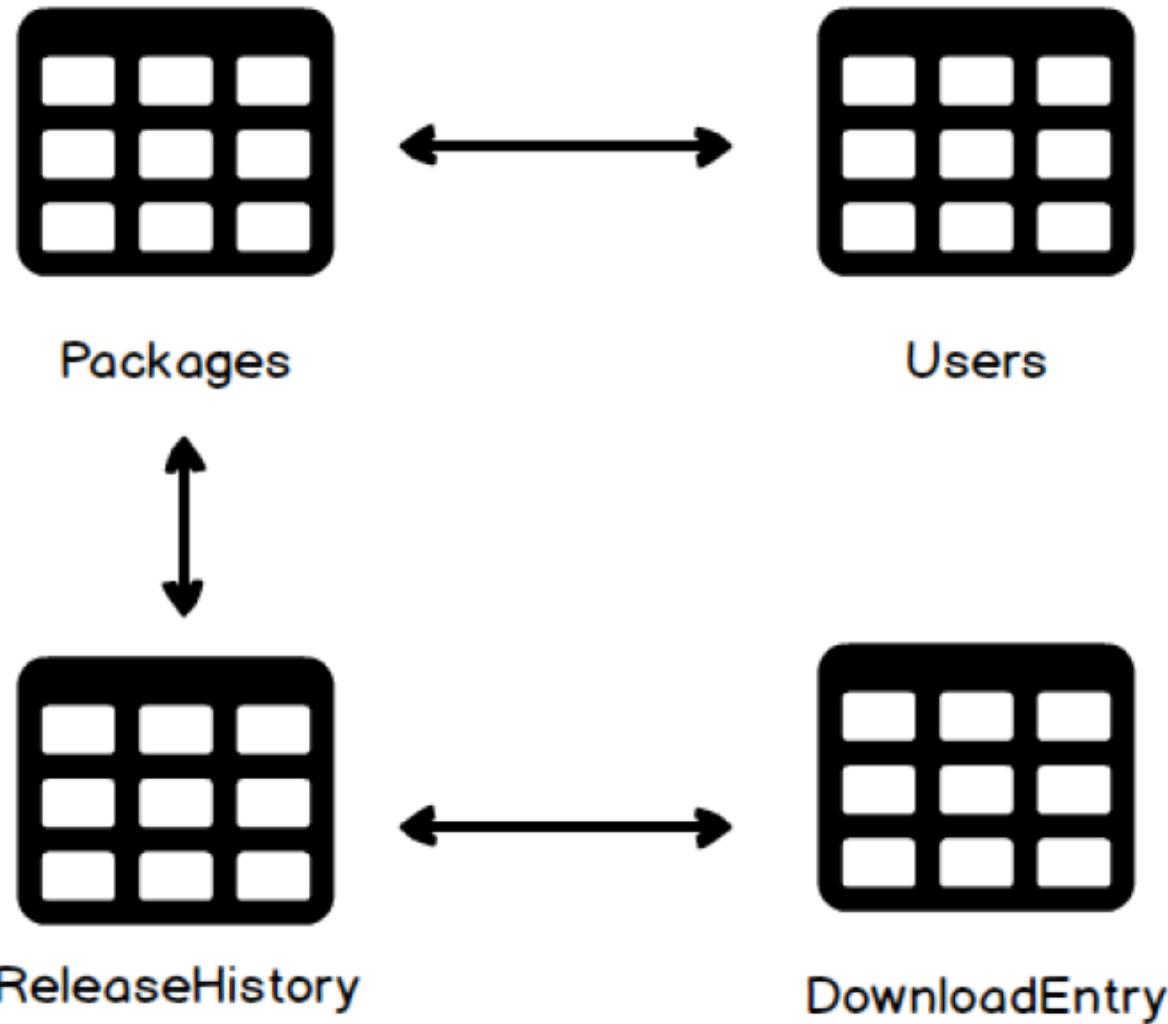
Our demo application

A screenshot of a web browser showing the PyPI project page for mongoengine 0.15.0. The page has a blue header with the PyPI logo, a search bar, and navigation links for Help, Donate, Log In, and Register. The main title is "mongoengine 0.15.0". Below it is a button with the command "pip install mongoengine" and a QR code. To the right is a green button labeled "Latest Version" with a checkmark and the date "Nov 6, 2017". A sub-header states "MongoEngine is a Python Object-Document Mapper for working with MongoDB." The page is divided into sections: "Navigation" (Project Description, Release History, Download Files), "About" (MongoEngine is a Python Object-Document Mapper for working with MongoDB. Documentation is available at <https://mongoengine-odm.readthedocs.io> - there is currently a [tutorial](#), a [user guide](#), and an [API reference](#).), "Supported MongoDB Versions" (MongoEngine is currently tested against MongoDB v2.4, v2.6, and v3.0. Future versions should be supported as well, but aren't actively tested at the moment. Make sure to open an issue or submit a pull request if you experience any problems with MongoDB v3.2+.), and "Installation" (We recommend the use of [virtualenv](#) and of [pip](#). You can then use `pip install -U mongoengine`. You may).

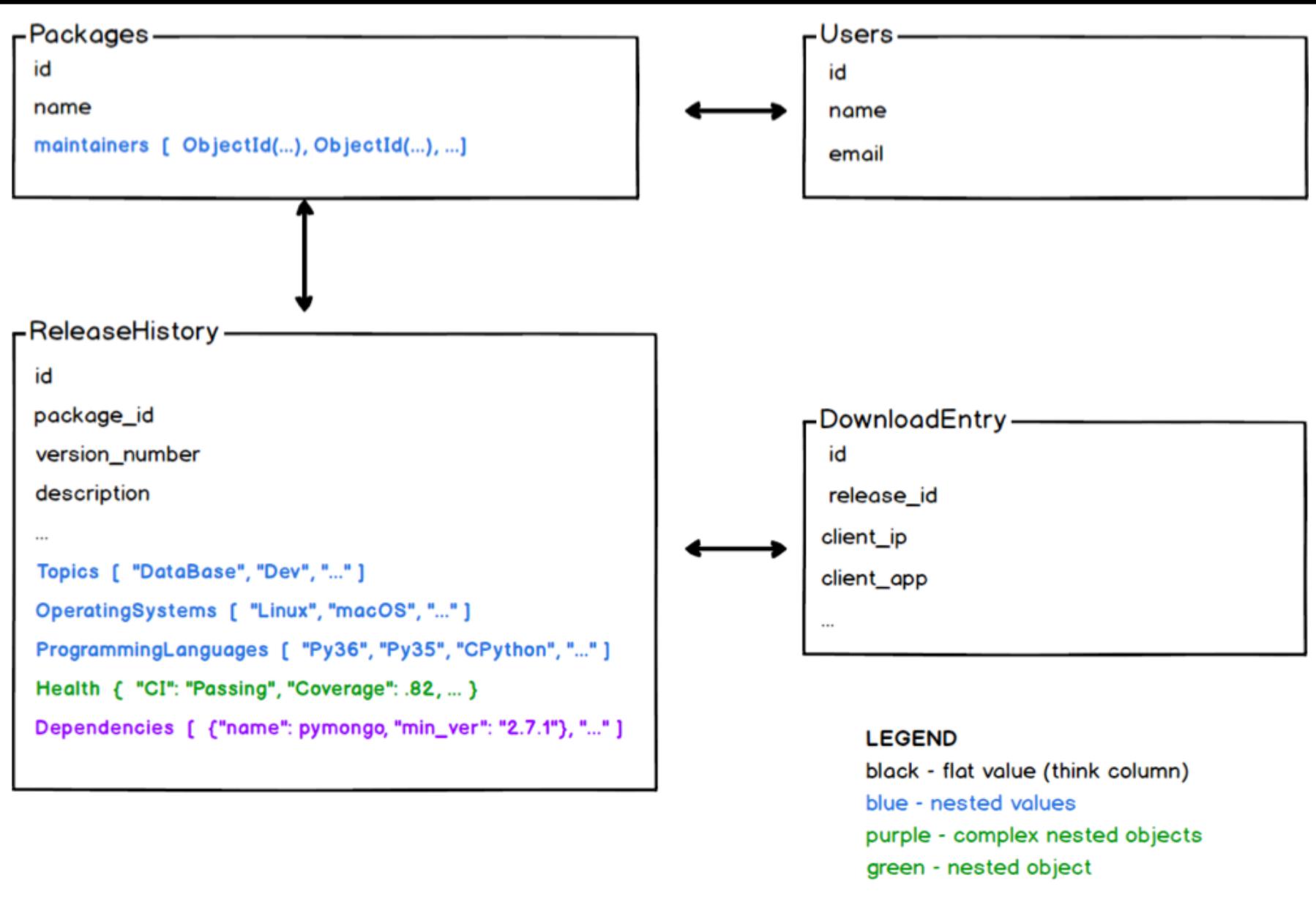
Relational model



Document DB model



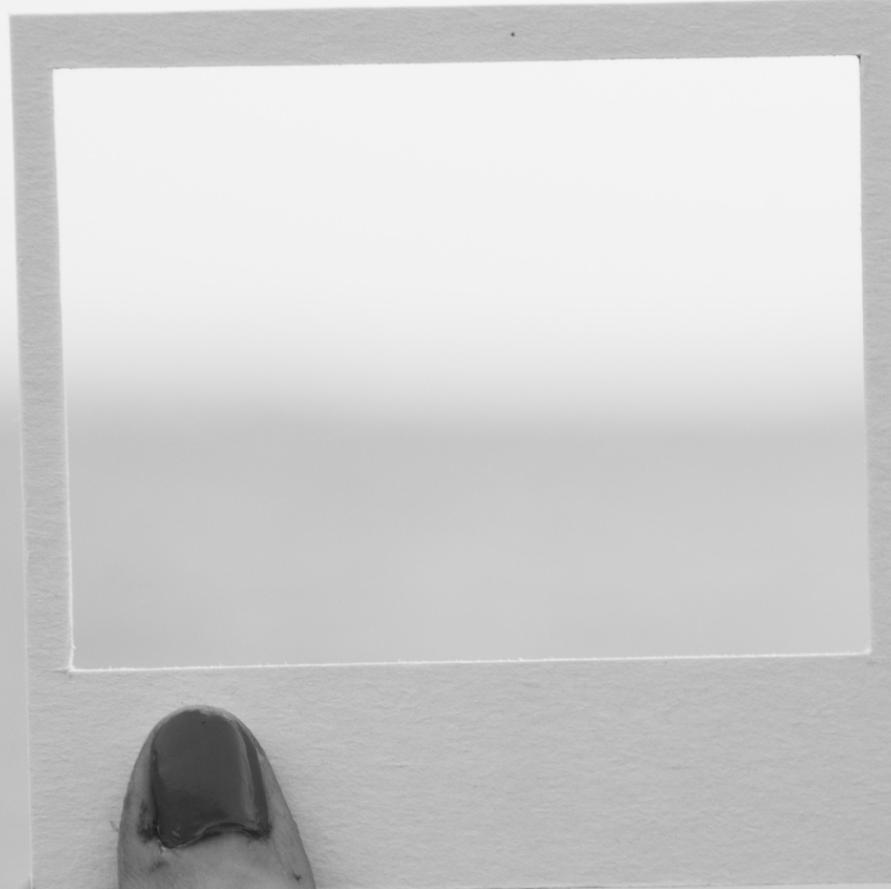
Document DB model (detailed)



Demo time



Concepts



The primary question becomes

To embed or not to embed?

1. Is the embedded data wanted **80% of the time**?
2. How often do you want the embedded data **without the containing document**?
3. Is the embedded data **a bounded set**?
4. Is that bound **small**?
5. **How varied** are your queries?
6. Is this an **integration DB** or an **application DB**?

Registering connections

```
import mongoengine

alias_core = 'core'
db = 'snakebnb'

mongoengine.register_connection(alias=alias_core, name=db)
```

Creating basic classes

```
import mongoengine

class Snake(mongoengine.Document):
    registered_date = mongoengine.DateTimeField()
    length = mongoengine.FloatField()
    name = mongoengine.StringField()
    species = mongoengine.StringField()
    is_venomous = mongoengine.BooleanField()
```

Nested data (fields and lists)

```
import mongoengine

class Cage(mongoengine.Document):
    name = mongoengine.StringField(required=True)
    price = mongoengine.FloatField(required=True)
    square_meters = mongoengine.FloatField(required=True)

    bookings = mongoengine.EmbeddedDocumentListField()

    def __str__(self):
        return self.name

    def __repr__(self):
        return self.name

    def __eq__(self, other):
        if not isinstance(other, Cage):
            return False
        return self.name == other.name

    def __ne__(self, other):
        if not isinstance(other, Cage):
            return True
        return self.name != other.name

    def __hash__(self):
        return hash(self.name)

    def __getstate__(self):
        state = self.__dict__.copy()
        state.pop('_meta')
        return state

    def __setstate__(self, state):
        self.__dict__.update(state)
        self._meta = {'db_alias': 'main', 'collection': 'cages'}
```

```
{
    "_id" : ObjectId("59d3951c3ae740cfffe16427d"),
    "registered_date" : ISODate("2017-10-03"),
    "name" : "Slither's first cage",
    "price" : 29.95,
    "square_meters" : 1.5,
    "is_carpeted" : true,
    "has_toys" : false,
    "allow_dangerous_snakes" : false,
    "bookings" : [
        {
            "check_in_date" : ISODate("2018-02-19"),
            "check_out_date" : ISODate("2018-02-28"),
            "rating" : 0
        },
        {
            "check_in_date" : ISODate("2018-03-01"),
            "check_out_date" : ISODate("2018-03-30"),
            "rating" : 0
        }
    ]
}
```

Querying (direct match)

```
def find_owner_by_email(email: str) -> Owner:  
  
    owner = Owner.objects().filter(email=email).first()  
    return owner
```

Filter on 1 or more fields. **email** must match passed **email** value.



Call **first** to execute the query and return 1 owner or None

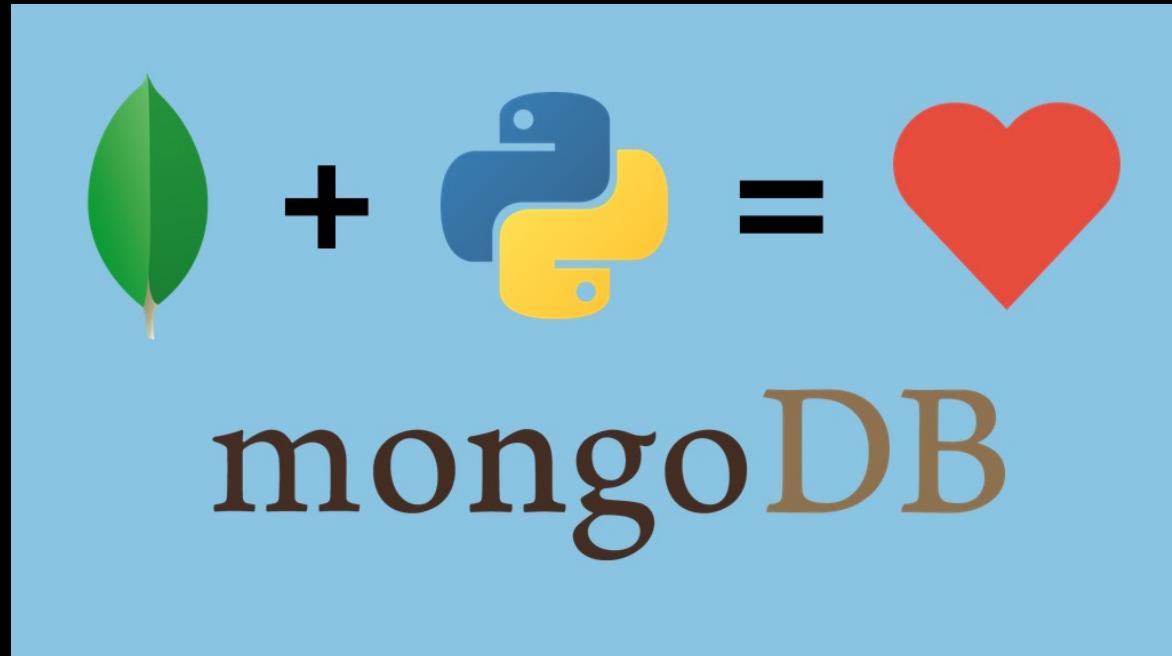
Querying (subdocuments)

```
def get_bookings_for_user(user_id: ObjectId) -> List[Booking]:  
    owner = Owner.objects(id=user_id).first()  
    booked_cages = Cage \  
        .objects(bookings_guest_snake_id_in=owner.snake_ids) \  
        .all()  
  
    return list(booked_cages)
```

Use `_` to separate / navigate levels in subdocuments

Best practice: Execute the query before returning results.

Want more?



MongoDB for Developers with Python, 7 hrs



MongoDB Quickstart with Python, 2.5 hrs

training.talkpython.fm