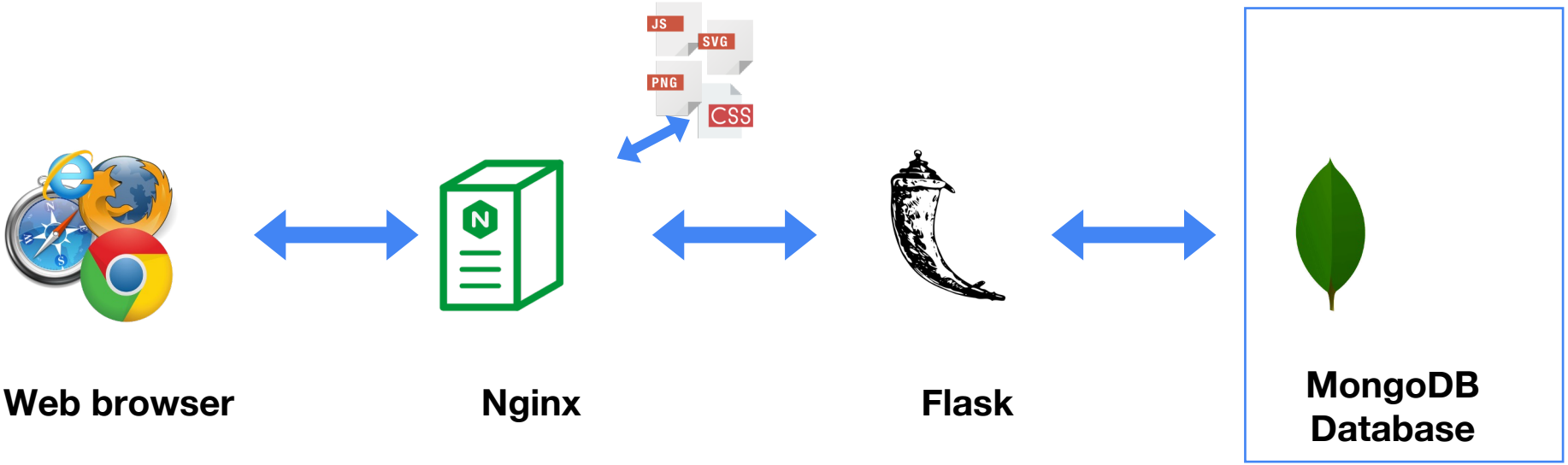
Web Application Development

Database


ESS
KEW
ELL
SE-
QUEL
SKEWL

Web Application Architecture



NoSQL Databases

⚠️ Caution! Different terminology

- Document databases
- Key-value databases
- Wide-column stores
- Graph databases

MongoDB Features

- Rich query language (CRUD, aggregate, text search, geospatial)
- High availability (replication, automatic failover, data redundancy)
- Horizontal scalability (sharding data)
- Multiple storage engines (file, encryption, in-memory)

JSON

Data types:

- String
- Number (int, float)
- Object (nested JSON)
- Array
- Boolean
- null

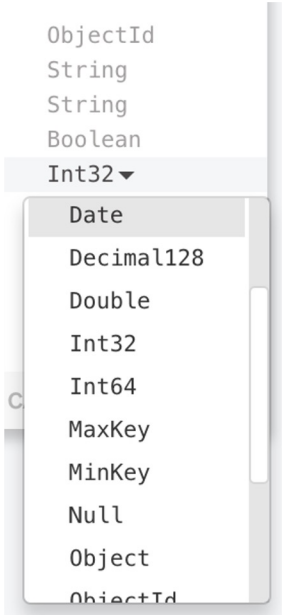
```
{
  "firstName": "John",
  "lastName": "Smith",
  "isAlive": true,
  "age": 27,
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    "postalCode": "10021-3100"
  },
  "phoneNumbers": [
    {
      "type": "home",
      "number": "212 555-1234"
    },
    {
      "type": "office",
      "number": "646 555-4567"
    }
  ],
  "children": [],
  "spouse": null
}
```

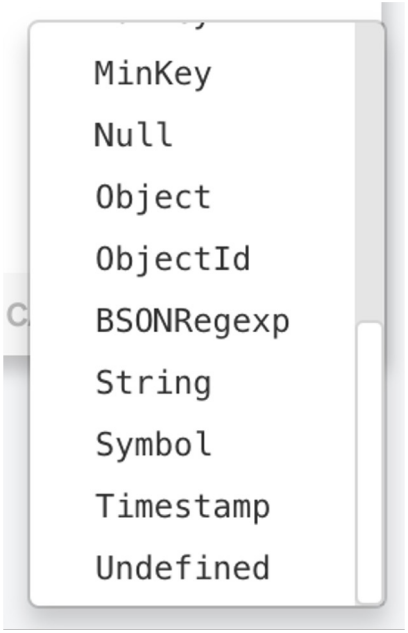
JSON representation

firstName	John	String
lastName	Smith	String
isAlive	true	Boolean
age	27	Int32
▼ address	{ 4 fields }	Object
streetAddress	21 2nd Street	String
city	New York	String
state	NY	String
postalCode	10021-3100	String
▼ phoneNumbers	[2 elements]	Array
▼ [0]	{ 2 fields }	Object
type	home	String
number	212 555-1234	String
▼ [1]	{ 2 fields }	Object
type	office	String
number	646 555-4567	String
▼ children	[0 elements]	Array
spouse	null	Null

BSON

- Binary
- Strictly typed
- Similar to JSON





Database & collection

```
{
    "_id" : ObjectId("5e9975a7e8de390f5968e28d"),
    "username" : "user",
    "password" : "123456",
    "registerDate" : ISODate("2020-04-18T12:00:00.000Z")
}
{
    "_id" : ObjectId("5e9975a7e8de390f5968e28d"),
    "username" : "user",
    "password" : "123456",
    "registerDate" : ISODate("2020-04-18T12:00:00.000Z")
}
```

```
{
    "_id" : ObjectId("5e9975a7e8de390f5968e28d"),
    "title" : "First article",
    "publishDate" : ISODate("2020-04-18T12:00:00.000Z")
}
```

***

***

***

Database & collection

Database

Collections

Documents

```
_id: ObjectId("5e9975a7e8de390f5968e28d")
username: "user"
password: "123456"
registerDate: 2020-04-18T12:00:00.000+00:00
```

```
1  _id: ObjectId("5e9976a42482b3456c1dc4f6")
2  firstName : "John "
3  lastName : "Smith "
4  isAlive : true
5  age : 27
6  › address : Object
7  › phoneNumbers : Array
8  › children : Array
9  spouse : null
```

Key-value

Limitations:

- Key cannot include: **null**, **$**, **.**
- Nested depth **100**
- Case insensitive db names
- Document size 16mb

Database operations

Access via:

- Command line interface (CLI)
- Visual (Compass, Robo 3T)
- Program (pymongo, mongoengine)

Operations:

- Insert
- Remove
- Find
- Update

Literature

- Databases and collections: https://docs.mongodb.com/manual/core/databases-and-collections/
- BSON in MongoDB: https://docs.mongodb.com/manual/reference/bson-types/
- Compass: https://www.mongodb.com/products/compass
- Robo 3T: https://robomongo.org/

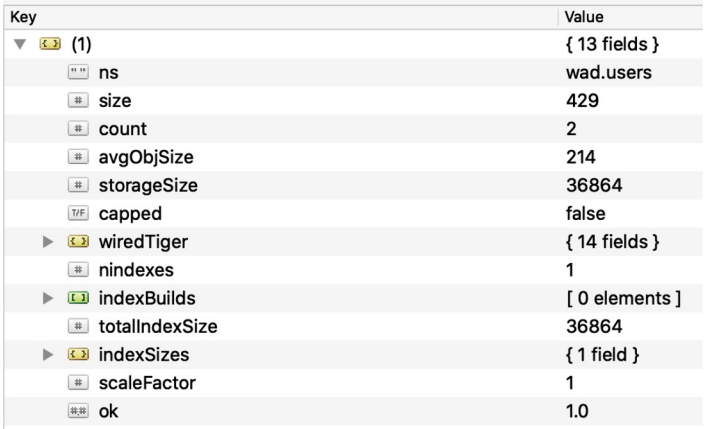
CLI Queries

List and use

```
> show dbs
admin   0.000GB
config  0.000GB
local   0.000GB
wad     0.000GB
> use wad
switched to db wad
> show collections
blogs
news
users
```

Iterate over collections

```
> db.getCollectionNames().forEach(function(collname) {
... print(collname + ": " + db[collname].count({}));
... })
blogs: 0
news: 0
users: 2
> db.users.stats()
{
        "ns" : "wad.users",
        "size" : 429,
        "count" : 2,
        "avgObjSize" : 214,
        "storageSize" : 36864,
        "capped" : false,
    ...
```



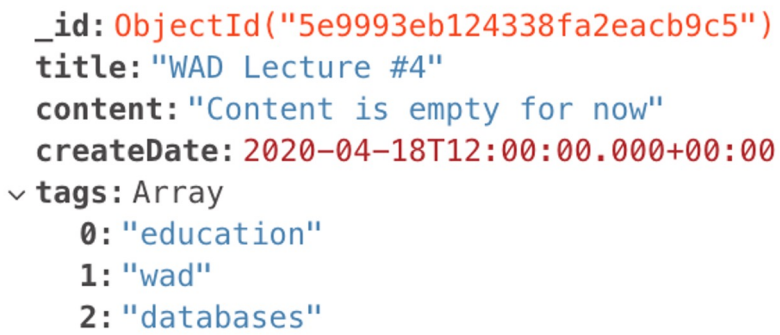
Key	Value
(1)	{ 13 fields }
ns	wad.users
size	429
count	2
avgObjSize	214
storageSize	36864
capped	false
wiredTiger	{ 14 fields }
nindexes	1
indexBuilds	[0 elements]
totalIndexSize	36864
indexSizes	{ 1 field }
scaleFactor	1
ok	1.0

Insert data

```
> db.users.insert({"username" : "admin", "password"
: "222", "registerDate" : ISODate("2020-04-
18T11:00:00Z")})
WriteResult({ "nInserted" : 1 })

> db.users.insert({"username" : "guest"})
WriteResult({ "nInserted" : 1 })
```

Insert data

```
> db.news.insert(
{"title": "WAD Lecture #4", "content": "Content is
empty for now", "createDate": ISODate("2020-04-
18T12:00:00Z"), "tags": ["education", "wad",
"databases"]
})
WriteResult({ "nInserted" : 1 })
```



ObjectId

- Default primary key
- Generates automatically
- 12 bytes size

Size	Description
4 bytes	Seconds since Unix epoch
3 bytes	Machine identifier
2 bytes	Process id
3 bytes	Counter starting from random value

ObjectId

```
> ObjectId()
ObjectId("5e999088124338fa2eacb9bb")
> ObjectId()
ObjectId("5e999089124338fa2eacb9bc")
> ObjectId()
ObjectId("5e999089124338fa2eacb9bd")
> ObjectId()
ObjectId("5e99908a124338fa2eacb9be")
> ObjectId()
ObjectId("5e99908a124338fa2eacb9bf")
> ObjectId()
ObjectId("5e99908b124338fa2eacb9c0")
> ObjectId()
ObjectId("5e99908b124338fa2eacb9c1")
> ObjectId()
ObjectId("5e99908c124338fa2eacb9c2")
```

>>> 0x5e999088
1587122312

Timestamp Converter

1587122312

Is equivalent to:

04/17/2020 @ 11:18am (UTC)

2020-04-17T11:18:32+00:00 in ISO 8601

Fri, 17 Apr 2020 11:18:32 +0000 in RFC 822, 1036, 1123, 2822

Friday, 17-Apr-20 11:18:32 UTC in RFC 2822

2020-04-17T11:18:32+00:00 in RFC 3339

Lookup data

```
> db.users.find()
{ "_id" : ObjectId("5e9975a7e8de390f5968e28d"), "username" : "user",
"password" : "123456", "registerDate" : ISODate("2020-04-
18T12:00:00Z") }
{ "_id" : ObjectId("5e9976a42482b3456c1dc4f6"), "firstName" :
"John", "lastName" : "Smith", "isAlive" : true, "age" :
NumberLong("27000000666660"), "AgE" : "123", "address" : {
"streetAddress" : "21 2nd Street", "city" : "New York", "state" :
"NY", "postalCode" : "10021-3100" }, "phoneNumbers" : [ { "type" :
"home", "number" : "212 555-1234" }, { "type" : "office", "number" :
"646 555-4567" } ], "children" : [ ], "spouse" : null }
{ "_id" : ObjectId("5e998e4b124338fa2eacb9b9"), "username" :
"admin", "password" : "222", "registerDate" : ISODate("2020-04-
18T11:00:00Z") }
{ "_id" : ObjectId("5e998e73124338fa2eacb9ba"), "username" : "guest"
}
```

Lookup data

```
> db.users.find({"username": "user"})
{
"_id" : ObjectId("5e9975a7e8de390f5968e28d"),
"username" : "user",
"password" : "123456",
"registerDate" : ISODate("2020-04-18T12:00:00Z")
}
> db.news.find({"tags.0": "education"})
{ "_id" : ObjectId("5e9993eb124338fa2eacb9c5"),
"title" : "WAD Lecture #4", "content" : "Content is
empty for now", "createDate" : ISODate("2020-04-
18T12:00:00Z"), "tags" : [ "education", "wad",
"databases" ] }
```

Lookup data

```
> db.news.find({"createDate": {$gte: new
ISODate("2012-01-12T20:15:31Z")}})
{ "_id" : ObjectId("5e9993eb124338fa2eacb9c5"),
"title" : "WAD Lecture #4", "content" : "Content
is empty for now", "createDate" : ISODate("2020-
04-18T12:00:00Z"), "tags" : [ "education", "wad",
"databases" ] }

> db.news.find({"createDate": {$lt: new
ISODate("2012-01-12T20:15:31Z")}})
>
```

https://docs.mongodb.com/manual/tutorial/query-documents/

Remove

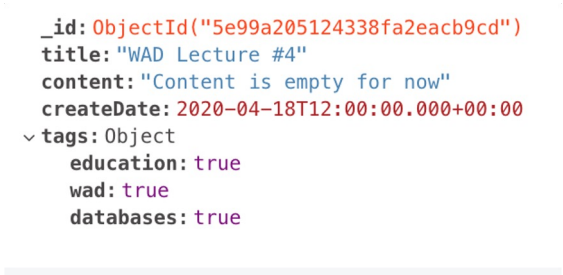
```
> db.users.remove({})
WriteResult({ "nRemoved" : 4 })
> db.news.remove({"tags.1": "wad"})
WriteResult({ "nRemoved" : 1 })

> db.news.insert({"title": "WAD Lecture #4", "content":
"Content is empty for now", "createDate": ISODate("2020-04-
18T12:00:00Z"), "tags": {"education":true, "wad":true,
"databases":true}})
WriteResult({ "nInserted" : 1 })

> db.news.remove({"tags.wad": true})
WriteResult({ "nRemoved" : 1 })
```

Remove

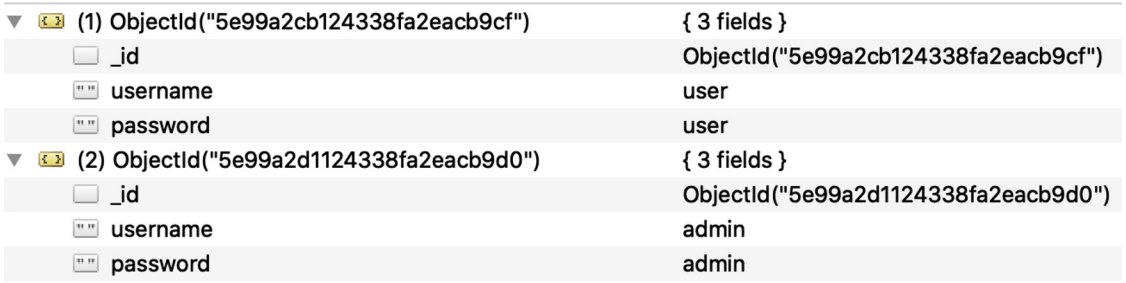
```
> db.users.remove({})
WriteResult({ "nRemoved" : 4 })
> db.news.remove({"tags.1": "wad"})
WriteResult({ "nRemoved" : 1 })

> db.news.insert({"title": "WAD Lecture #4", "content": "Content is empty for
now", "createDate": ISODate("2020-04-18T12:00:00Z"), "tags": {"education":true,
"wad":true, "databases":true}})
WriteResult({ "nInserted" : 1 })

> db.news.remove({"tags.wad": true})
WriteResult({ "nRemoved" : 1 })

> db.news.remove({"tags.wad": {$exists: true}})
WriteResult({ "nRemoved" : 1 })
```

Update

```
> db.news.updateMany( {"tags.wad": true}, {$set: {"tags.python":true}})
{ "acknowledged" : true, "matchedCount" : 2, "modifiedCount" : 2 }
```



Upsert



(1) ObjectId("5e99a2cb124338fa2eacb9cf")	{ 3 fields }
_id	ObjectId("5e99a2cb124338fa2eacb9cf")
username	user
password	user
(2) ObjectId("5e99a2d1124338fa2eacb9d0")	{ 3 fields }
_id	ObjectId("5e99a2d1124338fa2eacb9d0")
username	admin
password	admin

```
> db.users.update({"username":"user"}, {$set: {"password":
"newpass"}}, {upsert: true})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1
})
> db.users.update({"username":"guest"}, {$set: {"password":
"newpass"}}, {upsert: true})
WriteResult({
        "nMatched" : 0,
        "nUpserted" : 1,
        "nModified" : 0,
        "_id" : ObjectId("5e99a4ae2482b3456c1dd2cc")
})
```

Limit, Skip

```
> db.users.find({}).limit(3)
5e99a2cb124338fa2eacb9cf
5e99a2d1124338fa2eacb9d0
5e99a4ae2482b3456c1dd2cc

> db.users.find({}).limit(1).skip(0)
{ "_id" : ObjectId("5e99a2cb124338fa2eacb9cf"),
"username" : "user", "password" : "newpass" }
> db.users.find({}).limit(1).skip(1)
... 5e99a2d1124338fa2eacb9d0...
> db.users.find({}).limit(1).skip(2)
5e99a4ae2482b3456c1dd2cc
```

Sort

```
> db.users.find({}).sort({"username":1})
admin, guest, user
```

```
> db.users.find({}).sort({"username":-1})
user, guest, admin
```

Two fields:

```
> db.users.find({}).sort({"age": -1, "username":
-1})
```

Import and Export

```
mongodump --host "localhost" --port 27017 -d
wad -o ./output
```

```
mongorestore --host "localhost" --port 27017 -
d wad ./output
```

Literature

- MongoDB Basics: https://university.mongodb.com/courses/M001/about
- Dump & restore: https://docs.mongodb.com/manual/tutorial/backup-and-restore-tools/
- ObjectId: https://mongodb.github.io/node-mongodb-native/2.0/tutorials/objectid/

Demo

Python & MongoDB

Basic usage

```
from pymongo import MongoClient
client = MongoClient('localhost', 27017)
db = client.wad
```

Insert data

```
username = random.randint(0, 10)
password = random.randint(0, 100)
db.users.insert_one({
    "username": username,
    "password": password
})
```

db.users.insert_many(users)

```
db.users.find_one({"username": "user"})

db.users.find_one({"username": "user", "password":
                   "pass"})
```

```
for user in db.users.find({"username": 3}):
    print(user)
```

Flask-PyMongo

```
from flask import Flask, render_template, request, redirect
from flask_pymongo import PyMongo

app = Flask(name)
app.config["MONGO_URI"] = "mongodb://localhost:27017/wad"
mongo = PyMongo(app)


@app.route("/")
def home_page():
    online_users = mongo.db.users.find({})
    return render_template("list.html", users=online_users)


if name == "main":
    app.run(host='localhost', port=5000, debug=True)
```

Literature

- Pymongo docs: https://api.mongodb.com/python/3.8.0/
- Flask-PyMongo: https://flask-pymongo.readthedocs.io/en/latest/

Demo

Indexes

```python
from pymongo import MongoClient
import random
client = MongoClient('localhost', 27017)
db = client.wad

def generate_users():
    for user in range(1000):
        db.users.insert_many([{
            "username": random.randint(0, 1000000),
            "password": random.randint(0, 1000000)
    } for i in range(5000)])
```

5100000 documents in collection

With index

```
db.getCollection('users').createIndex({"username": 1})
```

```
db.getCollection('users').find({"username":811091})
```

users 0.05 sec.

Key	Value
(1) ObjectId("5e9a96e8472c0fe78d54d383")	{ 3 fields }
(2) ObjectId("5e9a9740487276165df70a88")	{ 3 fields }
(3) ObjectId("5e9a9755487276165d057e52")	{ 3 fields }
(4) ObjectId("5e9a975c487276165d0adc73")	{ 3 fields }
(5) ObjectId("5e9a9764487276165d107458")	{ 3 fields }
(6) ObjectId("5e9a9791487276165d30ff36")	{ 3 fields }
(7) ObjectId("5e9a9796487276165d34fa35")	{ 3 fields }
(8) ObjectId("5e9a979e487276165d3afd63")	{ 3 fields }

Without index

```
db.getCollection('users').find({"username":811091})
```

users 5.24 sec.

Key	Value
(1) ObjectId("5e9a96e8472c0fe78d54d383")	{ 3 fields }
(2) ObjectId("5e9a9740487276165df70a88")	{ 3 fields }
(3) ObjectId("5e9a9755487276165d057e52")	{ 3 fields }
(4) ObjectId("5e9a975c487276165d0adc73")	{ 3 fields }
(5) ObjectId("5e9a9764487276165d107458")	{ 3 fields }
(6) ObjectId("5e9a9791487276165d30ff36")	{ 3 fields }
(7) ObjectId("5e9a9796487276165d34fa35")	{ 3 fields }
(8) ObjectId("5e9a979e487276165d3afd63")	{ 3 fields }

Explain

db.getCollection('users').find({"username":811091}).explain("executionStats")

~5s

db.getCollection('users').find({"username":811091}).limit(1).explain("executionStats")

~0.08s

Explain with index

db.getCollection('users').find({"username":811091}).explain("executionStats")

~0.002s

db.getCollection('users').find({"username":811091}).limit(1).explain("executionStats")

~0.002s

Text search

⚙️ **Generate**

```
db.stores.insert_many(
    [
        { "name": "Java Hut", "description": "Coffee and cakes" },
        { "name": "Burger Buns", "description": "Gourmet hamburgers" },
        { "name": "Coffee Shop", "description": "Just coffee" },
        { "name": "Clothes Clothes Clothes", "description": "Discount clothing" },
        { "name": "Java Shopping", "description": "Indonesian goods" }
    ]
)
db.stores.create_index([("name", "text"), ("description", "text")])
```

🔎 **Query**

```
db.stores.find( { $text: { $search: "java coffee shop" } } )
```

```
db.getCollection('stores').find({ $text: { $search: "java coffee shop" } })
```

stores 0.003 sec.

Key	Value
(1) ObjectId("5e9a997a0c3169fb1ba1f726")	{ 3 fields }
_id	ObjectId("5e9a997a0c3169fb1ba1f726")
name	Coffee Shop
description	Just coffee
(2) ObjectId("5e9a997a0c3169fb1ba1f724")	{ 3 fields }
_id	ObjectId("5e9a997a0c3169fb1ba1f724")
name	Java Hut
description	Coffee and cakes
(3) ObjectId("5e9a997a0c3169fb1ba1f728")	{ 3 fields }
_id	ObjectId("5e9a997a0c3169fb1ba1f728")
name	Java Shopping
description	Indonesian goods

🔎 Query

```
db.stores.find( { $text: { $search: "\"coffee shop\"" } } )
```

```
db.getCollection('stores').find({ $text: { $search: "\"coffee shop\"" } })
```

stores 0.003 sec.

Key	Value
(1) ObjectId("5e9a997a0c3169fb1ba1f726")	{ 3 fields }
_id	ObjectId("5e9a997a0c3169fb1ba1f726")
name	Coffee Shop
description	Just coffee

🔎 Query

```
db.stores.find( { $text: { $search: "java shop -coffee" } } )
```

```
db.getCollection('stores').find({ $text: { $search: "java shop -coffee" } })
```

stores 0.004 sec.

Key	Value
(1) ObjectId("5e9a997a0c3169fb1ba1f728")	{ 3 fields }
_id	ObjectId("5e9a997a0c3169fb1ba1f728")
name	Java Shopping
description	Indonesian goods

Literature

- Create index: https://docs.mongodb.com/manual/indexes/#create-an-index
- Text search: https://docs.mongodb.com/manual/text-search/

Python & PostgreSQL

Basic usage

```
from flask_sqlalchemy import SQLAlchemy

app = Flask(__name__)
# app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///students.sqlite3'
app.config['SQLALCHEMY_DATABASE_URI'] = 'postgresql+psycopg2://{user}:{passwd}@{host}:{port}/{db}'.format(
       user='marco',
       passwd='foobarbaz',
       host='localhost',
       port='5432',
       db='db')
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
app.secret_key = 'foobarbaz'
db = SQLAlchemy(app)
```

SQL Model

```
class students(db.Model):
    id = db.Column('student_id', db.Integer, primary_key=True)
    name = db.Column(db.String(100))
    city = db.Column(db.String(50))
    addr = db.Column(db.String(200))

    def __init__(self, name, city, addr):
        self.name = name
        self.city = city
        self.addr = addr
```

SQL operations

```
# init db
db.create_all()

# create new student
student = students(request.form['name'], request.form['city'],
request.form['addr'])
db.session.add(student)
db.session.commit()

# find all
students = students.query.all()
```

Literature

- Using SQLAlchemy with Flask and PostgreSQL: https://stackabuse.com/using-sqlalchemy-with-flask-and-postgresql/
- Flask-SQLAlchemy docs: https://flask-sqlalchemy.palletsprojects.com/en/2.x/

Homework 2

Basic part:

Implement authentication feature

1.1 Listen on localhost:5000

1.2 Render authentication form at http://localhost:5000/

1.3 Redirect user to profile page if successfully authenticated

1.4 Show profile page for authenticated user only at http://localhost:5000/profile

1.5 User name and password are stored in Mongodb

Advanced part:

2.1 Implement feature that allows users to create new account, profile will be shown with data respected to each account.

2.2 Implement password hashing, logout and password change features

2.3 Allow users to update profile picture (new user will have a default profile picture)

2.4 Allow users to update profile information

Homework 2 — checklist

1. You have chosen an appropriate difficulty level for you:
   a. Basic (if you don’t have coding experience)
   b. Advanced (if you know how to code)
   c. Challenging (if you have a significant experience)
2. You have a homework2 solution committed to the https://github.com/itmo-wad/*yourname*-hw2

Questions?