# Databases for Data Science

Lecture 09 · 2022-10-05

# The plan

| Lectures | | |
|---|---|---|
| Today | 10/05 | MongoDB: Python, CRUD, Aggregation (MapReduce) |
| Friday | 10/07 | MongoDB: Exercises, Database Design |
| Monday | 10/10 | MongoDB wrap-up; discuss other NoSQL DBs |
| Wednesday | 10/12 | JSON in PostGreSQL; topics |
| | | |
| *After break* Monday | 10/24 | Final Exam |

# MongoDB: Python Interface

Previously, we've been using JavaScript to interact with MongoDB through the `mongo` shell.

We can also use the official `pymongo` library to control Mongo through python.

```
### Put this in a Jupyter notebook

import pymongo
client = pymongo.MongoClient("localhost",27017) # Connect to the local Mongo server
db = client.sample_airbnb # Select a database
```

The syntax of pymongo is very similar to what we've seen in Javascript.

```
db.listingsAndReviews.find().next()

db.listingsAndReviews.find({"address.country_code": "US"}).next()
```

**Exercise**: In your Jupyter notebook, run a query to find all listings with two bedrooms.

**MAKE SURE YOU SELECT `Python 3.6.8` IN THE TOP RIGHT.** (Otherwise, you'll get an error.)

# pymongo: Reading and Writing Data

We can use the `bson` library to print output.

- Non-JSON types like datetimes can't be directly serialized by the `json` library.

```
from bson.json_util import loads,dumps

coll = db.listingsAndReviews # for convenience

output = coll.find().next()

print(dumps(output))

with open('example.json', 'w') as f:
  f.write(dumps(output, indent=2)) # Use `indent` kwarg to pretty-print
```

**Exercise**: Use Python to write the output of your two-bedrooms query to file.

- Use `next()` to get only the first document (otherwise it takes a while).

# Recap: Query Operators

We've used this syntax for *equality* filters:

```
# { <field>: <value>}
coll.find({ 'property_type': "House" });
```

We can also use *operators* to define more general filters.

```
# { <field>: { <operator>: <value> } }

# Comparison
coll.find({ 'accommodates': { '$gt': 5 } })

# Inclusion
coll.find({ 'address.country_code': { '$in': ['US','CA'] } })
```

# Recap: Query Operators

```
coll.find({ 'accommodates': { '$gt': 5 } });
```

Full list: https://www.mongodb.com/docs/v4.4/reference/operator/query/

**Exercise**: Find all of the listings with a rating above least 90.

**Exercise**: Find all of the listings with more than two bedrooms and a weekly price of less than $1000.

**Exercise**: Find all of the condos and apartments in the US whose host is named "Alex".

*Note:* If you're not sure what properties exist on a document, you can call `.keys()`.

# Recap: Query Operators

Queries on array contents:

```
# Match one array item
coll.find({ 'amenities': "Internet" })

# Match two array items
coll.find({ 'amenities': { '$all': ["Internet", "TV"] } })

# Match *exact* array contents (this will be empty!)
coll.find({ 'amenities': ["Internet", "TV"] })

# Match document within array
coll.find({ "reviews.reviewer_name": "Xavier" })
```

**Exercise**: Find all of the apartments with TV *or* internet service.

# MongoDB | Projection

We can also choose which parts of each document to return.

```
# Get the street address and bedroom count of every listing
coll.find({}, { "address.street": 1, 'bedrooms': 1 })

# Get the amenity list and review score for every listing without internet
coll.find(
  { 'amenities': { '$ne': "Internet" } },
  { 'amenities': 1, "review_scores.review_scores_rating": 1 }
)
```

- What does this correspond to in SQL?

**Exercise**: Get the host name of every listing in Brazil.

# Mapping data in the output

In Python, we can use *list comprehensions* to apply a function to every item in the output.

```
# Get the number of properties in each output value
[len(i) for i in coll.find()]
```

This can also be done with `for`-loops, etc.

**Exercise**: Write a function to decide whether you would consider staying at a given listing. List the name of every property that meets your criterion.

# MongoDB | Inserting data

Select your own database:

- `db = client.[YOUR NAME HERE]`

Create a new collection:

- `db.create_collection("students")`

You can also create a collection *implicitly* by adding data to it!

# MongoDB | Inserting data

It's pretty simple:

```
# db.collection.insert( <object> )

db.students.insert_one({"name": "Alex"});
db.students.insert_one({"student_id": "N10234567"});

# Use an array to insert several objects
db.students.insert_many([{"name": "Charlie"}, {"name": "Dylan"}])
```

At this point, you can put *anything* in any collection.

- Why might this be helpful?
- Why might this be a problem?

# MongoDB | Inserting data

Python can be useful for automating tasks.

```
names = ["Alex","Bryce","Charlie","Dylan"]
new_students = []

for name in names:
  new_students.append({'name':name})

db.students.insert_many(new_students)
```

**Exercise:** Use Python to generate and insert your own set of students. Assign each one a unique email address based on their name.

# MongoDB | Update

Where we had `.find(filter, projection)`, we now have `.update_many(filter, command)`.

```
db.students.update_many({"name": "Alex"}, { '$set': {"gpa": 4.0}})
```

**Exercise:** Assign random GPAs to all of your students.

# MongoDB | Delete

Similarly to a `DELETE FROM tablename WHERE condition`,
we have `.delete_many(filter)`.

```
# Remove matching records
db.student.delete_many({"name": "Bryce"});

# Remove all records
db.student.delete_many({});
```

**Exercise:** Delete all records for students with GPA below 1.0.

# MongoDB | Documentation

**MongoDB Manual:**
https://www.mongodb.com/docs/v6.0/

**Cheat sheet:**
https://www.mongodb.com/developer/products/mongodb/cheat-sheet/

**PyMongo manual:**
https://pymongo.readthedocs.io/en/stable/