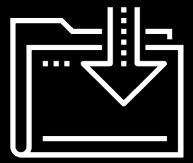
PyMongo and Advanced Queries

Data Boot Camp

Lesson 12.2



Class Objectives

By the end of today's class you will be able to:

Use the PyMongo library to interface with MongoDB and perform basic CRUD operations.

Select specific fields when retrieving documents from MongoDB.

Use comparison operators to find documents in MongoDB.

Use sort and limit with PyMongo when retrieving documents from MongoDB.

Instructor Demonstration

Introduction to PyMongo

What is PyMongo?

PyMongo is a library that allows interaction with a MongoDB database through Python.
It is a native driver for MongoDB.

Introduction to PyMongo

Instructions

Install PyMongo into your environment by following the steps provided by your instructor.

Once you have PyMongo installed, open Jupyter Notebooks and import the module in your first line of code.

Create a connection with a running instance.

```
# Import PyMongo
from pymongo import MongoClient

# The default port used by MongoDB is 27017
mongo = MongoClient(port=27017)
```

Introduction to PyMongo

Instructions

Create a database named `classDB` and assign it to a variable called `db` using `mongo.classDB`.

Next, create and insert our first document into the collection.

```
# Define the 'classDB' database in Mongo
db = mongo.classDB

# Insert a document into the 'classroom' collection
db.classroom.insert_one(
    {
        'name': 'Ahmed',
        'row': 3,
        'favorite_python_library': 'Matplotlib',
        'hobbies': ['Running', 'Stargazing', 'Reading']
    }
)
```

Introduction to PyMongo

Instructions

Update the document.

Then add an item to a document array.

```
# Update a document
db.classroom.update_one(
    {'name': 'Ahmed'},
    {'$set':
        {'row': 4}
     }
)
# Add an item to a document array
db.classroom.update_one(
    {'name': 'Ahmed'},
    {'$push':
        {'hobbies': 'Listening to country music'}
    }
)
```

Introduction to PyMongo

Instructions

Remove a field from a document.

Then add an item to a document array.

```
# Delete a field from a document
db.classroom.update_one({'name': 'Ahmed'},
                        {'$unset':
                               {'row': ""}
                        }
                       )

# Delete a document from a collection
db.classroom.delete_one(
    {'name': 'Ahmed'}
)
```

Questions?

Activity: Mongo Grove

In this activity, you will practice using PyMongo to enable Python to interact with MongoDB. Specifically, you'll build a command-line interface application for the produce department of a supermarket.

Suggested Time:

25 minutes

Activity: Mongo Grove

Use	Use PyMongo to create a `fruits_db` database and a `fruits` collection.
Insert	Insert two documents of fruit shipments received by your supermarket into the collection. They should contain the following information: • A vendor with the key, `vendor`. • The type of fruit with the key, `fruit`. • The number of cases with the key, `case_quantity`, and the number of cases received as an integer • The status of the fruit with the key, `ripeness` on a scale of 1 to 3, where the ratings are: 1 for unripe, 2 for ripe, 3 for over-ripe. • The date entered in the database in UTC datetime format with the key `date`.
Create	Create a Python script that asks the user for the above information, then inserts a document into the fruits collection.
Hint	Consult the documentation on the datetime library.

Time’s Up! Let’s Review.

Questions?

Instructor Demonstration

PyMongo with Imported CSV Data

Questions?

Activity: PyMongo with Imported JSON Data

In this activity, you will create your own Mongo database by importing a customer database in a JavaScript Object Notation (JSON) file, and use PyMongo to interface with the data.

Suggested Time:

20 minutes

PyMongo with Imported JSON Data

Instructions

In the terminal, use `cd` to navigate to the `Resources` folder that contains the `customer_list.json` file.

Use the following command to import this file into a Mongo database:

```
mongoimport --type json -d petsitly_marketing -c customer_list --drop --jsonArray customer_list.json
```

Create an instance of MongoClient by using Port Number 27017.

List the database names to confirm that the `petsitly_marketing` database was created.

Assign the `petsitly_marketing` database to a variable.

List the names of the collections in the database.

Use the `find_one` function to review a document in the `customer_list` collection of your database.

Assign the `customer_list` collection to a variable of your choice.

Use the `insert_one` function to insert the new customer into the database, and then run the query in the following cell to review this customer.

Time’s Up! Let’s Review.

Questions?

Instructor Demonstration

Selecting Fields

Count Documents

We can use `collectionName.count_documents(query)` to count how many results will be retrieved with our query.

```
# Create a query that finds the customers who have a Toyota
query = {'car_make': "Toyota"}

# Print the number of results
print("Number of documents in result:", customers.count_documents(query))
```

To count all documents, we send an empty query to `count_documents()`.

```
# Display the number of documents in the customers collection
customers.count_documents({})
```

Update Data Types in PyMongo

We use `update_many()` to update data types in PyMongo.

```
# Change the data type from String to Double for wages.hourly_rate
mechanics.update_many({}, [
                                {'$set':
                                     { "wages.hourly_rate" :
                                          {'$toDouble': "$wages.hourly_rate"}
                                     }
                                }
                            ]
                        )
```

Selecting Fields

To select specific fields to be returned in our results, we need to pass a second dictionary as an argument to the `find()` method in PyMongo.

```
# Select only the mechanic_name and wages.hourly_rate fields from the mechanics collection
query = {}
fields = {'mechanic_name': 1, 'wages.hourly_rate': 1}

# Capture the results to a variable
results = mechanics.find(query, fields)

# Pretty print the results
for result in results:
    pprint(result)
```

Selecting Fields

We can also deselect specific fields to remove them from being returned in our results.

```
# Select every field from the mechanics collection except the car_specialties field
query = {}
fields = {'car_specialties': 0}

# Capture the results to a variable
results = mechanics.find(query, fields)

# Pretty print the first two results
for i in range(2):
    pprint(results[i])
```

Questions?

Activity: Air Fields

In this activity, you will practice selecting specific fields from a Mongo database using PyMongo.

Suggested Time:

15 minutes

Air Fields

Instructions

Open AirFields_Unsolved.ipynb and follow the instructions to import the data for this activity.

Run the first 6 blocks to connect to the `epa` database and assign each collection to a variable. If you run into any errors, make sure that you have properly imported the data from your Resources folder.

Display the total number of documents in the `annual_aqi_by_county` collection using `count_documents()`.

Create a query that finds the documents that have a "parameter" of "Sulfur dioxide" in the `ohio_air` collection and print the number of results using `count_documents()`.

Pretty print just the first result from the previous "Sulfur dioxide" query using list indexing.

Air Fields

Instructions

Select only the `parameter`, `units_of_measure`, `observation_count`, `date_local`, `local_site_name`, `site_address`, `city`, and `county` fields from the `ohio_air` collection and use pretty print to print the first two results.

Select every field from the `ohio_daily_records` collection except the COUNTY_CODE and STATE_CODE fields and use pretty print to print the first two results.

In the `ohio_daily_records` collection, change the data types of the following fields:

- `CO.PERCENT_COMPLETE` should be converted to a double
- `CO.DAILY_AQI_VALUE` should be converted to an integer

Note

You can update data types in a single `update_many()` query, but you may prefer to update them separately.

Air Fields

Challenge

Create a query that finds the documents in the `ohio_daily_records` collection where `CO.UNITS` matches "ppm" and `NO2.UNITS` matches "ppb", and select only the following fields: `CBSA_NAME`, `COUNTY`, `Site Name`, `Date`, `CO`, `NO2`, and `SO2`. Use pretty print to print the first two results.

Create a query that finds the documents where the `State` is "Ohio" in the `annual_aqi_by_county` collection and returns only the `County`, `State`, `Days with AQI`, and `Max AQI fields`. Use pretty print to print the first four results.

Time’s Up! Let’s Review.

Questions?


Break

Instructor Demonstration

Comparison Operators

Comparison Operators

So far, we have only retrieved documents from a collection by searching for an exact match.

```
# Find the customers who have a Toyota
db.customers.find(
    {
        'car_make': "Toyota"
    }
)
```

But what do we do when we want to retrieve documents that aren't an exact match? Then, we can use **comparison operators**.

Comparison Operators

We can use comparison operators to match documents in a collection.

Operator	Function
`$gte`	Matches values that are greater than or equal to (≥) a specified value
`$lte`	Matches values that are less than or equal to (≤) a specified value
`$gt`	Matches values that are greater (>) than a specified value
`$lt`	Matches values that are less than (<) a specified value
`$in`	Checks if the value of the field is in a list of specified values
`$nin`	Checks if the value of the field is not in a list of specified values
`$eq`	Checks if the value of the field is equal to a specified value
`$ne`	Checks if the value of the field is not equal to a specified value
`$regex`	“Regular expression” can be used to find specific words in the value of a field

NOTE

For most data types, comparison operators only perform comparisons on fields where the BSON type matches the query value's type.

Comparison Operators

Instructions

Find the customers who have cars from 2010 or later.

```
# Create a query that finds the customers who have cars from 2010 or later.
query = {'car_year': {'$gte': 2010}}

# Capture the results to a variable
results = customers.find(query)

# Pretty print the first two results
for i in range(2):
    pprint(results[i])
```

Comparison Operators

Instructions

Find the customers who have cars manufactured earlier than 1990.

```
# Create a query that finds the customers who have cars that were manufactured
# before 1990
query = {'car_year': {'$lt': 1990}}

# Capture the results to a variable
results = customers.find(query)

# Pretty print the first two results
for i in range(2):
    pprint(results[i])
```

Comparison Operators

Instructions

Find the customers who have “Nye” in their name.

```
# Create a query that finds the customer(s) who have "Nye" in their name
query = {'full_name': {'$regex': "Nye"}}

# Capture the results to a variable
results = customers.find(query)

# Pretty print the first two results
for i in range(2):
    pprint(results[i])
```

Comparison Operators

Instructions

Find all the customers who have cars that "Dacey Cocom" can work on.

```
# To create a query that finds the customers who have cars that the mechanic "Dacey
# Cocom" can work on, first we must find the types of cars Dacey specializes in.
query = {'mechanic_name': "Dacey Cocom"}
fields = {'mechanic_name': 1, 'car_specialties': 1}

# Capture the results to a variable
results = list(mechanics.find(query, fields))

dacey_cars = results[0]['car_specialties']
dacey_cars
```

```
['Jaguar', 'Hummer', 'Mitsubishi', 'Geo', 'Holden', 'Rolls-Royce', 'Mercury', 'Subaru',
'Maybach']
```

Comparison Operators

Instructions

Find all the customers who have cars that "Dacey Cocom" can work on.

```
# Create a query that finds the customers who have cars that the mechanic "Dacey Cocom"
# can work on.
query = {'car_make': {'$in': dacey_cars }}

# Capture the results to a variable
results = customers.find(query)

# Pretty print the first two results
for i in range(2):
    pprint(results[i])
```

Questions?

Activity: Find Pets

In this activity, you will revisit the data from Petsitly Marketing and practice using comparison operators in MongoDB with PyMongo.

Suggested Time:

15 minutes

Find Pets

Instructions	Open FindPets_Unsolved.ipynb and follow the instructions at the top of the notebook to import the Petsitly data again.
	Create a query that finds the customers who had over 50 visits in 2021 and pretty print the first two results.
	Create a query that finds the customers who spent $250 or less in 2021 and pretty print the first two results.
	Create a query that finds the customer(s) who live in an apartment with "Suite" in the address and pretty print the first three results.
	Create a query that finds the customers who have turtles or fish and pretty print the first three results.
Hint	The `$regex` operator is used to match partial strings.

Time’s Up! Let’s Review.

Questions?

Instructor Demonstration

Sort and Limit

Sort

Pandas vs. PyMongo

Pandas

```
.sort_values(by=[column1, column2],
             ascending=True)
```

Accepts arguments:

- by: column name or a list of the columns to sort on
- ascending: boolean value or list of boolean values

PyMongo

```
.sort([(field1, 1), (field2, -1)])
```

Accepts a list of tuples to sort on, formatted as:

```
(field_name, sort_direction)
```

Sort direction:

-1	descending
1	ascending

Sort

Instructions

Sort in ascending order by last_service.

```
# Create a query that sorts in ascending order by last_service.
query = {}
sort = [('last_service', 1)]

# Capture the results to a variable
results = customers.find(query).sort(sort)

# Pretty print the first five results
for i in range(5):
    pprint(results[i])
```

Sort and Limit

Instructions

Sort in ascending order by last_service and limits to the first five results.

```
# Create a query that sorts in ascending order by last_service
# and limits the results to the first 5.
query = {}
sort = [('last_service', 1)]
limit = 5

# Pretty print the results
pprint(list(customers.find(query).sort(sort).limit(limit)))
```

Sort and Limit

Instructions

Find customers with a "Nissan" or "Hyundai".

Sort in in descending order by car_year, then ascending order by last_service, then limit to the first five results.

```
# Create a query that:
# finds customers with a "Nissan" or "Hyundai"
# sorts in descending order by car_year, then ascending order by last_service
# limits the results to the first 5
query = {'car_make': {'$in': ["Nissan", "Hyundai"]}}
sort = [('car_make', -1), ('last_service', 1)]
limit = 5

# Pretty print the results
pprint(list(customers.find(query).sort(sort).limit(limit)))
```

Questions?

Activity: Sort and Limit Pets

In this activity, you will revisit the data from Petsitly Marketing and practice using the sort and limit methods with PyMongo, while combining these new methods with other query building techniques.

Suggested Time:

15 minutes

Sort and Limit Pets

Instructions

Open SortAndLimitPets.ipynb and run the first few blocks of code to connect to the database and store the `customer_list` collection as a variable.

Create a query that finds customers who have cats or dogs, then sorts in descending order by `2021_Total_Spend`, and limits the results to the first 5. Pretty print the results.

Create a query that finds customers who spent less than $500 in 2021, then sorts in ascending order by `Customer_Last`, and limits the results to the first 5. Pretty print the results.

Pretty print the results of a query that:

- Finds customers who spent less than $500 **and** had more than 20 visits in 2021.
- Removes the `Address` and `Email` fields from the results.
- Sorts in ascending order by `2021_Visits`, then `2021_Total_Spend`.
- Limits the results to the first 8.

Time’s Up! Let’s Review.

Questions?


The End