

Databases in Flask



WEB 1.1

- Learning Outcomes
- Warm-Up: Review Databases
- CRUD Routes
- **BREAK**
- Lab Time
- Wrap-Up

By the end of today, you should be able to...

1. **Describe** the structure of a MongoDB database.
2. **Use** database operations to find, insert, update, and delete documents.
3. **Use** the Robo 3T program to look inside your database.

Review: Databases

What is MongoDB?

In this class, we'll be using **MongoDB** as our database, and **PyMongo** as the Python library to connect to it.

MongoDB is called a **NoSQL**, or **Document-based** database, because we don't have to specify ahead of time what our data will look like. Instead, we can store **any key-value pairs** in a database item.



Warm-Up (5 minutes)

Explain to your partner the difference between a **Document**, a **Collection**, and a **Database**. Pretend you are explaining to your 7-year-old cousin. Then, switch partners.

Review: Database Operations

We can import the PyMongo module in our code:

```
from flask import Flask
from flask_pymongo import PyMongo

app = Flask(__name__)
app.config["MONGO_URI"] = "mongodb://localhost:27017/myDatabase"

mongo = PyMongo(app)
```

Then, we can use the **mongo.db** object directly in our routes:

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


There are **four operations** we can do on a database document. You can remember them with the acronym **C.R.U.D.**

Create

Read

Update

Delete

We can **create a new document** using the operation `insert_one`.

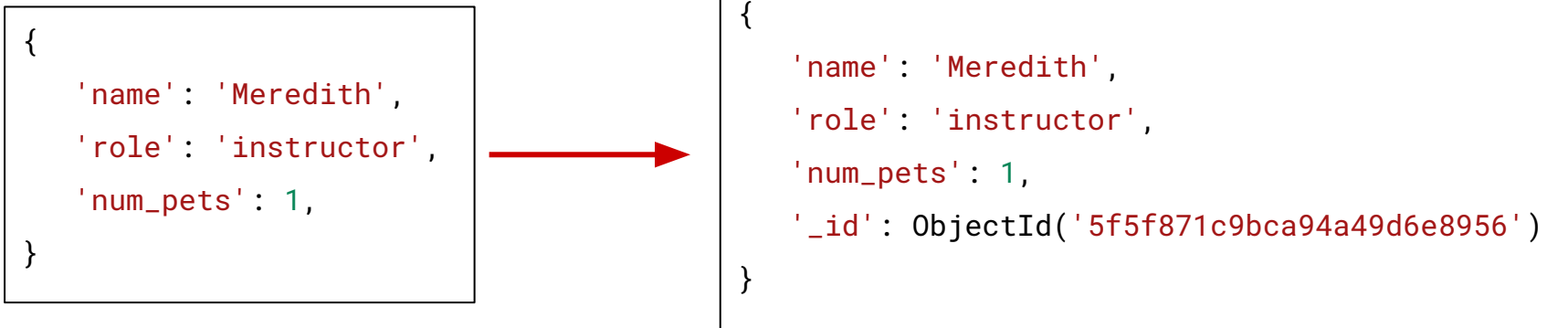
```
new_user = {  
    'first_name': 'Meredith',  
    'role': 'instructor',  
    'num_pets': 1  
}  
  
result = users_collection.insert_one(new_user)
```

This means that we are creating a new document in the **users** collection of the **db** database.

What is an ObjectId?

All new objects in the database are automatically given a field **_id** which is an randomly-generated 24-digit hexadecimal string.

However, this field is stored as type **bson.objectid.ObjectId**, not as a string.



We can get all objects in a collection by using **find**:

```
all_users = users_collection.find()
for user in all_users:
    print(user['name'])
```

We can also get all objects matching some constraint(s):

```
all_instructors = users_collection.find({'role': 'instructor'})
for user in all_instructors:
    print(user['name'])
```

Activity (10 minutes)

Use this [Repl.it](#) to practice using the **find** and **insert_one** operations.

We can get one single object using **find_one**:

```
user1 = users_collection.find_one({'first_name': 'Meredith'})  
print(user1)  
  
>>> {'name': 'Meredith', 'role': 'instructor', 'num_pets': 1,  
      '_id': ObjectId('5f5f871c9bca94a49d6e8956')}
```

We can update an existing entry using **update_one** and setting a field called **\$set** in the second parameter.

```
user1 = users_collection.update_one({
    'first_name': 'Meredith'
},
{
    '$set': { 'num_pets': 2 }
})
print(user1)

>>> {'name': 'Meredith', 'role': 'instructor', 'num_pets': 2,
'_id': ObjectId('5f5f871c9bca94a49d6e8956')}
```

Activity (5 minutes)

Use this [Repl.it](#) to practice using the **update_one** operation.

We can delete an entry by using **delete_one**.

```
result = users_collection.delete_one({  
    'first_name': 'Meredith'  
})  
print(result.deleted_count)  
  
>>> 1
```

Activity (5 minutes)

Use this [Repl.it](#) to practice using the **delete_one** operation.

Break - 10 min

CRUD Routes in Flask

Most simple applications with 1-2 resources follow a similar format. For each resource, they usually include:

- A **List** page (list all items of the resource)
- A **Detail** page (show 1 individual item of the resource)
- A **Create** page (create a new object of the resource)
- An **Edit/Update** page (update an existing object of the resource)
- A **Delete** page (delete an existing object of the resource)

Each of these pages typically corresponds to **one database operation**.

Route	Database Operation
List page	<code>find()</code>
Detail page	<code>find_one()</code>
Create page	<code>insert_one()</code>
Edit page	<code>update_one()</code>
Delete page	<code>delete_one()</code>

We can import the PyMongo module in our code:

```
from flask import Flask
from flask_pymongo import PyMongo

app = Flask(__name__)
app.config["MONGO_URI"] = "mongodb://localhost:27017/myDatabase"
mongo = PyMongo(app)
```

Database name

Then, we can use the **mongo.db** object directly in our routes:

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

Collection name

We can **create a new document** using the operation `insert_one`.

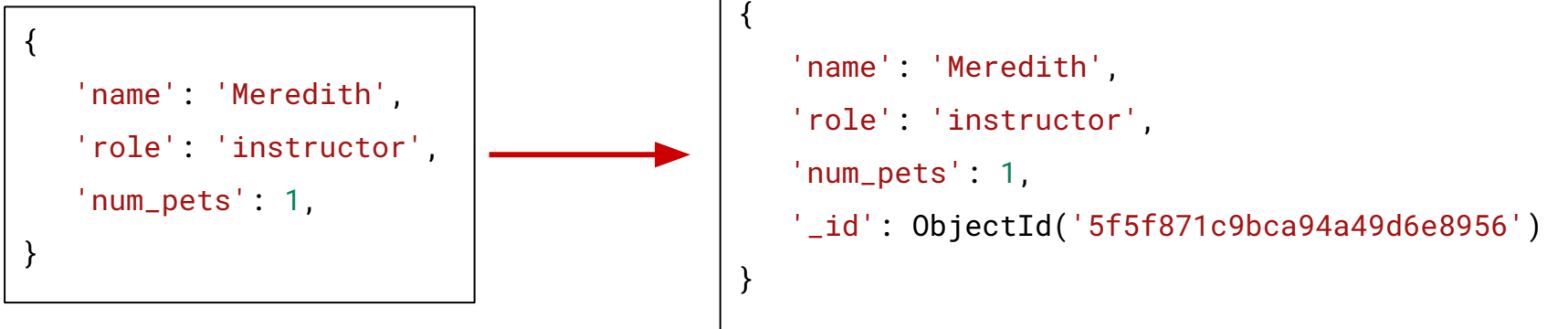
```
new_user = {  
    'first_name': 'Meredith',  
    'role': 'instructor',  
    'num_pets': 1  
}  
  
result = mongo.db.users.insert_one(new_user)
```

This means that we are creating a new document in the **users** collection of the **db** database.

What is an ObjectId?

All new objects in the database are automatically given a field **_id** which is an randomly-generated 24-digit hexadecimal string.

However, this field is stored as type **bson.objectid.ObjectId**, not as a string.



We can get all objects in a collection by using **find**:

```
all_users = mongo.db.users.find()
for user in all_users:
    print(user['name'])
```

We can also get all objects matching some constraint(s):

```
all_instructors = mongo.db.users.find({'role': 'instructor'})
for user in all_instructors:
    print(user['name'])
```

We can get one single object using **find_one**:

```
user1 = mongo.db.users.find_one({'first_name': 'Meredith'})
print(user1)

>>> {'name': 'Meredith', 'role': 'instructor', 'num_pets': 1,
'_id': ObjectId('5f5f871c9bca94a49d6e8956')}
```

We can update an existing entry using **update_one** and setting a field called **\$set** in the second parameter.

```
user1 = mongo.db.users.update_one({
    'first_name': 'Meredith'
},
{
    '$set': { 'num_pets': 2 }
})
print(user1)

>>> {'name': 'Meredith', 'role': 'instructor', 'num_pets': 2,
'_id': ObjectId('5f5f871c9bca94a49d6e8956')}
```

Whenever we do a MongoDB operation, we get a **return value** back. This can tell us things about the operation, such as whether it was successful, how many documents were operated upon, etc.

Explore the [Result class definitions](#) documentation page to learn more about these return types.

Run the code for the [Fruits Database Repl](#). Notice what happens after each database operation is run.

Note that this code is using a different library (PyMongo vs. Flask-PyMongo) - hence why we are operating on `my_collection` and not `mongo.db.fruits`. Otherwise, the database operations are the same.

Robo 3T

There are many programs that allow us to look inside of a database. The easiest one to use (in my experience) is Robo 3T.

[Download and install Robo 3T](#) (not Studio 3T, that's the paid version).

Your instructor will demonstrate how to use it.

Database Operations in Flask

You may have noticed in Homework 4 that after we have completed a database operation, we sometimes **redirect the user to a different page** instead of just rendering a template. Why?

This is to fix the **Double Submit Problem**: If a user submits a form and refreshes the page, it will re-submit the form.

So, it's a good practice to **always redirect the user** after a successful POST form submit.

We can use the Flask `redirect` method to send the user to another page on form submit.

```
@app.route('/send_email', methods=['GET', 'POST'])
def send_email():
    if request.method == 'POST':
        # ... Process form ...
        return redirect('/')

    else: # method was a GET
        return render_template('email_form.html')
```

Send the user to the homepage!

The method `url_for` is a shortcut that makes redirects easier to write. It **accepts the name of a route function** (as well as any parameters) and **returns a URL string**.

```
@app.route('/')  
def plants_list():  
    # ...  
  
@app.route('/plant/<plant_id>')  
def detail(plant_id):  
    # ...
```

`url_for('plants_list')` → `'/'`

`url_for('detail', plant_id=1)`
→ `'/plant/1'`

Let's do some live pair programming to demonstrate how to complete the **List Page** and **Creation Page** in Homework 4.

Lab Time