learn-co-curriculum / dsc-mongodb-lab Public

View license

0 stars 250 forks

Star Watch

Code Issues Pull requests Actions Projects Security Insights

solution

This branch is 6 commits ahead, 8 commits behind master. Contribute

Cheffrey2000 fixed reference to previous lti … on Oct 1, 2020 7

View code

README.md

MongoDB - Lab

Introduction

In this lesson, we'll get some hands-on experience with MongoDB!

Objectives

You will be able to:

- Create a MongoDB database
- Insert data into a MongoDB database
- Read data from a MongoDB database
- Update data in a MongoDB database

Getting Started

To begin this lab, make sure that you start up the mongoDB server in your terminal first! **You must do this lab locally on your computer.**

Connecting to the MongoDB Database

In the cell below, import the appropriate library and connect to the mongoDB server. Create a new database called `'lab_db'` .

```
import pymongo

myclient = pymongo.MongoClient('mongodb://localhost:27017')
mydb = myclient['lab_db']
```

Creating a Collection

Now, create a collection called `'lab_collection'` inside `'lab_db'` .

```
mycollection = mydb['lab_collection']
```

Adding Some Data

Now, we're going to add some example records into our database. In the cells below, create dictionary representations of the following customer records:

Name	Email	Mailing_Address	Balance	Note
John Smith	j.smith@thesmiths.com	123 mulberry lane	0.0	Calle techni suppo issue no resolv
Jane Smith	jane_smith@thesmiths.com	Null	25.00	Null
Adam Enbar	adam@theflatironschool.com	11 Broadway	14.99	Set up recurri billing c

Name	Email	Mailing_Address	Balance	Note
Avi Flombaum	avi@theflatironschool.com	11 Broadway	0.0	Null
Steven S.	steven.s@gmail.net	Null	-20.23	Refunde overpayr due to p matc guaran

Your first challenge is to take all of the data in the table above and create the corresponding documents and add then to our mongo database. Note that fields that contain 'Null' should not be included in the document (recall that since mongo is schema-less, each document can be different).

Create the documents from the table listed above, and then use `insert_many()` to insert them into the collection.

```
customer_1 = {'Name': 'John Smith', 'Email': 'j.smith@thesmiths.com', 'Mailing_Addre
customer_2 = {'Name': 'Jane Smith', 'Email': 'jane_smith@thesmiths.com', 'Balance':
customer_3 = {'Name': 'Adam Enbar', 'Email': 'adam@theflatironschool.com', 'Mailing_
customer_4 = {'Name': 'Avi Flombaum', 'Email': 'avi@theflatironschool.com', 'Mailing
customer_5 = {'Name': 'Steven S.', 'Email': 'steven.s@gmail.net', 'Balance': -20.23,

all_records = [customer_1, customer_2, customer_3, customer_4, customer_5]

insertion_results = mycollection.insert_many(all_records)
```

Now, access the appropriate attribute from the results object returned from the insertion to see the unique IDs for each record inserted, so that we can confirm each were inserted correctly.

```
insertion_results.inserted_ids
```

```
[ObjectId('5cab91e1f1210d35c8dbfd5f'),
 ObjectId('5cab91e1f1210d35c8dbfd60'),
 ObjectId('5cab91e1f1210d35c8dbfd61'),
 ObjectId('5cab91e1f1210d35c8dbfd62'),
 ObjectId('5cab91e1f1210d35c8dbfd63')]
```

Querying and Filtering

In the cell below, return the name and email address for every customer record. Then, print every item from the query to show that it worked correctly.

```
query_1 = mycollection.find({}, {'Name': 1, 'Email': 1})
for item in query_1:
    print(item)
```

```
{'_id': ObjectId('5cab91e1f1210d35c8dbfd5f'), 'Name': 'John Smith', 'Email':
'j.smith@thesmiths.com'}
{'_id': ObjectId('5cab91e1f1210d35c8dbfd60'), 'Name': 'Jane Smith', 'Email':
'jane_smith@thesmiths.com'}
{'_id': ObjectId('5cab91e1f1210d35c8dbfd61'), 'Name': 'Adam Enbar', 'Email':
'adam@theflatironschool.com'}
{'_id': ObjectId('5cab91e1f1210d35c8dbfd62'), 'Name': 'Avi Flombaum', 'Email':
'avi@theflatironschool.com'}
{'_id': ObjectId('5cab91e1f1210d35c8dbfd63'), 'Name': 'Steven S.', 'Email':
'steven.s@gmail.net'}
```

Great! Now, let's write a query that gets an individual record based on a stored key-value pair a document contains.

In the cell below, write a query to get the record for `'John Smith'` by using his name. Then, print the results of the query to demonstrate that it worked correctly.

```
query_2 = mycollection.find({'Name': 'John Smith'})
for item in query_2:
    print(item)
```

```
{'_id': ObjectId('5cab91e1f1210d35c8dbfd5f'), 'Name': 'John Smith', 'Email':
'j.smith@thesmiths.com', 'Mailing_Address': '123 mulberry lane', 'Balance': 0.0,
'Notes': 'Called technical support, issue not yet resolved.'}
```

Great! Now, write a query to get the names, email addresses, and balances for customers that have a balance greater than 0. Use a modifier to do this.

HINT: In the query below, you'll be passing in two separate dictionaries. The first one defines the logic of the query, while the second tells which fields we want returned.

```python
query_3 = mycollection.find({'Balance': {'$gt': 0}}, {'Name': 1, 'Email': 1, 'Balanc
for item in query_3:
    print(item)
```

```
{'_id': ObjectId('5cab91e1f1210d35c8dbfd60'), 'Name': 'Jane Smith', 'Email':
'jane_smith@thesmiths.com', 'Balance': 25.0}
{'_id': ObjectId('5cab91e1f1210d35c8dbfd61'), 'Name': 'Adam Enbar', 'Email':
'adam@theflatironschool.com', 'Balance': 14.99}
```

Updating a Record

Now, let's update some records. In the cell below. set the mailing address for `'John Smith'` to `'367 55th St., apt 2A'` .

```python
record_to_update_1 = {'Name': 'John Smith'}
update_1 = {'$set': {'Mailing_Address': '367 55th St., apt 2A'}}
mycollection.update_one(record_to_update_1, update_1)
```

```
<pymongo.results.UpdateResult at 0x15c44ddd248>
```

Now, write a query to check that the update worked for this document in the cell below:

```python
query_4 = mycollection.find({'Name': 'John Smith'})
for item in query_4:
    print(item)
```

```
{'_id': ObjectId('5cab91e1f1210d35c8dbfd5f'), 'Name': 'John Smith', 'Email':
'j.smith@thesmiths.com', 'Mailing_Address': '367 55th St., apt 2A', 'Balance':
0.0, 'Notes': 'Called technical support, issue not yet resolved.'}
```

Now, let's assume that we want to add birthdays for each customer record. Consider the following table:

Name	Birthday
John Smith	02/20/1986

Name	Birthday
Jane Smith	07/07/1983
Adam Enbar	12/02/1982
Avi Flombaum	04/17/1983
Steven S.	08/30/1991

We want to add birthdays for each person, but we want to do so in a way where we don't have to do the same repetitive task over and over again. This seems like a good opportunity to write a function to handle some of the logic for us!

In the cell below:

- Store the names in the `names_list` variable as a list.
- Store the birthdays in the `birthdays_list` variable as a list.
- Write a function that takes in the two different lists, and updates each record by adding in the appropriate key-value pair containing that user's birthday.

Hint: There are several ways that you could write this, depending on whether you want to use the `update_one()` method or the `update_many()` method. See if you can figure out both approaches!

```
names_list = ['John Smith', 'Jane Smith', 'Adam Enbar', 'Avi Flombaum', 'Steven S.']
birthdays_list = ['02/20/1986', '07/07/1983', '12/02/0982', '04/17/1983', '08/30/199

def update_birthdays(names, birthdays):
    for ind, name in enumerate(names):
        birthday = birthdays_list[ind]
        query = {'Name': name}
        update = {'$set': {'Birthday': birthday}}
        mycollection.update_one(query, update)
        # Alternative method is to create a dictionary for each person, store them a
        # list to update_many() at the end of the function.

update_birthdays(names_list, birthdays_list)
```

Now, write a query to check your work and see that the birthdays were added correctly.

```
[i for i in mycollection.find()]
```

```
[{'_id': ObjectId('5cab91e1f1210d35c8dbfd5f'),
  'Name': 'John Smith',
  'Email': 'j.smith@thesmiths.com',
  'Mailing_Address': '367 55th St., apt 2A',
  'Balance': 0.0,
  'Notes': 'Called technical support, issue not yet resolved.',
  'Birthday': '02/20/1986'},
 {'_id': ObjectId('5cab91e1f1210d35c8dbfd60'),
  'Name': 'Jane Smith',
  'Email': 'jane_smith@thesmiths.com',
  'Balance': 25.0,
  'Birthday': '07/07/1983'},
 {'_id': ObjectId('5cab91e1f1210d35c8dbfd61'),
  'Name': 'Adam Enbar',
  'Email': 'adam@theflatironschool.com',
  'Mailing_Address': '11 Broadway',
  'Balance': 14.99,
  'Notes': 'Set up on recurring billing cycle.',
  'Birthday': '12/02/0982'},
 {'_id': ObjectId('5cab91e1f1210d35c8dbfd62'),
  'Name': 'Avi Flombaum',
  'Email': 'avi@theflatironschool.com',
  'Mailing_Address': '11 Broadway',
  'Balance': 0.0,
  'Birthday': '04/17/1983'},
 {'_id': ObjectId('5cab91e1f1210d35c8dbfd63'),
  'Name': 'Steven S.',
  'Email': 'steven.s@gmail.net',
  'Balance': -20.23,
  'Notes': 'Refunded for overpayment due to price match guarantee.',
  'Birthday': '08/30/1991'}]
```

Great! It looks like the birthdays have been successfully added to every record correctly!

Summary

In this lesson, we got some hands-on practice working with MongoDB!

Releases

No releases published

Packages

No packages published

Contributors 4

sumedh10 Sumedh Panchadhar

mike-kane Mike Kane

LoreDirick Lore Dirick

Cheffrey2000 Jeffrey Hinkle

Languages

● **Jupyter Notebook** 100.0%