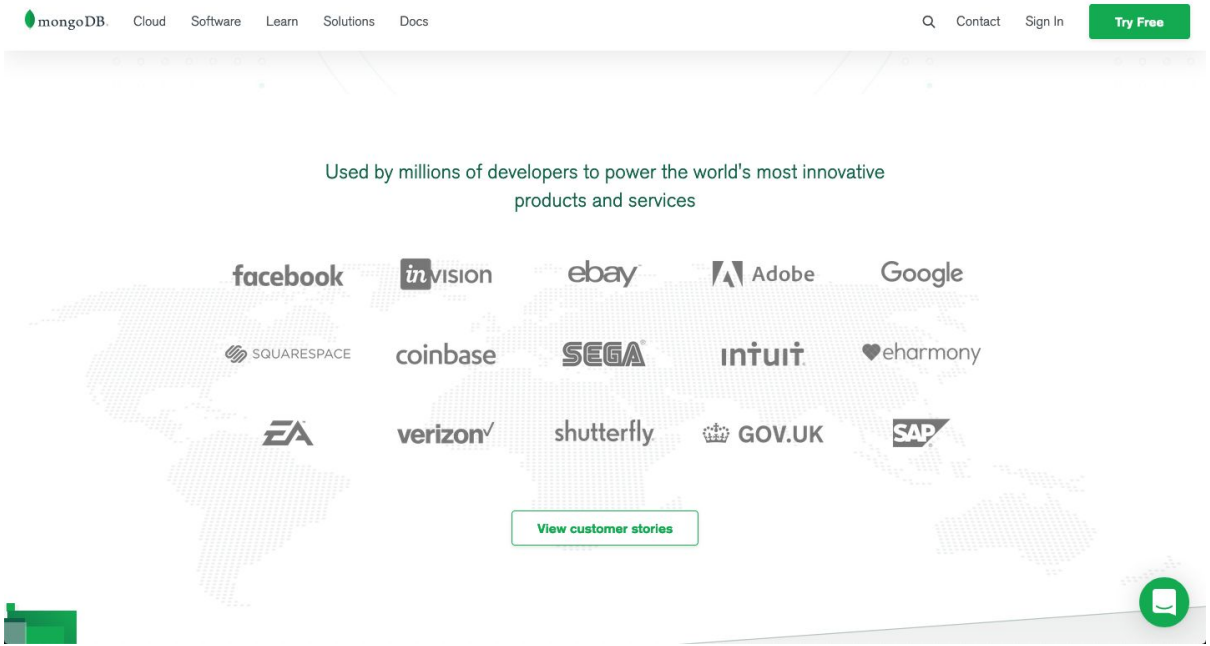
Introduction to MongoDB with Python

BuildingBloCS 2020

https://tinyurl.com/FA20MongoDB

BuildingBloCS 2020



Introduction to MongoDB with Python

Database

A database is an organized collection of data.

- Stored and accessed electronically from a computer system

How are the information of each item in a convenience store stored?

Fruits

Drinks

Database

Fruits

Drinks

Collections

Fruits

- Type (E.g. Banana, Apple)
- Brand (E.g. Dole)
- Price (E.g. $2.20. $3.00)
- Expiry Date (E.g. 23/09/2020)
- Discount Codes (E.g. 5% off)

Fields

Fruits

- Type: Banana
- Brand: Dole
- Price: $2.50
- Expiry Date: 23/09/2020
- Discount Codes: 5% off

Document in the ‘Fruits’ Collection

Summing it up briefly

- Database: Stores the entire information about the products in the convenience store
- Collection: A ‘sub-category’ storing the information about a particular product type, that lies within the database
- Document: Information about a particular product of a collection

The Convenience Store **database** stores information about many product types, such as *Necessities, Drinks, Fruits…* which are **collections** in the database. Within each collection, there are multiple products, where their information can be stored in a **document**.

Information that we need

- What are suitable data types for each field?
  - Must the data types be the same for each field?
- Must all fields be filled, or can certain fields be left empty?
  - Must each document have the same fields?

What are suitable data types for each field?

Fruits

- Type: Banana STRING
- Brand: Dole STRING
- Price: $2.50 FLOAT
- Expiry Date: 23/09/2020 STRING
- Discount Codes: “5% off” STRING

Must all fields be filled, or can certain fields be left empty?

Fru

Household Necessities

- Type: Toilet Paper STRING
- Brand: Kleenex STRING
- Price: $3.50 FLOAT
- ~~Expiry Date: 23/09/2020~~
- Discount Codes: [“10% off”, “SUPERTOILETPAPER”] ARRAY

MongoDB

- ★ Flexible and dynamic schema
  - The fields can take on different data types - no fixed structure
  - Different documents can take on different fields
- ★ Technical details:
  - Hierarchical data storage
  - Horizontally scalable

PyMongo

PyMongo 3.9.0 Documentation — PyMongo 3.9.0 documentation

```
$ python -m pip install pymongo
```

To get a specific version of pymongo:

```
$ python -m pip install pymongo==3.5.1
```

To upgrade using pip:

```
$ python -m pip install --upgrade pymongo
```

Setting up the MongoDB Server

```python
# Set up MongoDB server
import pymongo

client = pymongo.MongoClient("127.0.0.1", 27017)
print("Connected!")
client.close()
```

MongoDB Server

```
{ v: 2, key: { version: 1 }, name: "incompatible_with_version_32", ns: "admin.system.version" }
2018-11-12T11:46:50.018+0800 I INDEX    [initandlisten]          building index using bulk method; build
may temporarily use up to 500 megabytes of RAM
2018-11-12T11:46:50.025+0800 I INDEX    [initandlisten] build index done.  scanned 0 total records. 0 sec
s
2018-11-12T11:46:50.026+0800 I COMMAND  [initandlisten] setting featureCompatibilityVersion to 3.4
2018-11-12T11:46:50.300+0800 I NETWORK  [thread1] waiting for connections on port 27017
```

Creating a database

```
database = client.get_database("ConvenienceStore") # get database
You can also use: client["ConvenienceStore"]

# Check the databases we have
databases = client.list_database_names()
print("Databases:", databases)
```

```
>>> Databases: ['admin', 'config', 'entertainment', 'local']
```

Creating a collection

```
coll_fruits = database.get_collection("Fruits") # get collection for fruits
coll_household_necessities = database.get_collection("HouseholdNecessities") # get collection for household necessities
```

```
You can also use: coll_fruits = database["Fruits"]
```

Create Read Update Delete

insert

Insert One

```
# MongoDB assigns a unique ID to each document

coll_fruits.insert_one({"Type": "Banana",
                        "Brand": "Dole",
                        "Price": 2.50,
                        "ExpiryDate": "23/09/2020",
                        "DiscountCodes": "5% off"})

coll_household_necessities.insert_one({"Type": "ToiletPaper",
                                       "Brand": "Kleenex",
                                       "Price": 3.50,
                                       "DiscountCodes": ["10% off", "SUPERTOILETPAPER"]})
```

Key-Value pair dictionary object

Insert Many

```
fruits = [{"Type": "Apple", "Brand": "Fuji", "Price": 1.10, "ExpiryDate": "21/10/2020"},
          {"Type": "Strawberry", "Brand": "Foxi", "Price": 3.10, "Quantity": 12, "ExpiryDate":
"28/08/2020"}]
coll_fruits.insert_many(fruits)

# Check the Fruits collection
all_fruits = coll_fruits.find({})
print("Fruits:")
for fruit in all_fruits:
     print(fruit)
```

Dynamic Schema!

Insert many documents at once using an array or JSON object

Create Read Update Delete

find

Checking the items

```
# Check the Fruits collection
all_fruits = coll_fruits.find({})
print("Fruits:", all_fruits)

# Check the HouseholdNecessities collection
all_household_necessities = coll_household_necessities.find({})
print("Household Necessities:", all_household_necessities)

client.close()
```

```
>>> Fruits: <pymongo.cursor.Cursor object at 0x10d396700>
>>> Household Necessities: <pymongo.cursor.Cursor object at 0x10d396850>
```

Pointer to a memory address

Modifying how we check the items

```
# Check the Fruits collection
all_fruits = coll_fruits.find({})
print("Fruits:")
for fruit in all_fruits:
    print(fruit)

# Check the HouseholdNecessities collection
all_household_necessities = coll_household_necessities.find({})
print("Household Necessities:")
for household_necessities in all_household_necessities:
     print(household_necessities)
```

Output

```
>>> Fruits:
>>> {'_id': ObjectId('5e72d8c593467869fcb9197c'), 'Type': 'Banana', 'Brand': 'Dole', 'Price': 2.5,
'ExpiryDate': '23/09/2020', 'DiscountCodes': '5% off'}
>>> {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji', 'Price': 1.1,
'ExpiryDate': '21/10/2020'}
>>> {'_id': ObjectId('5e72dc4b6f1d6d17548e9105'), 'Type': 'Strawberry', 'Brand': 'Foxi', 'Price': 3.1,
'Quantity': 12, 'ExpiryDate': '28/08/2020'}

>>> Household Necessities:
>>> {'_id': ObjectId('5e72d8c593467869fcb9197d'), 'Type': 'ToiletPaper', 'Brand': 'Kleenex', 'Price': 3.5,
'DiscountCodes': ['10% off', 'SUPERTOILETPAPER']}
```

MongoDB assigns a unique ID to each document
→ No duplicate entries

Output

```
>>> Fruits:
>>> {'_id': ObjectId('5e72d8c593467869fcb9197c'), 'Type': 'Banana', 'Brand': 'Dole', 'Price': 2.5,
'ExpiryDate': '23/09/2020', 'DiscountCodes': '5% off'}
>>> {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji', 'Price': 1.1,
'ExpiryDate': '21/10/2020'}
>>> {'_id': ObjectId('5e72dc4b6f1d6d17548e9105'), 'Type': 'Strawberry', 'Brand': 'Foxi', 'Price': 3.1,
'Quantity': 12, 'ExpiryDate': '28/08/2020'}

>>> Household Necessities:
>>> {'_id': ObjectId('5e72d8c593467869fcb9197d'), 'Type': 'ToiletPaper', 'Brand': 'Kleenex', 'Price': 3.5,
'DiscountCodes': ['10% off', 'SUPERTOILETPAPER']}
```

Dictionary format: Key-Value pairs

What is `find({})` (equivalent to `find()`)?

```
.find({})
```

Method

Query

- `{}: Empty query: Returns all the documents in the collection`
- `find().limit(n): Returns the first n matching items`

Finding items that matches a filter

```
# Check the Fruits collection
query = {"Type": "Apple"}
all_fruits = coll_fruits.find(query)
print("Fruits:")
for fruit in all_fruits:
    print(fruit)
```

```
>>> Fruits:
>>> {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji', 'Price': 1.1,
'ExpiryDate': '21/10/2020'}
```

Finding items that matches a filter

```
# Check the Fruits collection
query = {"Price": {"$lt": 3.00}}
all_fruits = coll_fruits.find(query)
print("Fruits:")
for fruit in all_fruits:
    print(fruit)
```

Finding items where its ‘Price’ is less than 3.00

```
>>> Fruits:
>>> {'_id': ObjectId('5e72d8c593467869fcb9197c'), 'Type': 'Banana', 'Brand': 'Dole', 'Price': 2.5,
'ExpiryDate': '23/09/2020', 'DiscountCodes': '5% off'}
>>>  {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji', 'Price': 1.1,
'ExpiryDate': '21/10/2020'}
```

Other conditionals

`$eq`	Equals to
`$gt`	Greater than
`$gte`	Greater than or equal to
`$lt`	Less than
`$lte`	Less than or equal to
`$ne`	Not equal to
`$in`	In a specified `list`
`$nin`	Not in a specified `list`
`$or`	Logical OR
`$and`	Logical AND
`$not`	Logical NOT
`$exists`	Matches documents which has the named field

Create Read Update Delete

update

Updating items

Filter

Update value

```
coll_fruits.update_one({"Brand": "Fuji"}, {"$set": {"Brand": "Fuji Pte. Ltd."}})

# Check the Fruits collection
query = {"Type": "Apple"}
all_fruits = coll_fruits.find(query)
print("Fruits:")
for fruit in all_fruits:
     print(fruit)

>>> Fruits:
>>> {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji Pte. Ltd.', 'Price': 1.1,
'ExpiryDate': '21/10/2020'}
```

```
(filter, update, upsert=False, bypass_document_validation=False, collation=None,
    array_filters=None, session=None)
Update a single document matching the filter.
```

Updating items

Filter

Update value

```
coll_fruits.update_many({"Price": {"$lt": 3.00}}, {"$set": {"Price": 4.10}})

# Check the Fruits collection
query = {}
all_fruits = coll_fruits.find(query)
print("Fruits:")
for fruit in all_fruits:
     print(fruit)

>>> Fruits:
>>>  {'_id': ObjectId('5e72d8c593467869fcb9197c'), 'Type': 'Banana', 'Brand': 'Dole', 'Price': 4.1,
'ExpiryDate': '23/09/2020', 'DiscountCodes': '5% off'}
>>>  {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji Pte. Ltd.', 'Price': 4.1,
'ExpiryDate': '21/10/2020'}
>>>  {'_id': ObjectId('5e72dc4b6f1d6d17548e9105'), 'Type': 'Strawberry', 'Brand': 'Foxi', 'Price': 3.1,
'Quantity': 12, 'ExpiryDate': '28/08/2020'}
```

Create Read Update Delete

delete

Updating items

Filter

```
coll_fruits.delete_one({"Price": {"$lte": 3.10}})

# Check the Fruits collection
query = {}
all_fruits = coll_fruits.find(query)
print("Fruits:")
for fruit in all_fruits:
     print(fruit)

>>> Fruits:
>>>  {'_id': ObjectId('5e72d8c593467869fcb9197c'), 'Type': 'Banana', 'Brand': 'Dole', 'Price': 4.1,
'ExpiryDate': '23/09/2020', 'DiscountCodes': '5% off'}
>>>  {'_id': ObjectId('5e72dc4b6f1d6d17548e9104'), 'Type': 'Apple', 'Brand': 'Fuji Pte. Ltd.', 'Price': 4.1,
'ExpiryDate': '21/10/2020'}
```

Similarly, you can also delete many items that matches the filter using `delete_many`

Project

https://github.com/EmilyOng/PymongoWorkshop

ong.huiqi.emily@dhs.sg