MongoDB Setup

Quick Setup for MongoDB Atlas - Moritz Makowski

1) Cloud Setup

1. Log in at https://cloud.mongodb.com/ (https://cloud.mongodb.com/)

2. Create Cluster

3. Create Cluster-User-Identity under [Security -> Database Access -> Add New Database User]

4. Whitelist specific IP's under [Security -> Network Access -> Add IP Address]

5. Get your connection string under [Clusters -> Connect -> Connect Your Application -> Python]

2) Application Setup

In [5]:

```
# Set correct SSL certificate

import os
import certifi

os.environ["SSL_CERT_FILE"] = certifi.where()
```

In [6]:

```
# connect to your cluster

from secrets import MONGODB_CONNECTION_STRING
from pymongo import MongoClient

client = MongoClient(MONGODB_CONNECTION_STRING)

# This is the string you got in Step 1.5 . Remember to
# insert the username and passwort you created in Step 1.3
```

In [7]:

```
# connect to your database/collection

database = client.get_database("notebook_database")  # my example database name
collection = database["notebook_collection"]         # my example collectio name
inside that database
```

In [8]:

```
# count documents in collection
# you can pass a filter object to this function

collection.count_documents({})
```

Out[8]:

```
0
```

3) Inserting

In [18]:

```
# insert one record

new_person = {
    "name": "Peter",
    "happy": True
}


collection.insert_one(new_person)

collection.count_documents({})
```

Out[18]:

```
1
```

In [19]:

```
# insert many records at once

new_people = [
    {
        "name": "Mary",
        "happy": True
    },{
        "name": "Paul",
        "happy": False
    },{
        "name": "Ann",
        "happy": False
    },{
        "name": "Martin",
        "happy": True
    }
]

collection.insert_many(new_people)

collection.count_documents({})
```

Out[19]:

```
5
```

4) Deleting

In [20]:

```
# delete one record (the first one matching the filter)

collection.delete_many({"name": "Mary"})

collection.count_documents({})
```

Out[20]:

```
4
```

In [21]:

```
# delete many records (all the ones matching the filter)

collection.delete_many({"happy": True})

collection.count_documents({})
```

Out[21]:

```
2
```

In [29]:

```
# Empty the whole collection

collection.delete_many({})

collection.count_documents({})
```

Out[29]:

```
0
```

In [30]:

```
# Restoring the documents for further examples

new_people = [
    {
        "name": "Mary",
        "happy": True
    },{
        "name": "Paul",
        "happy": False
    },{
        "name": "Ann",
        "happy": False
    },{
        "name": "Martin",
        "happy": True
    }
]

collection.insert_many(new_people)

collection.count_documents({})
```

Out[30]:

```
4
```

5) Querying

In [31]:

```
# Find one record

collection.find_one({"happy": True})
```

Out[31]:

```
{'_id': ObjectId('5e8084a9d4282bdc82418eec'), 'name': 'Mary', 'happ
y': True}
```

In [33]:

```
# Find many records

list(collection.find({"happy": True}))
```

Out[33]:

```
[{'_id': ObjectId('5e8084a9d4282bdc82418eec'), 'name': 'Mary', 'happ
y': True},
 {'_id': ObjectId('5e8084a9d4282bdc82418eef'),
  'name': 'Martin',
  'happy': True}]
```

In [34]:

```
# include specific fields - this is called "projection"

list(collection.find({"happy": True}, {"name": 1}))
```

Out[34]:

```
[{'_id': ObjectId('5e8084a9d4282bdc82418eec'), 'name': 'Mary'},
 {'_id': ObjectId('5e8084a9d4282bdc82418eef'), 'name': 'Martin'}]
```

In [35]:

```
# excluse specific fields - this is called "projection"

list(collection.find({"happy": True}, {"_id": 0}))
```

Out[35]:

```
[{'name': 'Mary', 'happy': True}, {'name': 'Martin', 'happy': True}]
```

In [41]:

```
# You cannot mix inclusions and exclusion

from pymongo.errors import OperationFailure

try:
    list(collection.find({"happy": True}, {"happy": 0, "name": 1}))
except OperationFailure as e:
    print(f"OperationFailure: {e}")
```

```
OperationFailure: Projection cannot have a mix of inclusion and excl
usion.
```

6) Updating

In [44]:

```
# Update one record

collection.update_one({"name": "Ann"}, {"$set": {"happy": True}})

list(collection.find({"happy": True}, {"_id": 0}))
```

Out[44]:

```
[{'name': 'Mary', 'happy': True},
 {'name': 'Ann', 'happy': True},
 {'name': 'Martin', 'happy': True}]
```

In [47]:

```
# Update many records

collection.update_many({"happy": True}, {"$set": {"happy": False}})

print(list(collection.find({"happy": True}, {"_id": 0})))
print(list(collection.find({"happy": False}, {"_id": 0})))
```

```
[]
[{'name': 'Mary', 'happy': False}, {'name': 'Paul', 'happy': False},
{'name': 'Ann', 'happy': False}, {'name': 'Martin', 'happy': False}]
```

In [48]:

```
# Restoring the documents for further examples

collection.delete_many({})

new_people = [
    {
        "name": "Mary",
        "happy": True
    },{
        "name": "Paul",
        "happy": False
    },{
        "name": "Ann",
        "happy": False
    },{
        "name": "Martin",
        "happy": True
    }
]

collection.insert_many(new_people)

collection.count_documents({})
```

Out[48]:

```
4
```

7) Bulk Operations

In [49]:

```
# Combine multiple operations into one request

from pymongo.errors import BulkWriteError
from pymongo import InsertOne, UpdateOne, DeleteOne

operations = [
    InsertOne({"name": "Marcus", "happy": True}),
    UpdateOne({"name": "Ann"}, {"$set": {"happy": True}}),
    DeleteOne({"name": "Mary"})
]

collection.bulk_write(operations, ordered=True)

list(collection.find({}, {"_id": 0}))
```

Out[49]:

```
[{'name': 'Paul', 'happy': False},
 {'name': 'Ann', 'happy': True},
 {'name': 'Martin', 'happy': True},
 {'name': 'Marcus', 'happy': True}]
```

8) Indexes

Benefits:

- More performant queries when using the index as a filter-parameter
- Possibility to define fields that have to be unique

In [51]:

```
# Create an index

import pymongo

collection.create_index([('name', pymongo.ASCENDING)], unique=True)
```

Out[51]:

```
'name_1'
```

In [54]:

```
# Now when trying to insert a record with an already existing value in a unique
 field, it fails

from pymongo.errors import DuplicateKeyError

try:
    collection.insert_one({"name": "Paul", "happy": True})
except DuplicateKeyError as e:
    print(f"DuplicateKeyError: {e}")
```

```
DuplicateKeyError: E11000 duplicate key error collection: notebook_d
atabase.notebook_collection index: name_1 dup key: { name: "Paul" }
```

In [55]:

```
# See: nothing has been changed

list(collection.find({}, {"_id": 0}))
```

Out[55]:

```
[{'name': 'Paul', 'happy': False},
 {'name': 'Ann', 'happy': True},
 {'name': 'Martin', 'happy': True},
 {'name': 'Marcus', 'happy': True}]
```

Indexes are fully transparent - so whether or not you use indexes does not change anything about the query syntax!

9) Collection Settings

In [56]:

```
# renaming a collection

collection.rename("notebook_collection_example")
```

Out[56]:

```
{'ok': 1.0,
 '$clusterTime': {'clusterTime': Timestamp(1585482625, 2),
  'signature': {'hash': b'\x90-\xc4\xe3|\x0b}&\xbfd\tA\xd4"S\x11\x86
K\x10\xef',
   'keyId': 6807860005560123394}},
 'operationTime': Timestamp(1585482625, 2)}
```

In [57]:

```
# do operations still work on the collection object?

collection.count_documents({})
```

Out[57]:

```
0
```

In [58]:

```
# No! We have to redefine the collection object

collection = database["notebook_collection_example"]

collection.count_documents({})
```

Out[58]:

```
4
```

In [60]:

```
# renaming it back

collection.rename("notebook_collection")

collection = database["notebook_collection"]

collection.count_documents({})
```

Out[60]:

```
4
```