# Découverte de MongoDB

Xavier Gendre

## MongoDB

MongoDB est un système de gestion de base de données **orienté documents** distribué sous licence SSPL (non libre).

La communication se fait selon le principe **client-serveur**. MongoDB est un système populaire et réputé facile d'utilisation : http://db-engines.com/en/ranking

La langue maternelle de MongoDB est le **JavaScript**.

Les objets manipulés sont au format **BSON** (*Binary JSON*).

## NoSQL

MongoDB fait partie de la mouvance NoSQL. Cette présentation suivra le CRUD (*Creation, Read, Update, Delete*).

Un serveur MongoDB permet de gérer plusieurs bases de données. Chaque base de données contient des **collections** où sont stockés des documents enrichis d'une clé `_id`.

- **Imbrication de documents** au lieu de l'approche relationnelle normalisée.
- **Absence de schéma** : il suffit d'utiliser un document ou une collection pour les créer. Cette simplicité implique que **toute faute de frappe peut avoir des conséquences importantes**.

## MongoDB avec Onyxia

Le datalab Onyxia propose un service `Mongodb` dans la catégorie *Databases*.

Une fois le service lancé, Onyxia donne toutes les informations nécessaires pour se connecter au serveur MongoDB. Il y a même un code Python pour bien démarrer !

*Remarque :* le service `Mongodb` de Onyxia peut être supprimé sans perte de données. Si un nouveau service `Mongodb` est créé, il utilisera le même volume de stockage (au sens de *Kubernetes*).

## Module PyMongo

```
import pymongo

client = pymongo.MongoClient(
    "mongodb://user-...-ensae:...@mongodb-0.mongodb-headless:27017,"
    "mongodb-1.mongodb-headless:27017/defaultdb"
)

db = client.defaultdb
```

- `MongoClient` pour se connecter au serveur MongoDB.
- `client` est l'objet de connexion au serveur MongoDB.
- Le serveur utilise le port 27017 par défaut de MongoDB.
- `db` est l'objet relatif à la base de données utilisée (ici, il s'agit de `defaultdb`).

## Collection

Une **collection** est un groupe de documents et joue un rôle similaire à une table dans les bases de données relationnelles.

Pour accéder à une collection (ou pour en créer une), il suffit de la nommer à partir de la base de données.

```
test_collection = db["test"]
```

Si la collection n'existe pas, elle sera créée (une fois qu'un premier document y sera inséré).

Une faute de frappe dans le nom de la collection ne sera donc pas signalée.

## Document

Les documents manipulés dans une collection sont au **format JSON**. Côté Python, ils seront donc naturellement représentés par des **dictionnaires**.

En interne, les documents sont au **format binaire BSON**. En particulier, cela permet de gérer des types plus variés que le simple JSON comme les objets `datetime` de Python, …

```
import datetime

document = {
    "user": "Bob",
    "score": 170881,
    "time": datetime.datetime.now(tz=datetime.timezone.utc),
}
```

## Insérer des documents (*Create*)

La méthode `insert_one` permet d'insérer un document dans une collection.

```
result = test_collection.insert_one(document)
result.inserted_id
```

```
ObjectId('6820eeb154338a5401106966')
```

L'objet retourné contient l'identifiant `_id` ajouté au document par MongoDB.

La méthode `count_documents` avec un dictionnaire vide en paramètre compte le nombre de documents dans la collection.

```
test_collection.count_documents({})
```

```
1
```

---

Il est possible d'insérer plusieurs documents sous la forme d'une liste de dictionnaires avec la méthode `insert_many`.

```
documents = [
    {
        "user": "Joy",
        "score": 240482,
        "time": datetime.datetime(2024, 2, 6, 15, 5, 45),
    },
    {
        "user": "Ken",
        "score": 424242,
        "time": datetime.datetime(2024, 3, 12, 8, 42, 8),
    },
]

result = test_collection.insert_many(documents)
result.inserted_ids # Noter le pluriel
```

```
[ObjectId('6820eeb154338a5401106967'), ObjectId('6820eeb154338a5401106968')]
```

## RingsDB

La suite des exemples sera basée sur les données relatives à un jeu de cartes inspiré de l'univers du *Seigneur des anneaux* disponible via l'API du site RingsDB.

```python
import requests

r = requests.get("https://ringsdb.com/api/public/cards")

if r.status_code != 200:
    print(f"Erreur {r.status_code}")

cards = r.json() # Tableau de documents au format JSON

rings = db["rings"] # Nouvelle collection
result = rings.insert_many(cards) # Insertion des cartes

print(f"{len(result.inserted_ids)} cartes ajoutées")
```

```
1470 cartes ajoutées
```

---

L'absence de schéma permet d'insérer des documents qui n'ont pas la même structure.

```python
print(f"Avant : {rings.count_documents({})}")

rings.insert_one({"name": "Luke Skywalker", "outlier": True})

print(f"Après : {rings.count_documents({})}")
```

```
Avant : 1470
Après : 1471
```

Encore une fois, aucune erreur n'est signalée et toute faute de frappe peut être critique dans ces manipulations.

### Recherche (*Read*)

La méthode `find` permet de faire une recherche dans la collection. Cette fonction retourne un **itérateur** qui permet de parcourir les résultats dans une boucle.

Sans paramètre, toute la collection est retournée.

```
all_cards = rings.find()
print(all_cards[0])
```

```
{'_id': ObjectId('6820eeb554338a5401106969'), 'pack_code': 'Core', 'pack_name': 'Core Set',
```

---

Il est possible de définir des critères de recherche en passant un dictionnaire en argument `filter`.

```
contract_cards = list(rings.find(filter={"type_name": "Contract"}))
print(f"{len(contract_cards)} cartes de type 'Contract'")
```

```
14 cartes de type 'Contract'
```

```
contract_cards[0]
```

```
{'_id': ObjectId('6820eeb554338a5401106d5c'),
 'pack_code': 'ASitE',
 'pack_name': 'A Shadow in the East',
 'type_code': 'contract',
 'type_name': 'Contract',
 'sphere_code': 'neutral',
 'sphere_name': 'Neutral',
 'position': 74,
 'code': '21074',
 'name': 'Fellowship',
 'text': '<b>Side A</b>\r\nYou cannot play non-unique allies or put non-unique allies into pl
 'flavor': '"I will choose you companions to go with you, as far as they will or fortune allo
 'is_unique': False,
 'quantity': 1,
 'deck_limit': 1,
 'illustrator': 'Leanna Crossan',
 'octgnid': 'cf207eaa-2897-430e-9988-c557173055b5',
 'has_errata': False,
 'url': 'https://ringsdb.com/card/21074',
 'imagesrc': '/bundles/cards/21074.png'}
```

---

L'argument `projection` permet de limiter les clés des documents retournés par la méthode `find`.

```
contract_cards = list(
    rings.find(
        filter={"type_name": "Contract"},
        projection={"name": True, "illustrator": True},
    )
)
contract_cards[0]
```

```
{'_id': ObjectId('6820eeb554338a5401106d5c'),
 'name': 'Fellowship',
 'illustrator': 'Leanna Crossan'}
```

Par défaut, la clé `_id` est retournée sauf si `projection` contient `"_id": False`.

---

Le résultat des recherches se présente comme une liste de dictionnaires et il peut donc être facilement transformé en `DataFrame`

```
import pandas as pd

contract_cards = rings.find(
    filter={"type_name": "Contract"},
    projection={"_id": False, "name": True, "illustrator": True},
)

print(pd.DataFrame(contract_cards))
```

```
                          name        illustrator
0                   Fellowship     Leanna Crossan
1            The Burglar's Turn     Greg Bobrowski
2     Forth, The Three Hunters!      Justin Gerard
3             The Grey Wanderer      Justin Gerard
4           Council of the Wise      Borja Pindado
5         Messenger of the King      Justin Gerard
6            Bond of Friendship      Borja Pindado
7             The Last Alliance     Unknown Artist
```

```
8               The Riddle-game            Sansiia
9             A Perilous Voyage                NaN
10               Into the West    Donato Giancola
11    At the End of All Things     Michael Whelan
12  Beyond the Original Bargain  Magali Villeneuve
13             Thorin's Company         Chris Rahn
```

### Opérateurs de recherche

MongoDB met à disposition de nombreux opérateurs pour effectuer des recherches plus avancées. Ces opérateurs sont précédés par le caractère `$`.

- `$exist` pour tester l'existence d'une clé,

```
print(pd.DataFrame(
    rings.find(
        filter={"outlier": {"$exists": True}},
        projection={"_id": False, "name": True}
    )
))
```

```
            name
0  Luke Skywalker
```

---

- `$and`, `$or`, `$nor` et `$not` pour la logique,

```
print(pd.DataFrame(
    rings.find(
        filter={
            "$and": [
                {"type_name": "Contract"},
                {"illustrator": "Leanna Crossan"}
            ]
        },
        projection={"_id": False, "name": True, "illustrator": True}
    )
))
```

```
         name     illustrator
0  Fellowship  Leanna Crossan
```

---

- `$lt`, `$lte`, `$gt` et `$gte` pour comparer des nombres,

```
print(pd.DataFrame(
    rings.find(
        filter={"attack": {"$gte": 4}},
        projection={"_id": False, "name": True, "attack": True}
    )
))
```

```
                   name  attack
0               Gandalf       4
1               Saruman       5
2             Treebeard       4
3         Thunderstruck       6
4           Sea Serpent       6
5   Recurring Nightmare       5
6        Dagnir's Spawn       6
7       Gundabad Hunter       5
8                 Beorn       5
9               Gandalf       4
10             Skinbark       4
11      Wraith on Wings       6
12     Army of the Dead       6
13            Quickbeam       4
14           Giant Bear       4
15              Gandalf       4
16              Gandalf       4
17              Saruman       4
18              Gwaihir       4
19              Gandalf       4
20              Gandalf       4
21              Gandalf       4
22              Gandalf       4
23                Beorn       5
24      (MotK) Skinbark       4
```

---

- `$ne` pour tester la non-égalité,

```
print(pd.DataFrame(
    rings.find(
        filter={"pack_name": {"$ne": "Core Set"}},
        projection={"_id": False, "pack_name": True, "name": True}
    )
))
```

```
                                pack_name                 name
0        Revised Core Set (Campaign Only)      Mendor's Support
1        Revised Core Set (Campaign Only)                 Valor
2        Revised Core Set (Campaign Only)     Appointed by Fate
3        Revised Core Set (Campaign Only)                Mendor
4        Revised Core Set (Campaign Only)     Ungoliant's Swarm
...                                   ...                   ...
1393  ALeP - Messenger of the King Allies          (MotK) Balin
1394  ALeP - Messenger of the King Allies          (MotK) Birna
1395  ALeP - Messenger of the King Allies  (MotK) Bilbo Baggins
1396  ALeP - Messenger of the King Allies  (MotK) Frodo Baggins
1397                                  NaN        Luke Skywalker

[1398 rows x 2 columns]
```

---

- `$regex` pour utiliser des expressions régulières,

```
print(pd.DataFrame(
    rings.find(
        filter={"name": {"$regex": "^Z"}},
        projection={"_id": False, "name": True}
    )
))
```

```
          name
0  Zigil Miner
1  Zigil Miner
```

Et bien d'autres : `$in` pour l'inclusion, `$size` pour la taille d'un tableau, `$mod` pour le modulo, …

https://www.mongodb.com/docs/manual/reference/operator/query/

### Compter

La méthode `count_documents` permet de compter les documents qui vérifient une recherche.

```
rings.count_documents({"pack_name": "Core Set"})
```

```
73
```

```
rings.count_documents(
    {
        "$and": [
            {"pack_name": "Core Set"},
            {"type_name": {"$in": ["Hero", "Contract"]}},
        ]
    }
)
```

```
12
```

### Valeurs distinctes

La méthode `distinct` retourne la liste des valeurs distinctes prises par un clé parmi les documents vérifiant une recherche.

```
rings.distinct(key="type_name")
```

```
['Ally',
 'Attachment',
 'Campaign',
 'Contract',
 'Event',
 'Hero',
 'Player Objective',
 'Player Side Quest']
```

```
rings.distinct(
    key="illustrator",
    filter={"pack_name": "The Hunt for Gollum"},
)
```

```
['Ben Zweifel',
 'David A. Nash',
 'Felicia Cano',
 'Jake Murray',
 'John Gravato',
 'Joko Mulyono',
 'Katherine Dinger',
 'Magali Villeneuve',
 'Stu Barnes',
 'Tony Foti']
```

### Tri

L'argument `sort` de la méthode `find` permet de trier les résultats selon les valeurs d'une clé.

```
print(pd.DataFrame(
    rings.find(
        filter={"threat": {"$exists": True}},
        projection={"_id": False, "name": True, "threat": True},
        sort=[
            ("threat", pymongo.DESCENDING),
            ("name", pymongo.ASCENDING),
        ],
    )
))
```

```
              name  threat
0          Gandalf      14
1     (MotK) Beorn      13
2           Elrond      13
3           Elrond      13
4          Gwaihir      13
..             ...     ...
267  Frodo Baggins       0
268  Frodo Baggins       0
269  Frodo Baggins       0
270  Frodo Baggins       0
271    Sea Serpent       0

[272 rows x 2 columns]
```

### Modification (*Update*)

Pour modifier un ou plusieurs document(s) de la collection, il faut :

- identifier le ou les document(s) à modifier par une recherche comme avec la méthode `find`,
- apporter les modifications au(x) document(s).

La méthode `update_one` ne modifie qu'un seul document (le premier trouvé) tandis que `update_many` modifie tous les documents correspondants à la recherche.

### Opérateurs de modification

Comme pour la recherche, MongoDB met à disposition des opérateurs pour modifier les documents de la collection.

- `$set` pour changer la valeur d'une clé (ou la créer),

```
rings.update_one( # Un seul document est modifié
    filter={"name": "Gandalf"},
    update={"$set": {"cheveux": "gris"}}
)
print(pd.DataFrame(
    rings.find(
        filter={"name": "Gandalf"},
        projection={"_id": False, "name": True, "cheveux": True},
    )
))
```

```
      name cheveux
0  Gandalf    gris
1  Gandalf     NaN
2  Gandalf     NaN
3  Gandalf     NaN
4  Gandalf     NaN
5  Gandalf     NaN
6  Gandalf     NaN
7  Gandalf     NaN
8  Gandalf     NaN
```

---

```
rings.update_many( # Tous les documents sont modifiés
    filter={"name": "Gandalf"},
    update={"$set": {"cheveux": "gris"}}
)
print(pd.DataFrame(
    rings.find(
        filter={"name": "Gandalf"},
        projection={"_id": False, "name": True, "cheveux": True},
    )
))
```

```
      name cheveux
0  Gandalf    gris
1  Gandalf    gris
2  Gandalf    gris
3  Gandalf    gris
4  Gandalf    gris
5  Gandalf    gris
6  Gandalf    gris
7  Gandalf    gris
8  Gandalf    gris
```

---

- `$unset` pour supprimer une clé,

```
rings.update_many( # Tous les documents sont modifiés
    filter={"name": "Gandalf"},
    update={"$unset": {"cheveux": "gris"}} # Suppression
)
print(pd.DataFrame(
    rings.find(
        filter={"name": "Gandalf"},
        projection={"_id": False, "name": True, "cheveux": True},
    )
))
```

```
      name
0  Gandalf
1  Gandalf
2  Gandalf
```

```
3  Gandalf
4  Gandalf
5  Gandalf
6  Gandalf
7  Gandalf
8  Gandalf
```

---

Beaucoup d'autres opérateurs de modification sont disponibles :

- `$push`, `$pull`, `$addToSet`, … pour modifier un tableau,
- `$inc`, `$mul`, … pour modifier une valeur numérique,
- `$rename` pour renommer une clé,
- …

https://docs.mongodb.com/manual/reference/operator/update/

### Suppression (*Delete*)

La méthode `delete_one` supprime un document (le premier trouvé) vérifiant une recherche.

```
print(f"Avant : {rings.count_documents({})}")

rings.delete_one({"outlier": {"$exists": True}})

print(f"Après : {rings.count_documents({})}")
```

```
Avant : 1471
Après : 1470
```

---

La méthode `delete_many` supprime tous les documents vérifiant une recherche.

```
print(f"Avant : {rings.count_documents({})}")

rings.delete_many({"name": "Gandalf"})

print(f"Après : {rings.count_documents({})}")
```

```
Avant : 1470
Après : 1461
```

La méthode `drop` supprime une collection et ses documents.

```
test_collection.drop()
```

### Exporter une collection

Le module `pymongo` ne propose pas de fonction simple pour exporter une collection vers un fichier. Il existe néanmoins des outils pour le faire dans le module `bson` fourni avec PyMongo.

Une alternative simple consiste à utiliser Pandas

```
(
    pd.DataFrame(
        rings.find() # Tous les documents
    )
    .to_json(
        "rings.json",
        # Conversion forcée en chaînes de caractères
        # Perte des avantages du BSON ici
        default_handler=str,
        # Format NDJSON
        orient="records",
        lines=True,
    )
)
```

### Importer une collection

Le module `json` de la bibliothèque standard de Python suffit pour récupérer et insérer dans une collection l'ensemble des documents contenus dans un fichier au format NDJSON.

```
import json

print(f"1. Taille de rings : {rings.count_documents({})}")
# Suppression de la collection
rings.drop()
print(f"2. Taille de rings : {rings.count_documents({})}")
# Lecture du fichier NDJSON
```

```
with open("rings.json", "r") as f:
    documents = [json.loads(document) for document in f.readlines()]
rings.insert_many(documents)
print(f"3. Taille de rings : {rings.count_documents({})}")
```

```
1. Taille de rings : 1461
2. Taille de rings : 0
3. Taille de rings : 1461
```

---

To be continued...