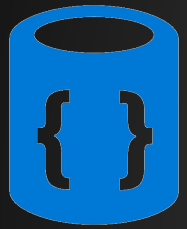
DB Non Structurées

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

Objectif du cours



Comprendre les différences avec le SQL. Comprendre les datas non & semi structurées

Établir un serveur MongoDB, s'y connecter et commencer une API. Faire des opérations de CRUD & des premières routes d'API

Projet en Python, nécessitant de gérer la DB & l'API. Faire un certains nombre de fonctionnalités

1. Les datas non structurées et le NoSQL

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

Le NoSQL

- ***Raisons de l'apparition du NoSQL***
  - Le **NoSQL** (**N**ot **O**nly **SQL**) répond à un besoin de performance, de disponibilité et de flexibilité dû aux big datas & aux applications web modernes
  - Il apporte de nouveaux schémas, de nouvelles possibilités d'optimisation et d'adaptation face à des situations nouvelles dû à notre utilisation d'internet
  - Il va en résulter un changement de paradigme dans la création d'une DB et de ses logiques d'utilisation. La rigidité est déplacée au niveau de l'application et de l'écriture, et non pas au moment de l'insertion dans la DB

Le NoSQL

- ***Changements dans la logique / paradigme***
  - Dans cette nouvelle logique, on va manipuler de la donnée dans le cadre de DB semi-structurées et/ou non-structurées
  - Ainsi, on passe d'une logique **ACID**, mettant l'emphase sur la cohérence, à une logique **BASE**, avançant une logique de disponibilité de la donnée.
  - Grâce à ces changements, la scalabilité horizontale est bien plus accessible et peut donc être envisagée plus fréquemment.

Le NoSQL

- ***Les scalabilités***
  - La **scalabilité verticale** consiste à augmenter les propriétés physiques d'un serveur. Ainsi, on augmentera la taille du disque, la RAM, le CPU etc.. Cette approche ne répond pas aux problématiques de disponibilité et ne peut croître indéfiniment.
  - La **scalabilité horizontale** consiste à multiplier les noeuds (instances serveurs). Les noeuds font partie d'un cluster d'une DB. Chaque noeuds aura un ou des shards primaires et/ou réplicas. Les données de la DB resteront accessibles grâce à une clé de sharding. Un shard est une partie de la DB duquel le noeud attaché est responsable pour les requêtes d'écriture & de lecture.

2. Théorème CAP & Approches ACID/BASE

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

CAP - ACID & BASE

- ***Principes du Théorème***
  - Énonce les contraintes qu'il y a dans les systèmes distribués (environnement où plusieurs appareils effectuent des actions sur un réseau)
  - **C** (Consistency) : Avoir la même donnée au même moment pour les utilisateurs
  - **A** (Availability) : Les requêtes doivent pouvoir s'effectuer à chaque moment
  - **P** (Partition Tolerance) : S'il y a des pertes dans une ou des parties du réseau, le système continue de fonctionner
  - Les contraintes révèlent une nécessité de faire des arbitrages afin de favoriser une contrainte au détriment d'une autre

CAP - ACID & BASE

- ***ACID***
  - Approche utilisé dans les SGBDR afin d'assurer de la cohérence et de la stabilité dans le système
  - **Atomicity** : Une requête est entièrement exécutable ou alors elle rollback
  - **Cohérence** : Après la requête, la DB passe d'un état valide à un autre état valide. Cela assure la cohérence des données pour l'ensemble des users
  - **Isolation** : Simule une séquentialité des requêtes afin que différents states de la DB ne s'écrasent pas. Assure la cohérence
  - **Durability** : Assure la persistance des données même après un problème technique, du moment où le commit de la requête est passé. Utilisation de disques et de journaux de transactions (reconstitution de requête si besoin)

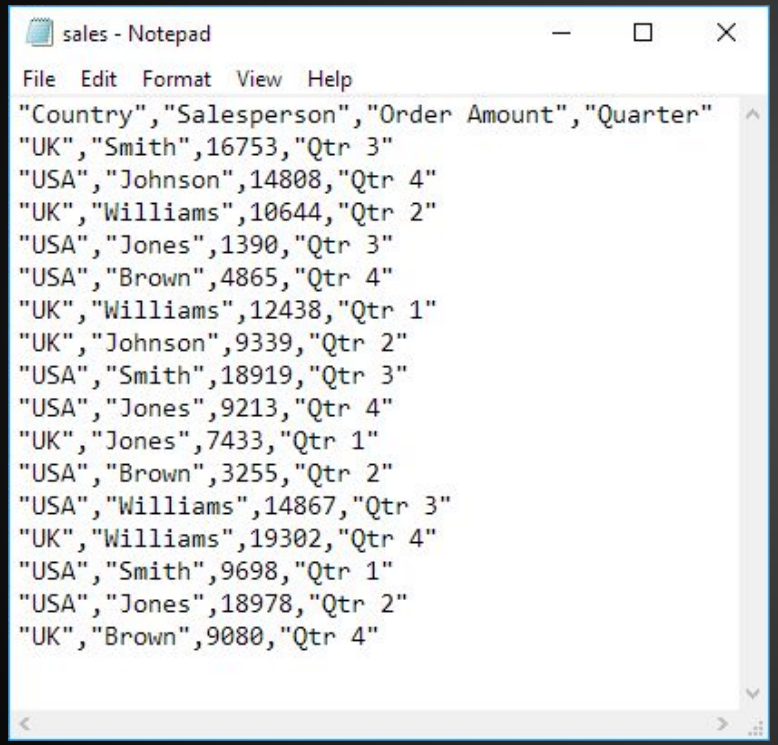
CAP - ACID & BASE

- ***BASE***
  - Approche associée aux solutions NoSQL. Plus flexible et permissive que l'approche ACID. Elle est néanmoins moins fiable mais permet une meilleure performance et gestion des contraintes.
  - **BA** (Basically Available) : Privilégie la disponibilité plutôt que la cohérence. Cette philosophie octroie des temps de réponse plus rapides.
  - **S** (Soft State) : L'état du système peut être en train d'évoluer durant la requête sans même une intervention explicite
  - **E** (Eventually Consistent) : Assure qu'après une synchronisation / état de transition, l'état reviendra vers la cohérence des données.

3. La donnée non-structurée et semi-structurée

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

Donnée non & semi structurée



Exercice récupération donnée sur de la donnée non structurée

- ***Consignes***
  - Télécharger la donnée disponible dans le .txt
  - Récupérer chaque ligne en mappant les valeurs de chaque lignes avec les colonnes
  - La sortie doit être une list de dictionnaires. Chaque dictionnaire possède en key les colonnes, en value les valeurs de chaque row
  - S’assurer des types (string, int, float)
  - Gérer les cas d’absence de donnée

4. MongoDB

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

MongoDB

MongoDB MacOS

https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-os-x/

MongoDB Server

https://www.mongodb.com/try/download/community

MongoDB Shell

https://www.mongodb.com/try/download/shell

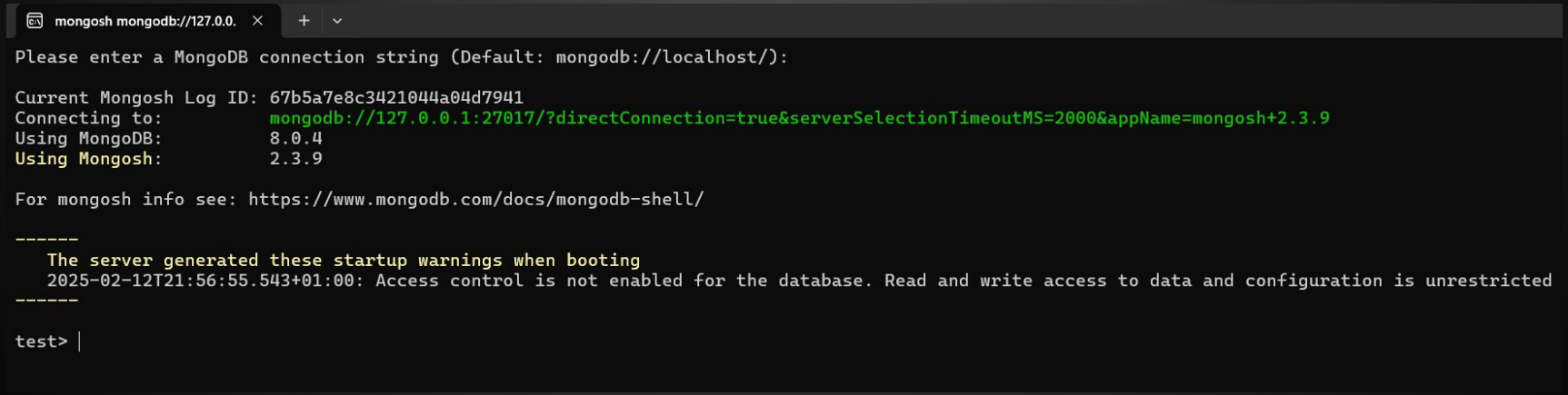
MongoDB Compass GUI

https://www.mongodb.com/try/download/compass

MongoDB

Pour savoir si votre serveur tourne bien :

- Lancez MongoSH
- Appuyer directement sur entrer pour vous connecter avec l'URL par défaut
- Si vous vous êtes connecté, vous devriez voir le screen suivant :



MongoDB

- ***Se connecter à sa DB***
  - Installer, via pip, pymongo -> *pip install pymongo*
  - Avec Python, se connecter à son serveur, puis à sa DB, pour accéder à ses collections

mdb.py > ...

```python
from pymongo import MongoClient
from pymongo.errors import ConnectionFailure
connected = False

try:
    client = MongoClient("mongodb://localhost:27017/", serverSelectionTimeoutMS=5000)
    db = client["my_db"]
    connected = True
    print("Connected to MongoDB")
except ConnectionFailure:
    print("MongoDB connection failed")

if connected:
    print(f"list_collection_names : {db.list_collection_names()}")
```

5. API & Flask Python

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

API & Flask

- ***Faire tourner une API***
  - Installer, via pip, flask -> *pip install flask*
  - Avec Python, se connecter à son serveur, puis à sa DB, pour accéder à ses collections

```python
from flask import Flask, jsonify
app = Flask(__name__)

def my_script():
    return "Script est exécuté"

# API Routes
@app.route("/run/a", methods=["GET"])
def run_a():
    result = my_script()
    return jsonify({"status": "success", "result": result})

# Start server
if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5000, debug=True)
```

6. Projet

Nicolas Triau
Mail : nicolas.triau@yahoo.fr
Discord : gillesleveau

Le projet

- ***Consignes***
  - Une seule personne pour le projet
  - Mettre en place une DB MongoDB (sur un thème e-commerce, products, customers, cart, orders...)
  - Mettre en place un projet Python avec une connexion à la DB
  - Mettre en place des fonctions pour le CRUD dans les collections choisies
  - Mettre en place une API ayant des routes afin de :
    - Faire du CRUD dans chaque collection
    - Dans la table produits, pouvoir récupérer 1 ou plusieurs items en fonction des certains filtres
    - Au moins une route API servant à exécuter un script faisant l'action de votre choix : statistiques sur la data voulu, création d'un XLSX ou autre..