

GESTION DE DONNÉES, PARTIE II

Philippe Helluy

CONTENTS

1. Rappels des objectifs
2. noSQL
3. Exemple pratique
4. Intégration avec Python

RAPPELS DES OBJECTIFS

- Les fondamentaux: bases de données relationnelles, SGDB, SQL, algèbre relationnelle, Python Pandas.
- **Avancées récentes: noSQL**
- outils cloud (API, JSON), Spark.
- Gestion des données semi-structurées: LLM, RAG.

NoSQL

- NoSQL (Not Only SQL) est une approche de gestion de données qui permet de stocker et de manipuler des données non structurées ou semi-structurées.
- Gestion de grandes quantités de données et pour être scalables horizontalement (i.e. parallélisable).
- Les principaux types de bases de données NoSQL (2025) sont :
 - **Clé-valeur** : Stockage de données sous forme de paires clé-valeur.
 - **Document** : *Stockage de données sous forme de documents (ex: JSON, JSONB) (ce cours).*
 - **Graphe** : Stockage de données sous forme de graphes, avec des nœuds et des arêtes.

- Les bases de données de type document stockent des données sous forme de documents, souvent au format JSON.
- JSON est un format léger de données, basés sur des dictionnaires et des listes. JSONB est une version binaire de JSON, optimisée pour le stockage.
- Une extension SQL dans PostgreSQL permet de stocker et d'interroger des documents JSON de manière efficace.
- Avantages de JSON:
 - Flexibilité du schéma : pas besoin de définir un schéma rigide.
 - Indexation et requêtes efficaces sur des champs spécifiques dans les documents.
 - Support (limité) pour des opérations de type NoSQL tout en utilisant un SGBD relationnel.

- JSON (JavaScript Object Notation) est un format de données léger et facile à lire.
- Il est utilisé pour échanger des données entre une application et un serveur.
- JSON est indépendant du langage, mais utilise des conventions familières aux programmeurs de la famille C.

- Les données sont organisées en paires clé/valeur.
- Les objets JSON sont entourés par des accolades {}.
- Les tableaux JSON sont entourés par des crochets [].
- Les clés doivent être des chaînes de caractères (entre guillemets).
- Les valeurs peuvent être des chaînes, des nombres, des objets, des tableaux, des booléens ou null.

```
{  
  "nom": "Jean Dupont",  
  "age": 30,  
  "estEtudiant": false,  
  "cours": ["Mathématiques", "Sciences", "Histoire"],  
  "adresse": {  
    "rue": "123 Rue Principale",  
    "ville": "Paris"  
  }  
}
```

```
{  
  "nom": "Dupont",  
  "prenom": "Jean",  
  "animaux": [  
    {  
      "nom": "Rex",  
      "espece": "Chien",  
      "age": 5  
    },  
    {  
      "nom": "Médor",  
      "espece": "Chien",  
      "age": 3  
    }  
  ]  
}
```

- La structure JSON est très bien adaptée aux dictionnaires et listes en Python.
- Utilisation du module `json` pour lire et écrire des données JSON.
- `json.loads()` pour décoder une chaîne JSON en objet Python.
- `json.dumps()` pour encoder un objet Python en chaîne JSON.

```
import json

# Exemple de chaîne JSON
json_string = '{"nom": "Jean Dupont", "age": 30, "estEtudiant": false}'

# Décodage de la chaîne JSON en objet Python
data = json.loads(json_string)
print(data["nom"]) # Affiche: Jean Dupont

# Modification de l'objet Python
data["age"] = 31

# Encodage de l'objet Python en chaîne JSON
new_json_string = json.dumps(data)
print(new_json_string)
```

Il faut faire attention aux erreurs de décodage JSON, par exemple en cas de format incorrect, ou de clé attendue manquante. On peut gérer ces erreurs avec des blocs try-except.

```
import json
json_string = '{"nom": "Jean Dupont", "age": 30, "estEtudiant": false}'
try:
    data = json.loads(json_string)
    print(data["name"])
except json.JSONDecodeError as e:
    print(f"Erreur de décodage JSON: {e}")
except KeyError as e:
    print(f"Clé manquante: {e}")
except Exception as e:
    print(f"Une erreur est survenue: {e}")
```

- Création d'une table avec une colonne JSONB :

```
CREATE TABLE animaux (  
  id SERIAL PRIMARY KEY,  
  data JSONB  
);
```

- Insertion d'un document JSONB :

```
INSERT INTO animaux (data) VALUES ('{"nom": "Rex", "espece": "Chien", "age":  
5, "proprietaire": {"nom": "Dupont", "prenom": "Jean"}}');
```

- Requête pour extraire des données d'un document JSONB :

```
SELECT data->>'nom' AS nom, data->>'espece' AS espece FROM animaux;
```

Résultat :

nom	espece
Rex	Chien

Les possibilités sont toutefois assez limitées. Il vaut mieux utiliser un SGDB conçu expressément pour le NoSQL, comme MongoDB.

MongoDB est une base de données **NoSQL** orientée **document**, qui stocke les données sous forme de documents **BSON** (Binary JSON).

Avantages:

- Flexibilité du schéma : les documents peuvent avoir des structures différentes
- Scalabilité horizontale : facile à distribuer sur plusieurs serveurs
- Performances élevées pour les opérations de lecture/écriture

Définitions:

- **Document** : unité de données (similaire à un enregistrement en SQL)
- **Collection** : ensemble de documents (similaire à une table en SQL)
- **Base de données** : ensemble de collections

SQL (Relationnel)	NoSQL (Document)
• Tables, lignes, colonnes • Schéma fixe	• Collections, documents • Schéma flexible
• Jointures • Exemple : PostgreSQL, MySQL	• Documents imbriqués • Exemple : MongoDB, CouchDB

EXEMPLE PRATIQUE

Tables en SQL:

- client : id, nom, prenom
- animal : id, nom, espece, age
- maitre : animal_id, client_id (clés étrangères)

id	nom	prenom
1	Dupont	Jean
2	Martin	Claire

id	nom	espece	age
1	Rex	Chien	5
2	Miaou	Chat	3
3	Panpan	Lapin	2

animal_id	client_id
1	1
2	2
3	2

Collection `clients` Chaque client est un document JSON :

```
{
  "_id": ObjectId("507f1f77bcf86cd799439011"),
  "nom": "Dupont",
  "prenom": "Jean",
  "animaux": [
    {
      "nom": "Rex",
      "espece": "Chien",
      "age": 5
    }
  ]
}
```

Avantages:

- Pas besoin de table de jointure maître
- Tous les animaux d'un client sont dans le même document

Document pour Claire Martin:

```
{
  "_id": ObjectId("507f1f77bcf86cd799439012"),
  "nom": "Martin",
  "prenom": "Claire",
  "animaux": [
    { "nom": "Miaou", "espece": "Chat", "age": 3 },
    { "nom": "Panpan", "espece": "Lapin", "age": 2 }
  ]
}
```

Trouver tous les clients:

```
db.clients.find()
```

Trouver les clients avec un chien:

```
db.clients.find({"animaux.espece": "Chien"})
```

Ajouter un animal à un client:

```
db.clients.updateOne(  
  { "nom": "Martin" },  
  {  
    $push: {  
      "animaux": { "nom": "Medor", "espece": "Chien", "age": 4 }  
    }  
  }  
)
```

Compter le nombre d'animaux par espèce:

```
db.clients.aggregate([
  { $unwind: "$animaux" },
  { $group: {
    _id: "$animaux.espece",
    count: { $sum: 1 }
  }
}]
```

Résultat:

```
[ { "_id": "Chien", "count": 2 },
  { "_id": "Chat", "count": 1 },
  { "_id": "Lapin", "count": 1 } ]
```

Cas d'usage:

- Données hiérarchiques ou imbriquées
- Schéma évolutif ou inconnu à l'avance
- Besoin de scalabilité horizontale
- Applications web modernes (JSON natif)

Cas à éviter:

- Données fortement relationnelles
- Besoin de transactions complexes
- Requêtes analytiques très complexes

INTÉGRATION AVEC PYTHON

Installation:

```
pip install pymongo
```

Exemple de connexion et requête:

```
from pymongo import MongoClient

client = MongoClient('mongodb://localhost:27017/')
db = client['veterinaire']
collection = db['clients']

# Trouver tous les clients
for client in collection.find():
    print(client)
```

Synthèse:

- MongoDB est une alternative flexible aux bases SQL pour les données non structurées ou semi-structurées
- L'exemple vétérinaire montre comment migrer d'un schéma relationnel vers un schéma document
- PyMongo permet une intégration facile avec Python

Prochaine étape: TP : Migration de la base SQLite vers MongoDB et requêtes avancées.