Bases de Données NoSQL

INFOA3 – Année 2025/2026

1 Contexte et évolution des systèmes de gestion de données

Le modèle relationnel domine depuis des décennies grâce à sa structure rigoureuse, soutenue par SQL, les contraintes d'intégrité et le respect du modèle ACID. Il offre des capacités robustes pour interroger des données fortement structurées et établir des relations explicites via les clés primaires et étrangères.

L'explosion du volume de données, l'arrivée des usages massifs en temps réel et les besoins croissants en scalabilité horizontale ont toutefois révélé les limites de ce modèle. Les systèmes relationnels restent performants mais deviennent plus complexes à faire évoluer lorsque la distribution sur plusieurs machines devient obligatoire.

C'est dans ce contexte qu'apparaissent les bases de données NoSQL, pensées pour gérer des données semi-structurées, des charges intensives et des architectures distribuées.

2 Typologie des systèmes NoSQL

Les SGBD NoSQL regroupent plusieurs approches selon les besoins et la nature des données :

- **Magasins clé–valeur** : structures simples et ultra-rapides comme Redis, souvent utilisées pour la mise en cache ou les traitements en mémoire.
- **Bases orientées documents** : stockage de documents JSON/BSON flexibles (MongoDB).
- **Bases en colonnes** : taillées pour les environnements massifs distribués.
- **Bases orientées graphes** : destinées aux réseaux complexes d'entités et de relations.

La suite du rapport développe un focus sur MongoDB, représentant majeur des bases orientées documents.

3 MongoDB : principes et fonctionnement

3.1 Modèle documentaire

MongoDB adopte un modèle basé sur des documents BSON regroupés au sein de collections. La souplesse du schéma permet de faire évoluer les structures sans migration lourde, un avantage notable pour les applications dont les données évoluent fréquemment.

3.2 Composants principaux

Une instance MongoDB repose sur plusieurs éléments :

- `mongod` : le serveur qui gère le stockage et les opérations ;
- `mongos` : le routeur utilisé lorsqu'un cluster est shardé ;
- les serveurs de configuration : stockage des métadonnées du cluster.

La haute disponibilité s'appuie sur les *Replica Sets*, tandis que la montée en charge horizontale repose sur le *Sharding*.

4 Domaines d'application

MongoDB s'intègre particulièrement bien dans des scénarios où :

- les données sont semi-structurées ou hétérogènes (catalogues, profils utilisateurs, contenu dynamique) ;
- les systèmes doivent absorber un trafic important ou croissant (IoT, télécom) ;
- le traitement de données non structurées est central, notamment dans les plateformes d'IA modernes.

La flexibilité du modèle documentaire et sa capacité à gérer des volumes importants en font une solution souvent adoptée.

5 Mise en place rapide avec Docker

L'environnement de travail du TP exploite Docker pour simplifier l'installation du serveur MongoDB :

```
# Télécharger l'image MongoDB
docker pull mongo

# Lancer un conteneur
docker run --name tp-mongodb -p 27017:27017 -d mongo

# Ouvrir un shell MongoDB
docker exec -it tp-mongodb mongosh
```

Les installations natives restent disponibles sur : `https://www.mongodb.com/docs/manual/installation/`

6 Manipulation des données

6.1 Lecture : `find`

```
db.collection.find(<filtre>, <projection>)
```

Opérations courantes :

- `sort({ champ: 1|-1 })` ;
- `limit(n)` ;
- `skip(n)` ;
- `pretty()`.

6.2 Modification : `updateOne` et `updateMany`

```
db.collection.updateOne(filter, update, options)
db.collection.updateMany(filter, update, options)
```

Opérateurs essentiels :

- `$set` : création ou modification d'un champ ;
- `$inc` : incrément ;
- `$unset` : suppression d'un champ.

6.3 Suppression : `deleteOne` et `deleteMany`

```
db.collection.deleteOne(filter)
db.collection.deleteMany(filter)
```

6.4 Agrégation : pipelines `aggregate`

```
db.collection.aggregate([ stage1, stage2, ... ])
```

Principaux types de stages :

- `$match` : filtrage ;
- `$group` : regroupements et calculs ;
- `$project` : transformations ;
- `$sort`, `$limit`, `$unwind`.

7 Optimisation par indexation

7.1 Création d'un index

```
db.collection.createIndex(
    { champ1: 1, champ2: -1 },
    { unique: true }
)
```

7.2 Inspection des index

```
db.collection.getIndexes()
```

7.3 Analyse d'une requête

```
db.collection.find({ filtre }).explain("executionStats")
```

Indicateurs clés :

- `totalDocsExamined` ;
- `executionTimeMillis` ;
- `winningPlan.stage`.

7.4 Suppression

```
db.collection.dropIndex({ champ: 1 })
db.collection.dropIndex("nom_index")
```