<https://dai.ly/x27wtco>

<https://www.octo.com/publications/19-hadoop-feuille-de-route/>

**MongoDB est une base de données NoSQL orientée documents.**

Les données prennent la forme de documents enregistrés eux-mêmes dans des collections, une collection contenant un nombre quelconque de documents.

- Collections <-> tables (modèle relationnel)

- Documents <-> enregistrements (modèle relationnel)

MongoDB permet de manipuler des objets structurés au format BSON (JSON binaire).

MongoDB ne nécessite pas de schéma prédéfini des données. Contrairement aux BDDR, les champs d'un enregistrement sont libres et peuvent être différents d'un enregistrement à un autre au sein d'une même collection. Le seul champ commun et obligatoire est le champ de clé principale ("id").

* Format :

|  |  |
| --- | --- |
| **BSON = JSON binaire**  **JSON = JavaScript Object Notation**  **Tout est clé/valeur : “clé” : “valeur”**  **Champ “\_id“ obligatoire**  **On peut ajouter des champs à la volée**  **On peut imbriquer des documents, mettre des listes, …** | **C:\Users\u174319\Documents\database\Doc MongoDB - DEP - Confluence_fichiers\image2021-4-12_14-56-38.png** |

* NOSQL :

Le NoSQL (Not Only SQL) regroupe de nombreuses BDD, récentes pour la plupart, qui se caractérisent par une logique de représentation de données non relationnelle et qui n'offrent donc pas une interface de requêtes en SQL.

C’est une approche de la conception des bases et de leur administration particulièrement utile pour de très grands ensembles de données distribuées.

Enjeu : résoudre les problèmes de performances en matière d'évolutivité et de Big Data que les BDDR ne sont pas conçues pour affronter.

MongoDB ne permet ni les requêtes très complexes standardisées, ni les jointures, mais permet de programmer des requêtes spécifiques en JavaScript (langage natif).

Il est également possible de manipuler les données en C, C++, Java, .NET, Perl, PHP, Python, ...

* Haute disponibilité :

MongoDB utilise une architecture basée sur le principe « maître/esclave ».

Le serveur primaire reçoit toutes les requêtes (lecture/écriture), et s’occupe de gérer la cohérence des données. Les serveurs secondaires appliquent également les instructions reçues par le serveur primaire pour avoir les mêmes données.

Il faut au minimum 3 serveurs (1 primaire et 2 secondaires) => **ReplicaSet**

Il est possible de modifier la gestion du ReplicaSet pour router les requêtes de lectures vers les secondaires.

Pour gérer les pannes, on peut ajouter un arbitre qui vérifie l’état du réseau. Si le primaire tombe, un système de vote va permettre d’élire le secondaire qui remplacera le serveur tombé en panne. L’arbitre est toujours sur un serveur à part.

Il est également possible de répartir les données sur plusieurs serveurs => **Sharding**.

|  |  |
| --- | --- |
| MONGODB | POSTGRESQL |
| * Stocker les données et les lire * Cycle de vie très simple * Orienté développeur | * Transactionnel (commit/no commit) * Modèle relationnel (MCD) * Sauvegarde à chaud * Plus robuste |