Cours
Mise en œuvre des Framework D'Intelligence Artificielle et Big data

Cycle Ingénieur
INDIA, Semestre 5

Pr. Abderrahim El Qadi
Département Mathématique Appliquée et Génie Informatique
ENSAM, Université Mohammed V de Rabat

A.U. 2023/2024

Plan

Ressources & Références 1ere partie

- Les bases de données NoSQL et le Big Data, Comprendre et mettre en œuvre. Rudi Bruchez. 2e edition
- MongoDB 4 Quick Start Guide. Doug Bierer
- MongoDB : Base de donnée orientée documents. Emmanuel Caruyer CR CNRS
- Trends of Artificial Intelligence and Big Data for E-Health. Nima Rezaei , Tehran University of Medical Sciences, Tehran, Iran
- Programming Hive. Data Warehouse and Query Language for Hadoop. Edward Capriolo, Dean Wampler, Jason Rutherglen - O'Reilly Media (2012)
- Bases de données documentaires et distribuées, Février 2023. Philippe Rigaux
- https://www.tutorialspoint.com/mongodb
- Neo4j website : http://neo4j.org/
- https://www.tutorialspoint.com/neo4j/

1. Big Data

– Définition

- Le Big data[1], mégadonnées ou les données massives désigne les ressources d'informations:
  - dont les caractéristiques en termes de **volume**, de **vélocité** et de **variété** imposent l'utilisation de technologies et de méthodes analytiques particulières pour créer de la valeur,
  - qui dépassent en général les capacités d'une seule et unique machine et nécessitent des traitements parallélisés.
- Grâce à lui, les entreprises peuvent collecter, analyser et interagir avec le maximum d'informations afin de comprendre comment leurs produits et services sont utilisés par leurs clients.

[1] https://fr.wikipedia.org/wiki/Big_data

– Big Data : principes clés

- Les bases du Big Data reposent sur un certain nombre de principes clés :
  - **La gestion des données** consiste à collecter les informations pertinentes provenant d'un système ou d'un service existant, puis à les organiser pour obtenir une vision plus cohérente de tout le processus.
  - **L'analyse des données** permet aux entreprises de comprendre ce qu'elles peuvent apprendre à partir de ces informations, telles que les tendances générales et le comportement des consommateurs.
  - **La science des données** est l'utilisation avancée des outils et algorithmes analytiques pour découvrir les relations complexes entre différents jeux de données.
  - **Le stockage des données** permet aux entreprises de sauvegarder et archiver cette grande quantité d'informations pour pouvoir y accéder rapidement lorsque nécessaire.

– Big Data : D'où viennent les données ?

- Bases de données clients
- Navigation sur internet
- Réseaux sociaux
- Dossiers médicaux
- Applications mobiles
- Systèmes de transaction commerciale
- Données générées par des machines
- Capteurs de données en temps réel : IoT

– Big Data : Structuration des données

- Les données du Big Data peuvent prendre plusieurs formes

Niveau de structuration	Modèle de données	Exemples	Facilité de traitement
Structuré	Système de données relationnel objet/colonne	Base de données d'entreprise	Facile (indexé)
Semi-structuré	XML, JSON, CSV, logs	Web, API Google, API Twitter, ...	Facile (non indexé)
Non structuré	Texte, image, vidéo	Web, e-mails, documents...	Complexe

– Caractéristiques de données massives : 4 V

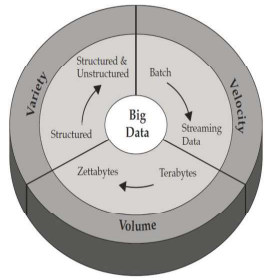
– **Volume** des données :
  - Le nombre d'observations n est grand
  - Le nombre de variables p est grand (devant n : n << p)

– **Variété** des données : données de formes diverses, données structurées, non structurées, semi structurées, données quantitatives, qualitatives, textuelles, images, vidéos, audio...

– **Vélocité** de l'acquisition : Gestion de la Vitesse d'acquisition des données en (quasi) continu, données en mouvement (temps réel)

– **Véracité** : données douteuses : incomplètes, incertaines, falsifiées, imprécises, ...

Quantité de données	En Octet
1Ko	1 024
1 Mo	$10\ 24^2$
1 Go	$10\ 24^3$
1 teraoctet (To)	$10\ 24^4$
1 pétaoctet (Po)	$10\ 24^5$
1 exaoctet (Eo)	$10\ 24^6$
1 zettaoctet (Zo)	$10\ 24^7$
1 yottaoctet (Yo)	$10\ 24^8$
1 brontooctet (Bo)	$10\ 24^9$



– Big Data : Cas d'utilisation (Exemples)

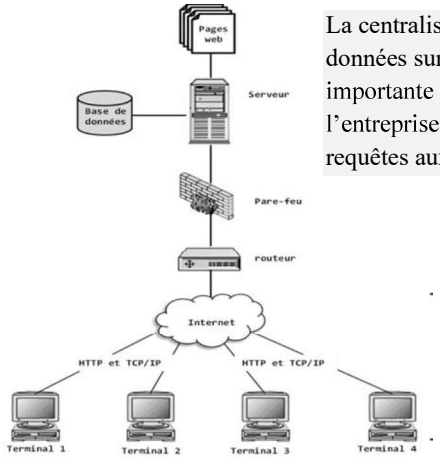
Développement de produits	Des entreprises comme Netflix utilisent le Big Data pour anticiper la demande des clients. Elles créent des modèles prédictifs pour de nouveaux produits et services, en classant les principaux attributs de produits ou services passés et présents et en modélisant la relation entre ces attributs et le succès commercial de leurs offres
Santé	Le Big data rend théoriquement possible une médecine préventive et personnalisée en lien avec des appareils connectés mesurant les données biométriques des patients et visant à leur proposer des conseils ou des traitements appropriés.
Augmenter l'efficacité opérationnelle	Grâce au Big Data, on peut analyser et évaluer la production, les commentaires et retours des clients, ainsi que d'autres facteurs, afin de réduire les pannes et d'anticiper les demandes à venir.
Transport	Le big data permet de modéliser les déplacements des usagers ou des salariés de l'entreprise afin d'optimiser les trajets et la fréquence des véhicules et donc d'accroître le taux de remplissage tout en faisant des économies de carburant.

– Défis du Big Data

- Capacité de traitements et de coût
- Qualité et gouvernance des données
- Défis de la conception d'une architecture :
  - Les équipes doivent s'adapter aux technologies disponibles
  - Déploiement et gestion de données massives exigent de nouvelles compétences.
- Service de gestion du Cloud :
  - Nécessite un contrôle
  - Migration de données est complexe
- Les pratiques et réglementations en matière de collecte de données :
  - Peu de restriction sur l'utilisation de données
  - Protection de la vie privée des consommateurs

2. Architecture de distribution de données

– **Approche traditionnelle** de gestion des données. Le stockage et le traitement des données sont centralisées dans un serveur.

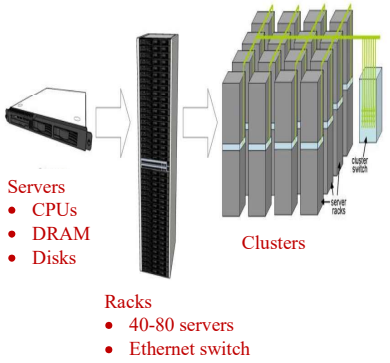


La centralisation du stockage et du traitement des données sur un serveur central crée une pression importante sur l'architecture informatique de l'entreprise, ce qui **augmente le temps de réponse** des requêtes aux clients (la latence).

Gérer de telles quantités de données pose deux problèmes principaux :

- Comment stocker ces informations : aucun disque dur seul n'est capable de stocker plusieurs milliers de téraoctets de données
- Comment organiser et rechercher dans ces informations : **une machine seule n'a pas la puissance** nécessaire pour effectuer des recherches suffisamment rapides sur les données.

– **Architecture distribuée – cluster computing** : Le stockage des données est distribué dans les nœuds d'un cluster et leur traitement y est parallélisé.

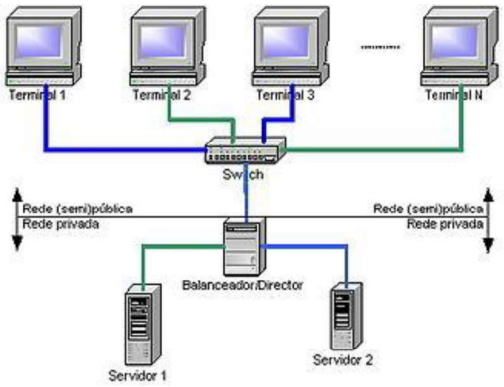


Il s'agit de grappe de serveurs[2], de cluster, de groupement de serveurs ou de ferme de calcul (**computer cluster**) pour désigner des techniques consistant à regrouper plusieurs ordinateurs indépendants appelés nœuds (**node**), afin de permettre une gestion globale et de dépasser les limitations d'un ordinateur pour :

- augmenter la disponibilité ;
- faciliter la montée en charge ;
- permettre une répartition de la charge ;
- faciliter la gestion des ressources (processeur, mémoire vive, disques durs, bande passante réseau).

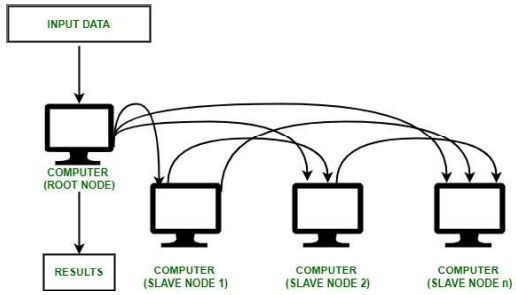
[2] https://fr.wikipedia.org/wiki/Grappe_de_serveurs

Exemple : Architecture cluster computing : Grappe de 2 serveurs en équilibrage de charge



Utiliser plusieurs machines :

- en partageant leur disque dur, les machines forment un groupe de stockage bien plus grand
- en partageant leur processeur (et/ou leur carte graphique), les machines forment un groupe de calcul bien plus performant

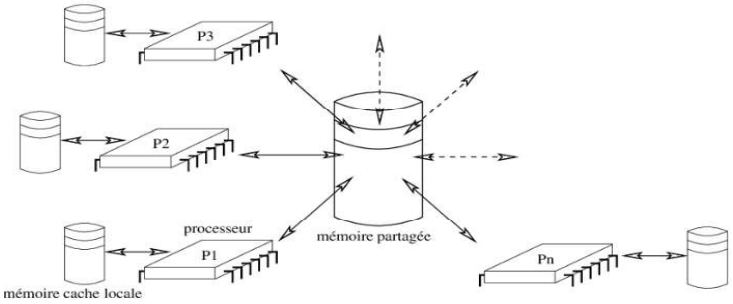


Cette organisation de machines dialoguant pour partager stockage et puissance de calcul est appelée un **cluster de machine** (et les machines de ce cluster sont alors appelées **des nœuds**).

- L'architecture distribuée présente des avantages :
  - Toutes **les données sont répliquées** à un certain facteur de réplication et partitionnées sur les différentes machines. Cela permet d'avoir une sécurité accrue au niveau des données.
  - Les calculs sont parallélisables, entraine **la rapidité de calcul**
  - Le passage à l'échelle est plus facile. En cas pas plus de stockage dans les machines, au lieu de changer d'ordinateur contre un ordinateur plus puissant (scalabilité verticale), **possibilité de rajouter** une machine à l'architecture, ce qui permet de résoudre le problème de stockage.
- En Data Science, pour réussir à effectuer des calculs avec une architecture distribuée et à **gérer le stockage** entre les différents nœuds de cette architecture distribuée, on utilise principalement Hadoop. Ce Framework est composé de trois sous-Framework : **HDFS**, **MapReduce** et **YARN**.

– Architecture à mémoire partagée

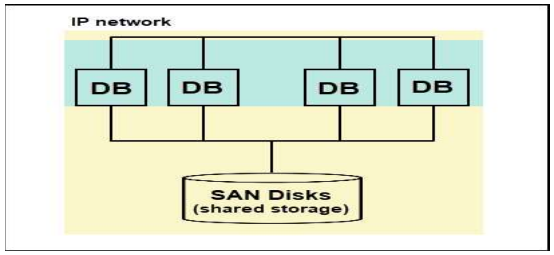
- La mémoire partagée désigne un bloc de mémoire vive qui est accédé par différentes unités de calcul au sein d'un ordinateur parallèle.



- Architecture à mémoire distribuée est une bibliothèque de communication portable permettant d'écrire de manière relativement performante et simple les échanges de données entre les différents processeurs.

– Architecture à disque partagé

- Le disque partagé est une architecture informatique distribuée dans laquelle tous **les nœuds du système sont liés au même périphérique de disque** mais ont leur **propre mémoire privée**.
- Les données partagées sont accessibles à partir de tous les nœuds du cluster et représentent généralement un disque partagé (comme une base de données) ou un système de fichiers partagé (comme un réseau de stockage ou un stockage en réseau).



– Cluster shared-nothing : Architecture sans partage (SN)

- Est une architecture informatique distribuée dans laquelle **chaque demande de mise à jour est satisfaite par un seul nœud** (processeur/mémoire/unité de stockage) dans un cluster d'ordinateurs.
- L'intention est d'éliminer les conflits entre les nœuds.
- Les nœuds ne partagent pas (n'accèdent pas indépendamment) à la même mémoire ou au même stockage.
- Une architecture alternative est tout partagée, dans laquelle les requêtes sont satisfaites par des combinaisons arbitraires de nœuds. Cela peut introduire des conflits, car plusieurs nœuds peuvent chercher à mettre à jour les mêmes données en même temps.

3. Bases de données SQL et NoSQL

3.1. BD SQL

- Une base de données SQL est une base de données relationnelle qui utilise le langage de requête structuré (SQL) pour stocker, récupérer et manipuler des données.
- Les bases de données SQL sont faciles à utiliser et à entretenir et offrent de nombreuses fonctionnalités qui les rendent bien adaptées à diverses applications.
- Les bases de données SQL offrent les avantages suivants :
  - **Facilité d'utilisation**.
  - **Sécurité robuste des données**.
  - **Évolutivité** : être étendues pour accueillir davantage de données et d'utilisateurs si nécessaire, plus de stockage de données.
  - **Haute performance, fiable**. Elles sont conçues pour gérer de grandes quantités de données et de transactions sans perdre ou corrompre la base de données.

- **Les règles de SQL** ACID **:**
  - La plupart des moteurs de SGBDs relationnels sont transactionnels.
  - Une transaction est une unité d'action qui doit respecter quatre critères, résumés par l'acronyme ACID (Atomicity Consistency Isolation Durability).

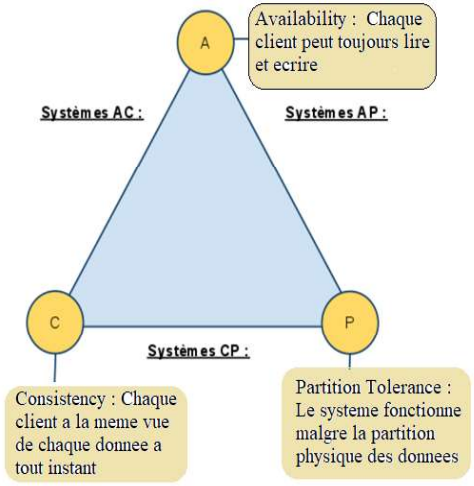
Critère	Définition
Atomique	**Tous les changements apportés aux données sont effectués totalement, ou pas du tout**. Par exemple, sur 5000 lignes devant être modifiées au sein d'une même transaction, si la modification d'une seule échoue, alors la transaction entière doit être annulée.
Cohérente (consistance)	**Les transactions doivent respecter les contraintes d'intégrité des données de la base de données**. Chaque transaction est soumise à un ensemble cohérent de règles (contraintes, déclencheurs, …)
Isolée	les écritures et lectures des transactions réussies ne seront pas affectées **par les écritures et lectures d'autres transactions, qu'elles soient ou non réussies**
Durable	Les transactions réussies survivront de façon permanente **et ne seront pas affectées par d'éventuelles pannes ou problèmes techniques.**

3.2. BD NoSQL

- Le terme « NoSQL » (Not Only SQL) a été inventé en 2009, désigne les différents types de bases de données non relationnelles.
- Le NoSQL offre :
  - Des schémas flexibles :
    - Les données de NoSQL sont stockées de manière semi-structurée.
    - Le NoSQL abandonne le principe des tables.
    - On ne cherche plus à savoir de quel type sont les informations stockées.
  - Une gestion de données volumineuses
  - Un traitement distribué. Le stockage distribué est soit réalisé par des fichiers plats sur un système de fichiers distribués, soit par un moteur de base de données distribué comme Cassandra ou Hbase, conçu pour fonctionner sur un large cluster de machines.
  - Des capacités de lecture/écriture élevées : de nombreux procédés sont mis en place pour assurer des débits élevés. Il y a par exemple le système de clé-valeur.

- La complexité des requêtes est également un facteur à prendre en compte pour les bases de données NoSQL.
  - Elles se révèlent très efficaces pour les requêtes portant sur une seule table.
  - Les bases de données NoSQL n'offrent généralement pas de jointures complexes, de sous-requêtes et d'imbrication de requêtes dans une clause WHERE.

- Les bases de données relationnelles implémentent les propriétés de Cohérence et de Disponibilité (système **AC**).
- Les bases de données NoSQL sont généralement des systèmes **CP** (Cohérent et Résistant au partitionnement) ou **AP** (Disponible et Résistant au partitionnement).



- C comme Coherence ou Consistency : tous les nœuds du système voient exactement les mêmes données au même moment.
- A comme Availability ou Disponibilité : la perte de nœuds n'empêche pas les survivants de continuer à fonctionner correctement.
- P comme Partition tolerance ou Résistance au partitionnement : Chaque nœud doit pouvoir fonctionner de manière autonome.

- Le théorème de **CAP** (Consistency Availibility Partition tolerance) stipule qu'il est impossible d'obtenir ces trois propriétés en même temps dans un système distribué et qu'il faut donc en choisir deux parmi les trois.

- Les bases de données SQL sont généralement **plus coûteuses à maintenir** que les bases de données NoSQL.
  - Les bases de données SQL nécessitent davantage d'administration, comme la création et la maintenance des index et des vues.
- Les bases de données SQL **doivent suivre les règles de ACID** (atomicité, cohérence, isolation et durabilité), ce qui peut les rendre plus lentes et plus compliquées.
  - Les données sont stockées de manière sûre et sécurisée, et que les transactions sont traitées de manière fiable et cohérente.

- NoSQL, en revanche, **ne suit pas** les règles ACID.
  - Plus flexible dans le stockage et le traitement des données.
  - Les données ne sont pas toujours stockées de manière sûre et sécurisée, et que le traitement des transactions peut être peu fiable et incohérent.

	SQL	NoSQL
Types de base de données	Base de données SQL	Plusieurs: key-value, document databases, et graph databases.
Exemples	MySQL, Postgres, Oracle Database	MongoDB, Cassandra, HBase, Neo4j
Model de stockage de données	• Une ligne dans une table avec découpage de l'information sur des colonnes. • Les types de données distincts sont séparés sur plusieurs tables. • Recours à la jointure pour interroger la base sur des données complexes.	Variable dépendant du type de BD : • Key-value : même chose que SQL mais seulement sur deux colonnes ("key" and "value"), avec plus de complexité au niveau de la colonne value • Document databases: En finir avec le modèle de table avec des lignes. Stocker toutes les données pertinentes ensemble uniquement sous forme d'un "document". (XML)
Schémas	Structure et types de données sont fixés à l'avance	Typiquement dynamique
Mise à l'échelle	Verticalement : un seul serveur doit être puissant pour faire face à une demande accrue.	Horizontalement : Un ajout de serveurs peut résoudre la monté en charge.
Model de Développement	Il y a des solutions Open Source.	Open-source
Support de transactions	Oui	Ca dépend des cas d'utilité de la base.
Manipulation des données	Langage d'interrogation spécifique	A travers Des API Orientées Objet
Cohérence	Peut-être configuré pour une forte cohérence	Selon le produit utilisé. Certains offrent une forte cohérence (par exemple, MongoDB), tandis que d'autres offrent une cohérence éventuelle (par exemple, Cassandra)

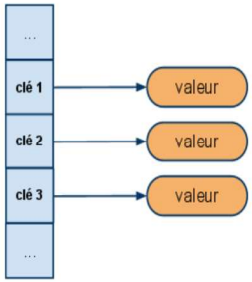
– Cas d'usage des bases de données NoSQL

• Google reader • Google maps • Blogger.com • Youtube • Gmail • Amazon • Sourceforge • Github • CollegeHumor • Mozilla	• Adobe • Facebook • LinkedIn • New York Times • Twitter • Yahoo! • Disney • Foursquare • IBM • Intel

3.3. Types de base NoSQL

3.3.1. Clé / valeur

- Base de données la plus basique.
- Chaque objet est identifié par une clé unique qui constitue la seule manière de le requêter.
- La structure de l'objet est libre et le plus souvent laissé à la charge du développeur de l'application (XML, JSON, ...), la base ne gérant généralement que des chaînes d'octets.

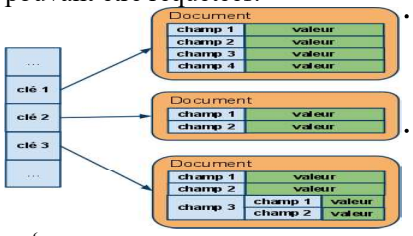


Ce modèle dispose généralement que des quatre opérations Create Read Update Delete (CRUD):

- Create : créer un nouvel objet avec sa clé → create (key, value)
- Read : lit un objet à partir de sa clé → read (key)
- Update : met à jour la valeur d'un objet à partir de sa clé → update (key, value)
- Delete: supprime un objet à partir de sa clé → delete(key)

3.3.2. Systèmes de BD document

- Sont constituées de collections de documents.
- Un document est composé de champs et des valeurs associées, ces dernières pouvant être requêtées.




```
{
    "name" : "MongoDB",
    "type" : "database",
    "count" : 1,
    "info" : {
        x : 203,
        y : 102
    }
}
```

- Les valeurs peuvent être, soit d'un type simple (**entier, chaine de caractère, date, ...**), soit elles-mêmes composées de plusieurs couples **clé/valeur**.
- Ces bases conservent des performances élevées. Elles disposent par ailleurs de fonctionnalités très intéressantes en terme de flexibilité du modèle documentaire, ce que les bases relationnelles ne peuvent offrir.

3.3.3. Bases orientées colonnes

- Les concepts essentiels sont les suivants :
  - Colonne : il s'agit de l'entité de base représentant un champ de donnée. Chaque colonne est définie par un couple clé / valeur.
  - Les colonnes sont regroupées par ligne et chaque ligne est identifiée par un identifiant unique. Elles sont généralement assimilées aux tables dans le modèle relationnel et sont identifiées par un nom unique.

	A	B	C	D
1	java	C++	Python	
2		C		
3			C#	Python

BD relationnelle

1	(A,java) (B,C++) (C,Python)
2	(B,C)
3	(C,C#) (D,Python)

BD orientée colonne

- Une colonne contenant d'autres colonnes est nommée supercolonne.
- Famille de colonnes : il s'agit d'un conteneur permettant de regrouper plusieurs colonnes ou supercolonnes.
- Les supercolonnes situées dans les familles de colonnes sont souvent utilisées comme les lignes d'une table de jointure dans le modèle relationnel.

Exemple : Supercolonne disposant chacune de 2 colonnes qui décrivent le firstname et le familyname de chaque personne, et Famille de colonnes

SuperColumns

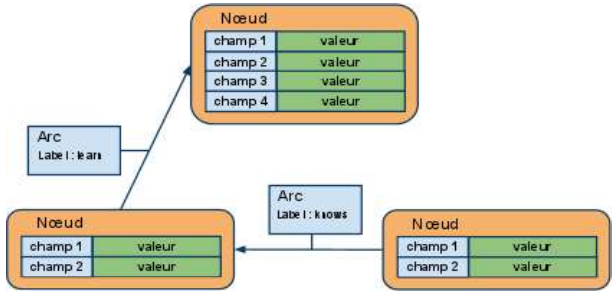
Key	Value
person1	Column: Name: firstName, Value: John; lastName, Calagan
person2	Column: Name: firstName, Value: George; lastName, Truffe

ColumnFamily

Key	Value
AddressBook	SuperColumns: person1 → Column: firstName, John; lastName, Calagan; person2 → Column: firstName, George; lastName, Truffe

3.3.4. Paradigme graphe

- Ce modèle s'appuie principalement sur deux concepts :
  - D'une part l'utilisation d'un moteur de stockage pour les objets (qui se présentent sous la forme d'une base documentaire, chaque entité de cette base étant nommée nœud).
  - D'autre part, vient s'ajouter à ce modèle un mécanisme permettant de décrire les arcs (relations entre les objets), ceux-ci étant orientés et disposant de propriétés (nom, date, …).



4. Solutions de base NoSQL

Type	Solution	Description
clé/valeur	Redis (open source)	Les bases de données sont exécutées directement en mémoire vive, sa vitesse d'exécution est donc excessivement augmentée.
	Berkeley DB	Fournit des propriétés de transactions ACID strictes par défaut.
Orienté colonne	BigTable by Google Cassandra by Facebook (Opensource) HBase (open source)	Permet de gérer des péta-octets de données répartis sur des milliers de serveurs.
Orienté document	MongoDB CouchDB	Le leader dans le domaine des BD NoSQL. La particularité est d'embarquer un serveur Web utilisant des protocoles standards.
Orienté graphe	Neo4j	Fournit une interface de REST (Representational State Transfert) bien conçues et documentées pour l'accès aux entités).

4.1. MongoDB (BD orientée documents)

- Les documents sont regroupés sous forme de collections, les collections étant l'équivalent des tables du SQL.

RDBMS	MongoDB
Table, View	Collection
Row	Document
Index	Index
Join	Embedded Document
Foreign Key	Reference
Partition	Shard

- Un document est essentiellement un tableau
  - Document == Objet JSON
  - Document == Tableau PHP
  - Document == Dictionnaire Python
  - etc.

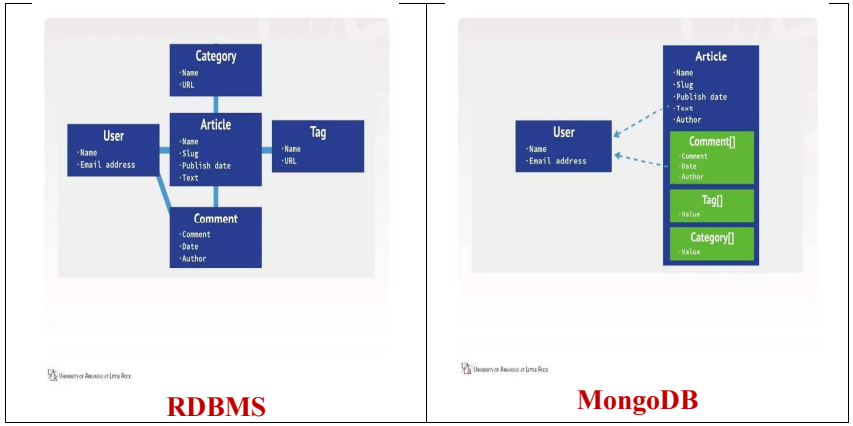
```
{
"nom": "Informatique",
"code": "420"
}

{
"num_ad": "20183383518",
"nom": "Patoche",
"prénom": "Alain"
}
```


- Pourquoi utiliser MongoDB ?
  - Stockage orienté document - Les données sont stockées sous forme de documents de style JSON.
  - Index sur n'importe quel attribut
  - Réplication et haute disponibilité
  - Partage automatique
  - Requêtes riches
  - Mises à jour rapides sur place
  - Assistance professionnelle par MongoDB

- MongoDB a des liaisons natives pour plus de 12 langues

4.1.1. MongoDB - Modélisation des données

- MongoDB propose deux types de modèles de données : Modèle de données intégré (Embedded) et modèle de données normalisé (Referenced).



- **Modèle de données intégré**, également appelé modèle de données dénormalisé : toutes les données associées peuvent être intégrer dans un seul document.

Exemple : Intégration de trois documents des employés Personal_details, Contact et Address, dans un seul.

```
db.users.insert (
   {    Emp_ID: "10025AE336",
        Personal_details:{
             First_Name: "Racha",
              Last_Name: "Mark",
            Date_Of_Birth: "1995-09-26"
         },
         Contact: [  {
mail:"racha_mark@gmail.com",
                  Phone : 9848022338 } ],
         Address: [  { city: "Rabat",
             State: " Morocco" }  ]
   }
)
```

Cette approche conserve toutes les données associées dans un seul document, ce qui facilite leur récupération et leur maintenance. L'ensemble du document peut être récupéré en une seule requête telle que :

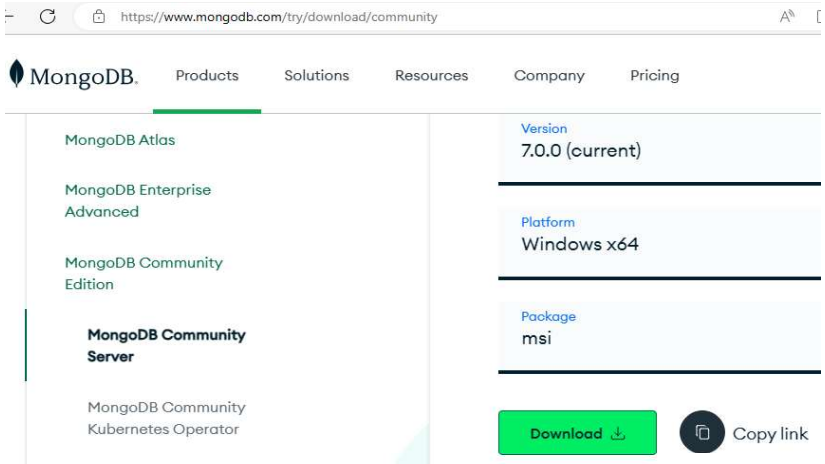
```
>db.users.findOne({"First_Name":"
Racha "},{"Address":1})
```


- **Modèle de données normalisé** : faire référence aux sous-documents du document original, à l'aide de références.

Employee:	**Personal_details:**
{ _id: <ObjectId101>, Emp_ID: "10025AE336" }	{ _id: <ObjectId102>, empDocID: "ObjectId101", First_Name: "Racha", Last_Name: "Mark", Date_Of_Birth: "1995-09-26" }
Contact:	**Address:**
{ _id: <ObjectId103>, empDocID: "ObjectId101", e-mail: "racha_mark@gmail.com", phone: "9848022338" }	{ _id: <ObjectId104>, empDocID: "ObjectId101", city: "Rabat", State: "Morocco" }

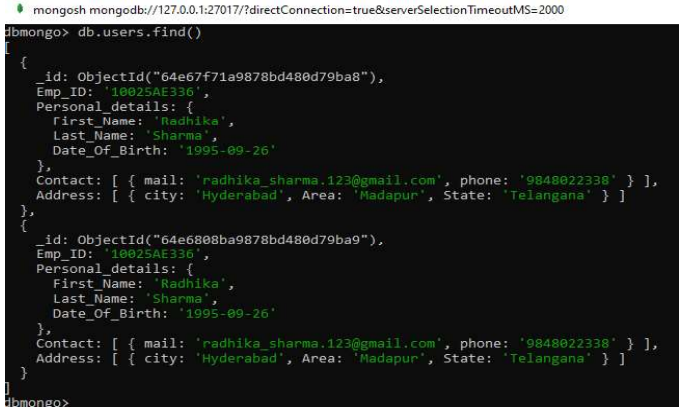
4.1.2. Développer avec MongoDB

- Installation et Démarrage du serveur MongoDB : mongod.exe



- Démarrage de l'interpréteur de commandes MongoDB : Mongo utilise le langage JavaScript pour la programmation dans le shell, avec un ensemble de méthodes permettant la gestion de la base et des données

c:\Users\...\mongodb-win-4.2.1\bin>mongo.exe



Requête	Expression
Connexion ou création d'une base de données	use *<nom_de_la_base>*
base de données courante	*db*
Suppression de la base	*db.dropDatabase()*
Afficher toutes les collections	*show collections*
Suppression des collections	*db.collection.dropCollection()*
Création d'une collection	*db.createCollection('livres')*
Insertion un document dans une collection	db.*<nom_de_la_collection>*.insert(*<document>*)
Insertion plusieurs documents	db.livres.insertMany([{ titre : 'Machine learning avec Scikit-learn : mise en oeuvre et cas concrets', auteur : 'Géron, Aurélien', editeur : 'Dunod', dateParution : '20/11/2019', nbpages :459 }, …..])

Lecture d'une collection	db.*<nom_de_la_collection>*.find(*[paramètres]*) db.livres.find() db.livres.find({auteurs: {$not: /^F/}})

– Opération d'interrogation du document sur la base d'une condition :

Opération	Exemple
Égalité	{<key>:{$eg;<value>}} db.livres.find({"editeur":"Dunod"}) db.livres.find({"editeur": "Dunod","titre": "C++"}})
Moins que	{<key>:{$lt:<value>}} db.livres.find({"PU":{$lt:350}})
Inférieur à égal	{<key>:{$lte:<value>}}
Plus grand que	{<key>:{$gt:<value>}}
Supérieur à égal	{<key>:{$gte:<value>}}

Pas égal	{<key>:{$ne:<value>}}
Valeurs dans un tableau	{<key>:{$in:[<value1>, <value2>,……<valueN>]}} db.livres.find({"titre":{$in:["C++", "Java", "Python"]}})
Valeurs ne figurant pas dans un tableau	{<key>:{$nin:<value>}} db.livres.find({"titre":{$nin:["SQL", "NoSQL"]}})
ET logique	{ $and: [{<key1>:<value1>}, { <key2>:<value2>}] } >db.livres.find({ $and: [{<key1>:<value1>}, { <key2>:<value2>}] })
OU logique	{ $or: [{key1: value1}, {key2:value2}] } db.livres.find({"$or": [{"editeur":"Dunod"}, {"titre": "C++"}] })

Requête	Expression
Modification : update()	db.livres.update({'auteur':'Géron, Aurélien'},{$set:{'auteur':'Géron, Alain'}}) Par défaut, MongoDB ne mettra à jour qu'un seul document. Pour mettre à jour plusieurs documents, définir un paramètre « multi » sur true. db.livres.update({'auteur':'Géron, Aurélien'}, {$set:{'auteur':'Géron, Alain''}},{multi:true})
save() :	db.COLLECTION_NAME.save ({_id:ObjectId(),NEW_DATA}) remplace le document existant par le nouveau document passé
findOneAndUpdate() met à jour les valeurs dans le document existant.	db.COLLECTION_NAME.findOneAndUpdate (SELECTIOIN_CRITERIA, UPDATED_DATA)
updateOne()	db.COLLECTION_NAME.updateOne(<filter>, <update>) met à jour un seul document qui correspond au filtre donné.
updateMany()	db.COLLECTION_NAME.updateMany (<filter>, <update>) met à jour tous les documents correspondant au filtre donné

Suppression d'une entrée	db.livres.remove({auteurs: 'Géron, Aurélien')
Suppression uniquement le premier enregistrement	db.COLLECTION_NAME.remove (DELETION_CRITERIA,1)
Récupération d'un sous-ensemble du document (projection)	db.livres.find({editeur: "Dunod"},{auteur: 1})
Tri des résultats	db.livres.find({}).sort({titre: -1})
Limiter le nombre des documents à afficher	db.livres.find({},{"titre":1,_id:0}).limit(2)
Interroger avec RegEx	db.livres.findOne({Titre : /^Programmer/})
Création d'un index avec createIndex	db.livres.createIndex({"auteur": 1}) db.livres.find({"auteur": "Géron"})
Comprendre l'exécution d'une requête : explain()	db.livres.find({"auteur": "Géron"}).explain()
Agrégation En SQL count(*) et avec group by est un équivalent de l'agrégation MongoDB	db.COLLECTION_NAME.aggregate(AGGREGATE_OPERATION) db.livres.aggregate([{$group : {_id : "$auteur", nombre : {$sum : 1}}}])

– Chaque document dispose d'une clé unique ("_id": ObjectId("50804d0bd94ccab2da652599")) permettant de l'identifier dans la collection.

```
db.users.findOne()
{
        "_id" : ObjectId("50804d0bd94ccab2da652599"),
        "username" : "fred.jones",
        "first_name" : "fred",
        "last_name" : "jones"
}
```

– _id est la clé primaire dans MongoDB
  - Indexé automatiquement
  - Créé automatiquement en tant qu'ObjectId s'il n'est pas fourni
  - Toute valeur unique et immuable peut être utilisée

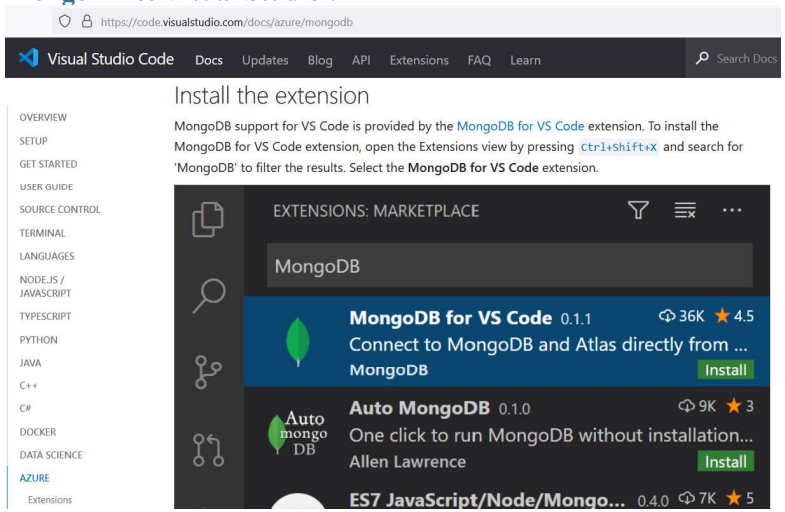
– ObjectId est une valeur spéciale de 12 octets
  - Garanti d'être unique dans le cluster
  - ObjectId("50804d0bd94ccab2da652599")

|------------||---------||-----||-------- |
ts mac pid inc.

– MongoDB et Java

Instruction	Expression
Création du client, connexion au serveur local	MongoClient mongoClient = new MongoClient("localhost" , 27017);
Connexion à la BD	MongoDatabase database = mongoClient.getDatabase("bibliographie");
Création de collection	MongoCollection collection = database.getCollection("article"); Document doc = new Document("auteurs", Arrays.asList("Antoine Dupont", "Frédéric Petit")) .append("titre", "Databases for the next generation") .append("editeur", "University Press") .append("categories", Arrays.asList("informatique", "Big Data", "technologies web")); collection.insertOne(doc);
Recherche des collections	Document myDoc = collection.find().first();

- **MongoDB et Visual Studio :**



4.1.3. Réplication

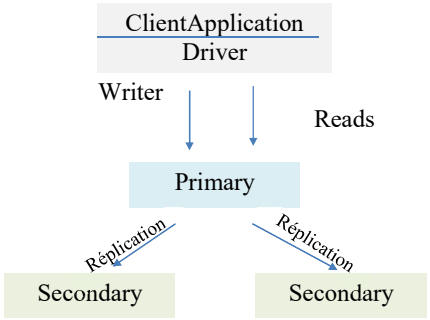
– La réplication est le processus de synchronisation des données sur plusieurs serveurs.

– La réplication assure la redondance et augmente la disponibilité des données avec plusieurs copies de données sur différents serveurs de base de données.

– Pourquoi la réplication ?
  - Haute disponibilité des données
  - Reprise après panne
  - Aucun temps d'arrêt pour la maintenance (comme les sauvegardes, les reconstructions d'index, le compactage)
  - Mise à l'échelle de lecture (copies supplémentaires à lire)
  - L'ensemble de réplicas est transparent pour l'application

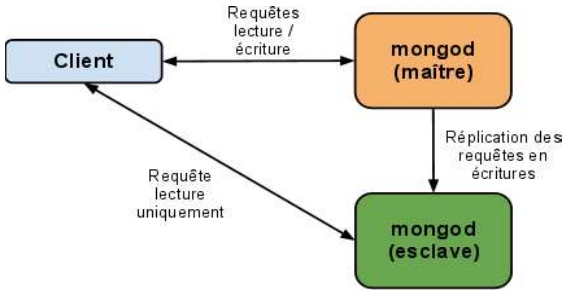
– Comment fonctionne la réplication dans MongoDB

  - Un diagramme typique de réplication MongoDB est présenté dans lequel l'application client interagit toujours avec le nœud principal et le nœud principal réplique ensuite les données vers les nœuds secondaires.

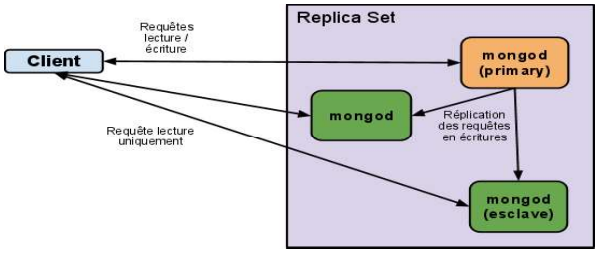


– Un serveur MongoDB peut gérer plusieurs bases de données.
  - Mongo a été conçu pour fonctionner selon plusieurs modes distincts :
    - Serveur seul : un seul processus nommé **mongod** est utilisé et traite directement les données issues des requêtes du client.

- Réplication par master-slave (un nœud primaire, plusieurs nœuds secondaires) : deux serveurs mongod nommés trivialement 'maître' et 'esclave' :
  - Les données sont écrites sur le maître uniquement.
  - Le maître réplique l'ensemble des écritures, avec une certaine latence.
  - Le client peut choisir de lire soit sur l'esclave s'il accepte les erreurs liées à la latence de réplication, soit sur le maître dans le cas contraire.



- Réplication par Replica Set :
  - Il s'agit d'un mode de réplication plus avancé que le mode maître / esclave
  - Ce mode permet d'éviter la présence d'un point de défaillance unique au sein de l'infrastructure par la méthode suivante :
    - On ajoute n serveurs au Replica Set. Chaque serveur dispose d'une priorité.
    - Un des serveurs est considéré comme le nœud primaire. Le nœud qui sera en charge des écritures et des réplications vers les autres serveurs.
    - En cas de défaillance du nœud primaire, un nouveau nœud au sein du Replica Set est choisi et devient nœud primaire.



- Conversion une instance MongoDB autonome en un jeu de réplicas :
  - Arrêter le serveur MongoDB déjà en cours d'exécution.
  - Démarrer le serveur MongoDB en spécifiant l'option -- replSet.

```
mongod --port "PORT" --dbpath "DB_DATA_PATH" --replSet "REPLICA_SET_INSTANCE_NAME"
```


Exemple :

- Démarrage une instance Mongod avec le nom rs0, sur le port 27017.

```
mongod --port 27017 --dbpath "D:\set up\mongodb\data" --replSet rs0
```

- Démarrage l'invite de commande et connecter à cette instance Mongod.
- Dans le client Mongo, exécuter la commande rs.initiate() pour lancer un nouveau jeu de réplicas.
- Pour vérifier la configuration du jeu de réplicas, exécuter la commande rs.conf().
- Pour vérifier l'état du jeu de réplicas, exécutez la commande rs.status().

- Ajouter des membres au jeu de réplicas :

```
rs.add(HOST_NAME:PORT)
```

  - Démarrage des instances mongod sur plusieurs machines.
  - Démarrage un client mongo et émettre une commande rs.add().

Exemple :

Supposons que le nom de l'instance mongod soit mongod1.net et qu'elle s'exécute sur le port 27017.

Pour ajouter cette instance au jeu de réplicas, exécuter la commande rs.add() dans le client Mongo.

```
rs.add("mongod1.net:27017")
```


4.1.4. Combiner deux collections

1. Agrégation : $lookup pour joindre deux collections
2. Opérateur pipeline pour joindre deux collections en fonction de la condition spécifiée
3. Opérateur $unwind pour aplatir un tableau avant de l'attacher aux documents résultants
4. Filtrage $project dans les requêtes d'agrégation pour joindre deux collections

Exemple : Creation de deux collections:

```
> use users
> db.createCollection('userInformation')
> db.createCollection('userAddress')
```

```
> db.userInformation.insertMany(
  [
    {
      fullname: 'Alaoui Majda',
      age: 30,
      gender: 'Female',
      nationality: 'Marocain'
    },
    {
      fullname: 'James Daniel',
      age: 45,
      sex: 'male',
      nationality: 'Canadian'
    }
  ])
```

```
> db.userAddress.insertMany(
  [
    {
      fullname: 'Alaoui Majda',
      block_number: 22,
      street: 'Rue Badr n40',
      city: 'Rabat'
    },
    {
      fullname: 'James Daniel',
      block_number: 30,
      street: 'Saint-Denis Street',
      city: 'Montreal'
    }
  ])
```


1. Agrégation : $lookup

- La fonction **$lookup** accepte quatre champs.
  - Le premier est le champ from, spécifie la collection qui est censée être jointe à l'autre collection.
  - Le second est le champ localField. C'est l'un des attributs (champ) des documents d'entrée de la collection spécifié dans le champ from.
  - Il est utilisé pour effectuer une correspondance entre le localField et le foreignField à partir des documents des collections.
  - Le troisième champ nommé foreignField effectue également la correspondance d'égalité entre foreignField et localField à partir des documents des collections.
  - Le nom du nouveau tableau pour le quatrième champ, est écrit par as

```
> db.userInformation.aggregate([
   { $lookup:
      {
        from: 'userAddress',
        localField: 'fullname',
        foreignField: 'fullname',
        as: 'address'
      }
   }
])
```

Résultat:

```
{
        "_id" : ObjectId("628bc4a45c544feccff5a566"),
        "fullname" : "Alaoui Majda",
        "age" : 30,
        "gender" : "Female",
        "nationality" : "Marocain",
        "address" : [
                {
                        "_id" :
ObjectId("628bc4ae5c544feccff5a568"),
                        "fullname" : "Alaoui Majda",
                        "block_number" : 22,
                        "street" : "Rue Badr n40",
                        "city" : "Rabat"
                }
        ]
}
.....
```


2. **Utilisation de l'opérateur pipeline** avec $lookup pour joindre deux collections en fonction d'une condition spécifique (tout comme la clause WHERE dans MySQL).

Exemple : joindre les deux collections où le fullname de userAddress est égal au fullname dans userInformation

```
db.userInformation.aggregate([{
   $lookup:{
      from: 'userAddress',
      let: {full_name: '$fullname'},
      pipeline: [{
         $match: {
            $expr: {
               $eq: ['$fullname', $$full_name']
            }
         }
      }],
      as: 'addressInfo'
}}])
```

Résultat:

```
{
        "_id" :
ObjectId("628bc4a45c544feccff5a566"),
        "fullname" : "Alaoui Majda",
        "age" : 30,
        "gender" : "Female",
        "nationality" : "Marocain",
        "addressInfo" : [
                {
                "_id" :
ObjectId("628bc4ae5c544feccff5a568"),
                 "fullname" : "Alaoui Majda",
                 "block_number" : 22,
                 "street" : "Rue Badr n40",
                 "city" : "Rabat"   }  ]  }
```


3. **Utilisation de l'opérateur $unwind** pour créer un tableau plat avant d'attacher aux documents résultants

```
> db.userInformation.aggregate([
   { $lookup:
      {
        from: 'userAddress',
        localField: 'fullname',
        foreignField: 'fullname',
        as: 'address'
      }
   },
   {
     $unwind: '$address'
   }
])
```

Résultat:

```
{
        "_id" :
ObjectId("628bc4a45c544feccff5a566"),
        "fullname" : "Alaoui Majda",
        "age" : 30,
        "gender" : "Female",
        "nationality" : "Marocain",
        "address" : {
                "_id" :
ObjectId("628bc4ae5c544feccff5a568"),
                "fullname" : "Alaoui Majda",
                "block_number" : 22,
                "street" : "Rue Badr n40",
                "city" : "Rabat"
        } }
```


4. **Utilisation du filtre $project** dans les requêtes d'agrégation pour joindre deux collections en une seule

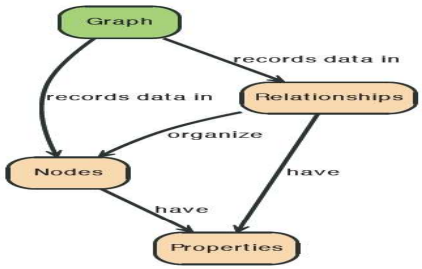
```
> db.userInformation.aggregate([
   { $lookup:
      {
        from: 'userAddress',
        localField: 'fullname',
        foreignField: 'fullname',
        as: 'address'
      }
   },
   {
     $unwind: '$address'
   },
   {
     $addFields: {
        street: '$address.street',
        city: '$address.city'
     }
   }
])
```

Résultat:

```
{
        "_id" :
ObjectId("628bc4a45c544feccff5a566"),
        "fullname" : "Alaoui Majda",
        "age" : 30,
        "gender" : "Female",
        "nationality" : "Marocain",
        "address" : {
                "_id" :
ObjectId("628bc4ae5c544feccff5a568"),
                "fullname" : "Alaoui Majda",
                "block_number" : 22,
                "street" : "Rue Badr n40",
                "city" : "Rabat"
        },
        "street" : "Rue Badr n40",
        "city" : "Rabat" }
```

4.2. Neo4j (BD orientée graphe)

- Neo4j est la première base de données graphique open source, développée à l'aide de la technologie Java.
- Les autres bases de données graphiques sont Oracle NoSQL Database, OrientDB, HypherGraphDB, GraphBase.
- La base de données graphique est une base de données utilisée pour modéliser les données sous forme de graphe. Les nœuds (sommets) d'un graphe représentent les entités tandis que les relations (arêtes) décrivent l'association de ces nœuds.



Le graphe enregistre les données dans → Nœuds qui ont → Propriétés

Les nœuds sont organisés par → Relations qui ont également → Propriétés »

RDBMS	Graph Database
les tables	Graphiques
Lignes	Nœuds
Colonnes et données	Propriétés et ses valeurs
Contraintes	Des relations
Rejoint	Traversée

4.2.1. A quoi servent les graphiques ?

- Recommandations
- Business Intelligence
- Réseaux sociaux
- Géo spatial
- Management des systèmes
- Internet des Objets
- Généalogie
- Données de séries chronologiques
- Catalogue des produits
- Web Analytiques
- Bio-informatique
- …..

4.2.2. Avantages de Neo4j

- Modèle de données flexible - simple mais puissant, qui peut être facilement modifié en fonction des applications.
- Haute disponibilité - pour les applications en temps réel des grandes entreprises avec des garanties transactionnelles.
- Récupération facile - des données connectées plus rapidement que d'autres bases de données.
- Évolutivité et fiabilité – évolution de la base de données en augmentant le nombre de lectures/écritures et le volume sans affecter la vitesse de traitement des requêtes et l'intégrité des données.
- Prend en charge les règles ACID complètes (atomicité, cohérence, isolement et durabilité).

- PAS de jointures complexes pour récupérer les données connectées/associées car il est très facile de récupérer son nœud adjacent ou les détails de sa relation sans jointures ni index.
- Utilise le langage de requête Cypher - déclaratif pour représenter visuellement le graphique, en utilisant une syntaxe ascii-art. Les commandes de ce langage sont dans un format lisible et très facile à apprendre.

Cypher

```
MATCH(p:Product)-[:CATEGORY]->(1:ProductCategory)-
[:PARENT*0.]->(:ProductCategory{name:"Dairy Products"})
RETURN p.name
```

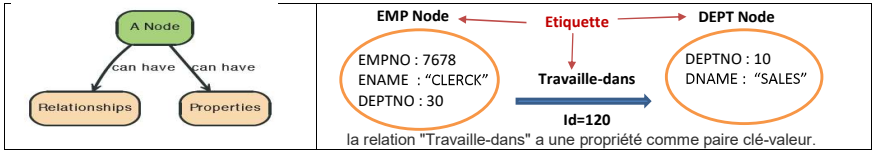
SQL :

```
SELECT p.ProductName
FROM Product AS p
JOIN ProductCategory pc ON (p.CategoryID = pc.CategoryID AND
pc.CategoryName = "Dairy Products")
```

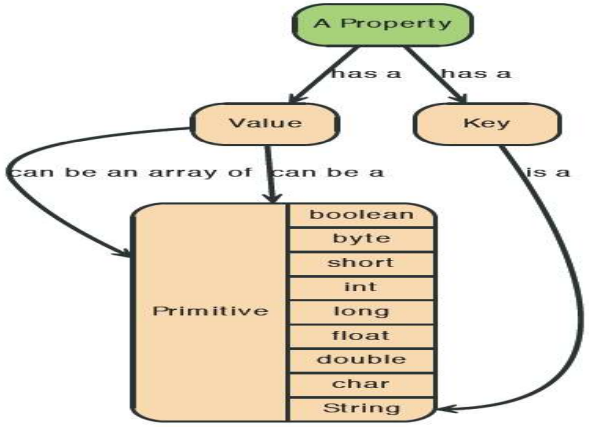

- Neo4j prend également en charge la réplication pour la sécurité et la fiabilité des données.
- Neo4j fournit une application Web de navigateur Neo4j intégrée.
- Indexation - Neo4j prend en charge les index en utilisant Apache Lucence.
- Pilotes - Neo4j peut fonctionner avec :
  - API REST pour travailler avec des langages de programmation tels que Java, Spring, Scala, etc.
  - Java Script pour fonctionner avec les frameworks Node JS.
  - Il prend en charge deux types d'API Java : l'API Cypher et l'API Java native pour développer des applications Java.

4.2.3. Modèle de données de graphique Neo4j

- La base de données Neo4j suit le **modèle Property Graph** pour stocker et gérer ses données.
  - Le modèle stocke les données dans les nœuds, les relations et les propriétés
  - Les propriétés sont des paires **clé-valeur**
  - Les nœuds sont représentés à l'aide d'un **cercle** et les relations sont représentées à l'aide **des touches fléchées**.
  - Les relations ont des directions : unidirectionnelles et bidirectionnelles
  - Chaque relation contient "**Start Node**" ou "From Node" et "To Node" ou "**End Node**".
  - Les nœuds et les relations contiennent des propriétés
  - Les relations connectent les nœuds.
  - L'étiquette associe un nom commun à un ensemble de nœuds ou de relations

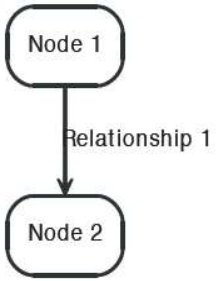


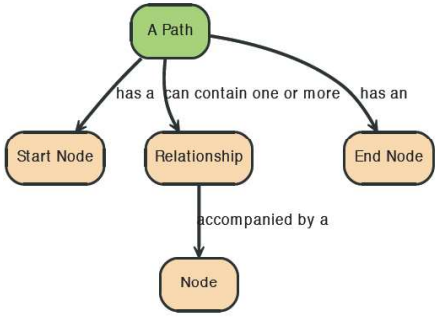
- Propriétés



– Chemin dans Neo4j

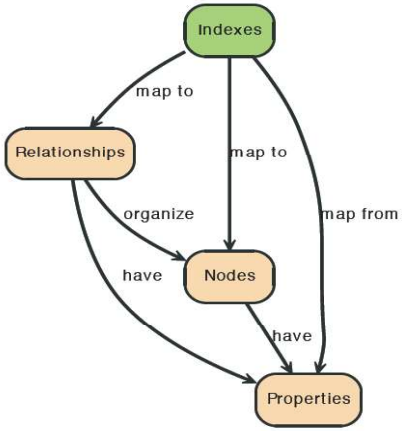
Un chemin est constitué d'un ou plusieurs nœuds avec des relations de connexion, généralement récupérés sous forme de requête ou de résultat de parcours.





– Indexes

- Un index — mappe de → Propriétés — vers → Nœuds ou relations »



-

4.2.4. Neo4j CQL (langage de requête Cyphe)

- Est un langage de requête pour Neo4j Graph Database.
- Suit la syntaxe SQL like.
- Les types de données sont similaires au langage Java. Ils sont utilisés pour définir les propriétés d'un nœud ou d'une relation.
- Les opérateurs de comparaison sont similaires au langage SQL.
- **Fonctions CQL Neo4j :**

Fonctions CQL	Usage
String	Ils sont utilisés pour travailler avec des littéraux de chaîne.
Agrégation	Ils sont utilisés pour effectuer certaines opérations d'agrégation sur les résultats de requête CQL.
Relation	Ils sont utilisés pour obtenir des détails sur les relations telles que startnode, endnode, etc.

- **Neo4j CQL Clauses de lecture**

Clauses de lecture	Usage
MATCH	utilisée pour rechercher les données avec un modèle spécifié.
MATCH OPTIONNEL	C'est la même chose que la correspondance, la seule différence étant qu'il peut utiliser des valeurs nulles en cas de parties manquantes du motif.
WHERE	utilisé pour ajouter du contenu aux requêtes CQL.
START	utilisée pour trouver les points de départ à travers les index hérités.
LOAD CSV	permet d'importer des données à partir de fichiers CSV.

-

- Neo4j CQL Clauses d'écriture

Clause d'écriture	Usage
CREATE	utilisée pour créer des nœuds, des relations et des propriétés.
MERGE	vérifie si le modèle spécifié existe dans le graphique. Sinon, cela crée le motif.
SET	utilisée pour mettre à jour les étiquettes sur les nœuds, les propriétés sur les nœuds et les relations.
DELETE	utilisée pour supprimer des nœuds et des relations ou des chemins, etc. du graphique.
REMOVE	permet de supprimer des propriétés et des éléments de nœuds et de relations.
FOREACH	utilisée pour mettre à jour les données d'une liste.
CREATE UNIQUE	À l'aide des clauses CREATE et MATCH, on peut obtenir un modèle unique en faisant correspondre le modèle existant et en créant le modèle manquant.
Importing CSV files	À l'aide de Charger un fichier CSV, vous pouvez importer des données à partir de fichiers .csv.

- Les clauses générales de Neo4j

Clauses générales	Usage
RETURN	permet de définir les éléments à inclure dans le jeu de résultats de la requête.
ORDER BY	utilisée pour organiser la sortie d'une requête dans l'ordre. Il est utilisé avec les clauses **RETURN** ou **WITH.**
LIMIT	permet de limiter les lignes du résultat à une valeur spécifique.
SKIP	permet de définir à partir de quelle ligne commencer, y compris les lignes de la sortie.
WITH	utilisée pour chaîner les parties de requête ensemble.
UNWIND	permet de développer une liste en une séquence de lignes.
UNION	utilisée pour combiner le résultat de plusieurs requêtes.
CALL	permet d'appeler une procédure déployée dans la base de données.

• Clause CREATE

Usage	Syntaxe
Création un seul nœud	`CREATE (node_name);`
Création de deux nœuds	`CREATE (node1),(node2)`
Une étiquette dans Neo4j est utilisée pour regrouper (classer) les nœuds	`CREATE (node:label)`
Création d'un nœud avec plusieurs étiquettes	`CREATE (node:label1:label2: . . . labeln)`
Les propriétés sont les paires clé-valeur à l'aide desquelles un nœud stocke les données	`CREATE (node:label { key1: value, key2: value, . . . })`
Création d'une relation	`CREATE (node1)-[:RelationshipType]->(node2)`

Création d'une relation entre les nœuds existants	`MATCH (a:LabeofNode1), (b:LabeofNode2) WHERE a.name = "nameofnode1" AND b.name = " nameofnode2" CREATE (a)-[:Relation]->(b) RETURN a,b`
Création d'une relation avec l'étiquette et les propriétés	`CREATE (node1)-[label:Rel_Type {key1:value1, key2:value2, . . . n}]->(node2)`
Création d'un chemin	`CREATE p = (Node1 {properties}-)-[:Relationship_Type]-> (Node2 {properties})[:Relationship_Type]->(Node3 {properties}) RETURN p`

• Clause MERGE

Vérifie si le modèle spécifié existe dans le graphique. Sinon, cela crée le motif.	`MERGE (node: label {properties . . . . })`
Fusion d'un nœud avec une étiquette	`MERGE (node:label) RETURN node`
Fusion d'un nœud avec des propriétés	`MERGE (node:label {key1:value, key2:value, key3:value . .. . . . })`
Fusionner une relation	`MERGE (node1)-[:RelationshipType]->(node2)`

• Clause MATCH

Utilisée pour rechercher les données avec un modèle spécifié	`MATCH (n) RETURN n`
Classement des nœuds par ordre décroissant, et limiter nombre de lignes	`MATCH (n) RETURN n ORDER BY n.name DESC LIMIT 3`
Compter le nombre de lignes	`MATCH (n { name: 'A' })-->(x) RETURN n, count(*)`
Définition d'une propriété	`MATCH (node:label{properties . .. }) SET node.property = value RETURN node`
Suppression d'une propriété	`MATCH (node:label {properties}) SET node.property = NULL RETURN node`

Définition d'une étiquette sur un nœud	`MATCH (n {properties . . . }) SET n :label RETURN n`
Suppression de tous les nœuds et relations	`MATCH (n) DETACH DELETE n`
Suppression d'un nœud particulier	`MATCH (node:label {properties . . . . }) DETACH DELETE node`
Suppression d'une propriété	`MATCH (node:label{properties . . . . }) REMOVE node.property RETURN node`
Suppression d'une étiquette d'un nœud	`MATCH (node:label {properties . . . . . }) REMOVE node:label RETURN node`
Suppression de plusieurs étiquettes	`MATCH (node:label1:label2 {properties . . . . }) REMOVE node:label1:label2 RETURN node`

Recherche tous les nœuds sous une étiquette spécifique	`MATCH (node:label) RETURN node`
Recherche des nœuds en fonction de la relation	`MATCH (node:label)<-[:Relationship]-(n) RETURN n`
Supprimer tous les nœuds	`MATCH (n) detach delete n`