Introduction au NoSQL

Explorez les horizons du NoSQL, une approche de gestion de données qui transcende les limitations des bases de données relationnelles traditionnelles. Avec le NoSQL, libérez-vous des schémas rigides et découvrez la liberté de stocker et de manipuler des données de manière flexible, permettant une adaptation rapide aux besoins changeants de votre entreprise. Grâce à sa capacité à gérer des volumes massifs de données de manière distribuée et à évoluer facilement, le NoSQL ouvre la voie à l'innovation continue, offrant des solutions agiles pour les défis de demain.

Sommaire

I. Rappels sur les bases de données relationnelles

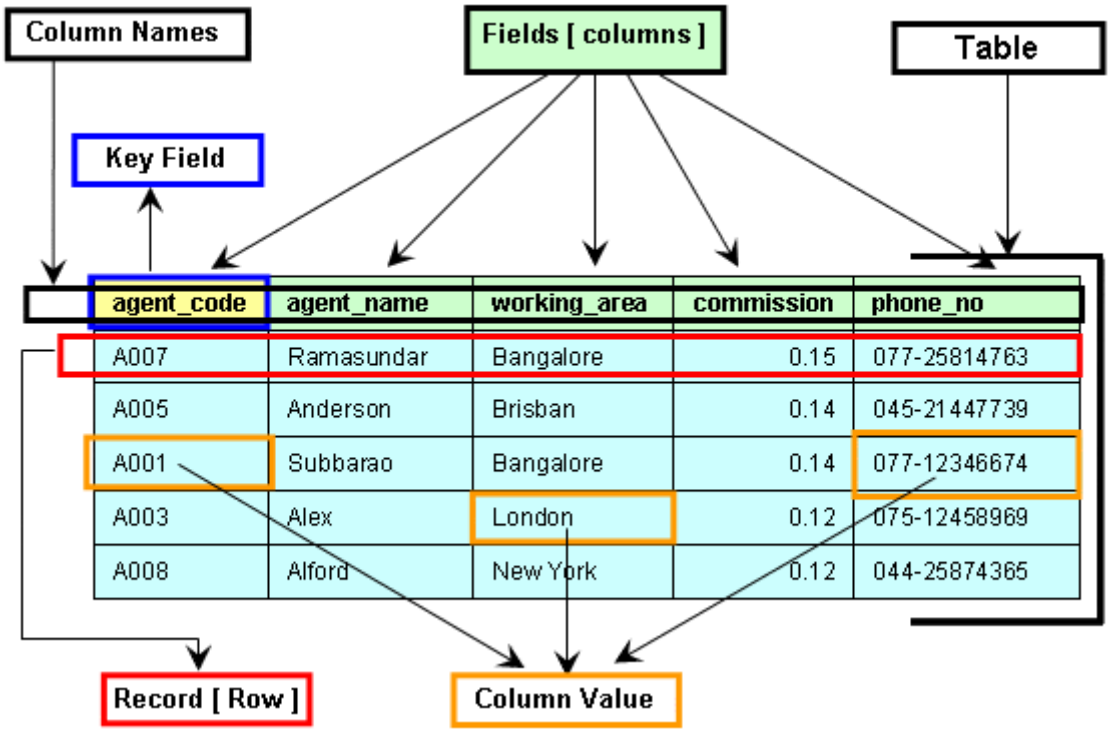
A. Concepts clés

Le SQL (Structured Query Language) est un langage informatique utilisé pour manipuler les bases de données relationnelles. Ces bases de données sont basées sur un modèle relationnel, qui organise les données en différentes tables, où chaque table représente une entité distincte et chaque ligne dans la table représente un enregistrement unique.

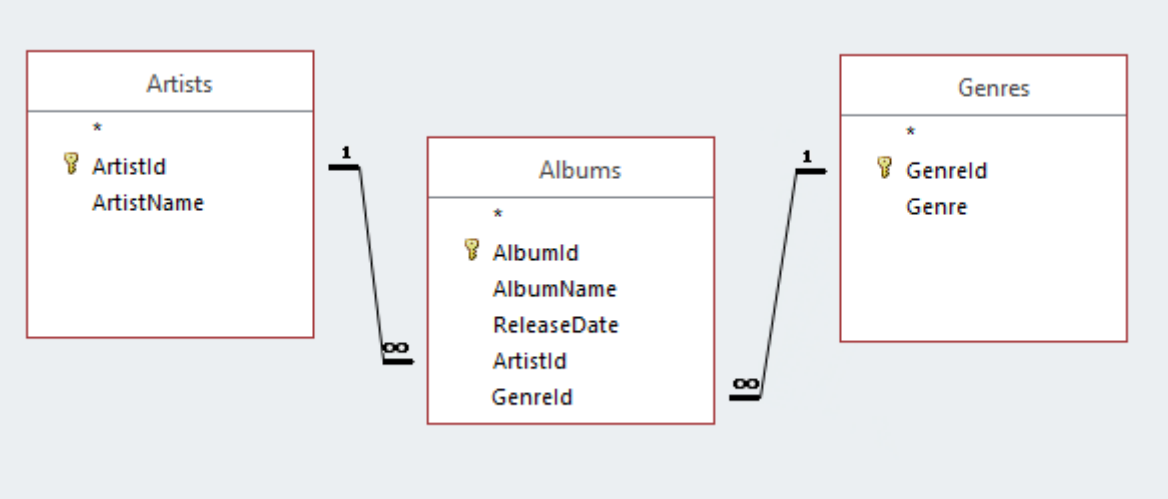
```sql
SELECT al.AlbumName, al.ReleaseDate, ar.ArtistName, g.Genre
FROM Albums al
JOIN Artist ar ON al.ArtistId = ar.ArtistId
JOIN Genres g ON al.GenreId = g.GenreId
ORDER BY al.ReleaseDate ASC;
```

Caractéristiques du SQL :

- **Données structurées :** Le SQL est conçu pour manipuler des données structurées, c'est-à-dire des données organisées en tables avec des lignes et des colonnes. Ces données sont généralement extraites de bases de données relationnelles où la structure est bien définie à l'avance.



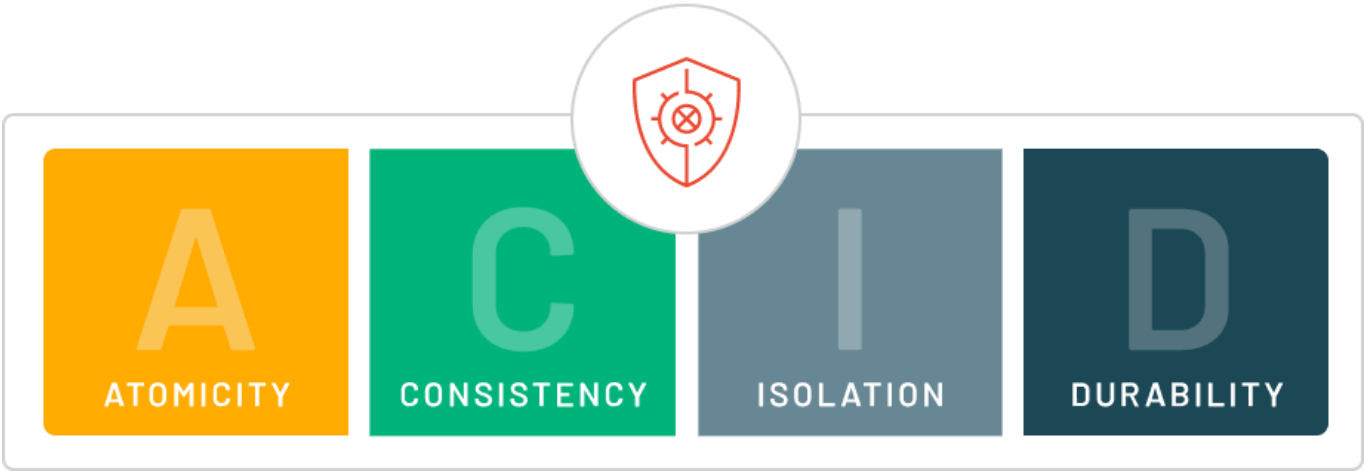
- **Modèle de données relationnel :** Le modèle relationnel consiste en des tables contenant des enregistrements organisés en colonnes. Les relations entre les tables sont établies à l'aide de clés primaires et de clés étrangères, ce qui permet de représenter des liens logiques entre les différentes entités. Ces relations utilisent des cardinalités afin de pouvoir relier un ou plusieurs éléments à un ou plusieurs autres grâce aux IDs.



Le SQL offre une syntaxe puissante et expressive pour interagir avec les bases de données relationnelles, permettant aux utilisateurs d'effectuer un large éventail d'opérations, telles que la sélection, l'insertion, la mise à jour et la suppression de données, ainsi que la définition et la manipulation de schémas de base de données.

B. Principe de gestion des données relationnel : ACID

Acronyme qui représente les propriétés fondamentales des transactions dans les bases de données relationnelles. Les transactions sont des opérations unitaires qui modifient les données d'une base de données. Pour garantir la fiabilité et la cohérence des données, chaque transaction doit respecter les propriétés suivantes :



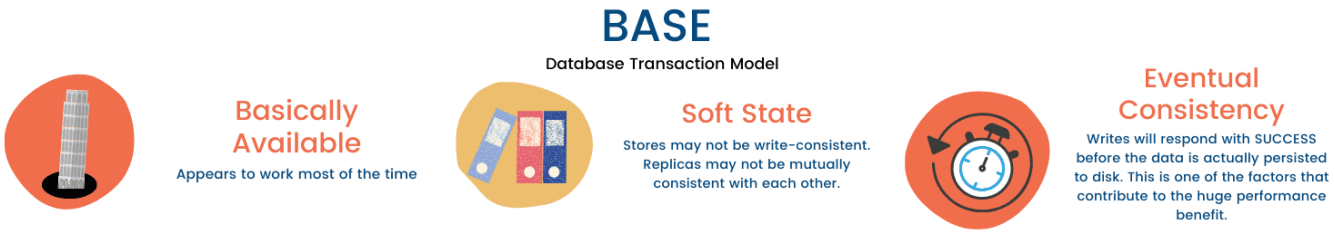
- **Atomicité** :
  - Une transaction est une opération tout ou rien. Cela signifie qu'elle est exécutée dans son intégralité ou pas du tout.
  - Si une partie de la transaction échoue pour une raison quelconque, toutes les modifications effectuées par cette transaction doivent être annulées (rollback), préservant ainsi la cohérence des données.
- **Cohérence** :
  - Après l'exécution d'une transaction, la base de données doit passer d'un état valide à un autre état valide, respectant toutes les règles d'intégrité référentielle et les contraintes définies.
  - Les modifications apportées par une transaction doivent préserver l'intégrité structurelle et logique des données.
- **Isolation** :
  - Les transactions doivent s'exécuter de manière isolée les unes des autres, comme si chaque transaction était la seule à accéder aux données de la base de données.
  - Cela signifie que les effets d'une transaction ne doivent pas être visibles pour les autres transactions tant que la transaction n'est pas terminée.
- **Durabilité** :
  - Une fois qu'une transaction est confirmée, ses modifications doivent être persistantes, même en cas de panne du système.
  - Les données modifiées doivent être sauvegardées de manière permanente sur le support de stockage, garantissant ainsi que les modifications ne seront pas perdues.

II. Transition vers les bases de données non relationnelles

Pour cette partie on va parler du NoSQL ou "Not Only SQL" qui désigne une approche de gestion de données qui diffère des bases de données relationnelles traditionnelles (SQL)

A. Principe de gestion des données non relationnel : BASE

Alternative au modèle ACID, souvent utilisée dans les systèmes distribués et les bases de données NoSQL. Contrairement à ACID, BASE met l'accent sur la disponibilité et la tolérance aux partitions réseau, au détriment parfois de la cohérence immédiate des données.



- **Basically Available** :
    - Le système garantit une disponibilité constante, même en cas de pannes ou de partitions du réseau.
    - Les services restent disponibles pour les requêtes même si une partie du système échoue, ce qui garantit une expérience utilisateur continue.
- **Soft State** :
    - Les données peuvent exister dans des états intermédiaires ou instables pendant un certain laps de temps sans être nécessairement en conflit.
    - Contrairement au modèle ACID, BASE permet aux données d'être temporairement dans un état de transition, tant que la cohérence finale est atteinte à terme.
- **Eventually Consistent** :
    - À terme, le système atteindra un état de cohérence où toutes les copies de données seront synchronisées.
    - Bien que les mises à jour des données puissent être propagées de manière asynchrone à travers le système, la cohérence finale est garantie, même si elle peut prendre un certain temps pour être réalisée.

B. Modèles de données et schémas flexibles

Là où les bases de données relationnelles utilisent un modèle de données tabulaire fixe où les schémas sont prédéfinis et rigides. Le NoSQL offre une flexibilité considérable en termes de modèles de données, de schémas et de distribution. Selon le type de NoSQL, les données peuvent être stockées de manière différente.

1. Document

Les bases de données de type document, tel que MongoDB, stocke des données sous forme de documents JSON. Ces documents ont une structure flexible qui permet de stocker des documents complexes de manière centralisée et peuvent varier d'un document à l'autre dans une même collection.

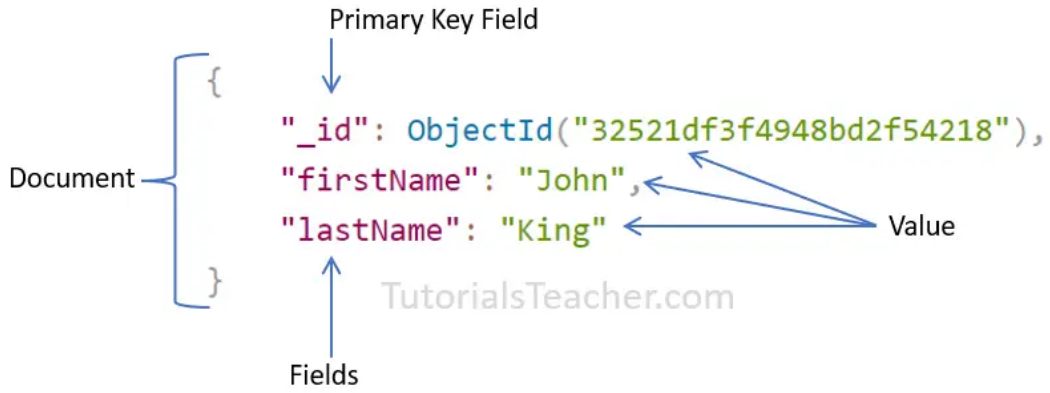


schéma logo

2. Clé-valeur

Les bases de données de type **clé-valeur**, telles que **Redis**, sont similaire à une table à 2 colonnes avec des clés primaires qui stockent des valeurs sans schéma prédéfini.

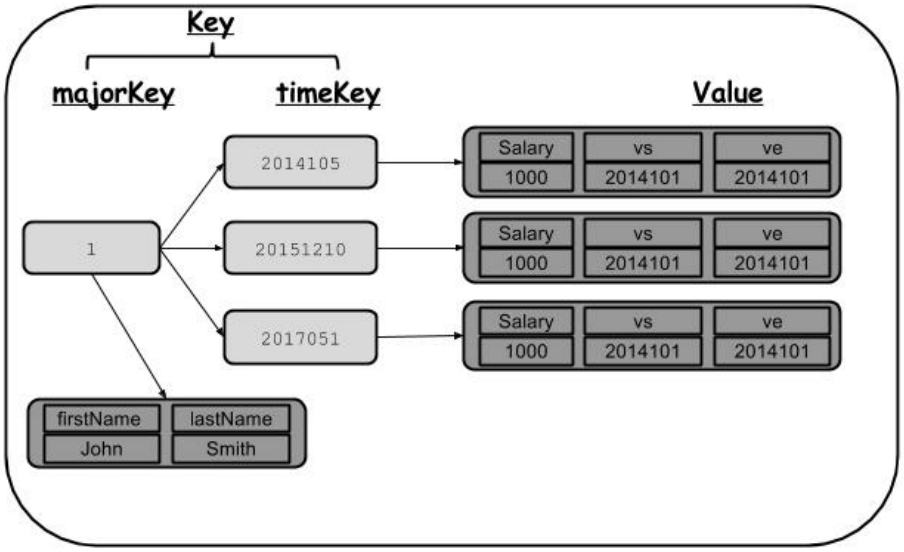


schéma logo

3. Colonnes larges

Les bases de données de type **colonnes larges**, telles que **Cassandra**, sont similaire aux tables relationnelles simples et organisent les données en colonnes plutôt qu'en lignes, ce qui permet une grande flexibilité dans la manière dont les données sont stockées et récupérées et donc elles ne possèdent ni clé étrangère, ni contrainte d'intégrité et on n'a pas besoin de faire de jointure.

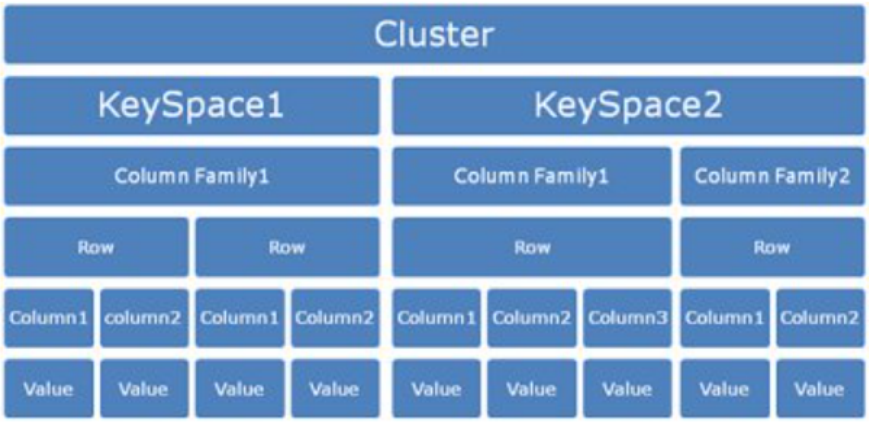
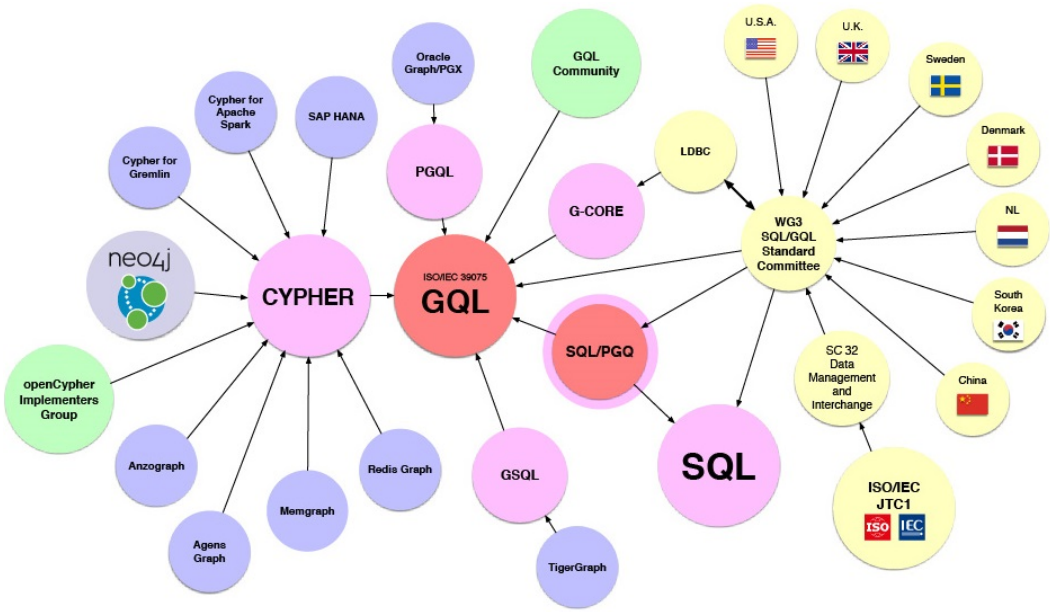


schéma logo

4. Graphes

Les bases de données de type graphes, telles que Neo4j, stockent des données sous forme de nœuds et des arcs, ce qui est particulièrement adapté pour modéliser des réseaux complexes de données interconnectées et facilite les requêtes de parcours de graphes.



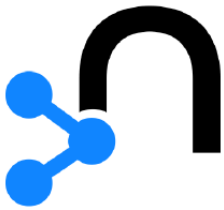
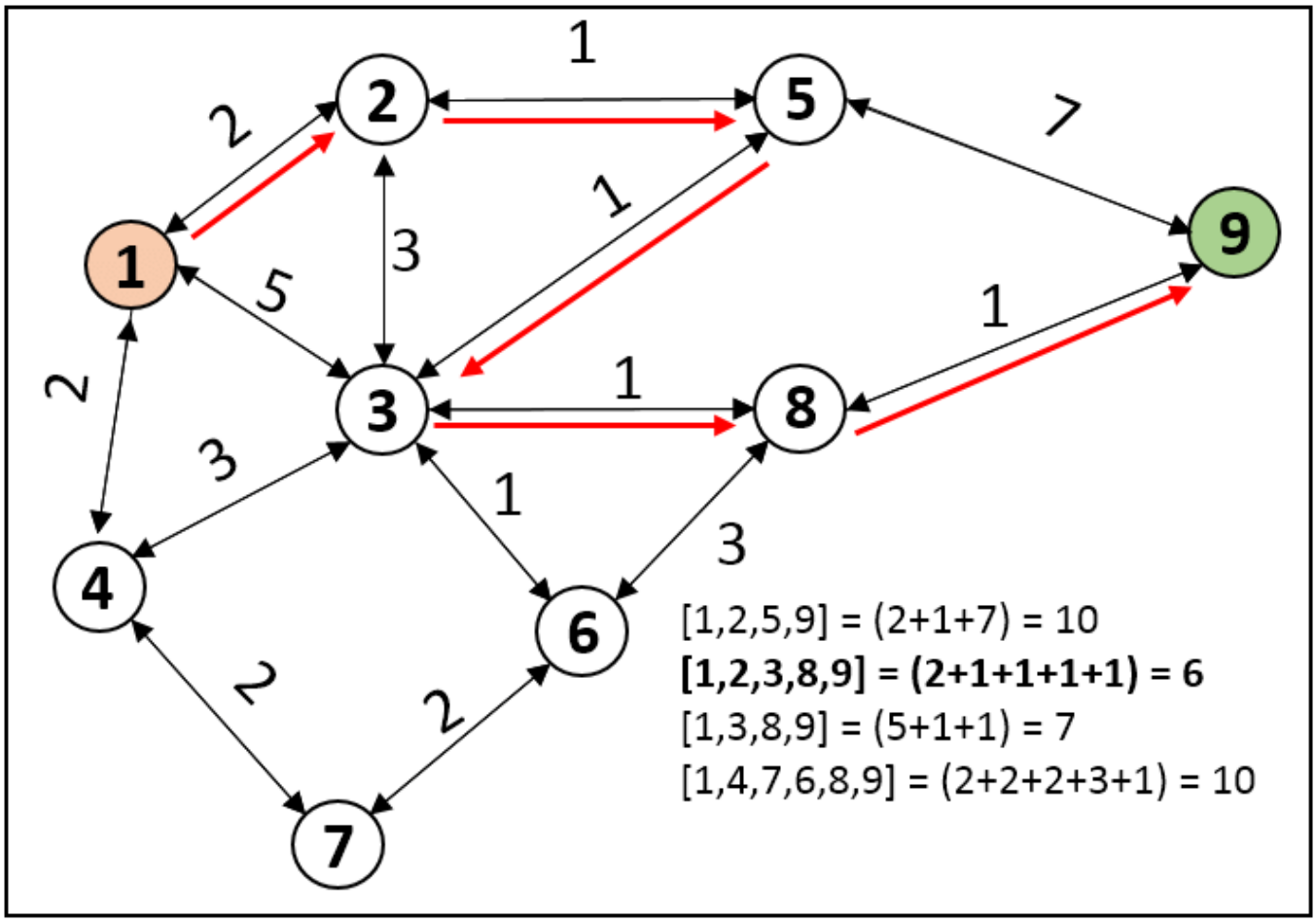


schéma logo

C. Requêtes spécifiques au modèle de données dans NoSQL

Contrairement au relationnel qui utilise le SQL pour ces requêtes qui stocke uniquement des données structurées, les requêtes des bases de données non-relationnel sont spécifiques aux modèles de données utilisés, elles peuvent stocker des données structurées, mais moins efficacement que le relationnel. Cependant, elles permettent de stocker des données semi-structurées comme du json, csv et non structurées comme des documents texte... Qui est impossible pour le relationnel.

Par exemple, les bases de données de **type document** utilisent souvent des **requêtes JSON**, format qui facilite le développement d'applications et permet une intégration transparente avec de nombreux langages de programmation, pour accéder et manipuler les données, tandis que les bases de données de type graphes offrent des requêtes optimisées pour la navigation dans les structures de graphes comme pour la robotique avec l'algorithme de Dijkstra.

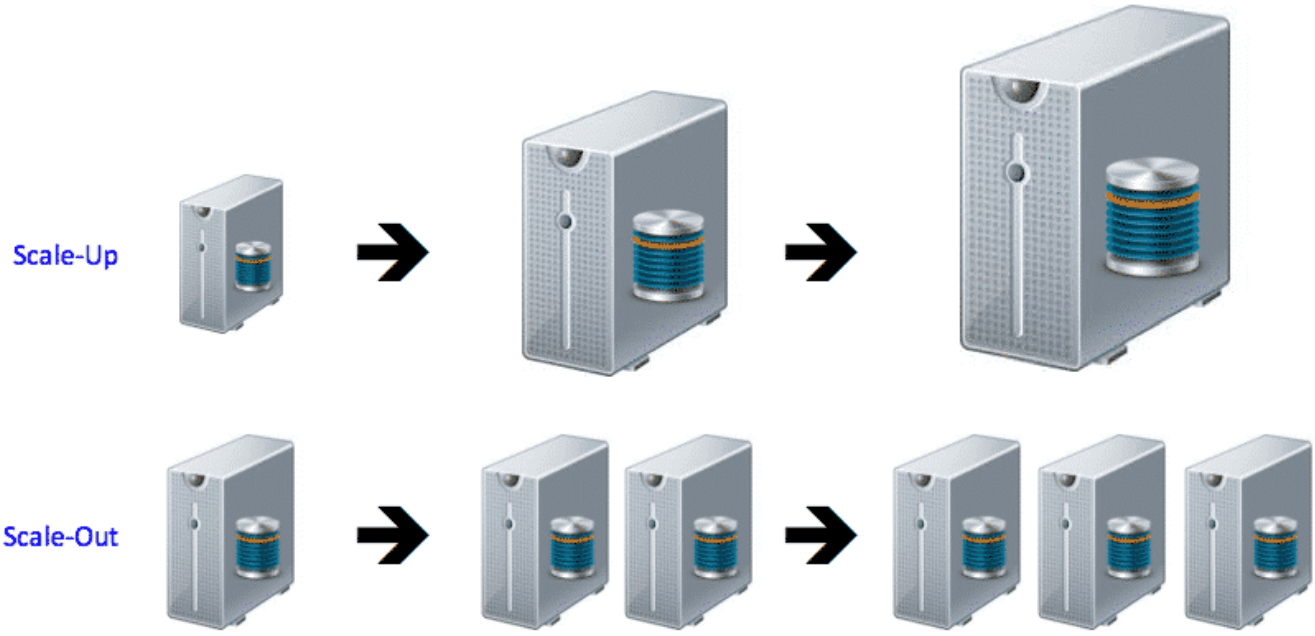


Ces différences de structure et façon de requêter la base de données offrent une plus grande flexibilité en termes de types de données qu'elles peuvent stocker et les performances pour récupérer les données.

Pour résumé, les bases de données SQL sont limitées aux données structurées avec des schémas rigides, tandis que les bases de données NoSQL offrent une plus grande souplesse en permettant le stockage de données diverses, ce qui les rend adaptées à une plus grande variété de cas d'utilisation, dont une plus grande variété d'importation de type de données.

D. Évolutivité vertical vs horizontale

L'évolutivité des bases de données peut être abordée de deux manières principales : **verticale** (scale-up) et **horizontale** (scale-out).



Chacune présente ses propres avantages et inconvénients en fonction des besoins et des contraintes spécifiques d'une application.

Évolutivité verticale

L'évolutivité verticale, consiste à augmenter la capacité d'une base de données en améliorant les ressources matérielles d'un seul serveur, telles que le processeur, la mémoire et le stockage. Cette approche est souvent utilisée lorsque les besoins de performance et de capacité de la base de données peuvent être satisfaits en ajoutant simplement plus de puissance à un seul serveur.

- Avantages :
  - Facilité de mise en œuvre et de gestion, car elle ne nécessite pas de modifications majeures de l'architecture de la base de données.
  - Peut-être moins coûteuse à court terme, car elle ne nécessite pas l'achat de matériel supplémentaire.
- Inconvénients :
  - Les limites matérielles peuvent être atteintes rapidement, entraînant des goulets d'étranglement en termes de performances et de capacité.
  - Les coûts peuvent devenir prohibitifs à mesure que les besoins de capacité augmentent, en raison des coûts élevés associés à l'acquisition et à la maintenance de matériel plus puissant.

Évolutivité horizontale

L'évolutivité horizontale consiste à répartir les données et la charge de travail sur plusieurs serveurs, formant ainsi un cluster ou un réseau de serveurs. Cette approche permet d'ajouter simplement de nouveaux serveurs au cluster pour augmenter la capacité et la résilience du système.

- Avantages :
    - Offre une évolutivité pratiquement illimitée en ajoutant simplement de nouveaux serveurs au cluster.
    - Améliore la résilience du système en répartissant la charge de travail et en répliquant les données sur plusieurs nœuds.
- Inconvénients :
    - Plus complexe à mettre en œuvre et à gérer, car elle nécessite la configuration et la synchronisation de plusieurs serveurs.
    - Peut entraîner une latence accrue dans certaines opérations, car les données doivent parfois être récupérées sur plusieurs nœuds.

En résumé, l'évolutivité verticale est adaptée lorsque les besoins de capacité peuvent être satisfaits en augmentant les ressources d'un seul serveur, tandis que l'évolutivité horizontale est préférable lorsque les besoins de capacité sont susceptibles de croître de manière exponentielle et nécessitent une approche distribuée pour maintenir les performances et la disponibilité du système, mais ceci est plus complexe à maintenir.

III. Les concepts clés du NoSQL

A. Introduction

Dans le domaine de la gestion des données, les bases de données NoSQL ont émergé comme une alternative novatrice aux bases de données relationnelles traditionnelles. Partant du concept BASE vu plus tôt, leur approche **distribuée** les rend particulièrement adaptées à la **gestion de gros volumes de données**, ce qui en fait des acteurs clé dans le domaine du **Big Data**. Cette distribution permet de répartir la charge de travail et d'assurer une meilleure **tolérance aux pannes**. Même en cas de défaillance d'un serveur ou d'une connexion réseau, les données restent accessibles et intègres.

De plus, les bases de données NoSQL offrent une capacité de **stockage flexible et extensible** avec les 4 types de stockages différents vu plus tôt, ce qui les rend idéales pour gérer des volumes de données en diverses et en constante augmentation. Leur architecture permet également d'exécuter efficacement des **opérations de lecture et d'écriture concurrente**, ce qui est essentiel dans les environnements où de nombreuses requêtes sont traitées simultanément.

Enfin, les bases de données NoSQL sont conçues pour offrir une cohérence final (Eventual Consistency) plutôt qu'une cohérence stricte en temps réel. Cela signifie qu'après une opération d'écriture, la cohérence des données sera établie à un moment ultérieur, ce qui permet d'optimiser les performances et la disponibilité du système dans des environnements distribués à grande échelle.

Dans l'ensemble, les bases de données NoSQL jouent un rôle crucial dans le traitement et la gestion des données à grande échelle, offrant une solution flexible, robuste et adaptée aux défis du Big Data.

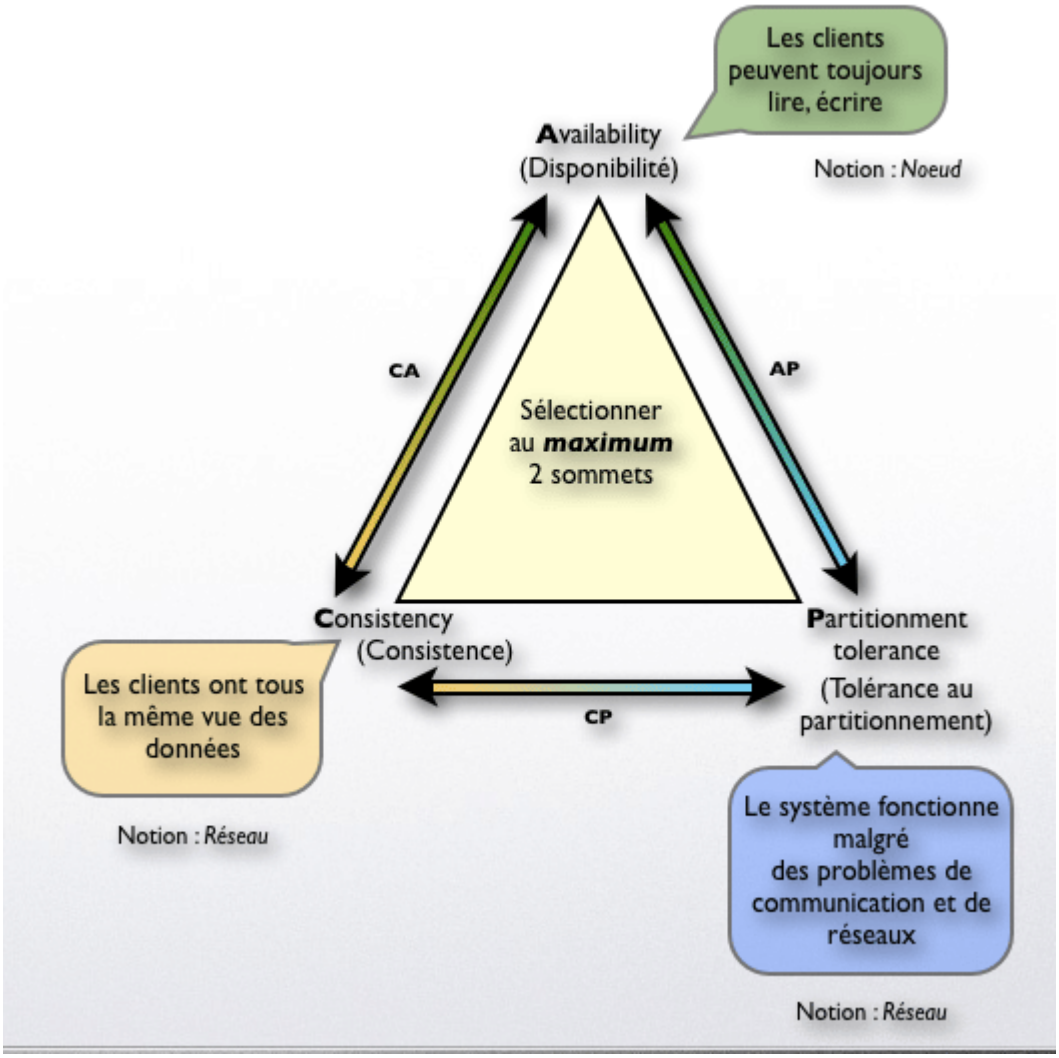
Malgré toutes ces qualités, ce système reste plus complexe à mettre en place et à maintenir.

Dans le but de tester leurs serveurs, Netflix a créé Chaos Monkey en 2011 dans le but de provoquer des pannes en environnement réel et de vérifier que le système informatique continue à fonctionner.

CHAOS MONKEY	Description

B. Théorème CAP

Concept fondamental en matière de conception de **systèmes distribués**. Il stipule que dans un système informatique distribué, **il est impossible de garantir simultanément les trois propriétés** suivantes : la **cohérence** (Consistency), la **disponibilité** (Availability) et la **tolérance aux partitions** (Partition-tolerance). En d'autres termes, dans un système distribué, vous devez choisir entre la cohérence et la disponibilité lorsque des partitions réseau se produisent.

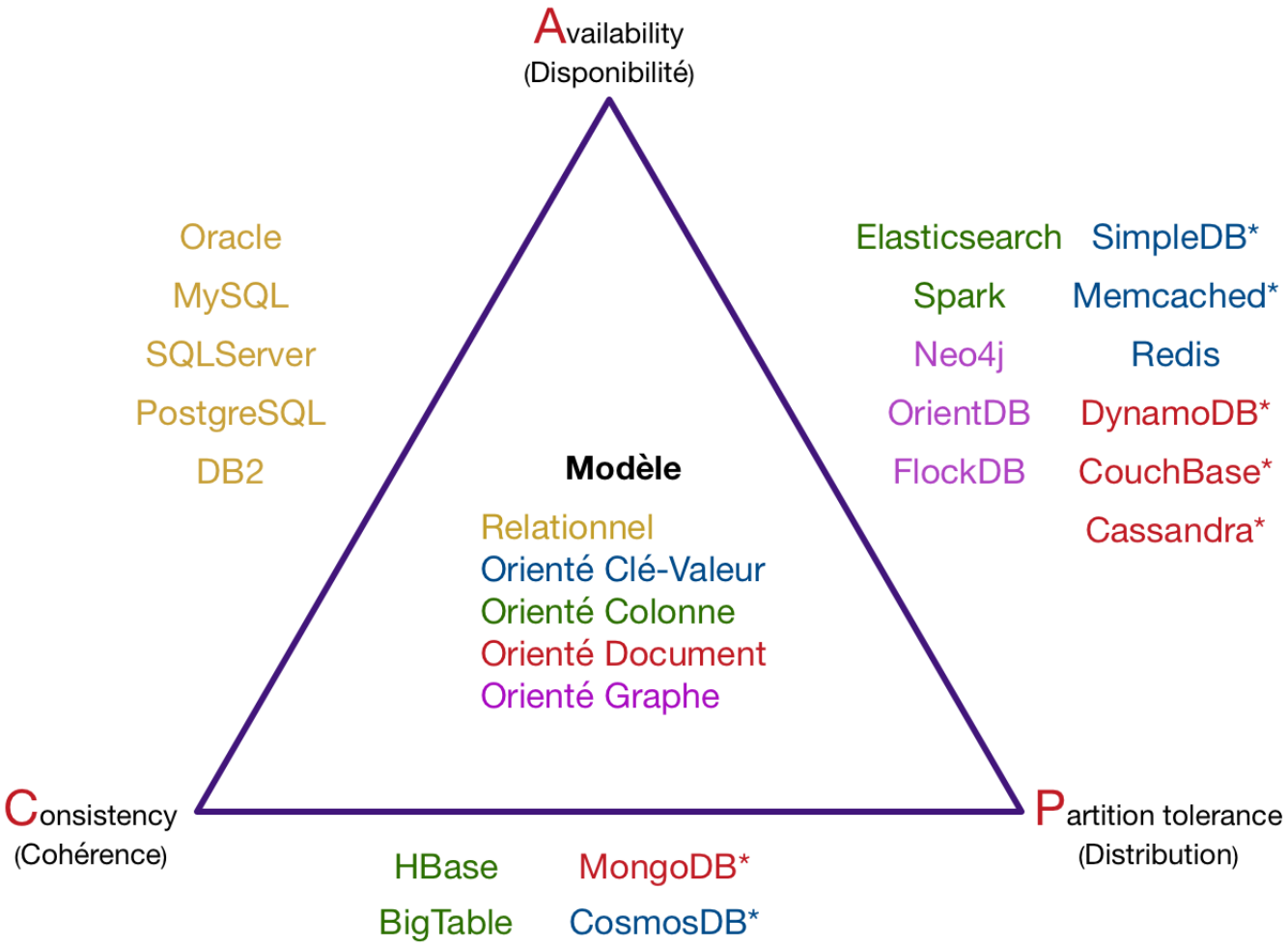


- **C : Consistency (Cohérence)**
  - Toutes les données d'un système distribué apparaissent de manière cohérente à tous les utilisateurs en même temps.
  - Dans un système qui privilégie la cohérence, les opérations de lecture et d'écriture garantissent que les données restent cohérentes en tout temps, même en cas de panne ou de partition réseau. Cela signifie que toutes les copies de données sont synchronisées et à jour.
- **A : Availability (Disponibilité)**
  - La disponibilité signifie que chaque requête reçue par un système distribué reçoit une réponse, qu'elle réussisse ou échoue. En d'autres termes, le système est toujours "disponible" pour répondre aux demandes de ses utilisateurs, même en cas de pannes ou de partitions réseau.

- Dans un système orienté vers la disponibilité, il est essentiel de répondre rapidement aux demandes des utilisateurs, même si cela signifie parfois renvoyer des données périmées ou sacrifier la cohérence des données.

- **P : Partition-tolerance (Tolérance aux partitions)**
  - La tolérance aux partitions se réfère à la capacité d'un système distribué à continuer de fonctionner malgré les partitions réseau, c'est-à-dire lorsque certaines parties du réseau ne peuvent pas communiquer avec d'autres parties. Les partitions peuvent survenir en raison de défaillances matérielles, de défaillances logicielles ou de problèmes de réseau.
  - Dans un système tolérant aux partitions, le système peut continuer à fonctionner même si certains nœuds ou segments du réseau sont inaccessibles ou ne peuvent pas communiquer entre eux.

Il est important de noter que le choix entre ces trois propriétés dépend des besoins spécifiques de votre application et des compromis que vous êtes prêt à faire en termes de cohérence, de disponibilité et de tolérance aux partitions. En fin de compte, la conception d'un système distribué implique souvent de trouver le bon équilibre entre ces différentes propriétés en fonction des besoins de l'application et de l'infrastructure sous-jacente.



> Ceux avec une * ont la capacité de changer leur pied d'appui pour passer en **CA** comme les bases de données relationnelles

C. Limites du NoSQL

Le NoSQL, malgré ses avantages en termes de scalabilité, de flexibilité et de traitement de grands volumes de données non structurées, présente également plusieurs limites.

Tout d'abord, la cohérence des données peut être compromise, en particulier dans les bases de données NoSQL où la cohérence forte n'est pas garantie, mais qui est contrebalancé par la grande quantité de données qu'il peut transiter contrairement au NoSQL, on parle de **Big Data**.

De plus, les opérations de modification des données peuvent être plus complexes dans un environnement NoSQL, car il n'y a pas de schéma fixe imposé aux données, ce qui rend parfois difficile la gestion des mises à jour et des requêtes complexes. L'absence de transactions dans de nombreuses bases de données NoSQL peut également entraîner des problèmes de race condition et de concurrence lors de l'accès et de la modification simultanée des données par plusieurs utilisateurs. En outre, bien que certains systèmes de gestion de bases de données relationnelles (SGBDR) intègrent désormais des fonctionnalités NoSQL, telle que le stockage et la manipulation de données au format JSON, ces solutions hybrides peuvent présenter des compromis en termes de performances et de fonctionnalités par rapport aux bases de données NoSQL dédiées.

IV. Exemples de typologie de projet

A. Choisir entre Relationnelle ou NoSQL

Le choix entre une base de données relationnelles et NoSQL dépend de facteurs tels que les besoins de l'application, l'échelle des données et des opérations, et la tolérance à la perte de cohérence. Il est important d'évaluer la structure des données, les performances, la scalabilité, les coûts et la disponibilité des compétences pour prendre la meilleure décision. Les bases de données relationnelles conviennent mieux aux applications avec des structures de données complexes et nécessitantes une cohérence stricte, tandis que les bases de données NoSQL sont plus adaptées aux environnements nécessitant une grande scalabilité horizontale et une flexibilité dans la modélisation des données.

B. Cas d'utilisations

- Applications web et mobiles : MongoDB est souvent utilisé dans le développement d'applications web et mobiles, notamment dans les domaines du commerce électronique, des médias sociaux, des jeux en ligne, des applications de contenu dynamique, etc. Sa flexibilité et sa scalabilité en font un bon choix pour gérer des charges de travail évolutives avec des structures de données variées. Exemple : stack **MERN** (MongoDB, ExpressJS, ReactJS et NodeJS) ou **MEAN** (AngularJS à la place de ReactJS), ou encore la stack **FARM** (FastAPI, React, and MongoDB)

- Catalogues de produits et de contenu : les systèmes de gestion de catalogues, tels que les catalogues de produits en ligne ou les systèmes de gestion de contenu (CMS), peuvent bénéficier de la flexibilité de MongoDB pour gérer des données non structurées, des hiérarchies complexes et des métadonnées associées.

- Analyses en temps réel : MongoDB est utilisé pour collecter et analyser des données en temps réel, notamment dans les domaines du suivi des activités utilisateur, de l'analyse des journaux, de la surveillance des performances, etc. Sa capacité à gérer des volumes importants de données avec des opérations de lecture et d'écriture rapides en fait un choix attrayant pour les applications nécessitant des analyses en temps réel.

- Gestion de contenu géospatiale : les données géospatiales, telles que les coordonnées GPS, les emplacements géographiques, les informations de cartographie, peuvent être efficacement stockées et requêtées avec les fonctionnalités géospatiales intégrées de MongoDB. Cela en fait un choix populaire pour les applications nécessitant des fonctionnalités de recherche ou d'analyse basée sur la géolocalisation.