Qualité et au-delà du relationnel

S4 - R4.03 - 2022/2023

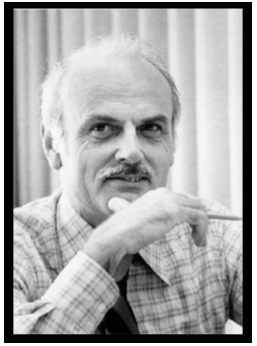
Plan de cette présentation

I. Introduction
   A. Problèmes avec les systèmes de gestion de bases de données relationnelles
   B. Architecture à serveur unique VS base de données distribuée
   C. NoSQL et changement de paradigme (paradigm shift)
II. Qu'est-ce que NoSQL ?
   A. Historique des bases de données NoSQL
   B. Relation entre Big Data et NoSQL
   C. Relation entre la mise à l'échelle horizontale et NoSQL
III. Classement de bases de données NoSQL
   A. Classement par usage
   B. Classement par schéma de données
IV. NoSQL et cohérence
   A. Le théorème CAP

Introduction

SGBDR

- En 1970, Edgar F. Codd (IBM) publie un article considéré comme à l'origine du modèle relationnel.
- Le modèle relationnel est régi par un ensemble de règles précises énoncées par Codd telles que :
  - séparation logique et physique,
  - langage déclaratif,
  - structuration forte des données,
  - représentation tabulaire,
  - cohérence transactionnelle forte.



Introduction

SGBDR

Les bases relationnelles sont bien adaptées au stockage et à la manipulation d'informations bien structurées, décomposables en unités simples et représentables sous forme de tableaux.

Dans les années 80-90, de nouveaux modèles sont apparus :

► Bases objet et objet-relationnel,

► Base de données dites semi-structurées (XML)

Cependant ... ces modèles n'ont pas connu un franc succès auprès des acteurs de l'informatique, et le modèle relationnel est resté très majoritairement dominant.

Les bases de données relationnelles font la preuve de leur efficacité depuis longtemps, et encore aujourd'hui !

Ranking > Complete Ranking

RSS RSS Feed

DB-Engines Ranking

The DB-Engines Ranking ranks database management systems according to their popularity. The ranking is updated monthly.

Read more about the method of calculating the scores.

trend chart

410 systems in ranking, February 2023

Rank Feb 2023	Rank Jan 2023	Rank Feb 2022	DBMS	Database Model	Score Feb 2023	Score Jan 2023	Score Feb 2022
1.	1.	1.	Oracle	Relational, Multi-model	1247.52	+2.35	-9.31
2.	2.	2.	MySQL	Relational, Multi-model	1195.45	-16.51	-19.23
3.	3.	3.	Microsoft SQL Server	Relational, Multi-model	929.09	+9.70	-19.96
4.	4.	4.	PostgreSQL	Relational, Multi-model	616.50	+1.65	+7.12
5.	5.	5.	MongoDB	Document, Multi-model	452.77	-2.42	-35.88
6.	6.	6.	Redis	Key-value, Multi-model	173.83	-3.72	-1.96
7.	7.	7.	IBM Db2	Relational, Multi-model	142.97	-0.60	-19.91
8.	8.	8.	Elasticsearch	Search engine, Multi-model	138.60	-2.56	-23.70
9.	↑ 10.	↑ 10.	SQLite	Relational	132.67	+1.17	+4.30
10.	↓ 9.	↓ 9.	Microsoft Access	Relational	131.03	-2.33	-0.23
11.	↑ 12.	11.	Cassandra	Wide column	116.22	-0.09	-7.76
12.	↓ 11.	↑ 15.	Snowflake	Relational	115.65	-1.60	+32.47
13.	13.	↓ 12.	MariaDB	Relational, Multi-model	96.81	-2.55	-10.30
14.	14.	↓ 13.	Splunk	Search engine	87.08	-1.32	-3.73
15.	15.	↑ 17.	Amazon DynamoDB	Multi-model	79.69	-1.87	-0.67
16.	16.	↓ 14.	Microsoft Azure SQL Database	Relational, Multi-model	78.75	-1.62	-6.20
17.	17.	↓ 16.	Hive	Relational	72.12	-2.22	-9.76
18.	18.	18.	Teradata	Relational, Multi-model	63.03	-2.40	-5.54
19.	19.		Databricks	Multi-model	60.33	-0.49	
20.	20.	20.	Neo4j	Graph	55.43	-0.41	-2.81
21.	↑ 22.	↑ 22.	FileMaker	Relational	52.80	-0.28	-1.33
22.	↓ 21.	↑ 24.	Google BigQuery	Relational	52.45	-1.97	+7.35
23.	23.	↓ 21.	SAP HANA	Relational, Multi-model	49.67	-1.67	-6.64
24.	24.	↓ 19.	Solr	Search engine, Multi-model	45.88	-0.51	-12.64
25.	25.	↓ 23.	SAP Adaptive Server	Relational, Multi-model	43.05	+0.21	-6.48

Introduction

Problèmes avec les SGBDR

- Nous devrions connaître l'intégralité du schéma à l'avance.
  - **Inflexibilité :** dans un SGBDR traditionnel, le schéma est fixe, ce qui signifie qu'une fois défini, il est difficile de le modifier. Cela peut être un problème dans les systèmes en évolution rapide où le modèle de données doit être mis à jour fréquemment.
  - **Manque d'agilité :** étant donné que le schéma est fixe, l'ajout de nouveaux champs ou la modification de champs existants peut prendre du temps et nécessiter des modifications importantes de la base de données. Cela peut ralentir le processus de développement et réduire l'agilité du système.
  - **Complexité accrue :** la définition d'un schéma nécessite beaucoup d'efforts et peut être complexe, en particulier dans les systèmes vastes et complexes. Cela peut rendre difficile le démarrage d'une nouvelle base de données et peut également augmenter la complexité du système dans son ensemble.

Introduction

Problèmes avec les SGBDR

- Nous devrions connaître l'intégralité du schéma à l'avance.
- Chaque enregistrement doit avoir les mêmes propriétés.
  - **Capacité réduite à accueillir des données non structurées :** les systèmes RDBMS sont conçus pour fonctionner avec des données structurées, et il peut être difficile de stocker et de récupérer des données non structurées dans ces systèmes. Cela peut limiter les types de données pouvant être stockées et traitées dans le système.
  - Structure rigide.

Introduction

Problèmes avec les SGBDR

- Nous devrions connaître l'intégralité du schéma à l'avance.
- Chaque enregistrement doit avoir les mêmes propriétés.
- L'évolutivité (scalability) coûte cher.
    - **Distribution des données :** dans un système distribué, les données sont généralement stockées sur plusieurs nœuds, qui peuvent être situés dans différentes parties du monde. La jointure de tables nécessite que les données des deux tables soient récupérées et combinées sur un seul nœud, ce qui peut être lent et gourmand en ressources, en particulier si les tables sont volumineuses et que les données sont réparties sur de nombreux nœuds.
    - **Latence du réseau :** la récupération de données à partir de plusieurs nœuds sur un réseau peut introduire une latence, ce qui peut ralentir l'opération de jointure. Plus la taille des données est grande et plus les nœuds sont éloignés, plus la latence sera importante.
    - **Augmentation de la charge de traitement :** la jointure de tables dans un système distribué nécessite une charge de traitement importante, car les données doivent être combinées et triées avant que la jointure puisse être effectuée. Cela peut être lent et gourmand en ressources, en particulier dans les systèmes volumineux et complexes.
    - **Absence d'indexation :** dans un système distribué, il peut être difficile de créer et de gérer des index pouvant être utilisés pour accélérer les opérations de jointure. Cela peut ralentir les performances de jointure, en particulier dans les systèmes volumineux et complexes.

Introduction

Problèmes avec les SGBDR

- Nous devrions connaître l'intégralité du schéma à l'avance.
- Chaque enregistrement doit avoir les mêmes propriétés.
- L'évolutivité (scalability) coûte cher.
- Normalisation.
  - **Augmentation du nombre de jointures :** la normalisation dans un SGBDR entraîne souvent une augmentation du nombre de jointures entre les tables, ce qui peut ralentir les performances des requêtes et augmenter la complexité du système. Dans un système distribué, les jointures peuvent être particulièrement lentes et gourmandes en ressources, car les données de plusieurs nœuds doivent être récupérées et combinées.
  - **Augmentation des E/S de disque :** la normalisation dans un SGBDR entraîne souvent une augmentation du nombre d'opérations d'E/S de disque, car les données sont réparties sur plusieurs tables. Cela peut ralentir les performances des requêtes et augmenter les besoins en ressources du système.

Introduction

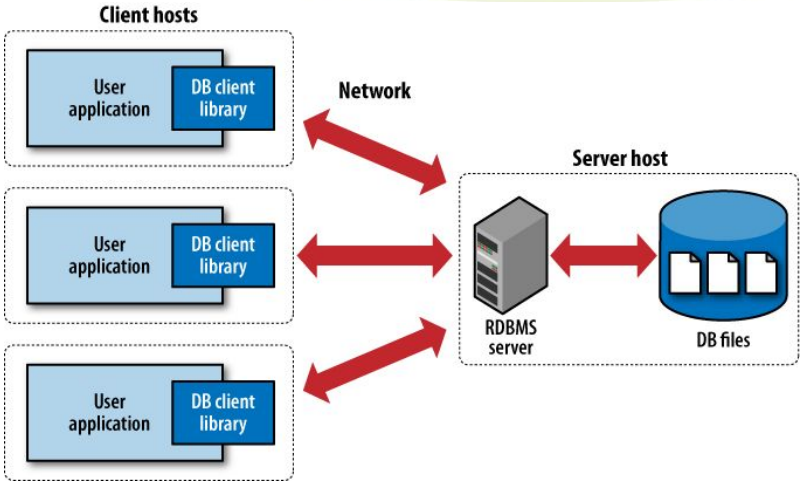
Problèmes avec les SGBDR

- Nous devrions connaître l'intégralité du schéma à l'avance.
- Chaque enregistrement doit avoir les mêmes propriétés.
- L'évolutivité (scalability) coûte cher.
- Normalisation.
- SQL a été conçu pour fonctionner sur un seul système de serveur.
  - SQL a été conçu comme un langage pour les SGBDR, qui stockent les données de manière structurée à l'aide de tables, de lignes et de colonnes. Cette structure facilitait le travail avec les données à l'aide de requêtes SQL, mais rendait également **plus difficile la mise à l'échelle horizontale** du système sur plusieurs serveurs.
  - Les premiers systèmes SGBDR étaient conçus pour fonctionner sur un seul serveur, et l'accent était mis sur l'optimisation des performances pour le matériel disponible. C'était suffisant pour de nombreuses applications à l'époque, mais à mesure que les volumes de données augmentaient et que les capacités de calcul et de mise en réseau s'amélioraient, les limites d'une architecture à serveur unique sont devenues plus apparentes et le besoin de systèmes de bases de données plus évolutifs est apparu.

Introduction

Architecture à serveur unique (Single server architecture)

- Une architecture à serveur unique fait référence à un système dans lequel tous les composants nécessaires à la gestion d'une base de données, y compris le logiciel de gestion de base de données, la base de données elle-même et le serveur d'applications, s'exécutent sur une seule machine.
- Dans une architecture à serveur unique, la base de données est stockée localement sur la même machine que le logiciel de gestion de base de données, et toutes les demandes d'accès ou de modification des données de la base de données sont traitées par cette machine.

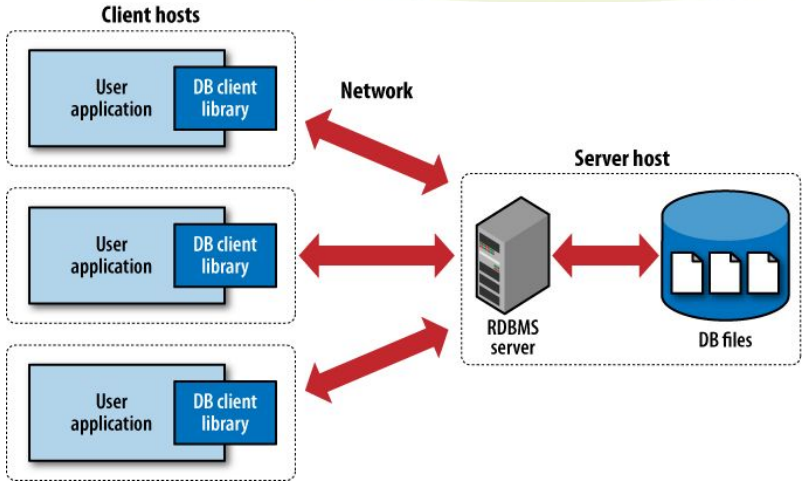


Introduction

Architecture à serveur unique (Single server architecture)

Limites de cette architecture :

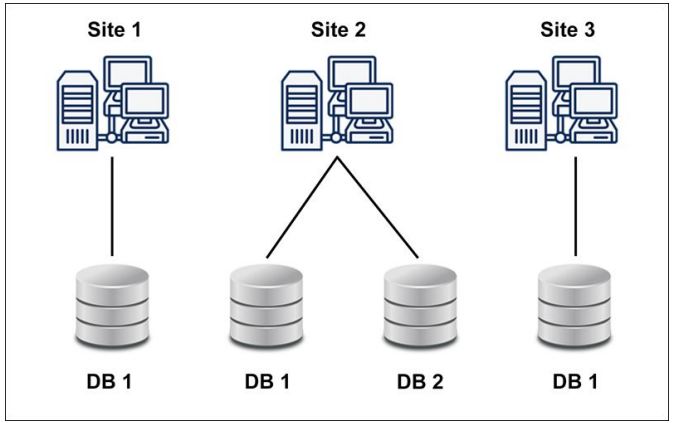
- Volumes de données.
  - ⇒ la demande pour plus de puissance de traitement a augmenté
- Un seul point de défaillance.
  - le logiciel de gestion de base de données et la base de données dépendent fortement des performances de la machine unique, et si cette machine tombe en panne ou devient indisponible, l'ensemble du système de base de données devient également indisponible.
- Maintenance - temps d'arrêt pendant la maintenance manuelle.



Introduction

Bases de données distribuées

- La base de données distribuée est un système dans lequel la base de données est stockée et gérée sur plusieurs machines, plutôt que sur une seule machine comme dans une architecture à serveur unique.
- Dans un système de base de données distribué, les données sont réparties sur plusieurs serveurs et le logiciel de gestion de base de données s'exécute sur chaque serveur, ce qui permet un traitement parallèle des requêtes et améliore les performances globales et l'évolutivité du système.
- En distribuant la base de données sur plusieurs machines, une base de données distribuée peut fournir une puissance de traitement accrue, une fiabilité améliorée et une meilleure tolérance aux pannes, car la défaillance d'une machine n'affecte pas l'ensemble du système.
- Vitesse - les fichiers sont récupérés à partir de l'emplacement le plus proche.



Introduction

Bases de données distribuées

Il existe plusieurs SGBDR qui peuvent être distribués, notamment :

- **MySQL :** MySQL est un SGBDR open source populaire qui peut être distribué sur plusieurs machines. MySQL Cluster Edition est conçu pour les applications à grande échelle et à haute disponibilité et prend en charge les bases de données distribuées.
- **Oracle :** Oracle est un SGBDR commercial qui prend en charge les bases de données distribuées et fournit des fonctionnalités telles que la réplication de données en temps réel, l'équilibrage de charge et le basculement automatique (automatic failover).
- **Microsoft SQL Server :** Microsoft SQL Server est un SGBDR commercial qui prend en charge les bases de données distribuées via ses fonctionnalités de clustering de basculement SQL Server et de groupes de disponibilité SQL Server Always On.
- **PostgreSQL :** PostgreSQL est un SGBDR open source populaire qui prend en charge les bases de données distribuées grâce à ses fonctionnalités intégrées de réplication et de partitionnement.
- **IBM Db2 :** IBM Db2 est un SGBDR commercial qui prend en charge les bases de données distribuées via sa fonction IBM Db2 pureScale, qui fournit un basculement automatique et un équilibrage de charge pour les applications à grande échelle.

MySQL Cluster

ORACLE DATABASE

Postgres-XL

Introduction

Bases de données distribuées

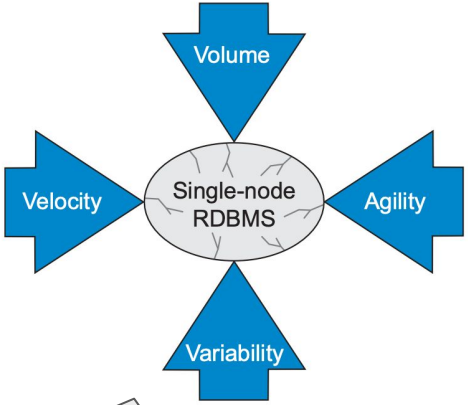
Au fur et à mesure que les grandes propriétés se sont déplacées vers les clusters, cela a révélé un nouveau problème : les bases de données relationnelles ne sont pas conçues pour être exécutées sur des clusters. Les bases de données relationnelles en cluster, telles qu'Oracle RAC ou Microsoft SQL Server, fonctionnent sur le concept d'un sous-système de disque partagé. Ils utilisent un système de fichiers compatible avec les clusters qui écrit sur un sous-système de disque hautement disponible, mais cela signifie que le cluster a toujours le sous-système de disque comme point de défaillance unique. Les bases de données relationnelles peuvent également être exécutées en tant que serveurs distincts pour différents ensembles de données, divisant efficacement la base de données. Bien que cela sépare la charge, tout le sharding doit être contrôlé par l'application qui doit garder une trace du serveur de base de données auquel parler pour chaque bit de données. De plus, nous perdons toutes les requêtes, l'intégrité référentielle, les transactions ou les contrôles de cohérence qui traversent les fragments. Une expression que nous entendons souvent dans ce contexte de la part de personnes qui ont fait cela est « actes contre nature ».

– NoSQL Distilled, Pramod Sadalage, Martin Fowler

Introduction

NoSQL et changement de paradigme (paradigm shift)

- Le scientifique-philosophe Thomas Kuhn a inventé le terme “**paradigm shift**” pour identifier un processus récurrent qu'il a observé dans la science, où des idées innovantes sont apparues en rafales et ont eu un impact sur le monde de manière non linéaire. Nous utiliserons le concept de changement de paradigme de Kuhn pour réfléchir et expliquer le mouvement NoSQL et les changements dans les modes de pensée, les architectures et les méthodes qui émergent aujourd'hui.
- De nombreuses organisations prenant en charge des systèmes relationnels à processeur unique sont arrivées à la croisée des chemins : **les besoins de leurs organisations évoluent**. Les entreprises ont trouvé de la valeur dans la capture et l'analyse rapides de **grandes quantités de données variables**, et dans la réalisation de changements immédiats dans leurs activités en fonction des informations qu'elles reçoivent.



Dans cette figure, nous voyons comment le volume, la vitesse, la variabilité et l'agilité des moteurs commerciaux exercent une pression sur le système à processeur unique, ce qui entraîne des fissures.

Introduction

NoSQL et changement de paradigme (paradigm shift)

L'émergence du "Big Data"

Le Big Data fait référence à des ensembles de données extrêmement **volumineux** qui peuvent être structurés, semi-structurés ou non structurés et qui ne peuvent pas être traités à l'aide de systèmes de traitement de données traditionnels.

Le Gartner définit le Big data comme **un phénomène caractérisé par une explosion des données**, données qui peuvent contenir des opportunités, non pas à l'aide d'approches technologiques traditionnelles, mais à l'aide d'approches technologiques innovantes.

Le **volume**, la **variété** et la **vitesse** des mégadonnées nécessitent de nouvelles technologies et approches pour pouvoir les stocker, les traiter et les analyser.

Sans aucun doute, le facteur clé qui pousse les organisations à rechercher des alternatives à leurs SGBDR actuels est le besoin d'interroger le Big Data à l'aide de clusters de processeurs.

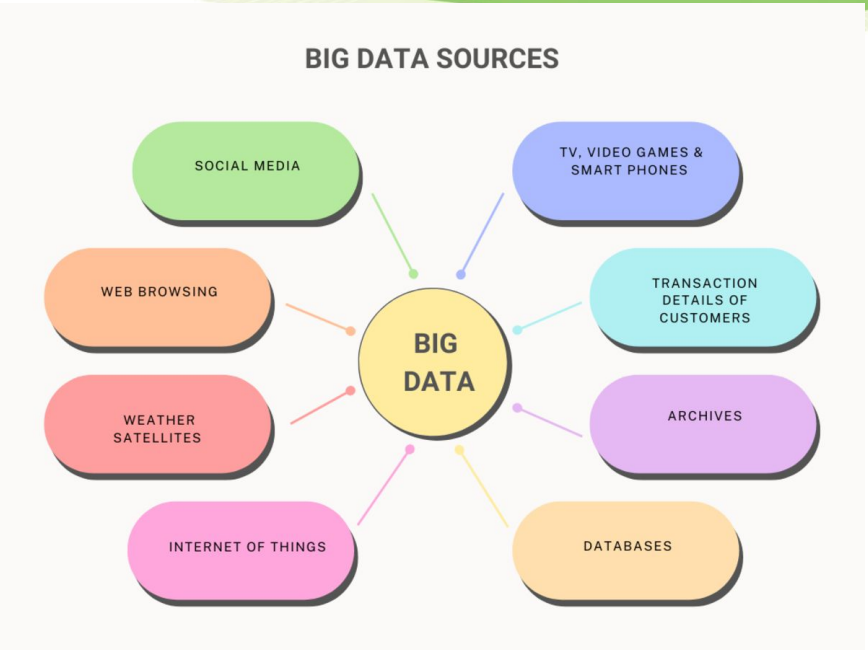
Introduction

NoSQL et changement de paradigme (paradigm shift)

L'émergence du "Big Data"

Le Big Data est devenu de plus en plus important ces dernières années en raison de la prolifération des entreprises axées sur les données et de la quantité croissante de données générées par les organisations et les individus.

Ces données peuvent être générées à partir de diverses sources, telles que les réseaux sociaux, les journaux Web, les capteurs et les appareils mobiles.



Introduction

NoSQL et changement de paradigme (paradigm shift)

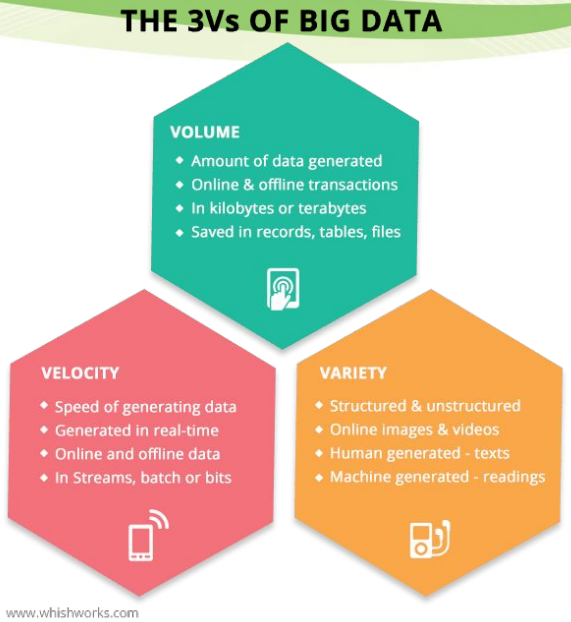
L'émergence du "Big Data"

Vitesse

La capacité d'un système à processeur unique à lire et écrire rapidement des données est essentielle. De nombreux SGBDR à processeur unique ne sont pas en mesure de répondre aux requêtes en temps réel.

Variabilité

Par exemple, si une unité commerciale souhaite capturer quelques champs personnalisés pour un client particulier, toutes les lignes client de la base de données doivent stocker ces informations même si elles ne s'appliquent pas. L'ajout de nouvelles colonnes à un SGBDR nécessite l'arrêt du système et l'exécution des commandes ALTER TABLE. Lorsqu'une base de données est volumineuse, ce processus peut avoir un impact sur la disponibilité du système, coûtant du temps et de l'argent.



Introduction

NoSQL et changement de paradigme (paradigm shift)

L'émergence du "Big Data" - **La solution Google**

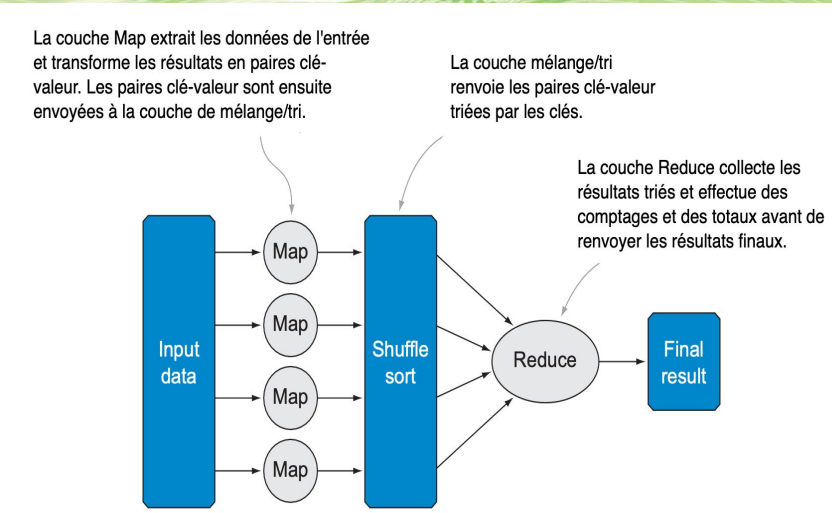
- Google est probablement la société la plus concernée par la manipulation de grands volumes de données, c'est pourquoi elle a cherché et développé une solution visant à relever ces défis.
- Google ne doit pas seulement gérer un volume extrêmement important de données afin d'alimenter son moteur de recherche, mais aussi ses offres différentes, comme Google Maps, YouTube, Google Groupes, etc.
- Afin de permettre le stockage de ces grands volumes de données, Google a développé il y a plus de quinze ans un système de fichiers distribué nommé **GoogleFS**, ou **GFS**, système propriétaire utilisé uniquement chez Google.

Introduction

NoSQL et changement de paradigme (paradigm shift)

L'émergence du "Big Data" - **La solution Google**

- En 2004, à l'occasion du sixième symposium OSDI (Operating System Design and Implementation) à San Francisco, Jeffrey Dean et Sanjay Ghemawat de Google présentèrent "**MapReduce : Simplified Data Processing on large Clusters**".
- Il s'agissait, en plus de stocker les données sur **GoogleFS**, de pouvoir effectuer des traitements sur ces données de façon également **distribuée** et de pouvoir en restituer les résultats. Pour ce faire, les ingénieurs de Google se sont inspirés des langages fonctionnels et en ont extrait deux primitifs, les fonctions **Map** et **Reduce**.



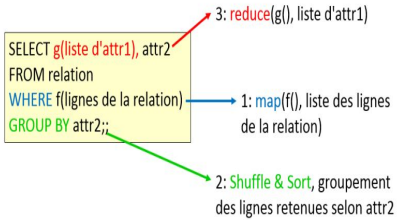
L'une des études de cas les plus influentes du mouvement NoSQL est le système Google MapReduce. Google a partagé son processus de transformation de gros volumes de contenu de données Web en index de recherche à l'aide de processeurs de base à faible coût.

Introduction

NoSQL et changement de paradigme (paradigm shift)

L'émergence du "Big Data" - **La solution Google**

- Dans une BdD relationnelle, une structure de requête classique est de la forme SELECT-FROM-WHERE-GROUPBY. L' application de la condition du Where (1) revient à appliquer sur chaque ligne de la relation la fonction de test booléenne **f(...).** Puis l'exécution du Group by va tout d'abord (2) trier les lignes retenues selon l'attribut **attr2** en regroupant celles présentant la même valeur pour cet attribut, et va ensuite (3) appliquer la fonction **g(...)** sur la liste des attributs **attr1** d'un ensemble de lignes de même attribut **attr2**.
- Ceci correspond à appliquer une fonction **map(f, lignes de la relation)**, puis à trier et regrouper les lignes retenues selon l'attribut **attr2**, et enfin à appliquer la fonction **reduce(g, liste d'attr1)** aux attributs **attr1** de chaque groupe de lignes.
- Une opération Map-Reduce, avec une fonction reduce appliquée par groupe de résultats de la fonction map, permet donc de refaire un "Select...From...Where...Group by...". **Sa mise en œuvre sera sans doute plus complexe que l'emploi de SQL**, mais sera aussi plus générique : **elle fonctionnera même si les éléments stockés dans la base ne sont pas de simples attributs de types scalaires comme dans une BdD relationnelle.** Par exemple, si les éléments stockés sont des données structurées complexes, comme des fichiers de données, un SGBDR ne pourrait pas être employé, alors qu'un mécanisme de Map-Reduce au dessus d'un système de fichiers distribués sera utilisable.

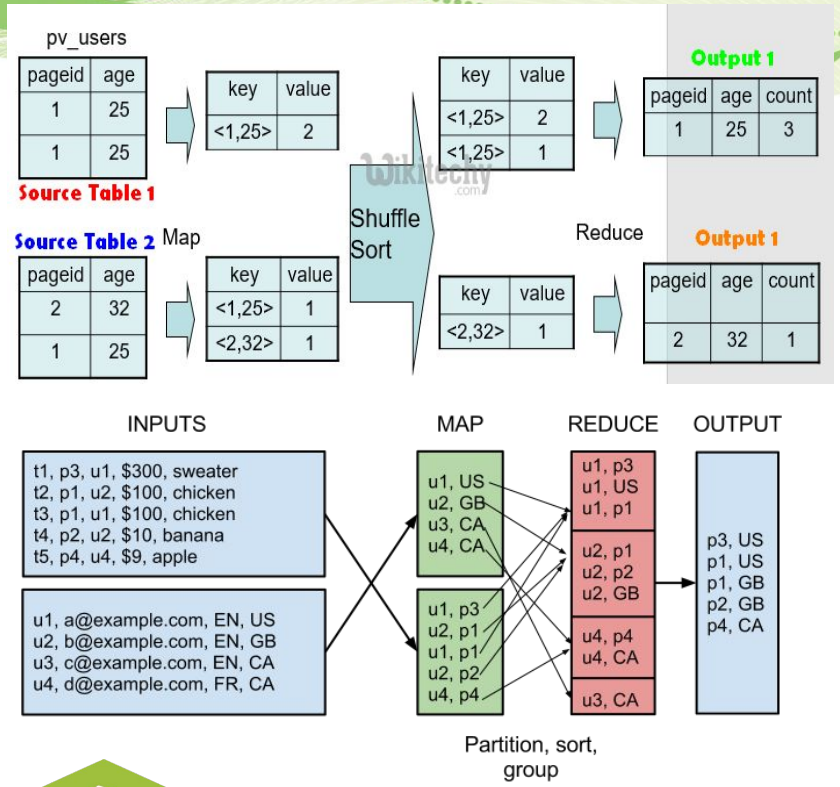


Introduction

NoSQL et changement de paradigme (paradigm shift)

L'émergence du "Big Data" - **La solution Google**

- La présentation par Google en 2003-2004 de son système de fichiers distribué et de son mécanisme de Map-Reduce distribué avait fait une forte impression, mais ces outils restaient propriété et exclusivité de Google. Inévitablement, une implantation libre vit le jour peu de temps après. Hadoop fut développé initialement par Doug Cutting pour mettre au point un moteur de recherche libre. Il repose sur le système de fichier distribué HDFS (Hadoop Distributed File System), et sur un mécanisme de Map-Reduce distribué exploitant HDFS.
- L'environnement de développement choisi fur l'écosystème extrêmement portable de Java. Hadoop a été assez rapidement adopté, et des sociétés comme Yahoo ! et Facebook annonçait déjà en 2012 avoir des dizaines de milliers de machines exploitées par Hadoop.
- Le développeur d'applications basées sur Hadoop écrit essentiellement les fonction map et reduce, sous forme de classes Java.



Qu'est-ce que NoSQL ?

Une petite note avant de continuer :

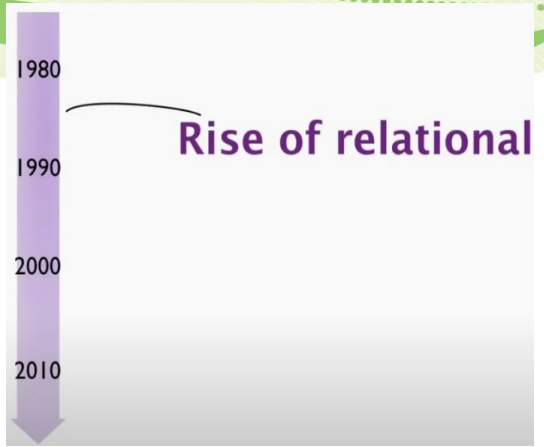
Le terme « NoSQL » a été inventé en 2009 lors d'un événement sur les bases de données partagées. Le terme est **vague**, **incorrect** (certains moteurs NoSQL utilisent des variantes du langage SQL, par exemple Cassandra), mais présente l'avantage d'avoir **un effet marketing** et polémique certain.

Qu'est-ce que NoSQL ?

Historique des bases de données NoSQL

Pour comprendre les bases de données NoSQL, nous devons savoir pourquoi elles sont là en premier lieu…

Au milieu des années 80, c'était le moment où les bases de données relationnelles arrivaient vraiment et commençaient leur ascension, il était assez difficile d'imaginer qu'il y aurait un temps sans bases de données relationnelles.



SGBDR nous a apporté de nombreux avantages:

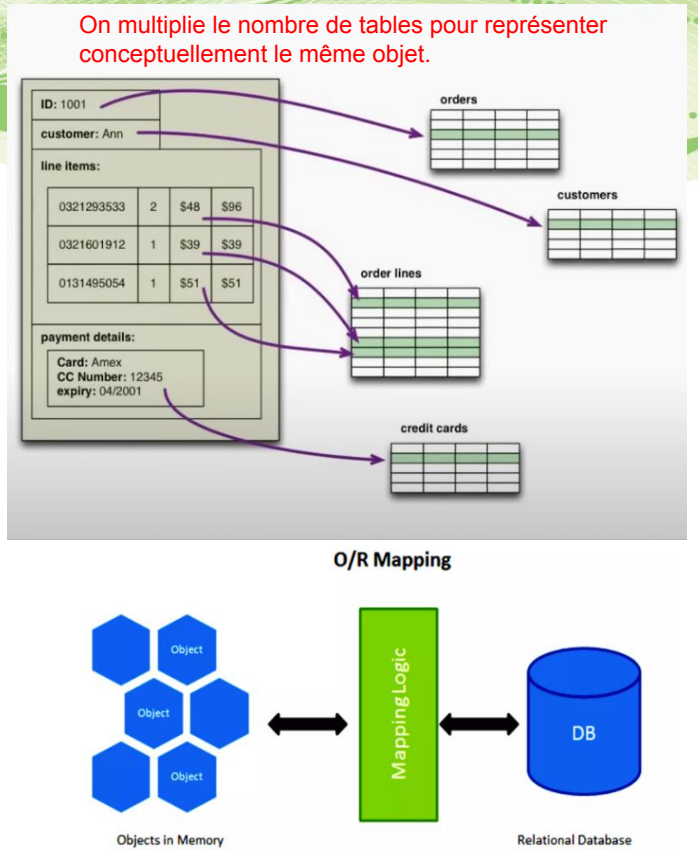
- Persistance de nos données
- Gestion de la concurrence par le biais de transactions
- SQL est devenu un langage standard de facto pour interroger ces bases de données
- Intégration et rapports

mais les bases de données relationnelles ont aussi quelques problèmes... !

Qu'est-ce que NoSQL ?

Historique des bases de données NoSQL

- L'une des rugosités du langage SQL est ce que les Anglo-Saxons appellent **le défaut d'impédance** (impedance mismatch) objet-relationnel.
- Par défaut d'impédance, les créateurs du terme veulent signifier que **le passage du relationnel à l'objet s'effectue avec une perte d'énergie et une résistance trop forte.**
- Comme on le voit sur cette figure, nous assemblons des structures d'objets en mémoire souvent en termes d'une sorte d'ensemble cohérent de choses, puis afin de l'enregistrer dans la base de données, **nous devons décomposer cette structure en parties** afin qu'elle entre dans ces lignes individuelles et ces tables individuelles.
- Pour traiter ce défaut d'impédance, des frameworks de **mapping objet-relationnel** (Object-Relational Mapping, ORM) ont été proposés.

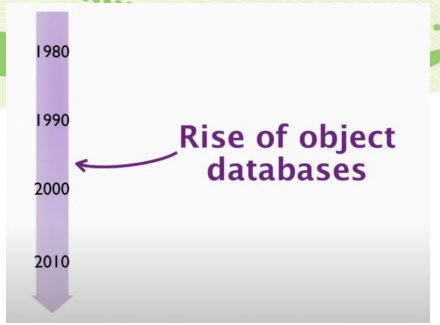


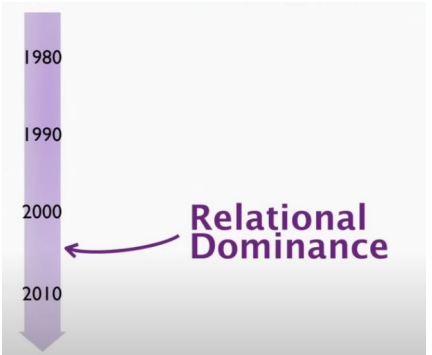
Qu'est-ce que NoSQL ?

Historique des bases de données NoSQL

Le **défaut d'impédance** est un problème suffisamment gênant pour qu'au milieu des années 90, les gens disaient que les bases de données relationnelles allaient disparaître et que les bases de données d'objets allaient arriver, de cette façon nous pouvons prendre nos structures en mémoire et les enregistrer directement sur disque sans aucune de ces correspondances (mapping) entre les deux.

… **Mais cela n'a pas été le cas et les bases de données relationnelles restent dominantes.**

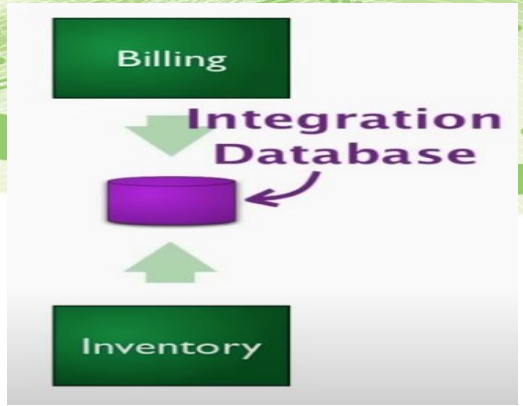




Qu'est-ce que NoSQL ?

Historique des bases de données NoSQL

- La raison pour laquelle les bases de données SQL sont restées dominantes est qu'elles étaient devenues un **mécanisme d'intégration** dans lequel de nombreuses personnes intégraient différentes applications via des bases de données SQL.
- En conséquence, cela a vraiment rendu très difficile l'arrivée de tout autre type de technologie et cela a conduit le relationnel à continuer à être dominant jusque dans les années 2000.
- Ainsi, le relationnel a eu 20 ans de domination complète de l'espace de données d'entreprise et de nombreux autres également.

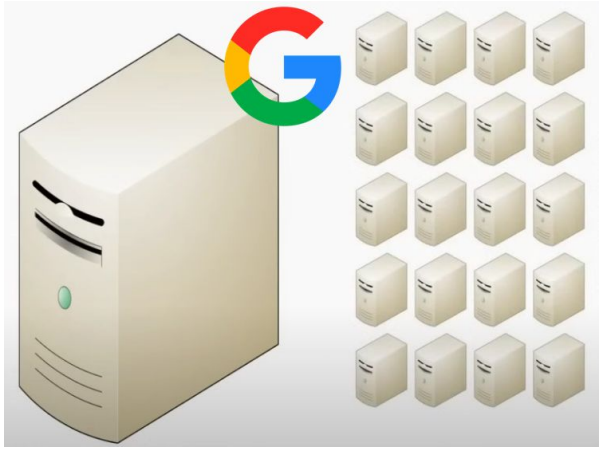


Le principal facteur était le rôle de SQL en tant que mécanisme d'intégration entre les applications. Dans ce scénario, **la base de données agit comme une base de données d'intégration, avec plusieurs applications, généralement développées par des équipes distinctes**, stockant leurs données dans une base de données commune. Cela améliore la communication car toutes les applications fonctionnent sur un ensemble cohérent de données persistantes.

Qu'est-ce que NoSQL ?

Historique des bases de données NoSQL

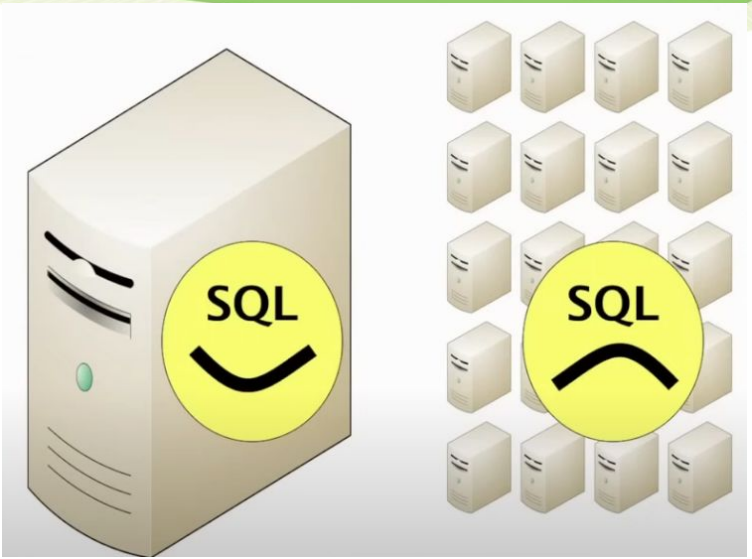
- Ce qui a vraiment changé (après l'an 2000), c'est l'essor d'internet et en particulier des sites qui ont beaucoup, beaucoup de trafic, les gros sites internet comme **Amazon** ou **Google**.
- Au fur et à mesure que vous recevez de grandes quantités de trafic, vous devez faire évoluer les choses et la seule voie évidente consiste à faire évoluer les choses en achetant des machines plus grosses (mise à l'échelle verticale), mais cette approche pose des problèmes …
- De nombreuses organisations, notamment Google, ont utilisé une approche complètement différente : **déployer de nombreuses petites boîtes (machines),** essentiellement des processeurs, des cartes mères, des disques, connus sous le nom de "commodity hardware" dans les grilles massives.



Qu'est-ce que NoSQL ?

Historique des bases de données NoSQL

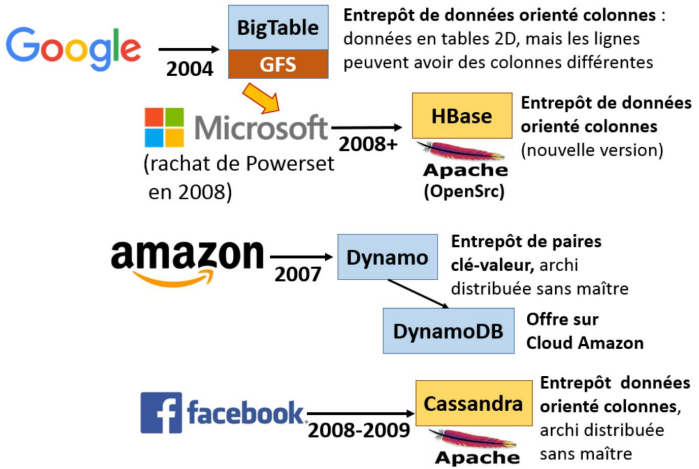
- SQL a été conçu pour fonctionner sur ces grandes boîtes conçues pour fonctionner comme un système de nœud de données unique, il ne fonctionne pas très bien avec de grands groupes de petites boîtes.
- Plusieurs acteurs du big data ont compris ce problème. Ils ont essayé de distribuer des bases de données relationnelles et de les exécuter sur des clusters, cela a fonctionné, mais cela reste "non naturel" et compliqué. Les bases de données SQL, de par leur conception, étaient censées fonctionner sur un nœud unique et centralisé.



Qu'est-ce que NoSQL ?

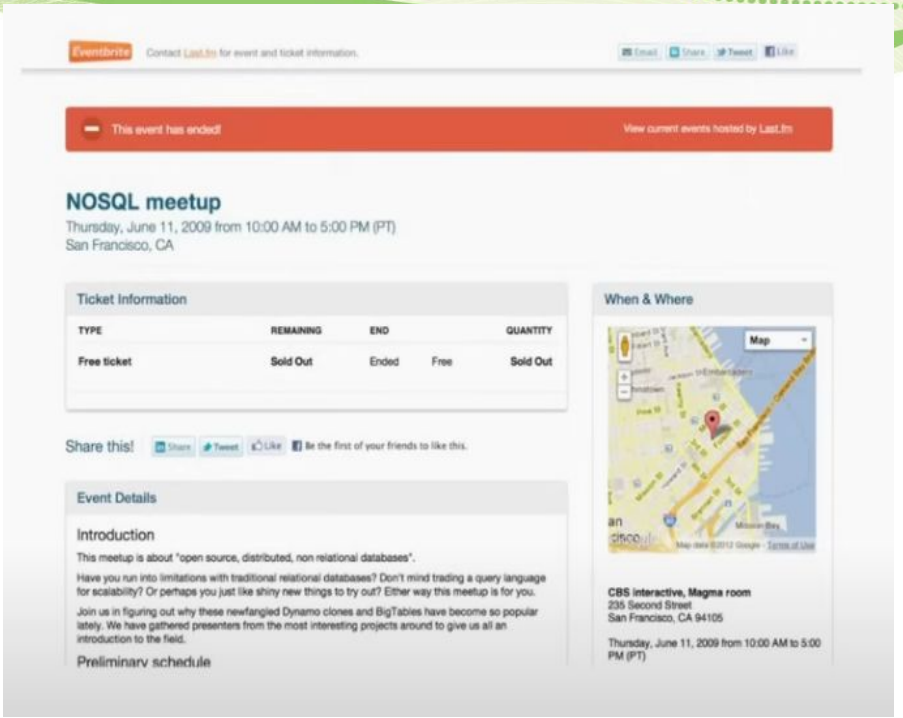
Historique des bases de données NoSQL

- Quelques organisations ont dit "**nous en avons assez, nous devons faire quelque chose de différent**" et elles ont développé leurs propres systèmes de stockage de données qui étaient vraiment très différents des bases de données relationnelles.
- Google avait tout d'abord développé **BigTable en 2004** : une BdD distribuée bâtie sur son système de fichiers distribués GFS et sur son propre mécanisme de Map-Reduce pour traiter les requêtes d'interrogation des données. Il s'agissait d'un entrepôt de données orienté colonnes. Les données sont encore organisées en lignes et colonnes, mais toutes les lignes n'ont pas forcément les mêmes colonnes (modèle beaucoup plus souple que celui des BdD relationnelles).
- En 2007 Amazon développe un entrepôt de paires clé-valeur appelé Dynamo, totalement distribué avec une architecture sans maître.
- Ces grandes organisations ont commencé à parler un peu de leurs solutions, ont publié des articles scientifiques **et c'est cela qui a vraiment inspiré tout un nouveau mouvement de bases de données qui est le mouvement NoSQL.**



Qu'est-ce que NoSQL ?

Le terme NoSQL est venu d'un hashtag Twitter pour annoncer une réunion à un moment donné (2009 à San Francisco). Le fait qu'il soit maintenant devenu le nom de tout un mouvement était complètement accidentel. Personne ne pensait que ce serait le cas.



Qu'est-ce que NoSQL ?

Le terme NOSQL est généralement interprété comme **Not Only SQL** et vise à signifier que de nombreuses applications ont besoin de systèmes autres que les systèmes SQL relationnels traditionnels pour augmenter leurs besoins en gestion de données.

La plupart des systèmes NOSQL sont des **bases de données distribuées** ou des **systèmes de stockage distribués**, mettant l'accent sur le stockage de données **semi-structurées**, les **hautes performances**, la **disponibilité**, la **réplication** des données et **l'évolutivité**, par opposition à l'accent mis sur la cohérence immédiate des données, les langages de requête puissants et les données structurées.

Qu'est-ce que NoSQL ?

Quelle est la relation entre Big Data et NoSQL ?

- Les bases de données NoSQL sont conçues pour gérer de grandes quantités de données qui peuvent ne pas tenir dans une base de données SQL traditionnelle.
- Les bases de données NoSQL sont souvent utilisées pour l'analyse de données volumineuses en raison de leur capacité à évoluer horizontalement et à gérer de gros volumes de données.
- Les bases de données NoSQL sont souvent utilisées dans des situations où les données sont structurées différemment que dans une base de données relationnelle traditionnelle, **et le schéma peut ne pas être connu à l'avance**. Cela les rend bien adaptés au traitement de données qui peuvent être non structurées ou semi-structurées, telles que les données de médias sociaux, les journaux Web ou les données de capteurs.

Qu'est-ce que NoSQL ?

Quelle est la relation entre la mise à l'échelle horizontale et NoSQL ?

- Les bases de données NoSQL ont généralement eu des implémentations distribuées, car le volume de données gérées dépasse de loin la capacité de stockage d'un seul nœud.
- La mise à l'échelle horizontale fait référence à la capacité d'augmenter la capacité d'un système **en ajoutant plus de nœuds**, plutôt qu'en mettant à niveau les capacités de nœuds individuels. Cela contraste avec la mise à l'échelle verticale, qui implique la mise à niveau du matériel ou des logiciels de nœuds individuels pour augmenter la capacité.
- Ceci est particulièrement utile pour gérer de gros volumes de données, car cela permet au système d'évoluer et d'ajouter plus de ressources selon les besoins, plutôt que d'être limité par les capacités d'un seul serveur.

Qu'est-ce que NoSQL ?

Même si les systèmes de bases de données relationnelles peuvent être distribués, pourquoi les bases de données NoSQL sont souvent considérées comme meilleures que le SGBDR pour la distribution ?

- Cela a à voir avec l'architecture et la conception des bases de données NoSQL. Contrairement aux SGBDR traditionnels, les bases de données NoSQL ont été spécialement conçues pour gérer de grandes quantités de données non structurées ou semi-structurées et pour évoluer horizontalement sur plusieurs nœuds dans un système distribué.
- Les bases de données NoSQL sont conçues pour évoluer horizontalement sur plusieurs nœuds, ce qui signifie que vous pouvez facilement ajouter plus de nœuds à votre système à mesure que vos données augmentent.
- Les bases de données NoSQL sont conçues pour être rapides et efficaces, et sont souvent optimisées pour des cas d'utilisation spécifiques, tels que le traitement de données en temps réel ou l'analyse de données à grande échelle. Cela peut faire des bases de données NoSQL un meilleur choix que RDBMS pour les applications qui nécessitent des performances élevées à grande échelle.

Qu'est-ce que NoSQL ?

- C'est plus que des lignes dans les tableaux
  - Les systèmes NoSQL stockent et récupèrent des données dans de nombreux formats : systèmes clé-valeur, bases de données de graphes, systèmes de familles de colonnes, systèmes de documents, etc.
- C'est sans jointure
  - Les systèmes NoSQL vous permettent d'extraire vos données à l'aide d'interfaces simples sans jointures.
- C'est sans schéma
  - Les systèmes NoSQL vous permettent de glisser-déposer vos données dans un dossier, puis de les interroger sans créer de modèle entité-relationnel.

Qu'est-ce que NoSQL ?

- NoSQL fonctionne sur de nombreux processeurs
  - Les systèmes NoSQL vous permettent de stocker votre base de données sur plusieurs processeurs et de maintenir des performances à haute vitesse.
- La définition ne s'agit pas du langage SQL
  - La définition de NoSQL n'est pas une application qui utilise un langage autre que SQL. SQL ainsi que d'autres langages de requête sont utilisés avec les bases de données NoSQL.
- Il n'y a pas que le big data
  - De nombreuses applications NoSQL, mais pas toutes, sont motivées par l'incapacité d'une application actuelle à évoluer efficacement lorsque le Big Data est un problème. Bien que le volume et la vitesse soient importants, NoSQL se concentre également sur la **variabilité** et l'**agilité**.

Qu'est-ce que NoSQL ?

Que signifie l'agilité dans les systèmes NoSQL ?

L'agilité fait référence à la capacité d'un système à s'adapter rapidement à l'évolution des exigences ou des besoins. Dans le contexte des bases de données NoSQL, l'agilité fait référence à la capacité de ces systèmes **à s'adapter rapidement à l'évolution des modèles de données et des charges de travail**, sans avoir besoin d'une planification initiale approfondie ou de migrations de données complexes.

Les bases de données NoSQL sont souvent plus agiles que les bases de données relationnelles traditionnelles car elles **disposent d'un modèle de données plus flexible, ce qui leur permet de stocker les données d'une manière plus adaptée aux besoins spécifiques de l'application**. Cela **facilite l'ajout ou la modification de structures de données selon les besoins**, sans qu'il soit nécessaire de reconcevoir l'ensemble de la base de données.

Qu'est-ce que NoSQL ?

Que signifie l'agilité dans les systèmes NoSQL ?

Exemple d'adaptation ou de modification d'une collection MongoDB

Imaginez que vous ayez une collection MongoDB appelée "customers" qui est utilisée pour stocker des informations sur les clients de votre entreprise. La collection a un schéma fixe, chaque document ayant les champs suivants :

Qu'est-ce que NoSQL ?

Que signifie l'agilité dans les systèmes NoSQL ?

Exemple d'adaptation ou de modification d'une collection MongoDB

Supposons maintenant que vous souhaitiez ajouter un nouveau champ à la collection pour stocker l'adresse du client. Vous pouvez le faire en utilisant l'opérateur **$addFields** dans la méthode **update()** :

```
db.customers.updateMany(
   {},
   [
      {
        $addFields: {
           address: {
              street: "1 Main St",
              city: "New York",
              state: "NY",
              zip: "10001"
           }
        }
      }
   ]
)
```

Qu'est-ce que NoSQL ?

Que signifie l'agilité dans les systèmes NoSQL ?

Exemple d'adaptation ou de modification d'une collection MongoDB

Cela ajoutera le champ d'adresse à chaque document de la collection clients. Les nouveaux documents auront désormais la structure suivante :

De cette façon, vous pouvez facilement adapter ou modifier une collection MongoDB pour répondre aux besoins changeants de votre application.

Classement de bases de données NoSQL

Classement par usage

Nous pouvons d'abord essayer de différencier les groupes de moteurs NoSQL suivant leur utilisation:

- **Amélioration des performances.**
  - Certains moteurs NoSQL ont pour but d'augmenter au maximum les performances de la manipulation des données, soit pour offrir un espace de cache en mémoire intermédiaire lors du requêtage de SGBDR, soit en tant que SGBD à part entière, qu'il soit distribué ou non.
  - Cet objectif est généralement atteint par trois mécanismes ; l'utilisation de la RAM- et nous parlons donc de bases de données en mémoire, la simplification du modèle de données en paires clé-valeur et la distribution du traitement sur les nœuds d'un cluster.

Classement de bases de données NoSQL

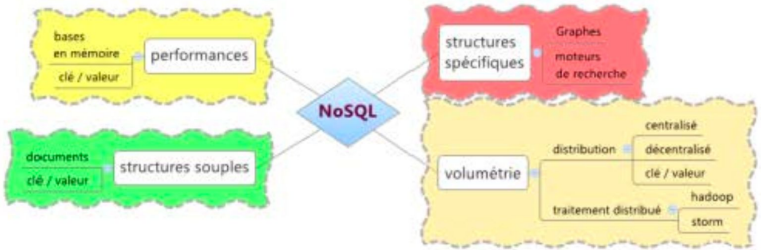
Classement par usage

- **Assouplissement de la structure :**
  - Pour s'affranchir de la rigidité du modèle relationnel, les moteurs NoSQL simplifient la plupart du temps la structure des données (utilisations de schéma souples comme le JSON, relâchement des contraintes, pas d'intégrité référentielle entre des tables, pas de schéma explicite au niveau du serveur).
- **Structures spécifiques :**
  - Certains moteurs NoSQL sont dédiés à des besoins spécifiques, et implémentent donc une structure et des fonctionnalités focalisées sur un cas d'utilisation. Citons par exemple les moteurs orientés graphes, ou les moteurs de recherche plein texte qui offrent également des fonctionnalités proches d'un SGBD, comme Apache Solr ou ElasticSearch.

Classement de bases de données NoSQL

Classement par usage

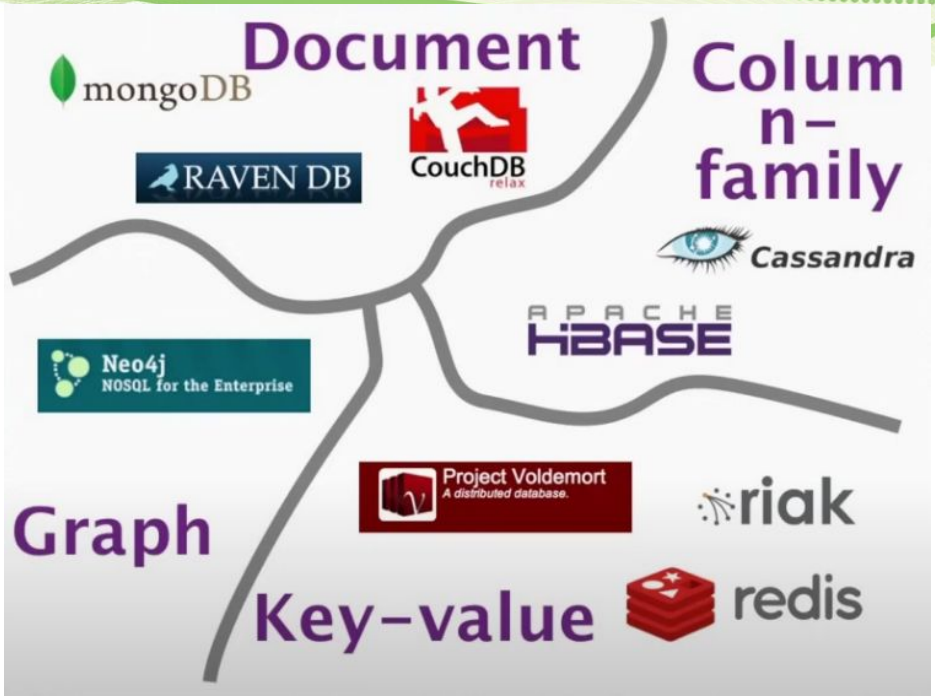
- **Volumétrie :**
  - L'un des aspects importants des moteurs NoSQL est leur capacité à monter en charge. C'est sans doute même la raison première de la création du mouvement NoSQL. Supporter des volumétries importantes passe par une distribution du stockage et du traitement. Hadoop est un système de distribution du traitement colocalisé avec un système de distribution de stockage. La distribution du traitement est très importante dans un conexte analytique et dans la plupart des applications Big Data. 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.



Classement de bases de données NoSQL

Classement par schéma de données

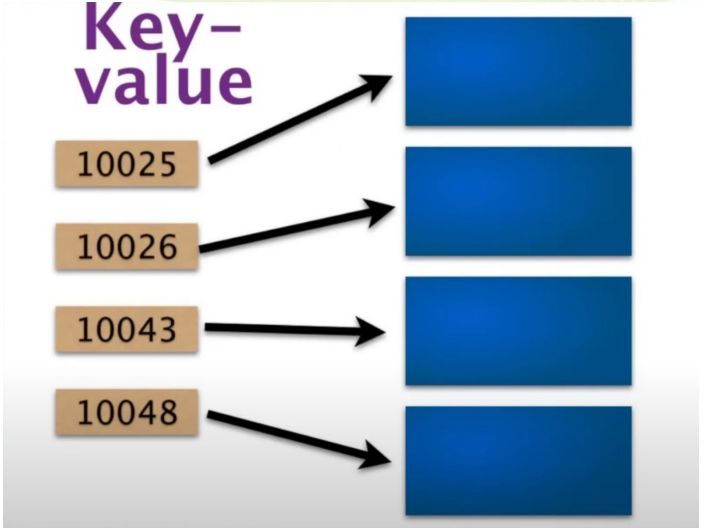
- Key-value — redis, riak
- Graph database — Neo4j the graph database, HyperGraphDB
- Document-oriented — mongoDB, CouchDB relax
- Column family — Cassandra, HBASE



Classement de bases de données NoSQL

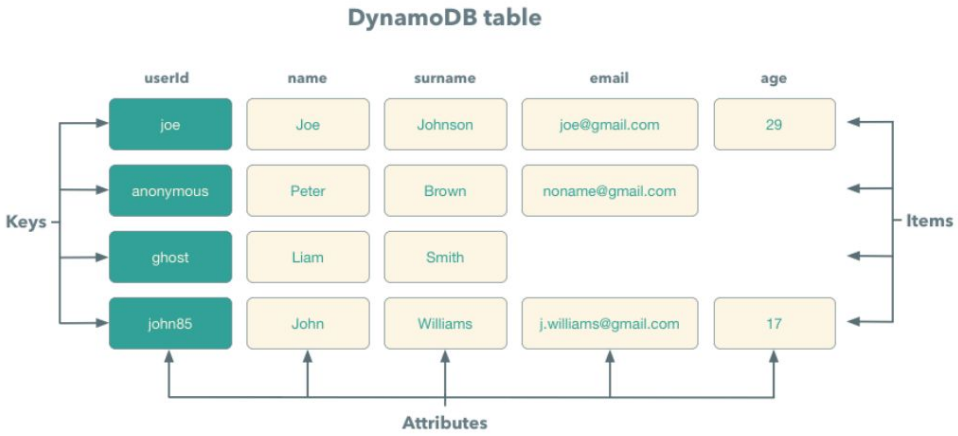
Key-Value

- Une base de données clé-valeur est un type de base de données NoSQL conçue pour stocker des données sous forme de paires clé-valeur. Dans une base de données clé-valeur, les données sont stockées sous la forme d'une collection de paires clé-valeur, où chaque clé est associée à une valeur. La clé est un identifiant unique pour les données, tandis que la valeur correspond aux données réelles stockées.
    - ça pourrait être un nombre unique, ça pourrait être un document complexe ou une image
- Les bases de données clé-valeur sont hautement évolutives et performantes, et sont bien adaptées aux applications qui nécessitent des lectures et des écritures à grande vitesse. Ils sont souvent **utilisés pour la mise en cache**, **la gestion des sessions** et d'autres applications nécessitant un accès rapide et à faible latence aux données.



Classement de bases de données NoSQL

Key-Value

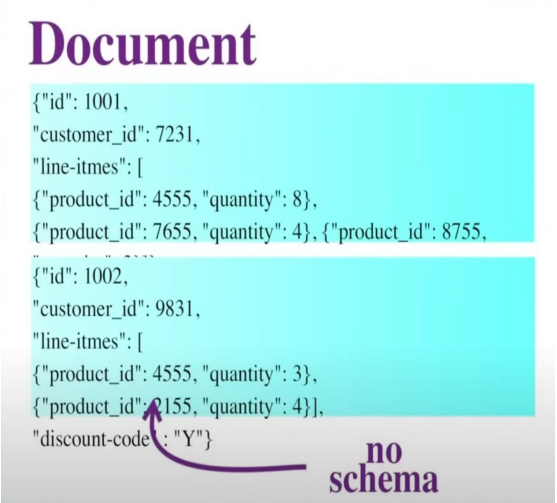


Key	Document
1001	{ "CustomerID": 99, "OrderItems": [{ "ProductID": 2010, "Quantity": 2, "Cost": 520 }, { "ProductID": 4365, "Quantity": 1, "Cost": 18 }], "OrderDate": "04/01/2017" }
1002	{ "CustomerID": 220, "OrderItems": [{ "ProductID": 1285, "Quantity": 1, "Cost": 120 }], "OrderDate": "05/08/2017" }

Classement de bases de données NoSQL

Modèle de données du document (Document Data Model)

- En relationnel, on a des lignes (des nuplets pour être précis) et des tables (des relations). **Dans le contexte du NoSQL, on va parler de documents et de collections**.
- **Dans un nuplet relationnel, on ne trouve que des valeurs dites atomiques, non décomposables**. Il ne peut y avoir qu'un seul genre pour un film. Si ce n'est pas le cas, il faut (processus de normalisation) créer une table des genres et la lier à la table des films.
  - **Cette nécessité de distribuer les données dans plusieurs tables est une lourdeur souvent reprochée à la modélisation relationnelle.**
- Avec un document structuré (en notation JSON), il est très facile de représenter les genres comme un tableau de valeurs, ce qui rompt la première règle de normalisation. On pourrait donc « encoder » une base relationnelle sous la forme de documents structurés, et chaque document pourrait être plus complexe structurellement qu'une ligne dans une table relationnelle.
  - Facilite l'ajout de nouveaux champs de données ou la modification du schéma de vos données sans avoir à apporter de modifications majeures à votre base de données.



Classement de bases de données NoSQL

Modèle de données du document (Document Data Model)

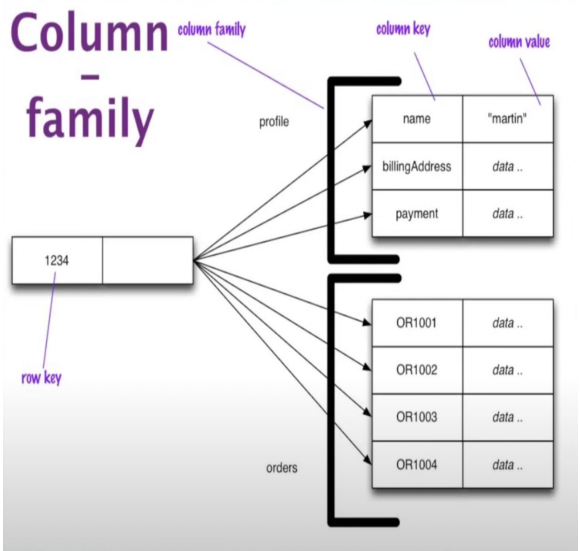
```
// customer info collection
{
    "id":42,
    "fName":"ebenezer",
    "mi":"j",
    "lName":"coot",
    "addresses": [
     { "addrType":"billingAddress",
       “mailingAddr”:"PO Box 99",
       "city":"Santa Fe",
       "St":"NM"
     },
     { "addrType": "shippingAddress",
       "streetAddr":"42 Wrinkled Way",
       "city":"Taos",
       "St":"NM"
     }]
}
```

```
// carts
{
  "id":74829312,
  "customer":42,
  "itemList":[
      {
         "UPC":293012429,
         "price":79.95,
         "name":"apple 85w power adapter",
         "shippingSpeed":"2nd day"
      },
      {
         "UPC":829381427,
         "price":59.95,
         "name":"apple touchpad mouse"
      }
  ],
 "paymentInfo":[
      {
        "ccard":"1234-5678-9876-5432",
        "exp":"0115",
        "ccxact":"111111"
      }
  ]
}
```

Classement de bases de données NoSQL

Bases de données orientées colonnes

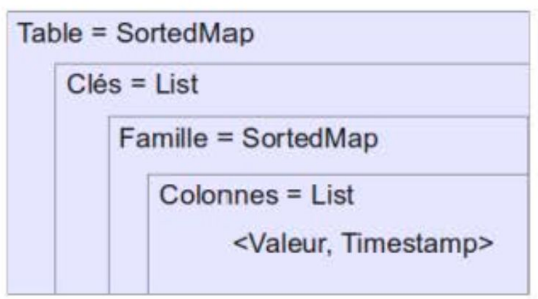
- Les bases de données orientées colonnes sont assez proches conceptuellement des tables relationnelles : elles comportent des colonnes avec un type de données. La structure des bases orientées colonnes est modélisée selon BigTable, la base de données de Google.
- Une table comporte des clés, souvent appelées **rowkeys**, les clés de lignes. Ces clés sont uniques et sont maintenues dans un ordre lexicographique.
- À l'intérieur de la table, des familles de colonnes sont définies et regroupées. La famille de colonnes est prédéfinie, et on lui attribue souvent des options, alors que les colonnes qui s'y trouvent ne sont pas prédéfinies, c'est-à-dire qu'il n'y a pas de description de schéma à l 'intérieur d'une famille de colonnes. D'une ligne à l'autre, les colonnes présentes dans une famille peuvent varier selon les données à stocker.

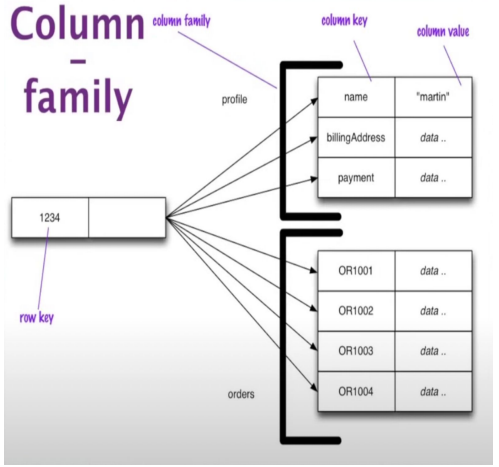


Classement de bases de données NoSQL

Bases de données orientées colonnes

- Le stockage sur le disque est organisé par famille de colonnes. On peut donc considérer les familles de colonnes comme des sous-tables. Si on représente cette organisation du point de vue de la structure des données qu'on peut manipuler dans un langage comme Java, on peut voir les choses comme illustrées à la figure suivante :
- **La table est un ensemble trié de clés. La clé est une liste de familles, la famille est un ensemble trié de colonnes, et la colonne est une liste de valeurs - timestamp. Cela donne en une ligne:**
  - **SortedMap<Cle, List<SortedMap<Colonne, List<Valeur, Timestamp>>>>**
- Les colonnes comportent donc un nom de colonne, une valeur et un horodatage (timestamp).
- Les bases orientées colonnes sont plus difficiles et complexes à mettre en place que les bases orientées documents ou paires clé-valeur, elles correspondent à des besoins plus larges.

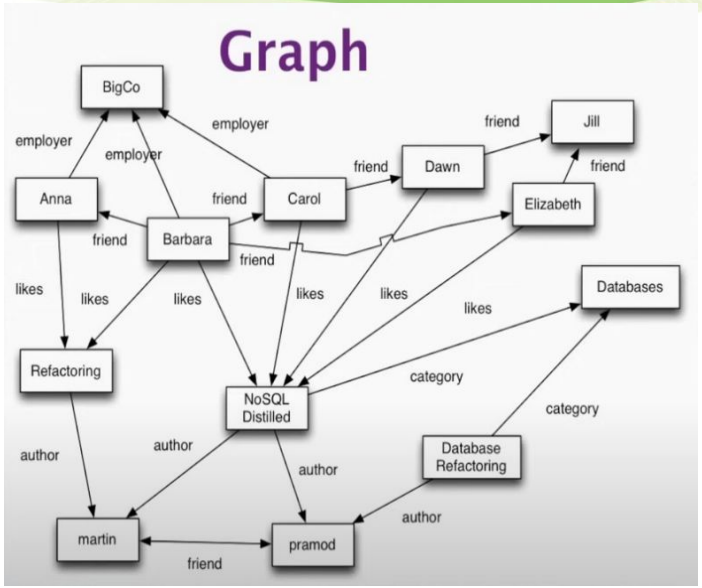




Classement de bases de données NoSQL

Bases de données orientées graphes

- Une base de données de graphes est une base de données NoSQL basée sur la théorie des graphes pour stocker des données **sur des environnements riches en relations**. La théorie des graphes est un domaine mathématique et informatique qui modélise les relations entre des objets appelés nœuds. La modélisation et le stockage de données sur les relations sont au cœur des bases de données de graphes.
- L'intérêt pour les bases de données de graphes **est né dans le domaine des réseaux sociaux.**
- Dans les données des réseaux sociaux, il peut y avoir des dizaines de relations différentes entre les individus qui doivent être suivies, et souvent les relations sont suivies à plusieurs niveaux (par exemple, amis, amis d'amis et amis d'amis d'amis). Il en résulte une situation où **les relations deviennent tout aussi importantes que les données elles-mêmes**. C'est le domaine où **brillent les bases de données de graphes.**



Classement de bases de données NoSQL

TABLE 16.2 NoSQL databases

NoSQL Category	Example Databases	Developer
Key-value databases	Dynamo Riak Redis Voldemort	Amazon Basho Redis Labs LinkedIn
Document databases	MongoDB CouchDB OrientDB RavenDB	MongoDB, Inc. Apache OrientDB Ltd Hibernating Rhinos
Column-oriented databases	HBase Cassandra Hypertable	Apache Apache (originally Facebook) Hypertable, Inc.
Graph databases	Neo4J ArangoDB GraphBase	Neo4j ArangoDB, LLC FactNexus

NoSQL et cohérence

Traiter la cohérence ⇒ Traiter le problème de nombreuses personnes essayant de modifier les mêmes données en même temps

Définition : La cohérence, dans le contexte des systèmes de gestion de bases de données, fait référence à l'exigence selon laquelle une base de données doit toujours refléter une vue valide et correcte des données, et que toutes les transactions qui modifient les données doivent être exécutées d'une manière qui préserve l'intégrité et cohérence des données.

NoSQL et cohérence

Le transactionnel et la cohérence des données

Dans le monde informatique, une transaction est une **unité d'action qui doit respecter quatre critères**, résumés par l'acronyme ACID, et qu'on nomme donc l'acidité de la transaction.

Critère	Définition
Atomique	Une transaction représente une unité de travail qui est intégralement validée ou totalement annulée. C'est tout ou rien.
Cohèrente	La transaction doit maintenir le système en cohérence par rapport à ses règles fonctionnelles. Durant l'exécution de la transaction, le système peut être temporairement incohérent, mais lorsque la transaction se termine, il doit être cohérent, soit dans un nouvel état si la transaction est validée, soit dans l'état cohérent antérieur si la transaction est annulée.
Isolée	Comme la transaction met temporairement les données qu'elle manipule dans un état incohérent, elle isole ces données des autres transactions de façon à ce qu'elle ne puisse pas lire des données en cours de modification.
Durable	Lorsque la transaction est validée, le nouvel état est durablement inscrit dans le système.

NoSQL et cohérence

Le transactionnel et la cohérence des données

- La notion de cohérence est importante, et c'est l'un des éléments les plus sensibles entre le monde du relationnel et le monde NoSQL.
- Les SGBDR imposent une règle stricte: d'un point de vue transactionnel, **les lectures de données se feront toujours sur des données cohérentes**. La base de données est visible en permanence sur un état cohérent. Les modifications (état intermédiaire) en cours sont donc cachées.
- Mais **cet état intermédiaire peut durer longtemps**, pour deux raisons: plusieurs instructions de modifications de données peuvent être regroupées dans une unité transactionnelle, qui peut donc être complexe. Ensuite, un SGBDR fonctionnant de façon ensembliste, une seule instruction de mise à jour, qui est naturellement et automatiquement transactionnelle, **peut très bien déclencher la mise à jour de milliers de lignes de table.** Cette modification en masse conservera des verrous d'écriture sur les ressources et écrira dans le journal de transaction ligne par ligne. Tout ceci prend bien évidemment du temps.

NoSQL et cohérence

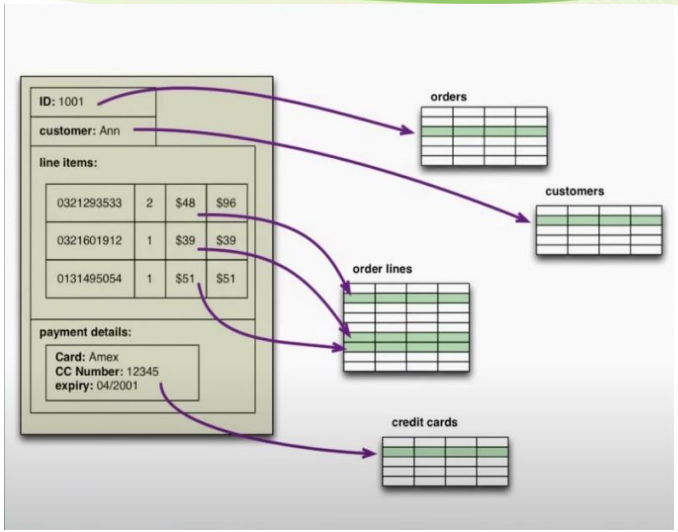
Le transactionnel et la cohérence des données

Une transaction, c'est donc un mécanisme d'isolation d'une opération simple ou complexe.

Elle est importante dans le cadre d'un SGBDR, car du fait du modèle relationnel, une donnée complexe va se décomposer dans plusieurs tables et devra donc faire l'objet de plusieurs opérations d'écriture.

Si je considère une donnée comme **une commande client**, qui comporte les informations du **client**, des **produits commandés** et de la **facture**, il faudra exécuter **plusieurs commandes SQL d'insertion**, sur les tables client, commande et facture. Le langage SQL ne permet de cibler qu'**une table à la fois** dans la commande INSERT.

L'opération logique de l'insertion d'une donnée complexe devra se traduire par plusieurs opérations physiques d'insertion, et devra être regroupée dans une transaction explicite pour s'assurer qu'il n'est pas possible de se retrouver dans la base de données avec des données partiellement écrites.



NoSQL et cohérence

Relâchement de la rigueur transactionnelle

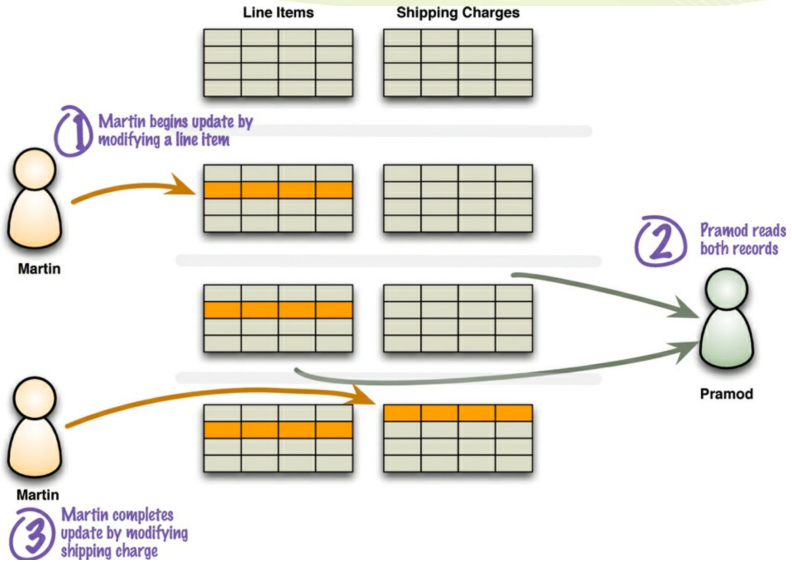
- On comprend bien qu'un moteur NoSQL, **basé sur le principe de l'agrégat**, ne présente pas ce genre de problème. Dans MongoDB par exemple, on écrit en une seule fois un document JSON qui agrège toutes les informations de la commande : plus besoin par conséquent d'une transaction explicite.
- De fait, les critères de la transaction changent de sens: **l'écriture est naturellement atomique car le document est écrit en une fois**, et **elle est naturellement isolée car le document est verrouillé durant l'écriture**.
- Il faut donc repenser le concept des critères transactionnels à la lumière d'un système qui gère différemment ses données, et dont les possibilités et les exigences sont autres.
- **Les systèmes NoSQL agrègent les données, mais quand ils sont distribués, ils les dupliquent.** Cela repose le problème transactionnel dans le cadre de la réplication. Si on écrit une donnée plusieurs fois, sur plusieurs nœuds d'un cluster, **cette écriture redondante sera-t-elle atomique, cohérente, isolée et durable?** C'est dans ce contexte qu'il faut comprendre les discussions autour de la cohérence (consistency) des écritures dans un moteur NoSQL.

NoSQL et cohérence

Un exemple d'incohérence logique

Imaginons que nous ayons une commande avec des articles et des frais d'expédition. Les frais d'expédition sont calculés en fonction des articles de la commande.

Si nous ajoutons un élément de ligne, nous devons donc également recalculer et mettre à jour les frais d'expédition. Dans une base de données relationnelle, les frais d'expédition et les éléments de ligne seront dans des tables distinctes. Le danger d'incohérence est que Martin ajoute un élément de ligne à sa commande, Pramod lit ensuite les éléments de ligne et les frais d'expédition, puis Martin met à jour les frais d'expédition. Il s'agit d'un conflit de lecture ou de lecture-écriture incohérent : Pramod a effectué une lecture au milieu de l'écriture de Martin.



NoSQL et cohérence

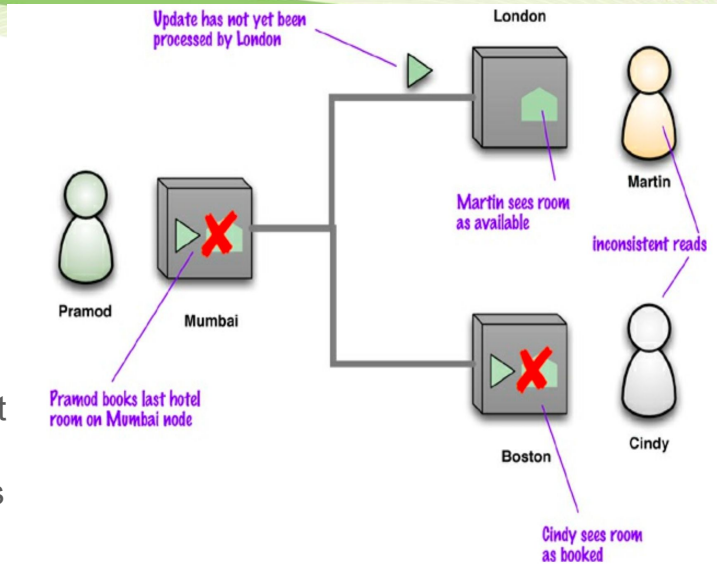
Un exemple d'incohérence logique

- Pour éviter un conflit lecture-écriture logiquement incohérent, **les bases de données relationnelles prennent en charge la notion de transactions.**
- Une affirmation courante que nous entendons est que les bases de données NoSQL ne prennent pas en charge les transactions et ne peuvent donc pas être cohérentes. **Une telle affirmation est généralement fausse** car elle passe sous silence de nombreux détails importants. Notre première clarification est que toute déclaration sur le manque de transactions ne s'applique généralement qu'à certaines bases de données NoSQL, en particulier celles orientées agrégats. En revanche, **les bases de données de graphes ont tendance à prendre en charge les transactions ACID** de la même manière que les bases de données relationnelles.
- Deuxièmement, les bases de données orientées agrégat prennent en charge les mises à jour atomiques, mais uniquement au sein d'un seul agrégat. **Cela signifie que vous aurez une cohérence logique au sein d'un agrégat, mais pas entre les agrégats**. Ainsi, **dans l'exemple, vous pouvez éviter de rencontrer cette incohérence si la commande, les frais de livraison et les éléments de ligne font tous partie d'un seul agrégat de commande.**

NoSQL et cohérence

Un exemple d'incohérence de réplication*

Imaginons qu'il y ait une dernière chambre d'hôtel pour un événement souhaitable. Le système de réservation d'hôtel fonctionne sur de nombreux nœuds. Martin et Cindy sont un couple qui envisage cette chambre, mais ils en discutent au téléphone car Martin est à Londres et Cindy est à Boston. Pendant ce temps, Pramod, qui est à Mumbai, va réserver cette dernière chambre. **Cela met à jour la disponibilité des chambres répliquées, mais la mise à jour arrive à Boston plus rapidement qu'à Londres.** Lorsque Martin et Cindy lancent leurs navigateurs pour voir si la chambre est disponible, Cindy la voit réservée et Martin la voit libre. Il s'agit d'une autre lecture incohérente, mais il s'agit d'une violation d'**une autre forme de cohérence que nous appelons la cohérence de la réplication** : garantir que le même élément de données a la même valeur lorsqu'il est lu à partir de différentes répliques.



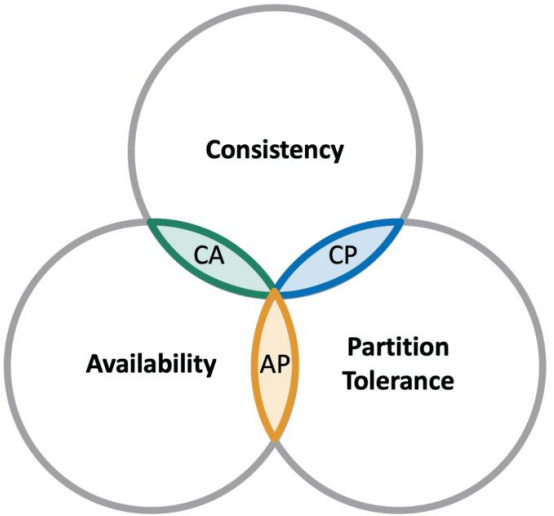
*La réplication est le processus de copie de données d'un serveur vers un ou plusieurs autres serveurs, afin de s'assurer que les données sont disponibles à plusieurs endroits.

NoSQL et cohérence

Théorème CAP

Théorème CAP : dans un système distribué il est **impossible** de garantir à chaque instant T **plus que deux parmi les trois propriétés** suivantes :

- **C**onsistency (cohérence) :
  - tous les nœuds sont à jour sur les données au même moment;
- **A**vailability (disponibilité) :
  - la perte d'un nœud n'empêche pas le système de fonctionner et de servir l'intégralité des données;
- **P**artition tolerance (résistance au morcellement / une panne partielle)
  - chaque nœud doit pouvoir fonctionner de manière autonome.

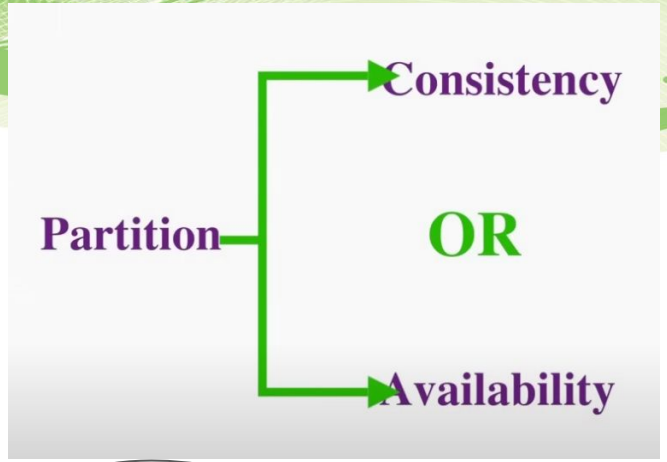


NoSQL et cohérence

Théorème CAP

Théorème CAP : dans un système distribué il est **impossible** de garantir à chaque instant T **plus que deux parmi les trois propriétés** suivantes :

- **C**onsistency (cohérence) :
  - tous les nœuds sont à jour sur les données au même moment;
- **A**vailability (disponibilité) :
  - la perte d'un nœud n'empêche pas le système de fonctionner et de servir l'intégralité des données;
- **P**artition tolerance (résistance au morcellement / panne partielle)
  - chaque nœud doit pouvoir fonctionner de manière autonome.

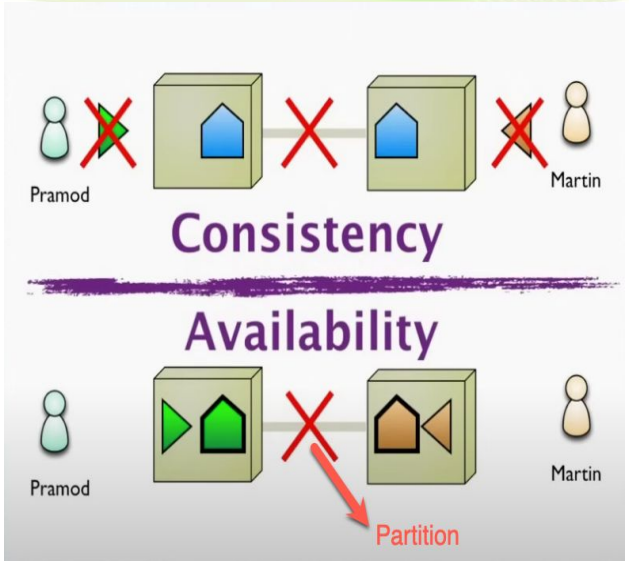


Si vous avez un système qui peut avoir une partition réseau et si vous avez un système distribué, vous avez le choix : voulez-vous être cohérent ou voulez-vous être disponible ?

NoSQL et cohérence

Théorème CAP

- Pramod et Martin veulent tous les deux réserver une chambre d'hôtel mais la ligne de communication est coupée et les deux nœuds ne peuvent pas communiquer (nous avons une “partition”).
- **Il y a deux choix :** L'un est que le système dit "**ah, notre ligne de communication est en panne, désolé, nous ne pouvons pas prendre vos réservations d'hôtel pour le moment, veuillez réessayer plus tard**".
- L'alternative est que le système dise "**oui, nous accepterons votre réservation car nous sommes vraiment fiables et à jour**".
- Ce que nous voyons, c'est un choix, c'est un choix entre la cohérence qui signifie "**non je ne ferai rien si mes lignes de communication sont coupées**" et la disponibilité qui dit "**oui je vais continuer mais au risque d'introduire un comportement incohérent**".
- Maintenant, la chose essentielle à réaliser ici est qu'il s'agit d'un choix et que **c'est un choix qui ne peut être fait qu'en connaissant les règles métier** (business rules) avec lesquelles vous travaillez.



NoSQL et cohérence

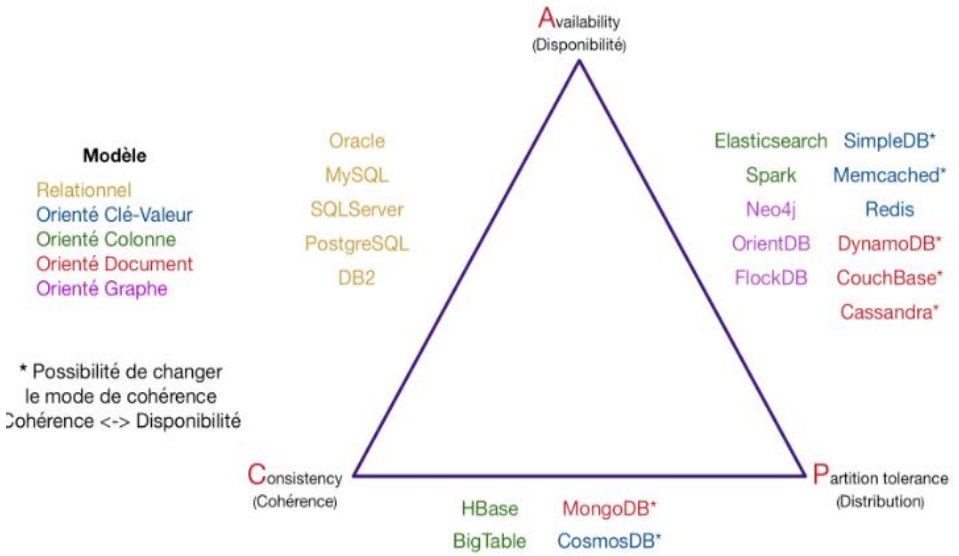
Théorème CAP

- Il y a des cas où vous pouvez traiter avec élégance des réponses incohérentes aux demandes. Ces situations sont étroitement liées au domaine et nécessitent une connaissance du domaine pour savoir comment les résoudre. **Ainsi, vous ne pouvez généralement pas chercher à les résoudre uniquement au sein de l'équipe de développement - vous devez parler à des experts du domaine.** Si vous pouvez trouver un moyen de gérer les mises à jour incohérentes, cela vous donne plus d'options pour augmenter la disponibilité et les performances. **Pour un panier (shopping cart), cela signifie que les acheteurs peuvent toujours faire leurs achats et le faire rapidement.**
- Dans certains cas, vous devez savoir dans quelle mesure vous tolérez les lectures obsolètes et combien de temps peut durer la fenêtre d'incohérence.
- Il est généralement préférable de ne pas penser au compromis entre cohérence et disponibilité, mais plutôt entre cohérence et latence. Nous pouvons alors considérer la disponibilité comme la limite de latence que nous sommes prêts à tolérer ; une fois que la latence devient trop élevée, nous abandonnons et traitons les données comme indisponibles.

NoSQL et cohérence

Théorème CAP

- Grâce à ce théorème de CAP, il est alors possible de classer toutes les bases de données en les plaçant sur le "triangle de CAP", tout en ajoutant des codes couleurs pour chaque modèle de stockage.
- Nous pouvons constater que les bases de données relationnelles se retrouvent sur la face CA du triangle, combinant disponibilité et cohérence. Nous retrouvons bien MongoDB pour le couple CP (cohérence et distribution) mais également les solutions orientées colonnes comme HBase ou BigTable.



Conclusion

Les bases NoSQL **ne remplacent pas les BD relationnelles** mais en sont une **alternative**, un **complément** apportant des solutions plus intéressantes dans certains contextes.

Exemple: Pour gérer un site de vente en ligne

- On utilisera une base relationnelle, comme PostgreSQL, pour enregistrer rigoureusement les transactions commerciales validées. Cela représente environ 1% de l'ensemble des accès aux données, les autres actions étant essentiellement de la consultation et de la construction de paniers d'achat.
- On utilisera une base orientée document, comme MongoDB, pour gérer l'interaction entre l'utilisateur et le catalogue de produits et enregistrer les paniers. Cela permet de proposer une interface plus réactive. Certaines incohérences se manifesteront parfois, par exemple on aura autorisé à ajouter un objet au panier alors qu'au moment de la validation (et donc de la vérification dans la base PostgreSQL) on découvrira qu'il n'est finalement pas disponible. On considère ici que ces quelques erreurs sont non critiques et acceptables au regard du gain de performance acquis.