NOSQL

NOSQL : POURQUOI ?

- NoSQL signifie “Not Only SQL”, “pas seulement SQL”
- ce terme désigne l’ensemble des bases de données qui s’opposent à la notion relationnelle des SGBDR
- NoSQL ne vient pas remplacer les BD relationnelles mais proposer une alternative ou compléter les fonctionnalités des SGBDR pour donner des solutions plus intéressantes dans certains contextes
- NoSQL répond à une nécessité de performance de traitement de données en très gros volumes
- avec NoSQL, on parlera souvent de scalabilité

BASES DE DONNÉES

- une base de données est une collection d'informations dont la structure reflète les relations qui existent entre ces données
- cette organisation et ce système relationnel distingue une base de données d'un simple système de fichiers ou d'une arborescence de répertoires
- une base de données peut être considérée comme la transcription d'une partie du monde réel (une entreprise par exemple) en informatique
- cette transcription passe par une phase de modélisation

BASES DE DONNÉES : SYSTÈME DE GESTION DE BASES DE DONNÉES

C'est un ensemble de logiciels permettant la gestion des données en très grand volume, notamment :

- la description des données
- la description des relations entre les données
- le stockage des données
- l'insertion, la mise à jour, la suppression des données
- la recherche efficace des données
- le partage des données par de multiples utilisateurs
- la sauvegarde des données
- la protection des données

BASES DE DONNÉES : FONCTIONNALITÉS ET RÈGLES

Indépendance physique (données / programmes)

- pouvoir modifier l'organisation physique sans modifier les programmes (accès local ou distant à la base de données, changement de stockage …)

Indépendance logique

- pouvoir modifier le schéma conceptuel sans modifier les programmes (beaucoup plus difficile)

Manipulation des données

- permettre à des utilisateurs qui n'ont pas la connaissance de l'organisation de la base ni les acquis techniques de manipuler des données

Efficacité de traitement des données

- permettre de traiter efficacement les données à l'aide de langages différents utilisant les fonctionnalités multicritères

BASES DE DONNÉES : FONCTIONNALITÉS ET RÈGLES

Administration centralisée de la base de données

- l'administrateur de la base définit la structure des données, leur stockage et leur contrôle : utilisateurs, droits d'accès

Intégrité des données

- l'administrateur définit des règles de cohérence : contraintes d'intégrité, contraintes de mises à jour et de suppression

Partage des données

- plusieurs utilisateurs ont accès aux données, et ce, via plusieurs applications différentes

Sécurisation et sauvegarde des données

- l'administrateur définit les droits d'accès des utilisateurs aux données et les modes de sauvegardes : export, snapshot, réplication etc.

BASES DE DONNÉES : NIVEAUX, ACTEURS ET RÔLES

Méthodologie du concepteur / développeur

- utilisation de méthodes et langages (Merise, UML) pour arriver à créer un modèle conceptuel
- ce modèle permet de définir :
  - les différents objets de la base de données
  - les relations entre les objets
  - les contraintes sur les données

Le niveau interne ou physique

- spécification du stockage physique des données (fichiers, disques, etc.) et des méthodes d'accès (index, chaînages, etc.)

Le niveau externe

- possibilité de création de vues externes pour différents groupes d'utilisateurs sur tout ou partie de la base de données

BASES DE DONNÉES : NIVEAUX, ACTEURS ET RÔLES

L'administrateur

- il administre les niveaux internes et externes, selon qu'il soit administrateur de la base de données et / ou administrateur d'application, notamment :
  - la définition du schéma conceptuel
  - l'évolution du schéma conceptuel
  - la création / l'évolution des tables
  - les modalités de protection des données
  - diverses tâches de maintenance et de sécurisation de la base de données (sauvegardes, restauration, optimisation, gestion des utilisateurs et droits d'accès …)

Le développeur

- il crée et maintient des bibliothèques de programmes de manipulation de la base (interrogation, mise à jour, etc.), à l'aide de langages de manipulation de données, liés ou non au moteur même de la base de données

BASES DE DONNÉES : HISTORIQUE

1950 - 1960

- apparition des premières organisations de fichiers (SGF : Systèmes de Gestion de Fichiers) et les méthodes d'accès à ces fichiers : accès séquentiel, direct, séquentiel indexé
- les SGF sont généralement incompatibles entre eux et la notion même de structure de fichiers n'est pas uniforme, ce qui rend les applications peu portables
- les langages de l'époque sont des langages procéduraux pour lesquels la gestion d'application multi-fichiers devient rapidement pénible, la programmation à l'aide d'outils non conçus initialement pour traiter des fichiers est donc lourde et contraignante
- il faut donc envisager un nouveau modèle de fonctionnement

BASES DE DONNÉES : HISTORIQUE

1962 – 1963

- apparition du concept de Base de Données.
- le terme «base de données» apparaît pour la première fois en 1964 dans le titre d'une conférence aux Etats-Unis : « *Development and Management of a computer-centered data base* »
- on s'intéresse pour la première fois au traitement de gros volumes d'informations avec un souci d'efficacité des accès
- les premières règles sont établies :
  - le SGBD est un système de stockage de la base de données qui en assure la maintenance
  - les données sont organisées en base de données, un ensemble de données logiquement liées
  - les données sont partageables (accès concurrents) entre plusieurs utilisateurs

BASES DE DONNÉES : HISTORIQUE

1965 - 1970

- Conception des SGBD de 1ère Génération (2 modèles : hiérarchique et réseau)
- IMS d'IBM (hiérarchique)
- IDS de General Electric (réseau)

1970 – 1985

- 2ème Génération des SGBD organisés sur le modèle relationnel, avec davantage de moyens d'accès aux données :
  - MRDS de Honeywell diffusé par CII-HB
  - QBE (Query By Example)
  - SQL/IDS d'IBM
  - INGRES de Relational Technology
  - ORACLE de Relational Software.
- Dans cette même période apparaissent les premières méthodes de conception (MERISE)

BASES DE DONNÉES : HISTORIQUE

Depuis …

- nous devons faire face à la prolifération de données plus variées : textes, sons, photos, vidéos, likes, …
- la technologie a été adaptée en conséquence :
  - bases de données réparties
  - bases de données orientées objets
  - bases de connaissances et Systèmes Experts : 3 acteurs, l'utilisateur final, l'expert du domaine et l'ingénieur de connaissances interagissent pour concevoir une base de connaissances, une base de faits et un moteur d'inférence (propositions, conclusions) effectuant une forme définie de raisonnement
  - bases de données déductives (règles de découverte de connaissances, données implicites) → Business Intelligence (informatique décisionnelle) / Big Data

BASES DE DONNÉES : HISTORIQUE

- nous sommes face à une évolution permanente du contexte technologique, hardware et software.
- du côté logiciel, nous sommes passés du néant à des produits évolués :
  - ateliers de développement
  - langages de plus en plus puissants
  - générateurs de programmes (formes, reports, menus …)
  - mécanismes assurant la cohérence du Système d'Information (référentiel, dictionnaire de données)
  - outils d'intégration continue (Jenkins)

BASES DE DONNÉES : HISTORIQUE

- nous sommes passés :
  - d'une informatique centralisée à une informatique orientée services aux utilisateurs, dans un environnement hétérogène : machines / logiciels / humains
  - d'une absence totale de méthodes à des méthodes d'analyse et de conceptions de systèmes informatiques qui privilégient désormais une approche horizontale de séparation des données et des traitements
- cela implique de concevoir des dépôts utilisés dans toutes les applications (potentielles), accessible à de multiples utilisateurs n'ayant pas les mêmes besoins, pas la même culture, pas les mêmes responsabilités et pas la même approche technique et humaine (management)

BASES DE DONNÉES : QUELQUES DATES

- 1970 : IBM crée le premier prototype de base de données sous le nom de System/R, ainsi que son langage, le SEQUEL
- 1973 : En se basant sur le System/R d'IBM, L'université de Berckeley crée le projet RDBMS, duquel découleront Sybase, Informix, NonStop SQL
- 1978 : arrivée d'Oracle
- 1981 : création d'Informix, fondation de Borland, apparition de dBase II sur IBM
- 1984 : Oracle V 4 et premier portage sur PC, création de Sybase
- 1986 : Oracle V5 avec architecture client / serveur
- 1989 : apparition de Postgres
- 1991 : MicroSoft SQL Server
- 1992 : Oracle V 7 (intégration des contraintes procédures et déclencheurs), première version de Microsoft Access
- 1996 : première version publique de MySQL
- 1997 : Oracle V 8 (objet, Java)

BASES DE DONNÉES : DU CENTRALISÉ AU DISTRIBUÉ

- avec l'apparition d'applications comme Amazon, Facebook, Google, Twitter … avec un énorme volume de données à gérer, il s'est rapidement avéré impossible de répondre à ces besoins là par des plateformes centralisées
- il faut donc distribuer les données et leur traitement sur beaucoup de serveurs → avènement des datacenters dont voici quelques caractéristiques :
  - Ils utilisent des réseaux avec 3 niveaux de communication :
    - les serveurs sont regroupés en racks : liaison réseau ultra rapide
    - un datacenter consiste en un grand nombre de racks, interconnectés par des routeurs
    - entre différents datacenters : communication internet ultra rapide
    - les serveurs communiquent par envoi de messages, ils ne partagent pas de disque ni de ressource de traitement = architecture shared nothing
- un datacenter Google contient entre 100 et 200 racks, chacun contenant une cinquantaine de serveurs
- il y a donc 5 000 à 10 000 serveurs par datacenter pour un total global de plus d'un million de serveurs

BASES DE DONNÉES : DU CENTRALISÉ AU DISTRIBUÉ

Les insuffisances des SGBD à traiter les gros volumes sont liées aux fondements des SGBD eux mêmes. Ils n'ont tout simplement pas été conçus pour cela, et présentent les lacunes suivantes :

- manque d'efficacité pour des données peu ou non structurées
- manque de puissance dans le cas de gros volumes
- difficultés d'évolution de la structure lors de la montée en charge
- forte nécessité de compétences (qui n'existent pas toujours) pour l'optimisation :
  - de la structure de la base de données
  - des stockages (type) et des volumes (supports)
  - des requêtes
- prix de certaines licences trop élevé (Oracle)

BASES DE DONNÉES : DU CENTRALISÉ AU DISTRIBUÉ

- le SGBD est plus adapté à la centralisation, parce que conçu pour cela à l'origine
- or la centralisation a ses limites par rapport à la distribution
- en effet, il n'y a pas dans les SGBD traditionnels de Map Reduce par exemple, à savoir la distribution des données sur un cluster de serveurs pour les traiter séparément
- il est également plus facile de transférer sur le réseau un programme (taille maîtrisée) qu'un trop fort volume de données

BASES DE DONNÉES - MAP REDUCE : DÉFINITION

- c'est un modèle de programmation parallèle pour le traitement de grands ensembles de données, développé par Google pour le traitement de gros volumes de données en environnement distribué
- il permet de répartir la charge sur un grand nombre de serveurs (cluster)
- il y a abstraction quasi-totale de l'infrastructure matérielle : il gère entièrement, de façon transparente le cluster, la distribution de données, la répartition de la charge, et la tolérance aux pannes
- ajouter des machines augmente la performance
- la librairie Map Reduce existe dans de nombreux langages (C++, C#, Java, Python, Ruby…)

BASES DE DONNÉES - MAP REDUCE : SON UTILISATION

- Map-Reduce est notamment utilisé par les grands acteurs du Web pour :
  - construire les index (Google Search)
  - la détection de spam (Yahoo)
  - le Data Mining (Facebook)
- mais aussi pour :
  - l'analyse d'images, l'apprentissage automatique (Machine Learning), des statistiques, …
  - le calcul de la taille de plusieurs milliers de documents
  - trouver le nombre d'occurrences d'un pattern dans un très grand volume de données
  - classifier de très grands volumes de données provenant par exemple de paniers d'achats de clients (Data Mining)

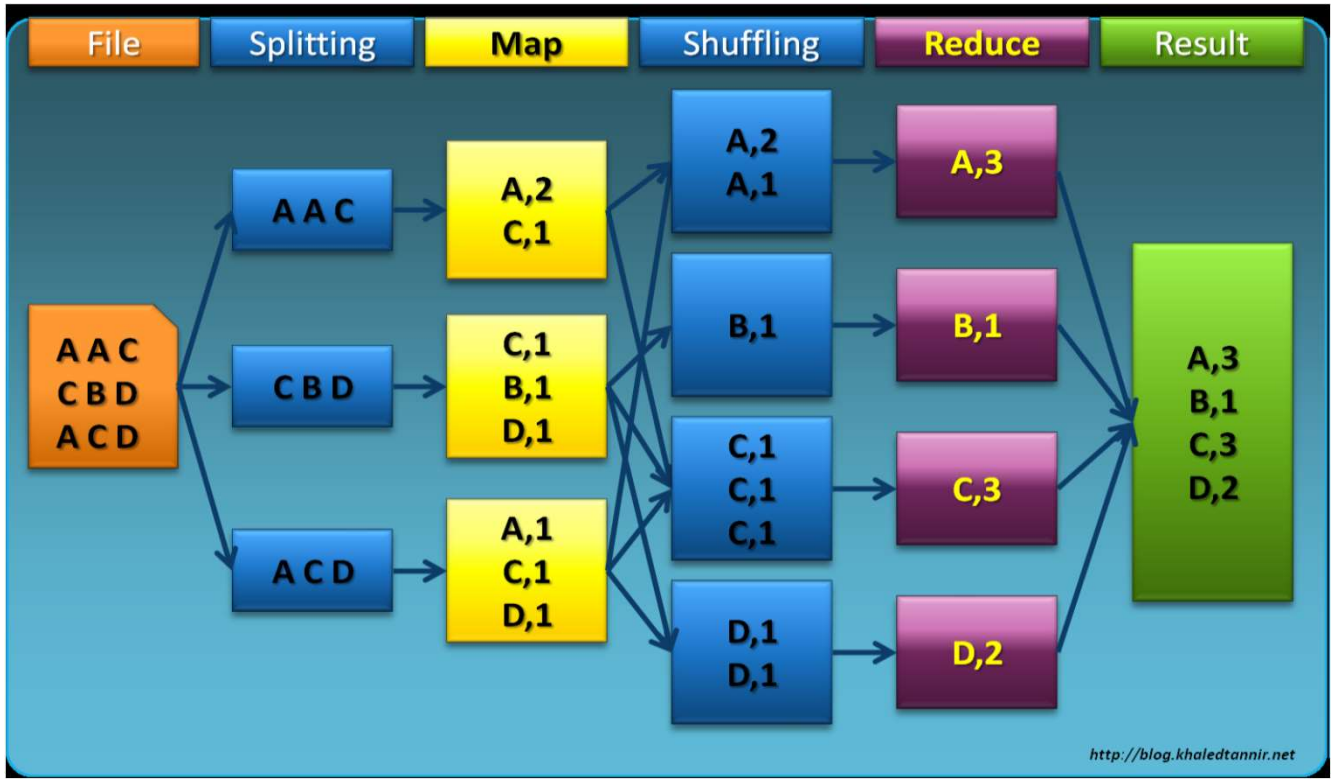
BASES DE DONNÉES : MAP REDUCE, EXEMPLE SIMPLE

Soit un fichier contenant 3 lignes.

Chaque ligne est composée de 3 mots parmi les mots : A, B, C ou D

On veut compter tous les mots contenus dans ce fichier de 3 lignes.

Voici le traitement Map Reduce correspondant :



BASES DE DONNÉES : LES CONTRAINTES ACID

- les SGBD relationnels sont généralement transactionnels, les transactions respectent les contraintes ACID (en anglais : Atomicity, Consistency, Isolation, Durability, en français : Atomicité, Cohérence, Isolation, Durabilité) qui garantissent qu'une transaction est fiable
- Atomicité
  - une transaction se fait complètement … ou pas du tout
  - si une partie de la transaction ne peut pas être réalisée, on annule la transaction et on restaure les données à leur état initial
- Cohérence
  - chaque transaction respecte les contraintes d'intégrités définies dans la base de données par le concepteur de la base de données
  - après chaque transaction complétée, le système est laissé dans un état valide

BASES DE DONNÉES : LES CONTRAINTES ACID

- Isolation
  - l'isolation assure qu'une transaction doit s'exécuter comme si elle était seule sur le système
  - cela implique que les transactions soient indépendantes
  - l'exécution simultanée de plusieurs transactions doit donner le même résultat que l'exécution d'une seule. C'est la contrainte la plus difficile à respecter, notamment lors de mises à jour concurrentielles
- Durabilité
  - la durabilité assure que lorsqu'une transaction a été confirmée, elle demeure enregistrée même à la suite d'un problème (coupure, panne …)
  - on parle de persistance des données : toute donnée enregistrée doit l'être de façon durable

BASES DE DONNÉES : LA SCALABILITÉ

- la scalabilité (mise à l'échelle) est la capacité d'un produit (matériel, logiciel) à s'adapter à la montée en charge, liée à la demande des utilisateurs (nombre, fréquence) ou à une augmentation des volumes d'informations à traiter
- la mise à l'échelle verticale (scale up/scale vertically) consiste à augmenter les capacités matérielles d'un système d'information
- elle présente 2 inconvénients :
  - les applications sont indisponibles pendant qu'on augmente les capacités de la machine
  - les coûts peuvent être rapidement prohibitifs pour une performance matérielle certes améliorée, mais pour combien de temps ?

BASES DE DONNÉES : LA SCALABILITÉ

- la mise à l'échelle horizontale (scale out/scale horizontally) consiste à lancer la même application sur plusieurs serveurs pour répartir la charge de travail
- avec cette solution, il n'y a plus de risque d'indisponibilité de l'application comme dans la mise à l'échelle verticale
- elle demande des compétences techniques élevées pour être optimisée
- il est souvent plus difficile de répartir des traitements et des données que d'installer de nouveaux matériels
- pour l'aspect financier, l'investissement matériel est remplacé par un investissement humain

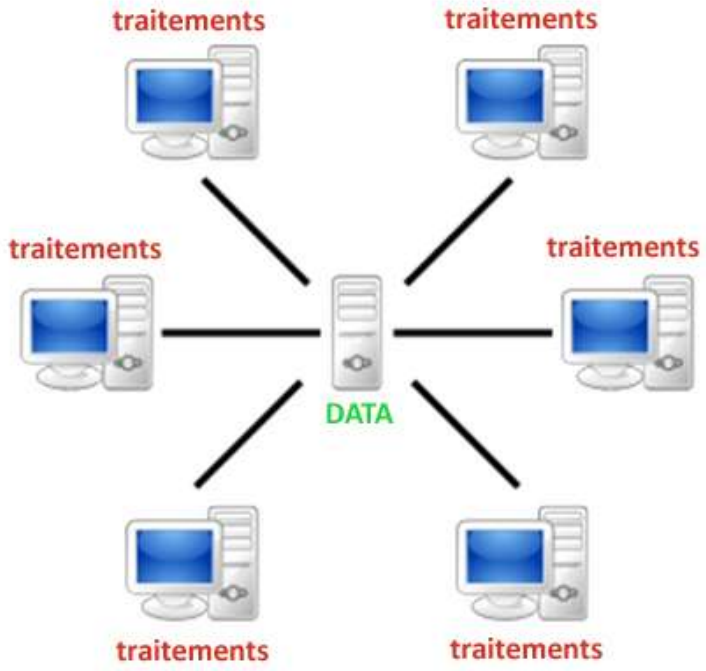
BASES DE DONNÉES : NOSQL ET LE SCALING

- NoSQL ne se substitue pas aux bases de données traditionnelles
- il est un complément et un palliatif à leurs lacunes, et repose sur des systèmes distribués, permettant de gérer et coordonner plusieurs ordinateurs reliés en un réseau et communiquant par envoi de messages
- le matériel n'est pas spécialisé, il est même considéré comme standard et peut ainsi être remplacé facilement
- dans ce type d'architecture, le nombre fait la force

TRAITEMENTS ET DONNÉES : 2 SOLUTIONS

Distribution des traitements

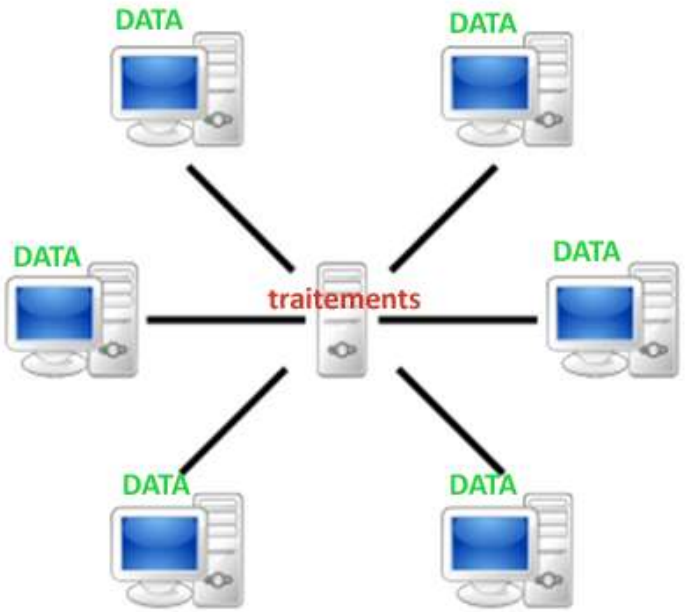
- on distribue les traitements sur un nombre important de serveurs afin d'absorber des charges très importantes
- on envoie les données vers les serveurs où se trouvent les traitements



2 SOLUTIONS

Distribution des données

- on distribue les données sur un nombre important de serveurs afin de stocker de très grands volumes de données
- on envoie les programmes vers les serveurs où se trouvent les données car il est plus efficace de transférer un petit programme sur le réseau plutôt qu'un grand volume de données



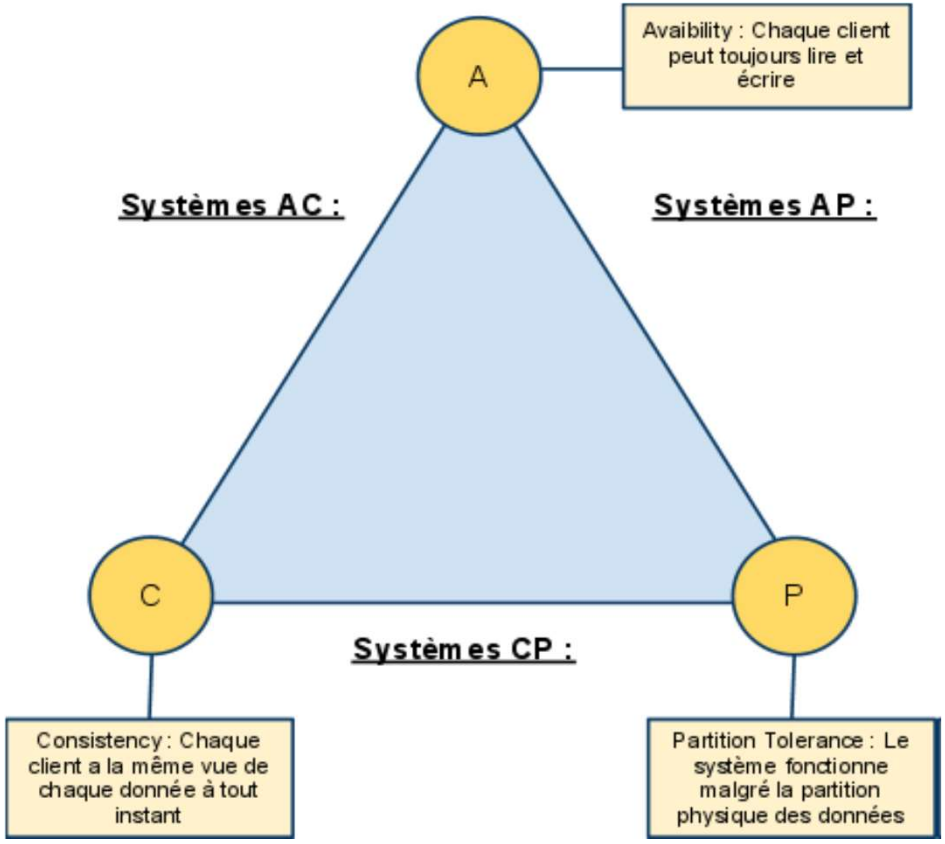
THÉORÈME CAP

- un système de calcul distribué ne peut garantir à un instant donné qu'au plus deux des trois contraintes suivantes :
  - Cohérence (Consistency) : tous les nœuds du système voient exactement les mêmes données au même moment
  - Disponibilité (Availability) : la perte de nœuds n'empêche pas le système de fonctionner correctement → chaque client peut toujours lire et écrire
  - Tolérance au partitionnement (Partition tolerance) : aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement → en cas de découpage en sous-réseaux, chacun doit pouvoir fonctionner de manière autonome

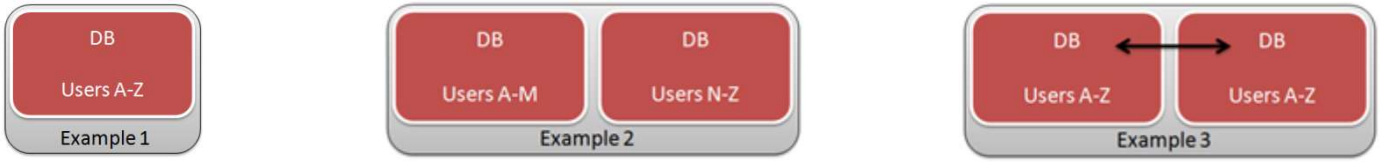
THÉORÈME CAP

- les SGBDR assurent les propriétés de Consistance et de Disponibilité (Availability) → système AC

- les bases de type NoSQL sont des systèmes AP (Disponible et Résistant au partitionnement) ou CP (Cohérent et Résistant au partitionnement)

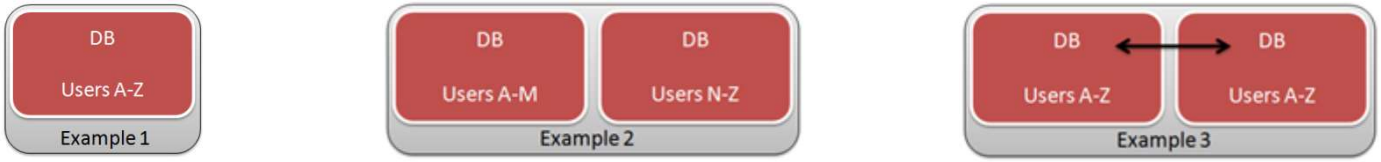


THÉORÈME CAP → EXEMPLE : COHÉRENCE



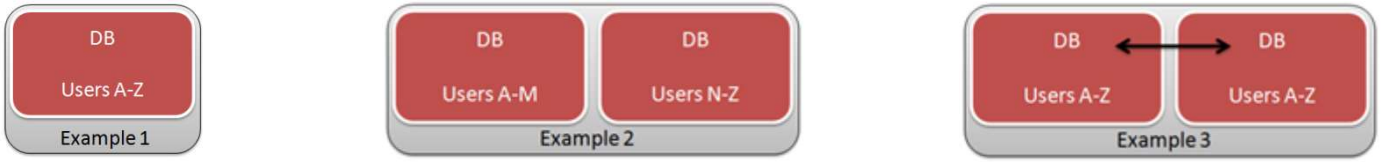
- exemple 1: une instance de la base de données est automatiquement pleinement consistante puisqu'il n'y a qu'un seul nœud qui maintient l'état
- exemple 2 : si 2 serveurs de la base de données sont impliqués, et si le système est conçu de telle sorte que toutes les clés de "a" à "m" sont conservées sur le serveur 1, et les clés "n" à "z" sont conservées sur le serveur 2, alors le système peut encore facilement garantir la cohérence

THÉORÈME CAP → EXEMPLE : COHÉRENCE



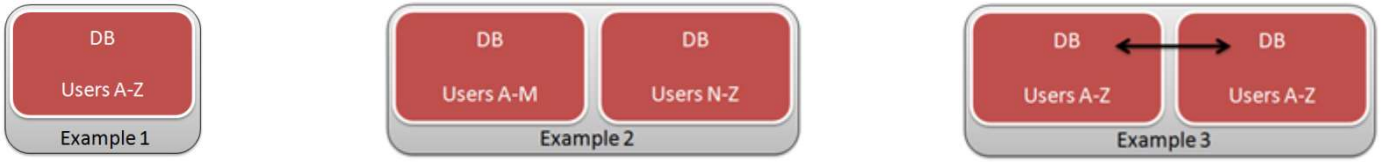
- exemple 3 : on a 2 bases de données, l'une étant la réplique de l'autre. Si une des 2 bases fait une opération d'insertion de ligne, cette opération doit être aussi exécutée dans la seconde base pour que l'opération soit considérée complète
- il y a un temps pendant lequel la cohérence du système n'est pas assurée
- plus le nombre de répliques est grand, plus les performances d'un tel système sont mauvaises

THÉORÈME CAP → EXEMPLE : DISPONIBILITÉ



- exemple 1 : si le seul serveur tombe en panne, on perd 100% des données
- exemple 2 : si un des deux serveurs tombe en panne, on perd 50% des données
- exemple 3 : la réplication d'un serveur sur l'autre assure une disponibilité de 100%
- augmenter le nombre de serveurs avec des répliques augmente directement la disponibilité du système, en le protégeant contre les défaillances matérielles
- les répliques peuvent aider à équilibrer la charge d'opérations concurrentes, notamment en lecture

THÉORÈME CAP → EXEMPLE : RÉSISTANCE AU PARTITIONNEMENT



- les exemples 1 et 2 ne sont pas concernés
- dans l'exemple 3, supposons que les 2 serveurs se trouvent dans 2 datacenters différents
- dans ce cas, si on perd la connexion entre les 2 datacenters, alors les 2 bases ne peuvent plus se synchroniser

THÉORÈME CAP : EXEMPLE CLASSIQUE DU COMPTE BANCAIRE

- une application bancaire doit conserver à tout moment l'état d'un compte sur l'ensemble de ses serveurs sinon elle présentera des enregistrements inconsistants
- si un client retire 50 euros à Lyon, cette action doit être immédiatement répercutée sur tous les serveurs susceptibles d'être sollicités par une autre opération sur le même compte
- si le système ne parvient pas à le faire, cela peut générer des problèmes de soldes faux en cas d'un autre retrait sur le même compte ailleurs
- en cas de panne (soit du réseau, soit d'un des nœuds du réseau), la disponibilité du cluster est perdue. Tous les comptes bancaires doivent alors être gelés jusqu'à ce que le réseau soit de nouveau opérationnel

NOSQL : LES 4 MODÈLES

Clé-Valeur (key value)

- collection de couples constitués d'une clé et d'une valeur
- produits : Redis, AmazonDB, Microsoft Azure Cosmos DB, Memcached

Colonne (wide column)

- modèle clé-valeur enrichi permettant de stocker plusieurs valeurs et associations one-to-many
- produits : Cassandra, Hbase, Microsoft Azure Cosmos DB

NOSQL : LES 4 MODÈLES

Document (document)

- modèle clé-valeur enrichi permettant de stocker une valeur complexe, un document
- chaque document est composé de champs et de valeurs associées
- produits : MongoDB, AmazonDB, Microsoft Azure Cosmos DB, Couchbase, Firebase, RavenDB

Graphe (graph)

- gestion de relations multiples entre les objets basé sur la théorie des graphes
- produits : Neo4J, Microsoft Azure Cosmos DB, Arango DB, OrientDB

Voir un lien pour suivre le classement de ces modèles et bases : https://db-engines.com/en/ranking

NOSQL : LE MODÈLE CLÉ-VALEUR

- la base de données se présente comme un tableau associatif unidimensionnel
- chaque objet de la base est représenté par un couple clé - valeur
- il est identifié par une clé unique qui est le seul moyen d'accès à l'objet
- la clé est le seul moyen de requêtage
- la valeur peut être une simple chaîne de caractères, ou un objet sérialisé
- la structure de l'objet est libre (XML, JSON, ...)chaînes d'octets

NOSQL : LE MODÈLE CLÉ-VALEUR

- absence de structure ou de typage → impact important sur le requêtage : toute l'intelligence portée auparavant par les requêtes SQL devra être portée par l'applicatif qui interroge la base : il faut donc de meilleurs développeurs

- enregistrement 1 : clé 1 – valeur 1
- enregistrement 2 : clé 2 – valeur 2
- …
- enregistrement n : clé n – valeur n

NOSQL : LE MODÈLE CLÉ-VALEUR

- on dispose des 4 opérations CRUD
  - Create (clé,valeur) : crée un couple (clé,valeur)
  - Read (clé) : lit une valeur à partir de sa clé
  - Update (clé,valeur) : modifie une valeur à partir de sa clé
  - Delete (clé) : supprime un couple (clé,valeur) à partir de sa clé

NOSQL : LE MODÈLE CLÉ-VALEUR

- cas d'utilisation :
  - requêtage simple pour des résultats rapides et en temps-réel
  - sessions web
  - fichiers de log
  - gestion de profils utilisateurs
  - données de panier d'achat …

NOSQL : LE MODÈLE CLÉ-VALEUR

- les plus :
  - simple, très performant en lecture et écriture
  - bonne mise à l'échelle horizontale pour les lectures et écritures
  - pas ou peu de maintenance du fait de la simplicité
- les moins :
  - interrogation seulement sur la clé
  - modèle de données trop simple donc pauvre pour les données complexes

NOSQL : LE MODÈLE COLONNE

- les données sont stockées en colonnes, et pas en lignes
- la colonne est l'entité de base représentant un champ de données
- chaque colonne est identifiée par un identifiant unique
- une colonne peut contenir d'autres colonnes
- une colonne contenant d'autres colonnes est nommée super-colonne
- les super-colonnes correspondent à une table de jointure dans le modèle relationnel

NOSQL : LE MODÈLE COLONNE

- ce modèle ressemble à une table dans un SGBD relationnel, à la différence qu'avec une base NoSQL orientée colonne, le nombre de colonnes est dynamique
- en effet, dans une table relationnelle, le nombre de colonnes est fixé lors de la création du schéma de la table et ce nombre reste le même pour tous les enregistrements dans cette table
- par contre, avec ce modèle, le nombre de colonnes peut varier d'un enregistrement à un autre ce qui évite de retrouver des colonnes ayant des valeurs NULL

- enregistrement 1 : clé 1 – nom – prénom – code postal – ville
- enregistrement 2 : clé 2 – nom – email
- enregistrement 3 : clé 3 – nom – prénom – email

NOSQL : LE MODÈLE COLONNE

- on dispose des 4 opérations CRUD
  - Create : crée un enregistrement (la clé et les valeurs de la colonne)
  - Read : lit une ou plusieurs valeurs à partir de la clé
  - Update : modifie une ou plusieurs valeurs à partir de la clé
  - Delete : supprime un enregistrement
- on peut exécuter des requêtes utilisant colonnes et super-colonnes

NOSQL : LE MODÈLE COLONNE

- cas d'utilisation :
  - analyse de données
  - traitement analytique en ligne (OnLine Analytical Processing (OLAP))
  - exploration de données (data mining)
  - entrepôt de données (data warehouse)
  - journalisation d'événements
  - stockage de listes (messages, posts, commentaires, ...)

NOSQL : LE MODÈLE COLONNE

- les plus :
  - bonne mise à l'échelle horizontale
  - nombre de colonnes dynamique : variable d'un enregistrement à un autre
- les moins :
  - ne supporte pas les données structurées complexes
  - maintenance nécessaire pour la modification de la structure en colonne
  - ajout de ligne couteux en temps de réponse

NOSQL : LE MODÈLE DOCUMENT

- la base de données stocke une collection de documents sur le modèle clé-valeur
- la valeur est un document lisible par un humain dans un format semi-structuré hiérarchique (JSON, XML …)
- l'avantage est de pouvoir récupérer via une seule clé un ensemble d'informations structurées de manière hiérarchique
- la même opération dans le monde relationnel impliquerait plusieurs jointures
- un document n'a pas de schéma (schemaless)
- c'est une structure arborescente qui contient une liste de champs
- un champ possède une valeur qui est soit de type simple (entier, caractère, date …) ou peut être elle-même composée de plusieurs couples clés / valeurs

NOSQL : LE MODÈLE DOCUMENT

Clé 1 → Document 1 : un document avec 3 valeurs

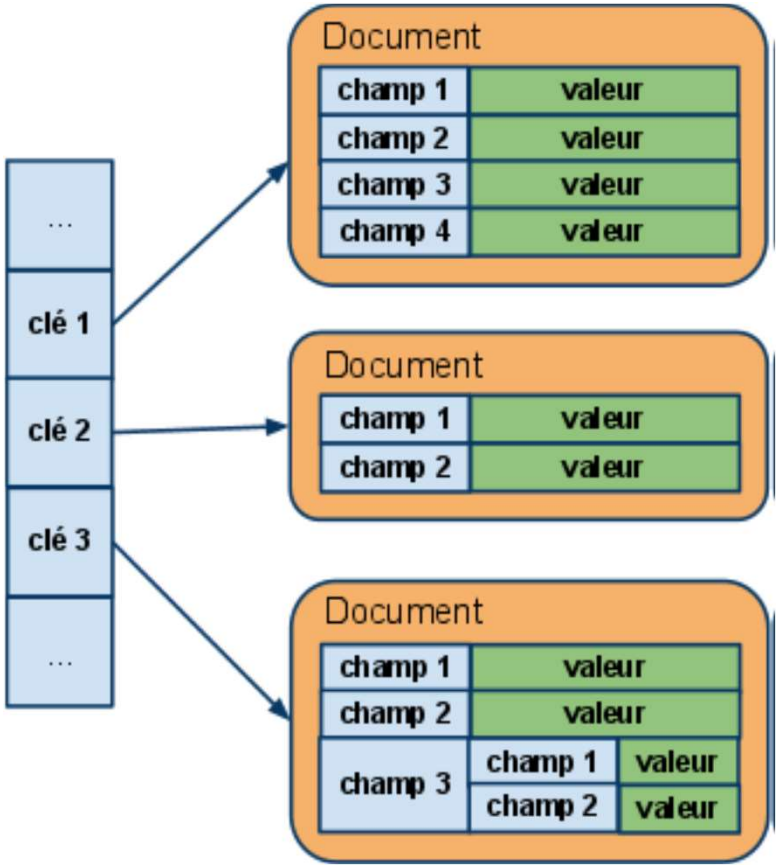
- Champ 1 → Valeur 1
- Champ 2 → Valeur 2
- Champ 3 → Valeur 3
- Champ 4 → Valeur 4

Clé 2 → Document 2 : un document avec seulement 2 valeurs

- Champ 1 → Valeur 1
- Champ 2 → Valeur 2

Clé 3 → Document 3 : un document avec 3 champs dont un (le champ 3) qui contient 2 autres champs (les champs 1 et 2 qui contiennent eux aussi des valeurs)

- Champ 1 → Valeur 1
- Champ 2 → Valeur 2
- Champ 3 → Champ 1 → Valeur
  → Champ 2 → Valeur



NOSQL : LE MODÈLE DOCUMENT

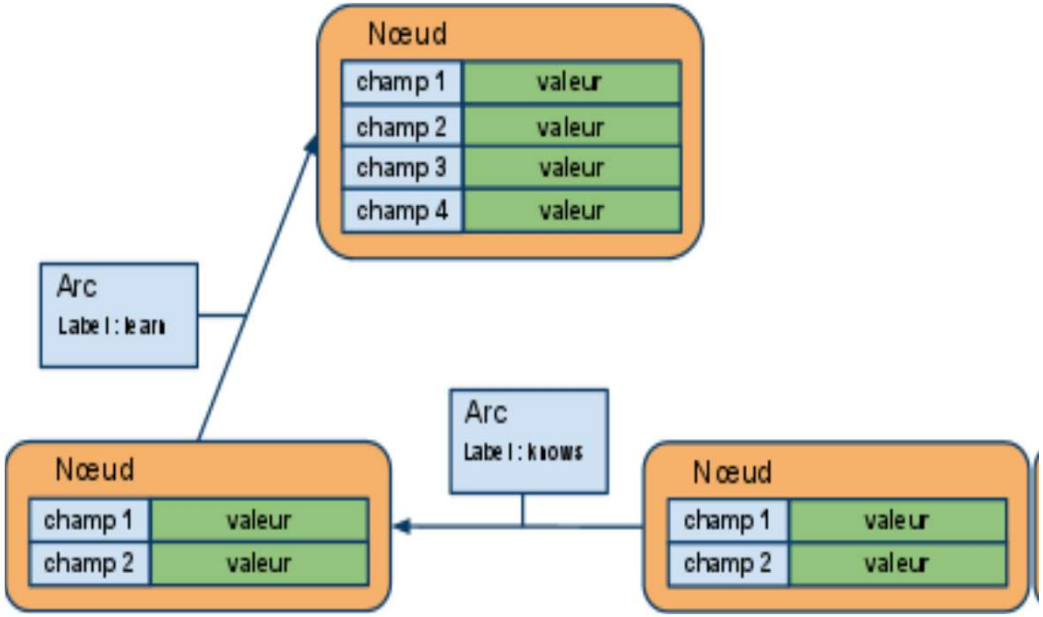
- on dispose des 4 opérations CRUD du modèle clé - valeur
- on peut exécuter des requêtes ou mettre en place des API sur les valeurs des documents
- cas d'utilisation :
  - catalogues de produits
  - web analytique
  - enregistrement d'événements / logs
  - stockage de profils utilisateur …

NOSQL : LE MODÈLE DOCUMENT

- les plus :
  - performances élevées
  - simplicité du modèle
  - possibilité de requêtes plus complexes que dans les autres modèles
  - utilisation de structures imbriquées
  - forte communauté
- les moins :
  - limité pour les interrogations autres que par clé
  - redondances de données

NOSQL : LE MODÈLE GRAPHE

- ce modèle est basé sur la théorie des graphes
- il permet la gestion d'un graphe (orienté ou non), sa modélisation, son stockage et la manipulation de données complexes liées par des relations
- il s'appuie sur les notions de nœuds, relations et propriétés



NOSQL : LE MODÈLE GRAPHE

- cas d'utilisation :
  - informatique décisionnelle
  - internet des objets (Internet of things (IoT))
  - données hiérarchiques (catalogue des produits, généalogie, …)
  - réseaux sociaux
  - réseaux de transport
  - cartographie
  - services de routage et d'expédition …

NOSQL : LE MODÈLE GRAPHE

- les plus :
  - modèle adapté aux situations où il faut modéliser beaucoup de relations
  - forte communauté : nombreux langages et API existants
  - très adapté aux objets organisés en réseau : cartes, réseaux sociaux …
- les moins :
  - modèle beaucoup trop typé car trop basé sur la théorie des graphes
  - donc à ne pas utiliser pour les cas simples

NOSQL : LE MODÈLE GRAPHE