14/03/2019

1

# BASES DE DONNÉES

CM :
*NoSQL*

Vincent COUTURIER
Maitre de conférences – Université Savoie Mont-Blanc
vincent.couturier@univ-smb.fr

2

## NoSQL

- Terme créé par Carl Strozzi
- = « Not Only SQL »
- =« base de données non relationnelle qui peut gérer des données sur un ensemble de serveurs de stockage distribués, est conçue pour être hautement disponible tout en étant capable de monter fortement en charge et prend en charge un schéma de données variable et différents formats de données. Les bases de données NoSQL évitent souvent les transactions ACID (atomiques, cohérentes, isolées et durables) et les jointures de tables afin de parvenir à un débit plus rapide »
- => Approche qui complète les outils existants pour en combler les faiblesses
- Technologies portées par des acteurs comme Google, Facebook ou Twitter
- Objectif : manipuler des volumétries de données très importantes dans un contexte distribué.

3

## SGBD Relationnels

- Règnent en maitres pour le stockage et la manipulation de données depuis 20 ans :
  - SQL est un langage standardisé
  - Technologie SQL moins coûteuse sur le plan de la formation que toute autre technologie
  - Gestion de l'intégrité des données
  - Transactions : indispensables pour les applications de gestion
  - Outils de développement (intégration dans les langages ou frameworks) ou d'exploitation (outils de sauvegarde, monitoring) matures
- Echec des initiatives concurrentes des années 1990/2000 : SGBD Objets, SGBD XML.

4

## Limites du modèle relationnel

- Ce n'est pas forcément le langage SQL qui est en cause
- Faiblesses du modèle relationnel :
  - Absence de gestion d'objets hétérogènes
  - Besoin de déclarer au préalable l'ensemble des champs représentant un objet.
  - => NoSQL permet l'utilisation d'objets hétérogènes apportant une plus grande flexibilité dans les modèles de données ainsi qu'une simplification de la modélisation.
- Coût important du modèle relationnel dans le contexte des systèmes distribués :
  - Nécessité de s'assurer en permanence que les données liées entre elles sont placées sur le même nœud du serveur.
  - Lorsque le nombre de relations au sein d'une base augmente, il devient de plus en plus difficile de placer les données sur des nœuds différents du système.

5

## Limites du modèle transactionnel

- Respect des contraintes **ACID** :
  - **A (Atomicity) :** les mises à jour de la base de données doivent être atomiques : elles doivent être totalement réalisées ou pas du tout. Par exemple, sur 5000 lignes devant être modifiées au sein d'une même transaction, si la modification d'une seule échoue, alors la transaction entière doit être annulée, afin d'éviter l'incohérence des données de la base.
  - **C (Consistency) :** les modifications apportées à la base doivent être valides, en accord avec l'ensemble de la base et de ses contraintes d'intégrité.
  - I (Isolation) : les transactions lancées au même moment ne doivent jamais interférer entre elles, ni même agir sur le fonctionnement de chacune. De fait, les transactions doivent s'enchaîner les unes à la suite des autres, et non de manière concurrentielle.
  - **D (Durability) :** Toutes les transactions sont lancées de manière définitive. Une base ne doit pas afficher le succès d'une transaction, pour ensuite remettre les données modifiées dans leur état initial. Pour ce faire, toute transaction est sauvegardée dans un fichier journal, de sorte que si un problème survient empêchant sa validation complète, la transaction pourra être correctement terminée lors de la disponibilité du système.

6

## Limites du modèle transactionnel

- Dans un contexte centralisé, les contraintes ACID sont aisées à garantir.
- Dans les systèmes distribués, nécessaire de distribuer les traitements de données entre différents serveurs.
  - => difficile de maintenir les contraintes ACID à l'échelle du système distribué entier tout en maintenant des performances correctes.
- La plupart des SGBD NoSQL ont donc été construits en faisant fi des contraintes ACID quitte à ne pas proposer de fonctionnalités transactionnelles.
  - => Ce sera au développeur de s'assurer de l'intégrité des données

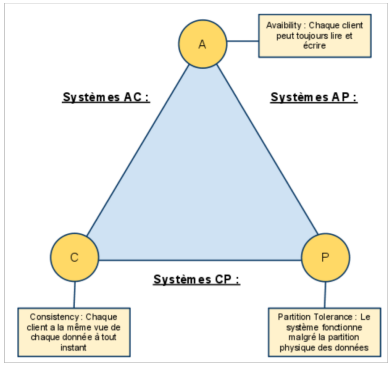
7

## Propriétés des systèmes distribués

- Théorème CAP (Consistency AvailibilityPartition tolerance) :
  - C (Coherence) : tous les nœuds du système voient exactement les mêmes données au même moment
  - A (Availability) : la perte de nœuds n'empêche pas les survivants de continuer à fonctionner correctement
  - P (Partition tolerance) = Résistance au partitionnement => aucune panne moins importante qu'une coupure totale du réseau ne doit empêcher le système de répondre correctement
- CAP stipule qu'il est impossible d'obtenir ces trois propriétés en même temps dans un système distribué et qu'il faut donc en choisir deux parmi les trois.

8

## Théorème CAP



9

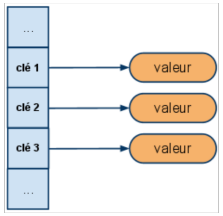
## Types de base NoSQL

- NoSQL != 1 approche unique en mesure d'apporter une solution unique au problème de gestion de grandes masses de données dans un contexte distribué
- 4 grandes catégories d'outils + 1 fourre-tout:
  - Paradigme clé/valeur
  - Bases documentaires
  - Bases orientées colonnes
  - Bases orientées graphes
  - Fourre-tout : Search engines, Time series DB (BD de séries chronologiques), etc.
- Près d'une 100aine d'outils existent aujourd'hui (en comptant ceux des chercheurs)
- Chaque implémentation à tendance à avoir son ensemble particulier de caractéristiques techniques et de comportement.
- Pas de norme aujourd'hui à ce sujet. Quelques tentatives de normalisation de langages de requêtes : CQL, UnQL, N1QL(SQL for JSON).

10

## Bases clé/valeur

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




11

## Bases clé/valeur

- 4 opérations (CRUD):
  - **Create** : créé un nouvel objet avec sa clé → `create(key, value)`
  - **Read** : lit un objet à partir de sa clé → `read(key)`
  - **Update** : met à jour la valeur d'un objet à partir de sa clé → `update(key, value)`
  - **Delete** : supprime un objet à partir de sa clé → `delete(key)`
- Disposent généralement d'une interface HTTP REST permettant de procéder très simplement à des requêtes, et ceci depuis n'importe quel langage de développement.
- + : performances exceptionnellement élevées en lecture et en écriture, scalabilité horizontale considérable.
- - : besoins très simples de requêtes, pas d'intégrité relationnelle des données
- Exemples : Redis (VMWare), Riak (Apache), Voldemort (LikedIn), Oracle NoSQL Database

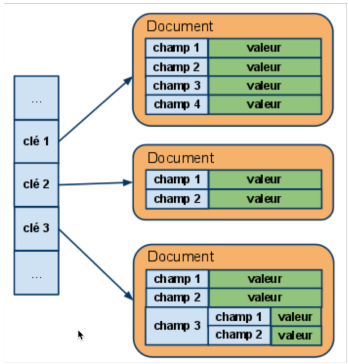
12

## Bases documentaires

- Extension des bases clé/valeur (+ possibilité d'effectuer des requêtes sur le contenu des objets)
- Constituées de collections de documents.
  - Un document est composé de champs et des valeurs associées.
  - Les valeurs associées peuvent aussi être requêtées.
- Les valeurs peuvent être de type simple (entier, chaine de caractères, date, ...), soit composées de plusieurs couples clé/valeur.
- Bases "schemaless" :
  - Pas nécessaire de définir au préalable les champs utilisés dans un document.
  - Les documents peuvent être très hétérogènes au sein de la base.

13

## Bases documentaires




14

## Bases documentaires

- Interface d'accès HTTP REST permettant d'effectuer simplement des requêtes sur la base mais celle-ci est plus complexe que l'interface CRUD des bases clés/valeurs.
- Formats de données JSON et XML sont le plus souvent utilisés afin d'assurer le transfert des données sérialisées entre l'application et la base.
- + : performances élevées, flexibilité du modèle documentaire, très adapté à la gestion des contenus des CMS
- Exemples : MongoDB, RavenDB (plateforme .NET), CouchDB (Apache), Couchbase, DynamoDB (Amazon), Terrastore (Apache), (MySQL, MariaDB PostgreSQL).

15

## Bases orientées colonnes

- Les plus complexes
- Organisation des données adaptée aux traitements d'analyse de données et traitements massifs
- Concepts :
  - Colonne : entité de base représentant un champ de données. Chaque colonne est définie par un couple clé / valeur.
  - Une colonne contenant d'autres colonnes est nommée super-colonne. Correspond à une ligne d'une table. La clé permet d'identifier la super colonne tandis que la valeur est la liste des colonnes qui la compose.
  - Famille de colonnes : conteneur permettant de regrouper plusieurs colonnes ou super-colonnes.
    - Les colonnes sont regroupées par ligne et chaque ligne est identifiée par un identifiant unique.
    - Famille ≈ table dans le modèle relationnel, identifiée par un nom unique.

16

## Bases orientées colonnes

| Column |
| --- |
| name : Firstname |
| value : John |

1 *column* est constituée de 2 champs :
- Name (Key)
- Value

17

## Bases orientées colonnes

***Le concept de supercolumn n'existe pas dans tous les outils.***

| SuperColumns | |
|---|---|
| **Key** | **Value** |
| person1 | Column<br>**Name** : **Value**<br>firstName : John<br>lastName : Calagan |
| person2 | Column<br>**Name** : **Value**<br>firstName : George<br>lastName : Truffe |

*SuperColumn : clé* (permet d'identifier la ligne), *value* (liste des colonnes qui la compose)

2 *SuperColumn*. Chaque *SuperColumn* est ici constituée de 2 *Column* (fistname et lastname)

18

## Bases orientées colonnes

| ColumnFamily | |
|---|---|
| **Key** | **Value** |
| AddressBook | SuperColumns<br>**Key** : **Value**<br>person1 : Column (**Name** : **Value** ; firstName : John ; lastName : Calagan)<br>person2 : Column (**Name** : **Value** ; firstName : George ; lastName : Truffe) |

19

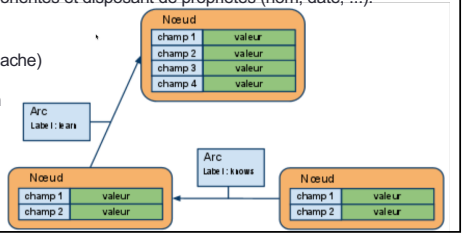
## Bases orientées colonnes

- Les super-colonnes situées dans les familles de colonnes sont souvent utilisées comme les lignes d'une table de jointure dans le modèle relationnel.
- Comparativement au modèle relationnel, les bases orientées colonnes offrent plus de flexibilité :
  - Possible d'ajouter une colonne ou une super colonne à n'importe quelle ligne d'une famille de colonnes à tout instant.
- Exemples : Cassandra (Apache), Amazon SimpleDB, Google BigTable, Hbase (Apache), Spark SQL (Apache), Elasticsearch.

20

## Bases orientées graphes

- Utilisée pour les problématiques liées aux réseaux (cartographie, relations entre personnes)
- Principes :
  - Utilisation d'un moteur de stockage pour les objets (qui se présentent sous la forme d'une base documentaire, chaque entité de cette base étant nommée nœud).
  - Mécanisme permettant de décrire les arcs (relations entre les objets), ceux-ci étant orientés et disposant de propriétés (nom, date, ...).
- Exemples :
  - Neo4J
  - OrientDB (Apache)
  - Info Grid
  - Infinite Graph



21

## Search engines (moteurs de recherche)

- Outils dédiés à l'indexation et la recherche des données.
- Fournissent un moteur de recherche distribué et multi-entité souvent via une interface REST.
- Intègrent souvent un SGBD NoSQL (souvent orienté-colonnes ou orienté-documents) => il s'agit plus d'une surcouche.
- Exemple : Elasticsearch, Splunk

22

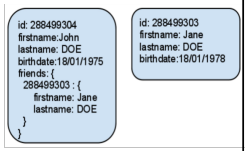
## Time series DB

- Conçues pour gérer des données de séries chronologiques ou des tableaux de nombres indexés par heure (une date / heure ou une plage de date / heure).
- Exemples de time series :
  - Relevés périodiques de températures
  - Série chronologique de consommation d'énergie,
  - Cours d'actions (bourse)
  - Etc.
- Dédiés à gérer une logique complexe ou des règles métier et un volume de transactions élevé pour les données de séries temporelles
- Exemples : Riak-TS, InfluxDB, eXtremeDB.
- Il existe aussi des SGBD relationnels (voire objet) dédiés à la gestion des TS comme Informix-TS

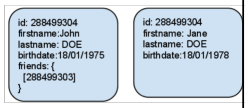
23

## Conception des bases NoSQL

- Modèle relationnel : utilisation des données sous forme normalisée et jointures entre les tables au besoin.
- NoSQL : dénormalisation des données
  - Dans certains cas, les données seront embarquées dans la donnée parente. C'est notamment le cas lorsque le volume de la données est faible et que celle-ci est peu mise à jour. + : rapidité de lecture, - : mises à jour.




  - Dans les autres cas, on stockera simplement les identifiants permettant d'accéder aux données comme on le ferait dans le modèle relationnel. La différence réside dans le fait que c'est le plus souvent à l'application cliente de procéder par la suite aux jointures. + : rapidité de mise à jour , - : lecture moins rapide.




24

## Quelle sélection ?

- Meilleures solutions :
  - Base clé/valeur : Redis, Oracle NoSQL Database, Riak
  - Documentaires : MongoDB, CouchDB / Couchbase (http://www.couchbase.com/couchbase-vs-couchdb)
  - Orientés colonnes : Cassandra
  - Graphes : Neo4J
  - Azure Cosmos DB (gère les 4 types !!)
  - Search engines : Elasticsearch
  - Times series DB : InfluxDB
- Critères :
  - Maturité la plus grande,
  - Potentiel le plus important
  - Meilleure intégration aux outils de développement actuels.
- Sources :
  - « Les bases de données NoSQL » 2ème édition (Rudi Bruchez, Editions Eyrolles)
  - http://blog.3pillarglobal.com/selection-criteria-nosql-database

25

## Bilan (d'après Oracle)

- Quand les utiliser ?
  - Data Capture : capturer et interroger les données entrantes provenant d'une multitude de points de données, tels que la surveillance du réseau, les réseaux de capteurs dans une usine automatisée.
  - Data Service : des services orientés Web, de haute-performance et axés sur le client tels qu'Amazon, LinkedIn ou Facebook.
- Avantages :
  - Facilité pour augmenter la capacité de calcul et de stockage en ajoutant de nouveaux serveurs,
  - Requêtes simples et rapides
  - Approche flexible et aisée pour gérer le schéma.
- Inconvénients :
  - En n'utilisant pas SQL, les systèmes de base de données NoSQL perdent la capacité de faire des requêtes très structurées avec une certitude mathématique.
  - Incapacité d'exécuter des requêtes complexes
  - Incapacité à effectuer des jointures multi-tables,
  - Support transactionnel limité
  - Nécessité d'apprendre une nouvelle approche de technologie de base de données

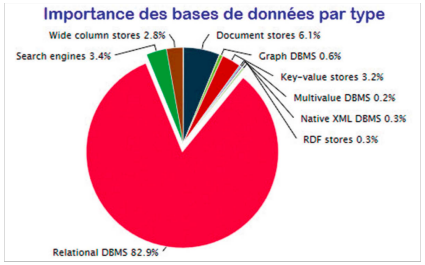
26

## Bilan (d'après moi)

- Ne remplacera (sans doute) jamais les SGBD relationnels.
- Impossible à utiliser pour des applications de gestion (ERP, progiciels comptables, de paye, etc.).
  - D'ailleurs Facebook, Google, Twitter et LinkedIn utilisent WebScaleSQL
- A terme, ajout de certaines caractéristiques NoSQL aux bases SQL
- Orienté-colonnes :
  - Très intéressant pour stocker des lignes de données avec valeurs nulles nombreuses
  - Utilisé dans le cadre du projet BATIMETRE de l'INES
    - -> passage en cours à InfluxDB
- Graphes :
  - Très intéressant pour stocker des graphes de données, par exemple des graphes de concepts (ontologies) et le poids des relations entre concepts : outils d'enrichissement de résultats de requêtes, etc.

27

## Bilan




28

## Bilan

14/03/2019

29

# Bilan

343 systems in ranking, January 2019

| Rank Jan 2019 | Rank Dec 2018 | Rank Jan 2018 | DBMS | Database Model | Score Jan 2019 | Score Dec 2018 | Score Jan 2018 |
|---|---|---|---|---|---|---|---|
| 1. | 1. | 1. | Oracle | Relational DBMS | 1268.84 | +14.39 | -73.11 |
| 2. | 2. | 2. | MySQL | Relational DBMS | 1154.27 | -6.98 | -145.44 |
| 3. | 3. | 3. | Microsoft SQL Server | Relational DBMS | 1040.26 | -0.08 | -107.81 |
| 4. | 4. | 4. | PostgreSQL | Relational DBMS | 466.11 | +5.48 | +79.93 |
| 5. | 5. | 5. | MongoDB | Document store | 387.18 | +8.57 | +56.24 |
| 6. | 6. | 6. | IBM Db2 | Relational DBMS | 179.85 | -0.90 | -10.43 |
| 7. | 7. | ↑ 9. | Redis | Key-value store | 149.01 | +2.19 | +25.88 |
| 8. | 8. | ↑ 10. | Elasticsearch | Search engine | 143.44 | -1.26 | +20.89 |
| 9. | 9. | ↓ 7. | Microsoft Access | Relational DBMS | 141.62 | +2.10 | +14.92 |
| 10. | 10. | ↑ 11. | SQLite | Relational DBMS | 126.80 | +3.78 | +12.54 |
| 11. | 11. | ↓ 8. | Cassandra | Wide column store | 122.98 | +1.17 | -0.89 |
| 12. | 12. | ↑ 15. | Splunk | Search engine | 81.43 | -0.76 | +17.42 |
| 13. | ↑ 14. | ↑ 17. | MariaDB | Relational DBMS | 78.82 | +1.56 | +20.52 |
| 14. | ↓ 13. | ↓ 12. | Teradata | Relational DBMS | 76.19 | -2.98 | +3.56 |
| 15. | 15. | ↑ 18. | Hive | Relational DBMS | 69.91 | +2.52 | +14.42 |
| 16. | 16. | ↓ 14. | Solr | Search engine | 61.48 | +0.13 | -2.89 |
| 17. | 17. | ↓ 16. | HBase | Wide column store | 60.39 | +0.38 | -1.24 |
| 18. | 18. | ↑ 19. | FileMaker | Relational DBMS | 57.15 | +0.50 | +1.95 |
| 19. | 19. | ↑ 20. | SAP HANA | Relational DBMS | 56.64 | +0.33 | +10.47 |
| 20. | ↑ 21. | ↑ 22. | Amazon DynamoDB | Multi-model | 55.09 | +0.80 | +17.17 |
| 21. | ↓ 20. | ↓ 13. | SAP Adaptive Server | Relational DBMS | 55.04 | -0.78 | -10.42 |
| 22. | 22. | ↓ 21. | Neo4j | Graph DBMS | 46.80 | +1.24 | +6.83 |
| 23. | 23. | 23. | Couchbase | Document store | 34.59 | -1.00 | +3.16 |
| 24. | 24. | ↑ 25. | Memcached | Key-value store | 29.54 | -0.07 | +1.39 |
| 25. | 25. | ↑ 26. | Microsoft Azure SQL Database | Relational DBMS | 27.20 | +0.17 | +4.17 |

30

# Evolution

- NewSQL :
  - = « catégorie de SGBD relationnelles modernes qui cherchent à fournir la même puissance évolutive que les systèmes NoSQL pour le traitement transactionnel en ligne (lecture-écriture), tout en maintenant les propriétés ACID d'un système de base de données traditionnel. »
  - => Combiner le meilleur de SQL et NoSQL
  - « NoSQL Is Out And NewSQL Is In » Google
  - Exemples :
    - MemSQL
    - **Google Spanner** (successeur de BigTable) : http://en.wikipedia.org/wiki/Spanner_%28database%29
    - **Cockroach** ("copie" de Google Spanner) https://www.cockroachlabs.com/docs/stable/sql-feature-support.html#
    - VoltDB
    - **PostqreSQL**
    - **MariaDB**

31

# MONGODB

32

# Présentation

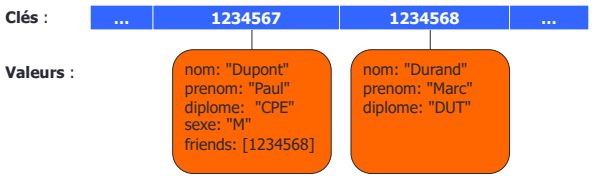
- Base NoSQL Orientée documents
- Licence AGPL
- Développée en C++
- Classée 6e dans le classement des SGBD les plus populaires tous types confondus et 1ère pour les SGBD NoSQL (source : developpez.com, db-engines.com)
- Certains frameworks implémentent un support de MongoDB sous forme de plugins : CakePHP, Zend, Doctrine2 (Object Document Mapper), Rails (MongoMapper), Spring (Spring MongoDB), etc.
- Utilisée par beaucoup d'organisations comme MTV, Disney, SourceForge, Foursquare, etc.

33

## Documents

- Les documents sont regroupés sous forme de collections
- 1 collection = 1 table relationnelle
- Chaque document est stocké au format BSON (JSON binaire), plus compacte que le format JSON
- Chaque document dispose d'une clé (key) unique permettant de l'identifier dans la collection

**Collection** : users




34

## Architecture

- 4 modes distincts de fonctionnement :
  - serveur seul
  - réplication maitre / esclave
  - réplication via ReplicaSet (ensemble de serveurs traitant les mêmes données)
  - partitionnement des données selon leur clé (key) via sharding (partitionnement des données sur plusieurs serveurs)

35

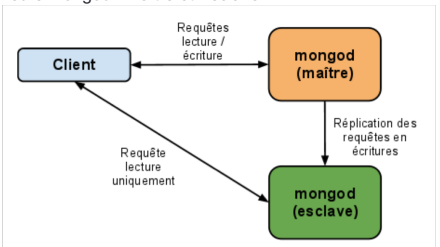
## Architecture

- Serveur seul :
  - un seul processus nommé mongod est utilisé et traite directement les données issues des requêtes du client.


36

## Architecture

- Réplication maître/esclave
  - 2 serveurs mongod : Maitre et Esclave



37

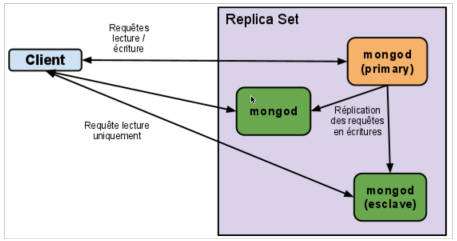
# Architecture

- Réplication maître/esclave
  - Données écrites sur le maître uniquement.
  - Maître réplique l'ensemble des écritures, avec une certaine latence.
  - Le client peut choisir de lire soit sur l'esclave s'il accepte les erreurs liées à la latence de réplication, soit sur le maître dans le cas contraire.
  - - : faible tolérance à la panne, car en cas de panne du serveur maître, l'application cliente est incapable d'écrire sur la base (nécessaire de redémarrer le serveur esclave en mode maitre)
  - + : mode intéressant car permet aux développeurs de tester les effets de bords liés à la réplication et donc d'assurer que l'application développée sera à même de fonctionner avec de telles contraintes. Cela est d'autant plus vrai que le délai de réplication minimale peut être fixé manuellement afin de simuler une latence réseau allant jusqu'à quelques secondes.

38

# Architecture

- Réplication Replica Set




39

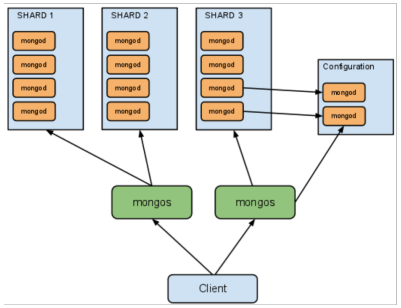
# Architecture

- Réplication Replica Set
  - Principe du maitre-esclave mais plus avancé.
  - Nécessite au moins trois machines physiques.
  - Permet d'éviter la présence d'un point de défaillance unique au sein de l'infrastructure :
    - On ajoute n serveurs au Replica Set. Chaque serveur dispose d'une priorité
    - Un des serveurs est considéré comme le nœud primaire. On peut voir le rôle de nœud primaire comme celui du serveur maître dans le modèle maître / esclave : il s'agit du nœud qui sera en charge des écritures et des réplications vers les autres serveurs.
    - En cas de défaillance du nœud primaire, un nouveau nœud au sein du Replica Set est choisi et devient nœud primaire. Dans le processus d'élection d'un nouveau nœud sa priorité est prise en compte. Cela permet de privilégier certains nœuds, typiquement les plus puissants.

40

# Architecture

- Partitionnement via Sharding



41

# Architecture

- Partitionnement via Sharding
  - Dans les 2 systèmes précédents, la lecture repose sur plusieurs serveurs, mais l'écriture sur 1 seul
  - Sharding :
    - Un shard contient un ou plusieurs serveurs : un serveur seul, un couple maitre/esclave ou un replica set. L'utilisation d'un replica set est conseillée en production pour une meilleure sécurité
    - Pour partitionner une collection sur plusieurs shards on définit sur celle-ci une shard key. Il s'agit d'une liste de champs du document qui permettront au système de définir sur quel shard un document doit être stocké. Les données seront réparties automatiquement de manière la plus homogène possible entre les différents shards.

42

# Architecture

- Partitionnement via Sharding
  - 2 éléments sont ajoutés :
    - Serveur **mongos** :
      - Un ou de plusieurs serveurs dont le rôle et de router les requêtes du client vers le shard approprié. Conseillé de disposer d'au moins 2 serveurs pour éviter que ce serveur soit un point de défaillance unique.
      - Par ailleurs, ces serveurs sont susceptibles de recevoir une charge significative. En effet, lors de requêtes faisant intervenir plusieurs shards, ils sont responsables entre autres de l'agrégation des résultats.
    - Configuration **mongod** :
      - Serveurs stockant la configuration du sharding, utilisés par les autres instances afin d'être en mesure de déterminer sur quel shard les opérations doivent être effectuées.
      - Conseillé d'utiliser plusieurs serveurs afin d'éviter l'apparition d'un point de défaillance unique.





43

# Qualité de service

- Cohérence des données (système CP) :
  - Ecritures sont toujours cohérentes car effectuées sur le même serveur
  - Cohérence des données en lecture : doit être assurée par le développeur qui peut choisir de lire sur un serveur esclave quitte à ce que celui-ci n'ait pas pu répliquer toutes les données écrites depuis le serveur maitre
- Transactions :
  - Pas de mécanisme transactionnel
- Tolérance à la panne :
  - Replica Set : permettent au système de résister face à la disparition de l'un de ses nœuds de stockage.
  - Système légèrement plus centralisé que certains autres systèmes NoSQL lorsqu'on utilise le sharding (présence des nœuds de routage) mais ceux-ci peuvent être dupliqués sans problème.

44

# Qualité de service

- Sauvegardes :
  - Via **mongodump** (export de la base de données)
  - Restauration via **mongorestore**

45

## Développement / API

- Outil mature.
- Deux modes d'accès principaux :
  - Driver spécifique au langage de programmation permettant une communication directement en BSON
  - API REST disponible sous forme d'un serveur HTTP embarqué optionnel
- Langages supportés :
  - Nombreux au travers de drivers : Java, Python, PHP, Ruby, C++, C#/.NET, JavaScript,…

46

## Avantages / inconvénients

- + :
  - 1 des outils NoSQL les plus matures
  - Grand nombre de bibliothèques supportées par son éditeur (permettant son intégration dans un grand nombre de langages de programmation)
  - Intégration de MongoDB dans un nombre croissant de frameworks parmi les plus populaires.
  - Paradigme documentaire simple à prendre en main
  - Construction aisée de requêtes complexes
  - API bien documentée
- - :
  - Nécessité d'avoir un nombre important de serveurs dès lors que l'on souhaite distribuer les données
  - Temps de configuration nécessaire important surtout pour les modes Replica Set et Sharding



47

## FONCTIONNALITES NOSQL DE **POSTGRESQL**

48

## Présentation

- PostgreSQL intègre des fonctionnalités NoSQL depuis la version 9.3 :
  - Version 9.3 : Type de données JSON ⇔ Clé/valeur
    - 1 clé -> 1 document JSON
  - Version 9.4 :Type de données JSONB ⇔ orienté-documents
    - 1 clé -> 1 Document JSON (stocké au format binaire)
    - Chaque champ du JSON est adressable par une clé.
- Il est ainsi possible :
  - D'ajouter un champ supplémentaire dans une table qui contiendra un document JSON.
  - Permet de mixer relationnel et NoSQL, et même relationnel-objet et NoSQL

14/03/2019

49

## Exemple : type de données JSON (clé/valeur)

```
CREATE TABLE ClientJSON (
        id integer,
        client json
);
insert into ClientJSON values (1, '{
        "nom" : "DURAND",
        "prenom" : "Jean",
        "societe" : false,
        "adresse" : "12 rue de la Gare, 74000, Annecy",
        "categories" : ["EURL", "Informatique"]
}');
insert into ClientJSON values (2, '{
        "raison sociale" : "INFORMATIQUE SA",
        "societe" : true,
        "adresse" : "9 rue de la Gare, 74000, Annecy",
        "categories" : ["Grande entreprise", "Informatique"]
}');
```

50

## Exemple : type de données JSON (clé/valeur)

```
select * from ClientJSON;
```

| id integer | client json |
|---|---|
| 1 | {"nom":"DURAND","adresse":"12 rue de la Gare, 74000, Annecy","prenom":"Jean","categories":["EURL","Informatique"],"societe":false} |
| 2 | {"raison sociale":"INFORMATIQUE SA","adresse":"9 rue de la Gare, 74000, Annecy","societe":true,"categories":["Grande entreprise","Informatique"]} |

```
select * from ClientJSON where id=1;
select * from clientJSON where client::text like '%DURAND%';
```

Recherche dans le contenu JSON (le JSON doit être casté en TEXT)

| id integer | client json |
|---|---|
| 1 | {"nom":"DURAND","adresse":"12 rue de la Gare, 74000, Annecy","prenom":"Jean","categories":["EURL","Informatique"],"societe":false} |

```
SELECT client->'nom', client->'prenom' FROM clientJSON WHERE
client @> '{"nom": "DURAND"}';
```

Data Output | Explain | Messages | Query History

```
ERROR:  operator does not exist: json @> unknown
```

=> Impossible de rechercher dans le contenu d'un type JSON

51

## Exemple : type de données JSON (clé/valeur)

- Plutôt que de gérer un ID, on peut aussi utiliser les fonctionnalités orientées objet de PostgreSQL.

```
CREATE TABLE ClientJSON (
        client json
) WITH OIDS;
```

52

## Exemple : type de données JSONB (orienté document)

```
CREATE TABLE ClientJSONB (
        id integer,
        client jsonb
);
insert into ClientJSONB values (1, '{
        "nom" : "DURAND",
        "prenom" : "Jean",
        "societe" : false,
        "adresse" : "12 rue de la Gare, 74000, Annecy",
        "categories" : ["EURL", "Informatique"]
}');
insert into ClientJSONB values (2, '{
        "raison sociale" : "INFORMATIQUE SA",
        "societe" : true,
        "adresse" : "9 rue de la Gare, 74000, Annecy",
        "categories" : ["Grande entreprise", "Informatique"]
}');
```

53

# Exemple : type de données JSONB (orienté document)

```
select * from ClientJSONB;
```

| id integer | client json |
|---|---|
| 1 | {"nom":"DURAND","adresse":"12 rue de la Gare, 74000, Annecy","prenom":"Jean","categories":["EURL","Informatique"],"societe":false} |
| 2 | {"raison sociale":"INFORMATIQUE SA","adresse":"9 rue de la Gare, 74000, Annecy","societe":true,"categories":["Grande entreprise","Informatique"]} |

```
SELECT client->'nom', client->'prenom' FROM clientJSONB WHERE
client @> '{"nom": "DURAND"}';
```

| ?column? jsonb | ?column? jsonb |
|---|---|
| DURAND | Jean |

=> Possible de rechercher dans le contenu d'un type JSONB

Il est nécessaire de renommer les colonnes :

```
SELECT client->'nom' as nom, client->'prenom' as prenom FROM
clientJSONB WHERE client @> '{"nom": "DURAND"}';
```

54

# Exemple : type de données JSONB (orienté document)

- Comme l'ID est adressable, il n'est pas nécessaire de gérer un ID en dehors du document JSON

```
CREATE TABLE ClientJSONB2 (
        client jsonb
);
insert into ClientJSONB2 values ('{
        "id" : 1,
        "nom" : "DURAND",
        "prenom" : "Jean",
        "societe" : false,
        "adresse" : "12 rue de la Gare, 74000, Annecy",
        "categories" : ["EURL", "Informatique"]
}');
```

55

# Exemple : type de données JSONB (orienté document)

- Il est possible d'ajouter (et même grandement conseillé) un index sur le champ de type JSONB afin d'améliorer les performances de la recherche :

```
SELECT client->'nom' as nom, client->'prenom' as prenom
FROM clientJSONB WHERE client @> '{"nom": "DURAND"}';
```

- Sans index :

| QUERY PLAN text |
|---|
| Seq Scan on clientjsonb (cost=0.00..25.88 rows=1 width=64) |
| Filter: (client @> '{"nom": "DURAND"}'::jsonb) |

- Avec index GIN (type d'index spécifique au JSONB) :

```
CREATE INDEX ix_clientJSONB_client ON clientJSONB USING
gin(client);
```

| QUERY PLAN text |
|---|
| Seq Scan on clientjsonb (cost=0.00..1.03 rows=1 width=64) |
| Filter: (client @> '{"nom": "DURAND"}'::jsonb) |

- **ATTENTION, le plan d'exécution est le même même si l'index est utilisé.**
- A chaque nouvelle version de PostgreSQL, les performances de l'index GIN sont améliorées.