Bases de données avancées : de SQL à NoSQL à NewSQL

christophe.cerin@univ-paris13.fr

2023 - 2024

Définitions (informelles) pour planter le décor

Quelques définitions autour des notions SQL

SQL (*Structured Query Language)* : langage d'interrogation normalisé de bases de données relationnelles – Les données sont organisées via des tables – Les données sont structurées – Au delà du langage de manipulation :

- *le langage de définition des données* permet de créer et de modifier l'organisation des données dans la base de données ;
- le *langage de contrôle de transaction* permet de commencer et de terminer des transactions ;
- le *langage de contrôle des données* permet d'autoriser ou d'interdire l'accès à certaines données à certaines personnes.

Nota : le langage SQL n’est pas un langage de programmation comme Java, Python… mais on y trouve quand même des opérations comme insérer, supprimer, modifier…

Quelques définitions autour des notions NoSQL

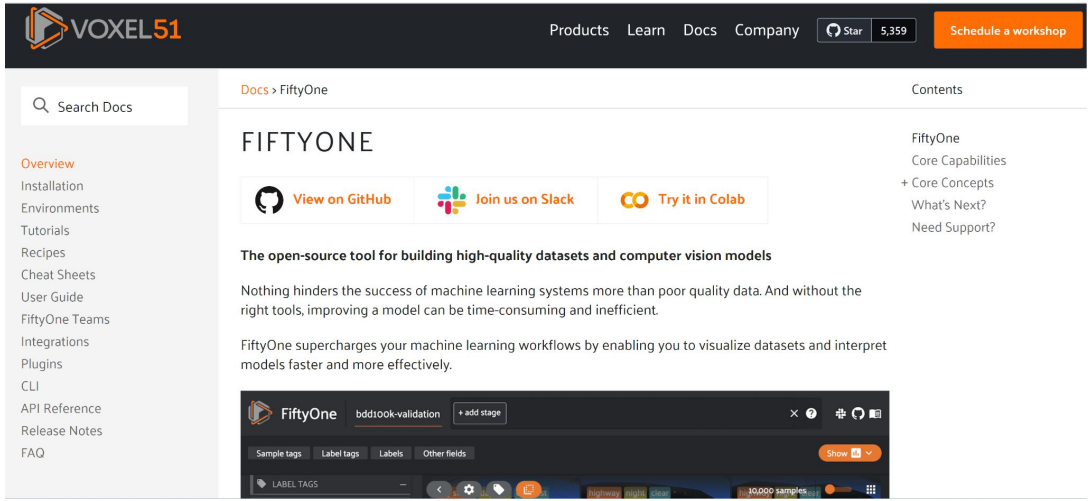
Une définition : NoSQL = Not Only SQL ou alors NoREL = Not Only RELational

Notion qui est apparue avec les GAFAM / BATX à travers des projets comme Google (BigTable), Amazon (DynamoDB), LinkedIn (Voldemort), Facebook (Cassandra puis HBase), SourceForge.net (MongoDB), Ubuntu One (CouchDB), Baidu (Hypertable) ;

Besoins : 1) gérer un cluster de machines ou les BDs sont distribuées + 2) gérer des données stockées à plat (avec beaucoup moins de structure que le relationnel)

Un modèle typique en NoSQL est le système clé-valeur (l'unique data structure). Applications : analyses temps-réel, statistiques, du stockage de logs (journaux), etc

Exemple de besoin dans un cadre industriel (½)

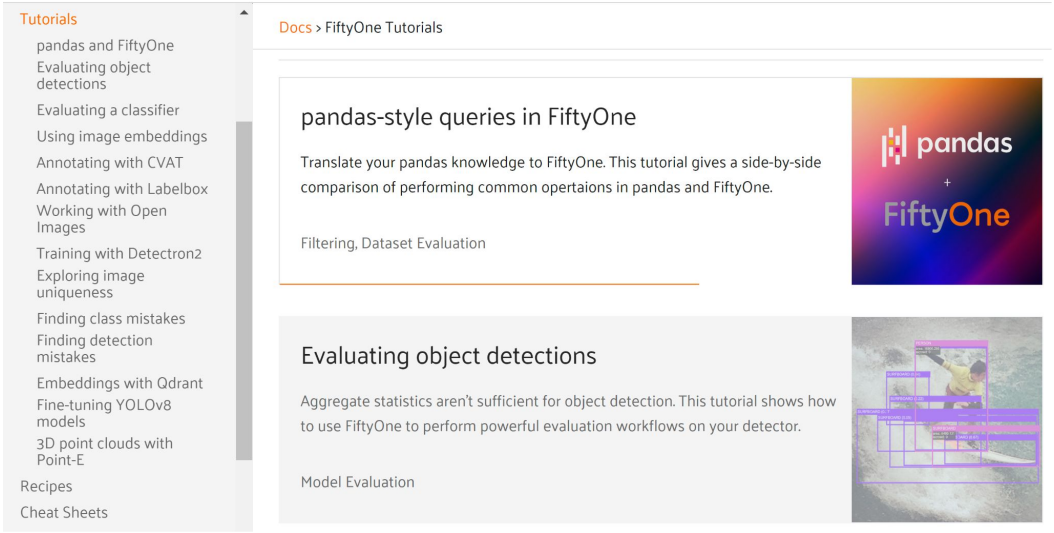


FITYONE, : L'outil open-source pour la construction d'ensembles de données et de modèles de vision par ordinateur de haute qualité ;

Facilement installable via pip (l'outil est un outil Python) ;

Le backend pour la gestion des images est MongoDB ;

Exemple de besoin dans un cadre industriel (2/2)



Database utilities

https://docs.voxel51.com/api/fiftyone.core.odm.database.html

`to_dict`([extended])

Serializes this document to a BSON/JSON dictionary.

`to_json`([pretty_print])

Serializes the document to a JSON string.

`to_mongo`(*args, **kwargs)

Return as SON data ready for use with MongoDB.

Quelques définitions autour des notions NewSQL

NewSQL est une classe de systèmes de gestion de bases de données relationnelles qui cherchent à offrir l'évolutivité des systèmes NoSQL pour les charges de travail de traitement des transactions en ligne (OLTP) tout en maintenant les garanties **ACID** d'un système de base de données traditionnel.

Mot clé ici : ACID – voir plus loin.

Idée : tirer le meilleur des différents mondes i.e, être en mesure de gérer des données très structurées, moins structurées, pas du tout structurée au sein d'un même système. C'est l'idée d'un SGBD universel (pour tout type d'application).

Nos objectifs (surtout les votre)

Expérimenter avec des gestionnaires NoSQL (Redis, MongoDB) et NewSQL (CockroachDB) + faire des parallèles / comparaisons avec le modèle SQL ;

En bref : se familiariser avec ce genre d'outils ;

Heureux étudiants : accès, à une machine virtuelle où les outils sont déjà préinstallés ! (voir plus loin)

Votre travail : réaliser les TPs proposés, les résumer dans un document (compte rendu de TP), restituer votre travail (exposé court devant jury).

Retour sur ACID à des fins de classification

ACID

Le terme générique ACID concerne les transactions (séquences d'opérations/requêtes) :

- A*tomicité* : Une transaction s'effectue entièrement ou pas du tout;
- C*ohérence* : Le contenu d'une base doit être cohérent au début et à la fin d'une transaction ;
- I*solation* : Les modifications d'une transaction ne sont visibles/modifiables que quand celle-ci a été validée ;
- D*urabilité* : Une fois la transaction validée, l'état de la base est permanent (non affecté par les pannes ou autre) ;

Question : peut-on et doit-on être en mesure d'appliquer ces propriétés dans le cadre du NoSQL et du NewSQL ?

La question précédente est difficile...

Propriétés applicables ? Exemple d'une transaction de cinq opérations (lecture/écriture) ⇒ une synchronisation entre cinq serveurs pour garantir l'atomicité, la cohérence et l'isolation. Au final, cela se traduit par des latences dans les transactions (en cours et en concurrence car le monde n’est pas synchrone) ⇒ pas tolérable lorsque justement on veut éviter ces latences en distribuant les calculs sur des serveurs.

Attention, cela s'aggrave : on fait de la réplication ⇒il va falloir synchroniser toutes mises à jour avec tous les réplicas de la donnée !

OVH et l'incendie de 2021 à Strasbourg (pourquoi répliquer ?)

OVH ajoute par ailleurs avoir augmenté la cadence de fabrication de ses serveurs de **2500 à 3000 serveurs par jour pour rétablir le plus grand nombre de sites possibles dans des délais raisonnables**. OVH précise que si vous avez opté pour le Backup FTP vos données sont sans doute récupérables dans leur intégralités, ces sauvegardes étant stockées dans un autre site. Il reste à évaluer quelles seront les conséquences de l'incident pour l'hébergeur français.

A priori, il est permis de penser que cet incident finira par être oublié malgré sa gravité, surtout si OVH prend les décisions nécessaires, comme l'assure Octave Klaba, pour que ce genre d'accidents ne se reproduise jamais. Il est cependant difficile d'ignorer la **montée en puissance de Amazon Web Services, Microsoft Azure et Google Cloud dans l'hébergement**. Des firmes qui ont une taille mondiale et un grand nombre d'installations redondantes près des clients, ce qui les expose sans doute un peu moins à ce genre de risques.

Les propriétés BASE pour les systèmes NoSQL

Du coup, les propriétés BASE ont été proposées pour caractériser les systèmes de gestion NoSQL :

- Ba*sically Available* : quelle que soit la charge de la base de données (données/requêtes), le système garantie un taux de disponibilité de la donnée ;
- S*oft-state* : La base peut changer lors des mises à jour ou lors d'ajout/suppression de serveurs. La base NoSQL n'a pas à être cohérente à tout instant ;
- E*ventually consistent* : À terme, la base atteindra un état cohérent ;

Idée : relâcher certaines contraintes, telles que la synchronisation des réplicas, pour favoriser l'efficacité.

Efficacité : pourquoi des dictionnaires en NoSQL

En informatique, un tableau associatif (aussi appelé dictionnaire ou table d'association) est un type de données associant à un ensemble de *clefs*, un ensemble correspondant de *valeurs*.

Deux structures de données se montrent efficaces pour représenter les tableaux associatifs : la table de hachage et l'arbre équilibré. Les avantages et inconvénients respectifs de ces deux solutions sont les suivants :

- Les tables de hachage ont une meilleure complexité en moyenne pour l'insertion et la recherche (O(1)), alors que les arbres équilibrés ont une meilleure complexité dans le pire des cas (O(log(*n*)), au lieu de O(*n*) pour certaines tables de hachage) ;
- Les tables de hachage ont une représentation mémoire généralement plus compacte ;
- Les arbres équilibrés préservent l'ordre des clefs, permettant notamment d'effectuer un parcours des clefs dans l'ordre ou de localiser efficacement une clé proche d'une valeur donnée. Les tables de hachage, en revanche, ne préservent pas l'ordre des clefs (lorsqu'il existe)

Efficacité : pourquoi des dictionnaires en NoSQL

Les dictionnaires sont présents nativement dans de nombreux langages de programmation : C++, Javascript, PHP, Perl, Python… donc on peut penser que le passage d'une description NoSQL vers un de ces langages est immédiate !

```
monuments = {"La tour Eiffel": "à Paris",
             "La statue de la liberté": "à New-York",
             "Le nombre de visiteurs de la tour Eiffel": 6930000
}
for clef in monuments.keys():
     print("{} est {}".format(clef, monuments[clef]))
```

Théorème de Brewer (ou de CAP)

En 2000, Eric A. Brewer a formalisé un théorème très intéressant reposant sur 3 propriétés fondamentales pour caractériser les bases de données (relationnelles, NoSQL et autres) :

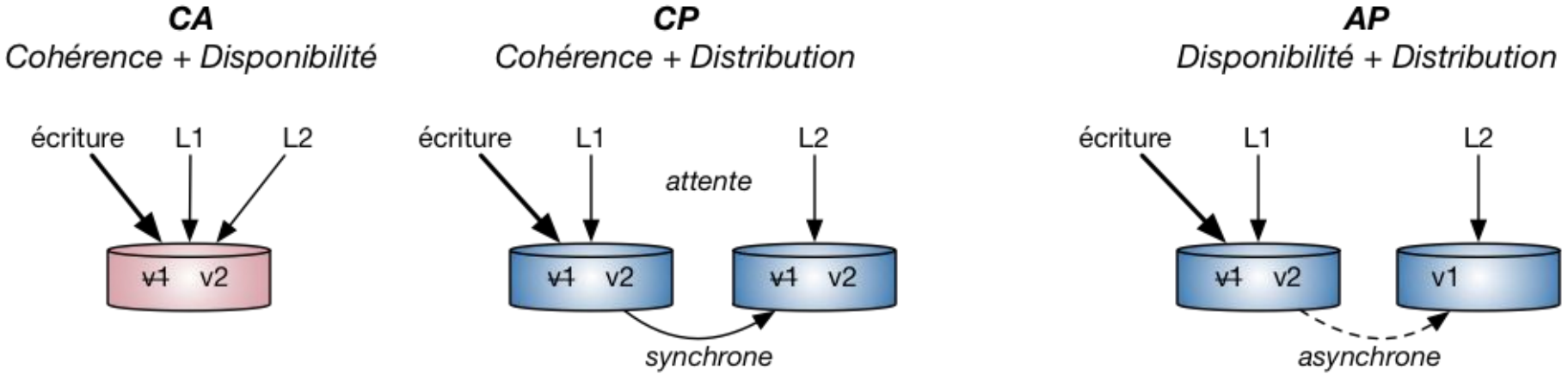
1. C*onsistency* (Cohérence) : Une donnée n'a qu'un seul état visible quel que soit le nombre de réplicas ;
2. A*vailability* (Disponibilité) : Tant que le système tourne (distribué ou non), la donnée doit être disponible ;
3. P*artition Tolerance* (Distribution) : Quel que soit le nombre de serveurs, toute requête doit fournir un résultat correct ;

Le théorème de CAP dit :

> Dans toute base de données, vous ne pouvez respecter au plus que 2 propriétés parmi la *cohérence*, la *disponibilité* et la *distribution*.

Théorème de Brewer

Cela s'illustre assez facilement avec les bases de données relationnelles, elles gèrent la cohérence et la disponibilité, mais pas la distribution.



Théorème de Brewer

Prenons le couple CA (Consistency-Availability), il représente le fait que lors d'opérations concurrentes sur une même donnée, les requêtes L1 et L2 retournent la nouvelle version (v2) et sans délai d'attente.

Cette combinaison n'est possible que dans le cadre de bases de données transactionnelles telles que les SGBDR.

Théorème de Brewer

Le couple CP (Consistency-Partition Tolerance) propose maintenant de distribuer les données sur plusieurs serveurs en garantissant la tolérance aux pannes (réplication). En même temps, il est nécessaire de vérifier la cohérence des données en garantissant la valeur retournée malgré des mises à jour concurrentes.

La gestion de cette cohérence nécessite un protocole de synchronisation des réplicas, introduisant des délais de latence dans les temps de réponse (L1 et L2 attendent la synchronisation pour voir v2). C'est le cas de la base NoSQL *MongoDB*.

Théorème de Brewer

Le couple AP (Availability-Partition Tolerance) à contrario s'intéresse à fournir un temps de réponse rapide tout en distribuant les données et les réplicas. De fait, les mises à jour sont asynchrones sur le réseau, et la donnée est "*Eventually Consistent*" (L1 voit la version v2, tandis que L2 voit la version v1). C'est le cas de *Cassandra* dont les temps de réponses sont appréciables, mais le résultat n'est pas garanti à 100% lorsque le nombre de mises à jour simultanées devient important.

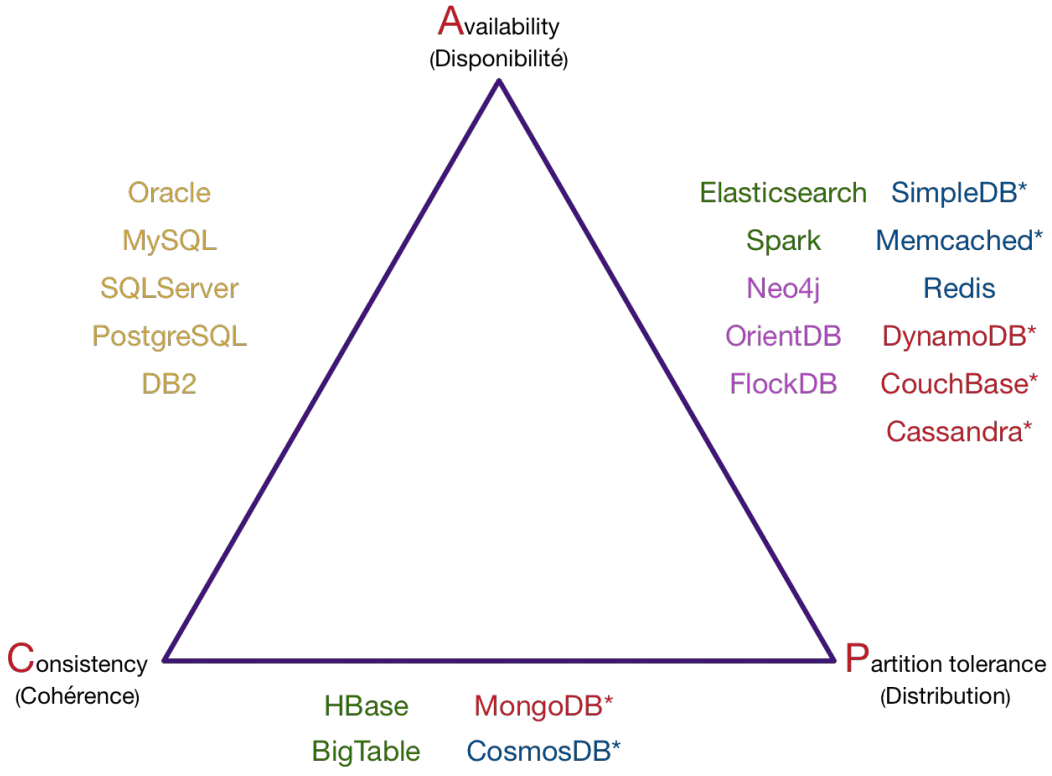
Ainsi, la cohérence des données est incompatible avec la disponibilité dans un contexte distribué comme les bases NoSQL.

Classification : triangle de CAP

Modèle

Relationnel
Orienté Clé-Valeur
Orienté Colonne
Orienté Document
Orienté Graphe

* Possibilité de changer
le mode de cohérence
Cohérence <-> Disponibilité



Classification : il reste à placer CockroachDB (NewSQL)

Rappel : NewSQL est censé être universel en étant capable de gérer du très structuré au pas du tout structuré ⇒ il pourrait être placé sur chacune des arêtes du triangle ;

Cependant des arguments sur la page https://www.cockroachlabs.com/blog/limits-of-the-cap-theorem/ affirment que : (suspense)

Classification : CockroachDB (sources : lui même)

The CAP theorem is a fundamental part of the theory of distributed systems. It states that in the presence of partitions (i.e. network failures), a system cannot be both consistent and available, and must choose one of the two.

CockroachDB chooses consistency, and is therefore a CP system in the terminology of the CAP theorem (as opposed to an AP system, which favors availability over consistency). Both consistency and availability are crucial to any business, and you might wonder how you are expected to choose between such important goals.

Banc d’essai

Image Ubuntu avec les gestionnaires préinstallés

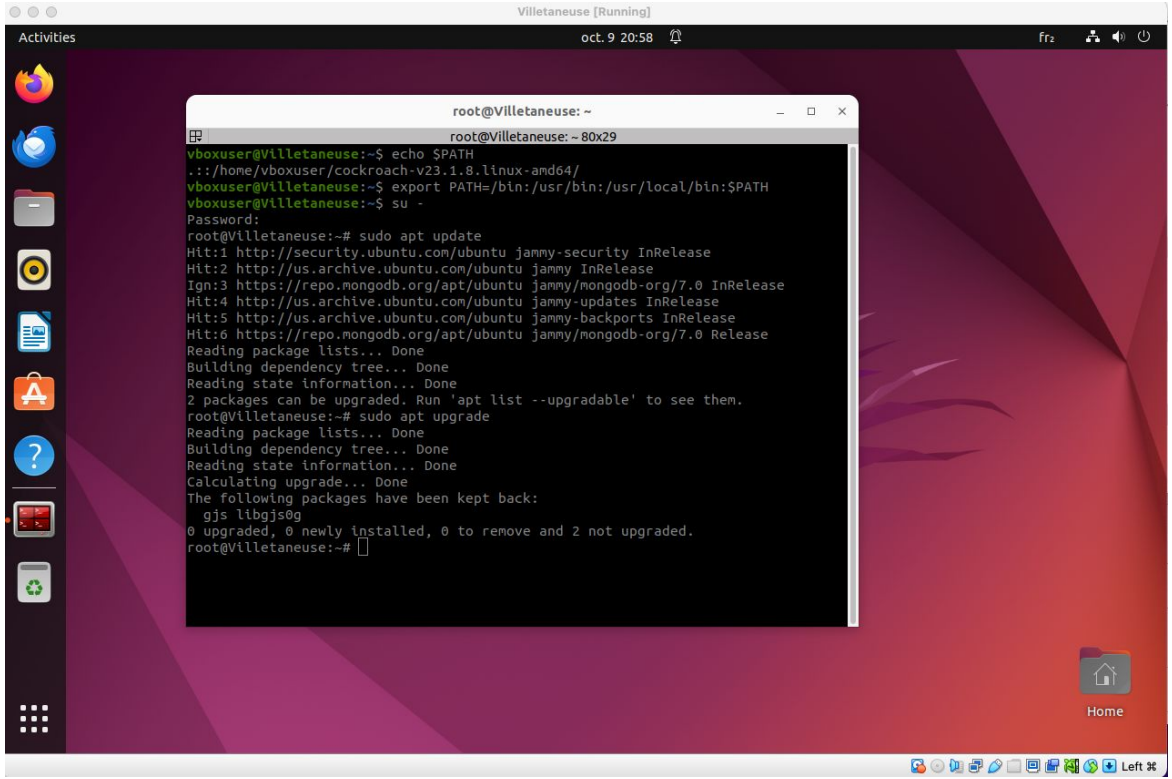
Voici comment l'image Ubuntu a été créée et configurée :

1) On a besoin de VirtualBox pour gérer des machines virtuelles. Tutoriels d'installation très répandus ;
2) On télécharge une distribution Ubuntu, on l'a lance depuis VirtualBox et on la configure. Nombreux tutoriels ; (Nota : su - pour passer root et villetaneuse est le mot de passe root pour l'image que je fourni)
3) On installe les gestionnaires de SGBD dans l'image en suivant :
    a) https://redis.io/docs/getting-started/installation/install-redis-on-linux/
    b) https://www.mongodb.com/docs/manual/tutorial/install-mongodb-on-ubuntu/
    c) https://www.cockroachlabs.com/docs/stable/install-cockroachdb-linux

Dernière mise à jour de l'image Ubuntu : 09/10/2023

Remarquez les différentes étapes de conf ;

Attention : il se peut que votre clavier soit en mode qwerty et donc a et q sont inversés ;



Comment interagir avec les gestionnaires

1) Soit à travers une CLI (Command Line Interface) qui vient avec l'installation du SGBD ;
2) Soit en se connectant au serveur (point d'entrée) puis en tapant des commandes ;
3) Soit en utilisant un connecteur (une bibliothèque), que l'on nomme aussi un *client*, pour un de nos langages de programmation (Java, Python,...)

Cas de Redis (redis-cli et redis-py)

CLI : voir https://www.tutorialspoint.com/redis/redis_commands.htm ou encore https://redis.io/docs/ui/cli/

Redis client : voir https://redis.io/docs/clients/python/

Cas de MongoDB (mongosh et PyMongo)

CLI (shell d'interaction) :
https://www.mongodb.com/docs/mongodb-shell/run-commands/

Client Python MongoDB :
https://www.mongodb.com/languages/python/pymongo-tutorial et aussi
https://pypi.org/project/pymongo/

Cas de CockroachDB

CLI : https://www.cockroachlabs.com/docs/stable/cockroach-commands

Client Python : https://www.cockroachlabs.com/docs/v23.1/build-a-python-app-with-cockroachdb-psycopg3 (attention, c'est un peu dense comme explications)

Nota : on peut aussi utiliser Django… mais on verra plus tard (https://www.cockroachlabs.com/docs/stable/build-a-python-app-with-cockroachdb-django?filters=local) Django est un framework web *open source* en Python. Il a pour but de rendre le développement d'applications web simple et basé sur la réutilisation de code. Qui dit web dit BD (c.f. pile LAMP)

Exercices : lot 1

Normalisation et dénormalisation

Le problème : comment passer du relationnel au NoSQL ?

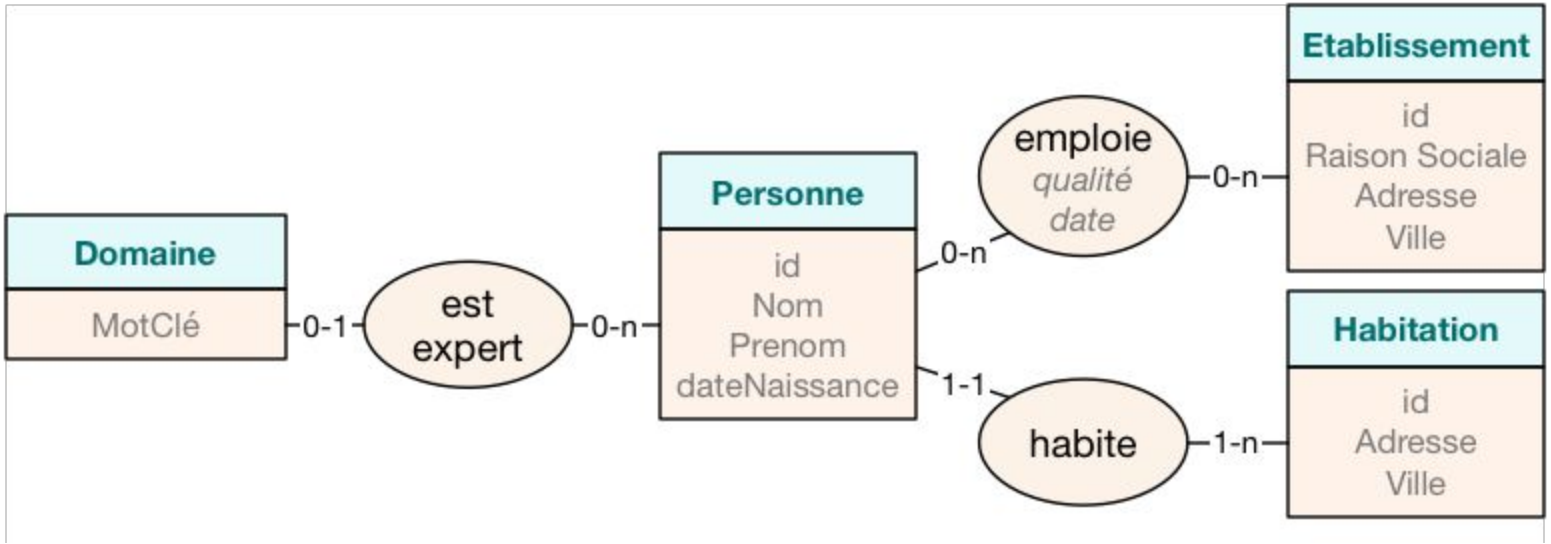
On part de tables (SQL) et on veut une représentation pour Redis et MongoDB

Rappel : normaliser c'est jouer avec les formes normales ;

Dénormaliser c'est partir de plusieurs tables pour n'en ⟪ *fabriquer* ⟫ plus qu'une seule (c'est ce que j'appelle une structure à plat) ;

Quelles règles de transformations appliquées ? Ca dépend du but recherché. En général, on recherche à minimiser les jointures nécessaires aux opérations du schéma relationnel.

Exemple de dénormalisation

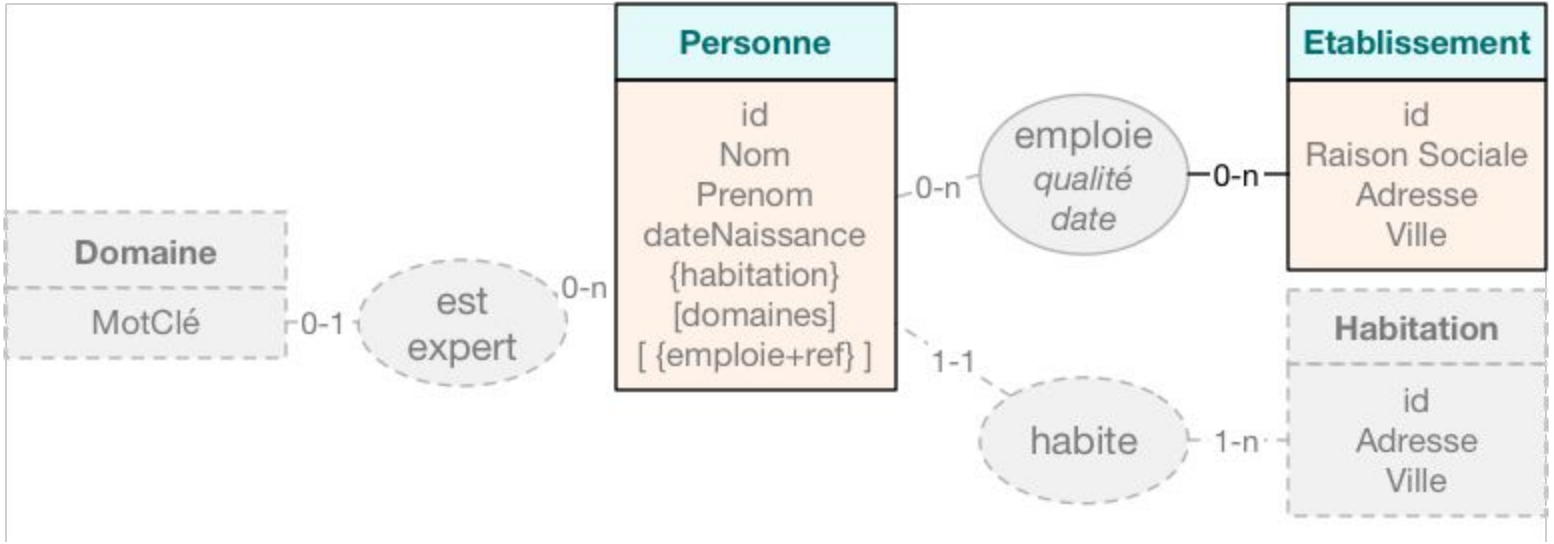


Source : https://openclassrooms.com/fr/courses/4462426-maitrisez-les-bases-de-donnees-nosql/4474601-decouvrez-le-fonctionnement-de-mongodb

Critères de fusion (tout en prenant soin de la cohérence)

- **Des données fréquemment interrogées conjointement sont rassemblées.**
  Par exemple, les requêtes demandent fréquemment le lieu d'habitation d'une personne. De fait, la jointure devient coûteuse. Accessoirement, cette information étant peu mise à jour, cela pose peu de problèmes. Résultat, l'entité 'Habitation' et l'association 'habite' sont intégrés à l'entité Personne. Habitation devient un document imbriqué à l'intérieur de Personne, représenté par : "{habite}"
- **Toutes les données d'une entité sont indépendantes.**
  Prenons l'exemple des domaines d'expertise d'une personne, ils sont indépendants des domaines d'une autre personne. De fait, rapatrier les données de cette entité n'impacte aucune autre instance de Personne. Ainsi, la liste des domaines est importé dans Personne et représenté par : "[domaines]"
- **Une association avec des relations 1-n des deux côtés.**
  Cette fois-ci, c'est plus délicat pour l'entité Etablissement. Une personne peut avoir plusieurs emplois et un employeur, plusieurs employés. De fait, une imbrication de l'employeur dans Personne peut avoir de gros impacts sur les mises à jour (tous les employés à mettre à jour !). Il est donc peu recommandé d'effectuer une fusion complète. Pour cela, seule l'association est imbriquée sous forme d'une liste de documents, intégrant les attributs (qualité et date), ainsi qu'une référence vers l'employeur. Ainsi : "[{emploie+ref}]"
- **Même taux de mises à jour.**
  Dans le cas des emplois d'une personne, là également nous pourrions effectuer une fusion de l'association "emploie". En effet, le taux de mises à jour des emplois est équivalent à celui de la Personne, de fait, sans incidence sur les problèmes de cohérence de données.

On obtient cette dénormalisation (avec ces critères)



Représentation JSON de la dénormalisation

```
{
  "_id" : 1,    "nom" : "Travers",   "prenom" : "Nicolas",
  "domaines" : ["SGBD", "NoSQL", "RI", "XML"],
  "emplois" : [
    {"id_etablissement" : "100", "qualité" : "Maître de Conférences",
        "date" : "01/09/2007"},
    {"id_etablissement" : "101", "qualité" : "Vacataire",
        "date" : "01/09/2012"}
  ],
  "Habite" : {"adresse" : "292 rue Saint Martin", "ville" : "Paris"}
}
```

Exercices

1) Construisez des dénormalisations (en choisissant des paires de table) jusqu'à une expression en JSON à partir des tables que vous allez trouver ici : https://sites.google.com/site/lebbah/bdd puis aller sur **Les fichiers de la BDD utilisés en TP à télécharger <ici>** Il s'agit de la base de données Pilotes vue en première année !
2) Le site https://unehistoireduconflitpolitique.fr/ contient des données pour le livre de Julia Cagé et Thomas Piketty intitulé Une histoire du conflit politique. Aller à l'onglet Télécharger. Récupérer le CSV pour les diplômes. Quelle représentation JSON de cette table à plat vous paraît la plus pertinente et pourquoi ?
3) Sur le site précédent, rechercher s'il existe des dataset ventilés dans plusieurs tables et qui nécessiteraient des dénormalisations.

Exercices

4) A l'exercice 2) vous avez associé une description JSON à une table à plat (un CSV). Est-ce que votre description JSON est celle obtenue en sortie du Programme Python décrit ici https://www.askpython.com/python/examples/convert-csv-to-json ? Si c'est non, pouvez-vous modifier le script Python pour produire la description obtenue au 2) ? Justifier ce que vous faites.

5) Écrire un programme Python qui prend en paramètre deux descriptions JSON et un attribut (commun) et qui retourne la jointure au sens qui est présenté dans http://lipn.univ-paris13.fr/~cerin/jointure.pdf Choisissez d'implémenter la jointure avec l'une des trois méthodes explicitées au lien précédent.

Etude de Redis

Redis data struture

Rappel : une structure de données sert à représenter l'univers du problème à traiter. On a besoin de structurer à la fois les programmes et les données :

- Par exemple, la notion de class sert à structurer les programmes ;
- Par exemple, les notions de liste, dictionnaires, tuples servent à structurer les données manipulées par un programme. On les appelle parfois des Type de données abstraits ;
- Les types de base (char, int, float...) servent aussi à structurer les données des programmes

Redis data structure

Les structures de données (évoluées) de Redis sont principalement les chaînes de caractères, les dictionnaires, les listes, les ensembles, les ensembles ordonnés ;

Il y a ensuite, au niveau du langage des éléments un peu curieux :

- HyperLogLog : ensemble où l'on peut rechercher de manière approchée ;
- Des mécanismes de publication / souscription qui est un modèle d'interaction entre composants ;
- Les transactions : groupes de commandes qui s'exécutent en un seul "pas".

Redis data structure

Pour illustrer le propos, le plus simple est de visiter https://www.tutorialspoint.com/redis/index.htm

La liste exhaustive des Data Structures est ici :https://redis.io/docs/data-types/

Ensuite, on peut s'intéresser aux différentes manières d'interagir avec le serveur Redis :

- https://www.tutorialspoint.com/redis/redis_scripting.htm
- CLI : https://redis.io/docs/ui/cli/
- Via un client Python : https://redis.io/docs/clients/ et comme on va travailler en Python, on regarde https://redis.io/docs/clients/python/

Mise en garde

Attention : dans les commandes (le ⟪ *langage* ⟫) Redis, il n'y a pas les moyens d'itérer un traitement, voire de choisir entre deux alternatives ⇒ C'est de la responsabilité du client Redis :

- C'est lui qui implémente l'algorithme qui résout votre problème, par composition d'actions élémentaires ;
- En gros, le ⟪ *langage* ⟫ Redis c'est une collection d'opérations atomiques sur la base de données … et pis c'est tout !

Exercices Redis (lot 2)

Exercices

1) Pour vous faire la main, commencer par rejouer https://redis.io/docs/clients/python/
2) Sur la page https://redis.io/commands/ vous allez trouver des commandes qui font référence à des Bloom filters. Qu'est-ce que c'est que cette notion ? Mettez-là en oeuvre à travers un "petit" exemple. Exercice à faire en utilisant le prompt d'invite de Redis, directement ;
3) Récupérer le dictionnaire http://rali.iro.umontreal.ca/DEM//DEM-1_1.csv et rentrer les valeurs (mot, définition) dans un Bloom filter et dans une liste. Construisez un test permettant de comparer les temps d'appartenance de 100000 mots selon la structure de données choisie ; Quelle structure (Bloom filter ou List) est la plus efficace ?
4) Même question avec les Cockoo Filters.
5) Même question avec les opérations JSON.xxx.v Vous pouvez commencer par lire https://redis.io/docs/data-types/json/
6) Sélectionner et rejouer les codes (en Python) situés à partir de https://developer.redis.com/howtos/quick-start/

Exercices

6) Publication / souscription. Implémenter le mécanisme suivant. Un premier script Python publie, de manière espacée (toutes les 1 à 3 secondes) un couple (date, valeur_de_CO2), représentant une date de mesure et la valeur prise sur un capteur de CO2. Ce script répète indéfiniment ces publications (les valeurs de CO2 seront obtenues par tirage aléatoire).

Un deuxième script Python, calcule toutes les minutes, la moyenne des valeurs de CO2 reçues sur la dernière minute, les dernières 30 minutes et la dernière heure, puis affiche ces 3 valeurs.

Nota : vous pouvez commencer par lire https://redis.io/docs/interact/pubsub/ ainsi que https://www.stackhero.io/fr-fr/services/Redis/documentations/Utilisation-avec-Python/Comment-utiliser-Pub-Sub-avec-Redis-et-Python

Exercices

7) Rejouer https://redis.io/docs/manual/patterns/twitter-clone/ Attention, vous allez être amenés à installer des outils supplémentaires par rapport à ce qui est fourni dans l'image Ubuntu ! Il s'agira de faire un compte rendu concis qui résume, en une page maximum, les difficultés rencontrées !

8) Rejouer l'exemple du 《 Chapter 7 – Simple Redis Application 》du livre Redis for dummies que vous allez trouver en accès libre sur le Web (https://redis.com/redis-for-dummies/).

Etude de MongoDB

MongoDB

On suit la démarche précédente : 1) vue générale 2) les data structures/types 3) l'interaction avec le serveur 4) les exercices :

- Vue générale de l'écosystème MongoDG : https://welovedevs.com/fr/articles/mongodb/
- MongoDB a faire le choix des performances, donc de la duplication, via de nombreux indexes : https://welovedevs.com/fr/articles/mongodb-index/ (autrement dit, MongoDB incite à la dénormalisation outrancière !)
- Interactions : https://welovedevs.com/fr/articles/mongodb-client puis https://www.mongodb.com/languages/python (pour programmer en Python)

Exercices MongoDB (lot 3)

Exercices

1) Rejouer tous les exemples à partir de la section « Comment démarrer avec MongoDB ? » de la page https://welovedevs.com/fr/articles/mongodb/
2) Rejouer les codes exemples depuis https://www.mongodb.com/languages/python
3) Refaire le plus possible d'exercices présentés pour Redis en donnant cette fois-ci des implémentations MongoDB (qui utilisent les concepts, les outils et les méthodes MongoDB). Pour l'exercice sur le Pub/Sub, lire d'abord https://www.mongodb.com/blog/post/pubsub-with-mongodb

Etude de CockroachDB

CockroachDB in a nutshell

Le côté sympathique : une/plusieurs colonnes d'une table (au sens relationnel) peut-être un/des objet/s JSON ⇒ disponibilité d'opérations pour créer, consulter, modifier de telles colonnes ;

On est donc dans un monde mixte avec que des tables ou que des dictionnaires ou un mélange table/dictionnaires, donc dans un monde structuré ou pas structuré !

Exemple un peu complet à discuter : https://www.cockroachlabs.com/docs/stable/demo-json-support

Conséquence : avec CockroachDB on peut enfin refaire des bons vieux SELECT * FROM …. Donc, on peut passer directement aux exercices !!!

Exercices CockroachDB (lot 4)

Exercices

1) Rejouer https://www.cockroachlabs.com/docs/cockroachcloud/quickstart qui est un exemple ou l'on programme en partie en JAVA (et pas Python).
2) Rejouer l'exemple https://www.cockroachlabs.com/docs/v23.1/build-a-python-app-with-cockroachdb (ou cette fois on programme en Python)
3) Rejouer l'exemple ;https://www.cockroachlabs.com/docs/cockroachcloud/learn-cockroachdb-sql pour apprendre à (re)faire du SQL ;
4) Rejouer https://www.cockroachlabs.com/docs/v23.1/build-a-python-app-with-cockroachdb-sqlalchemy?filters=local (attention, il faut déployer un cluster local)

Exercices

5) Refaire le plus possible d'exercices présentés pour Redis en donnant cette fois-ci des implémentations CockroachDB (qui utilisent les concepts, les outils et les méthodes de CockroachDB).

Note : il n'est pas possible de refaire l'exercice sur le Pub/Sub. Disons que cela serait un peu compliqué pour notre niveau même si vous pouvez lire https://www.cockroachlabs.com/blog/cdc-with-google-pubsub-and-cockroachdb/

Rédiger un compte rendu de TPs
Biliographie

Votre travail pour ce module

Toutes les manipulations qui vous paraissent les plus intéressantes à raconter doivent être insérées dans un document Compte rendu des TPs qui ne devra pas excéder 15 pages.

NOTA : vous devez faire des choix ! Vous, pas nous les Profs !

Comment rédiger un compte rendu de TPs ? Voir les recommandations sur la page https://lipn.univ-paris13.fr/~cerin/LPASUR/ et à la section PREAMBULE.

La dernière séance sera consacrée à une soutenance individuelle où vous présenterez votre compte rendu et où les chargés de TPs vous interrogerons.

Bibliographie

[1] Pavlo, Andrew; Aslett, Matthew (2016). "What's Really New with NewSQL?" (PDF). *SIGMOD Record*. Retrieved February 22, 2020.

[2] Cattell, R. (2011). "Scalable SQL and NoSQL data stores" (PDF). *ACM SIGMOD Record*. 39 (4): 12–27. CiteSeerX 10.1.1.692.2621. doi:10.1145/1978915.1978919. S2CID 3357124.

[3] Aslett, Matthew (2011). "How Will The Database Incumbents Respond To NoSQL And NewSQL?" (PDF). 451 Group (published April 4, 2011).

[4] Leavitt, Neal (2010). "Will NoSQL Databases Live Up to Their Promise?" (PDF). *IEEE Computer*. 43 (2): 12–14. doi:10.1109/MC.2010.58. S2CID 26876882

[5] Strauch, Christof. "NoSQL Databases" (PDF). pp. 23–24.

[6] Stéphane Vialle. http://www.metz.supelec.fr/metz/personnel/vialle/course/BigData-2A-CS/ (voir la partie III)