Gestion des données et accès à l'information

Bases de données noSQL

Olivier Schwander <olivier.schwander@sorbonne-universite.fr>

Master Statistiques
Sorbonne Université

2024-2025

Not Only SQL

Données non-structurées

- Pas de schéma
- Pouvoir s'adapter à de nouvelles données
- Stocker au fur et à mesure et traiter plus tard

Langage de requête non-standardisé

- Spécifique à chaque système

Un pas en avant, deux pas en arrière ?

Langage spécifique

- Similaire dans l'idée au SQL
- `ALLER CHERCHER` tel truc `À L'ENDROIT` machin `AVEC LES CONTRAINTES` bidule
- Parfois plus puissant, parfois mieux connu du développeur

Contraintes inutiles

- Éviter les redondances: stockage de masse pas cher
- Optimiser les requêtes: requêtes souvent simples

Localité

- Données logiquement proches physiquement proches

Théorème CAP

Dans un système distribué

Cohérence (*Coherency*)

- Tous les nœuds voient la même version

Disponibilité (*Availability*)

- Chaque requête obtient une réponse

Résistance au partitionnement (*Partition tolerance*)

- Perdre un nœud ou un message ne bloque pas le système

Théorème: au plus deux propriétés sur les trois

Théorème CAP

Relationnel

- Cohérence
- Disponibilité

Non-relationnel

- Disponibilité
- Résistance au partitionnement

Orienté graphe

Données et relations

- Stockage de données avec beaucoup de relations complexes
- En évitant les jointures

Avantages et inconvénients

- Beaucoup de données ressemblent à des graphes
- Approprié pour parcourir les relations
- Pas pour filtrer selon des contraintes

Orienté clé-valeur

Opérations CRUD

- Create: créer un objet
- Read: lire à partir de la clé
- Update: mettre à jour à partir de la clé
- Delete: suppression à partir de la clé

Indexation

- Seulement la clé
- On ne regarde pas le contenu de la valeur

Avantages et incovénients

- Facile à utiliser, performances élevées
- Pas de structure, pas de requêtes complexes

Orienté document

Collection de documents

- Clé/Valeur
- On regarde dans la valeur

Indexation

- Identifiant unique par document

Avantages et inconvénients

- Modèle simple mais puissant
- Requêtes complexes possibles
- Mauvais passage à l'échelle pour des requêtes complexes

Orienté colonne

Stockage colonne par colonne

- Et pas ligne par ligne

Indexation

- Par colonne

Avantages et inconvénients

- Passe à l'échelle
- Lecture difficile pour des données complexes

De plus en plus

Intégration

- Documents structurés directement dans une base relationnelle
- Syntaxe JSON pour les manipuler
- Moins besoin de jointures
- Dans PostgreSQL
  https://www.postgresql.org/docs/current/functions-json.html

Passerelle

- Coté application: interface noSQL
- Coté serveur: vrai serveur SQL
- MongoDB vers SQL https://github.com/FerretDB/FerretDB

Structures

Bases de données

- Contiennent des *collections*

Collections

- Contiennent des *documents*
- Chaque document a une clé primaire unique

Document

- Dictionnaire de valeurs
- Aucune structure imposée

Document


```
{
  "address": {
     "building": "1007",
     "coord": [ -73.856077, 40.848447 ],
     "street": "Morris Park Ave",
     "zipcode": "10462"
  },
  "borough": "Bronx",
  "cuisine": "Bakery",
  "grades": [
     { "date": { "$date": 1393804800000 }, "grade": "A", "s
     { "date": { "$date": 1378857600000 }, "grade": "A", "s
     { "date": { "$date": 1358985600000 }, "grade": "A", "s
     { "date": { "$date": 1322006400000 }, "grade": "A", "s
     { "date": { "$date": 1299715200000 }, "grade": "B", "s
  ]
```

Base et collection

Sélection de la base

- `use NOM_DE_LA_BASE`

Sélection de la collection

- `db.NOM_DE_LA_COLLECTION.xxxx`
- `xxxx` est l'instruction à effectuer

Insertion

```
db.restaurants.insert(
    {
        "address" : {
            "street" : "2 Avenue",
            "zipcode" : "10075",
            "building" : "1480",
            "coord" : [ -73.9557413, 40.7720266 ],
        },
        "name" : "Vella",
    }
)
```

Recherche

Conditions

- `db.restaurants.find({"borough": "Manhattan" })`
- `db.restaurants.find({"address.zipcode": "10075" })`
- `db.restaurants.find({"grades.grade": "B" })`
- `db.restaurants.find({"grades.score": {$gt: 30 } })`

Logique

- ET: `db.restaurants.find({"cuisine": "Italian", "address.zipcode": "10075"})`
- OR:

```
db.restaurants.find(
   { $or: [ { "cuisine": "Italian" }, { "address.zipcode":
)
```

Mise à jour

```
db.restaurants.update(
    { "name" : "Juni" },
    {
      $set: { "cuisine": "American (New)" },
      $currentDate: { "lastModified": true }
    }
)
```

- `$currentDate` est une fonction qui met à jour le champ "lastModified"

Suppression

Tous les documents

```
db.restaurants.remove( { "borough": "Manhattan" } )
```

Seulement un

```
db.restaurants.remove( { "borough": "Queens" },
                       { justOne: true } )
```

Tri

```
db.restaurants.find().sort(
    { "borough": 1, "address.zipcode": 1 }
    )
```

Aggrégation

```
db.restaurants.aggregate(
    [
      { $match: { "borough": "Queens", "cuisine": "Brazilian
      { $group: { "_id": "$borough", "count": { $sum: 1 } }
    ]
);
```

Opérateurs

- $sum, $avg, $min, $max, $first, $last, $avg

Calculs

```
db.collection.aggregate(
    [
        { $group: { _id: "$item", avgAmount: {
          $avg: {$multiply:["$price", "$quantity"]}
          }
}}])
```

Géographie

```
db.restaurants.find({ location:
   { $geoWithin:
      { $centerSphere: [ [ -73.93414657, 40.82302903 ],
                         5 / 3963.2 ] } } })
```