MAC - Méthode d'accès aux données

Introduction aux bases de données NoSQL

09 November 2025

Table des matières

1 Modèle relationnel et approches concurrentes

Le modèle de données relationnel s'est imposé comme la norme dominante pour la gestion des bases de données depuis les années 1970. Cependant, avec l'essor du web et des applications distribuées, de nouvelles approches de gestion des données ont émergé, notamment les bases de données NoSQL.

1.1 NoSQL

Le terme « NoSQL » fait référence à une catégorie de systèmes de gestion de bases de données qui ne suivent pas le modèle relationnel traditionnel. Les bases de données NoSQL sont conçues pour gérer de grandes quantités de données non structurées ou semi-structurées, offrant une flexibilité et une scalabilité accrues par rapport aux bases de données relationnelles.

Les principales caractéristiques des bases de données NoSQL incluent :

- Besoin d'une plus grande **scalabilité** que ce que les bases de données relationnelles peuvent offrir.
- **Opération de requête spécialisées** qui ne sont pas bien prises en charge par le modèle relationnel.
- Frustration face au **caractère restrictif des schémas relationnels** et désir de données plus dynamiques et expressives.

2 Impedance mismatch et montée en charge

Le terme « impedance mismatch » fait référence au décalage entre le modèle de données relationnel utilisé dans les bases de données relationnelles et les modèles de données utilisés dans les applications modernes, souvent basés sur des objets ou des structures de données plus flexibles. Ce décalage peut entraîner des difficultés lors de la manipulation et de la gestion des données, notamment en termes de performance, de complexité du code et de maintenance.

2.1 Approches pour éviter l'impedance mismatch

2.1.1 ORM

Les ORM (Object-Relational Mappers) sont des outils qui permettent de mapper les objets d'une application aux tables d'une base de données relationnelle, facilitant ainsi la gestion des données. Cependant, ils ne résolvent pas toujours complètement le problème de l'impedance mismatch.

2.1.2 Approche Document

Si nous prenons le cas d'un CV LinkedIn, nous pouvons voir que l'approche classique relationnelle peut être contraignante. Une approche dite `Document` par exemple sous forme de JSON permet de structurer les données de manière plus naturelle et flexible, en encapsulant toutes les informations pertinentes dans un seul document.

Approche relationnelle

Dans une approche dite relationnelle, pour construire l'affichage d'un CV nous devons parcourir plusieurs tables. Cette approche est couteuse en temps de requêtes et en complexité.

On se rend compte que pour afficher un simple CV, nous devons faire appel à de nombreuses jointures entre les différentes tables (utilisateur, expériences, formations, compétences, etc.). Cela peut entraîner des performances médiocres et une complexité accrue dans la gestion des données.

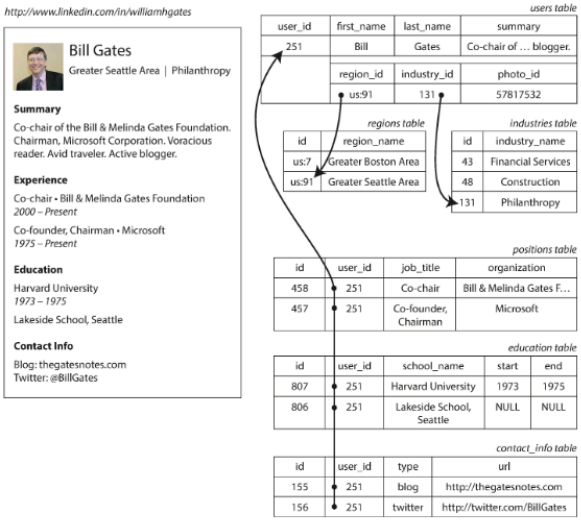

Fig. 1. – Capture des slides du cours – Exemple de CV

Une approche plus simple serait de stocker toutes les informations relatives à un utilisateur dans un seul document JSON.

```
{
  "user_id": 251,
  "first_name": "Bill",
  "last_name": "Gates",
  "summary": "Co-chair of the Bill & Melinda Gates... Active blogger.",
  "region_id": "us:91",
  "industry_id": 131,
  "photo_url": "/p/7/000/253/05b/308dd6e.jpg",
  "positions": [
    {
      "job_title": "Co-chair",
      "organization": "Bill & Melinda Gates Foundation"
    },
    {
      "job_title": "Co-founder, Chairman",
      "organization": "Microsoft"
```

```
    }
  ],
  "education": [
    {
      "school_name": "Harvard University",
      "start": 1973,
      "end": 1975
    },
    {
      "school_name": "Lakeside School, Seattle",
      "start": null,
      "end": null
    }
  ],
  "contact_info": {
    "blog": "http://thegatesnotes.com",
    "twitter": "http://twitter.com/BillGates"
  }
}
```

> ⚠ **Warning**
>
> Dans l'approche `Document`, les relations $1 : N$ sont assez simples à gérer en utilisant une structure arborescente explicite. Cependant les relations $M : N$ peuvent être plus complexes à modéliser et nécessitent souvent des compromis en termes de performance et de flexibilité.

2.2 Montée en charge

Les bases de données relationnelles peuvent rencontrer des difficultés lorsqu'il s'agit de gérer de grandes quantités de données ou un grand nombre d'utilisateurs simultanés. De plus elles ne sont pas toujours adaptées aux architectures distribuées modernes.

2.2.1 Scalabilité

Lors-ce que l'on souhaite améliorer les performances d'une base de données relationnelle, on peut envisager deux approches principales : la scalabilité verticale et la scalabilité horizontale.

Scalabilité verticale consiste à augmenter la capacité d'un seul serveur (par exemple, en ajoutant plus de CPU, de RAM ou de stockage). Les limites de cette approche sont le prix et les contraintes matérielles.

Scalabilité horizontale consiste à ajouter plus de serveurs pour répartir la charge. Cette approche est plus complexe à mettre en œuvre avec des bases de données relationnelles en raison des contraintes d'intégrité des données et des transactions distribuées. Cependant, cette approche est souvent plus économique et permet une meilleure résilience.

3 Modèles NoSQL

Les bases de données NoSQL adoptent différents modèles de données pour répondre aux besoins spécifiques des applications modernes. Voici les principales technologies de NoSQL :

- **n'utilisent pas le modèle relationnel** ni le langage SQL.
- **sont conçues pour la scalabilité horizontale** et la gestion de grandes quantités de données.
- n'ont **aucun schéma fixe**, les champs peuvent être ajoutés ou supprimés dynamiquement.
- sont **open-source** pour la plupart.
- **réplication aisée** (tolérence aux pannes, efficacité des requêtes).
- **API simples** pour les développeurs.
- *Eventually consistent* donc pas ACID.

3.1 Catégories principales

3.1.1 Base de données orientées documents

Les bases de données orientées documents stockent les données sous forme de documents, généralement au format JSON, BSON ou XML. Chaque document peut contenir des structures de données complexes, y compris des tableaux et des objets imbriqués, ce qui permet une grande flexibilité dans la modélisation des données.

Les documents sont stockés dans la partie clé-valeur cela permet donc d'avoir:

- Stores clés-valeurs ou les valeurs sont **examinables**
- Création d'**index** de recherche sur divrese clés/champs dans les documents

3.1.1.1 Cas d'utilisation

- Gestion de contenu et CMS: articles, blogs, documents et médias avec structure flexible
- Applications Web et mobiles: profils utilisateurs, contenus personnalisés et feeds dynamiques
- Machine Learning/AI: stockage de modèles, paramètres et métadonnées pour pipelines ML
- Plateformes sociales: messages, commentaires, interactions avec schéma flexible

3.1.1.2 Exemples

- mangoDB
- Couchbase
- RavenDB
- ArangoDB

3.1.2 Stores orientés colonnes

Les bases de données orientées colonnes stockent les données en regroupant les valeurs de chaque colonne ensemble, plutôt que de stocker les lignes complètes. Cette approche est particulièrement efficace pour les opérations analytiques et les requêtes qui impliquent la lecture de grandes quantités de données sur un nombre limité de colonnes.

Le store orientés colonnes peut-être vu comme une **map à deux niveaux**:

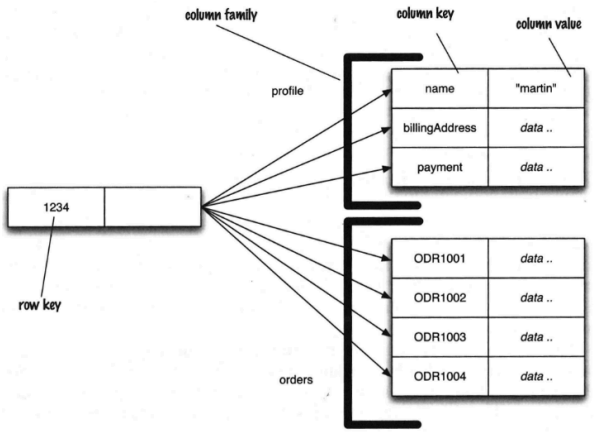

Fig. 2. – Capture des slides du cours – Store orienté colonnes

3.1.2.1 Cas d'utilisation

- Logs d'événements et journalisation (fort volume, données semi-structurées)
- Comptage (vote en ligne, clics, monitoring, etc.)
- Recherche de produits (filtrage par attributs)
- Reporting à grande échelle

3.1.2.2 Exemples

- Cassandra
- HBase

3.1.3 Base de données graphes

Les bases de données graphes sont conçues pour stocker et gérer des données sous forme de graphes, où les entités sont représentées comme des nœuds et les relations entre elles comme des arêtes. Ce modèle est particulièrement adapté pour les applications qui nécessitent une exploration complexe des relations entre les données, telles que les réseaux sociaux, les systèmes de recommandation et la gestion des connaissances.

3.1.3.1 Cas d'utilisation

- Réseaux sociaux (recommandation, plus court chemin, cluster, etc.)
- Réseaux SIG (routes, réseau électrique, etc)
- Web social (Linked Data)

3.1.3.2 Exemples

- Neo4j

3.1.4 Stores clé-valeur

Les stores clé-valeur sont structurés autour d'une simple paire clé-valeur, où chaque clé est unique et associée à une valeur. Ce modèle est particulièrement efficace pour les opérations de lecture et d'écriture rapides, mais il peut être limité en termes de complexité des requêtes et de relations entre les données.

La clé est gérée comme une chaîne de caractères unique, tandis que la valeur peut être un simple type de données (comme une chaîne, un nombre ou un booléen) ou une structure plus complexe (comme un tableau ou un objet sérialisé).

Grâce à cette approche nous pouvons utiliser une table de hachage distribuée pour répartir les données sur plusieurs nœuds, ce qui permet une scalabilité horizontale efficace et une haute disponibilité des données.

3.1.4.1 Cas d'utilisation

- Gestion de session à grande échelle
- Préférences utilisateur et stores de profils
- Recommandations de produits: les derniers articles consultés sur un site Web entraînent les futures recommandations de produits
- Publicités: les habitudes d'achat des clients se traduisent par des publicités personnalisées, des coupons, etc. pour chaque client en temps réel
- Peut fonctionner efficacement comme un cache pour les données très consultées mais rarement mises à jour

3.1.4.2 Exemples

- Riak
- Redis
- Couchbase