Les Bases de Données NoSQL : Focus sur les Bases de Données Orientées Colonnes

Par Amira Abbadi

Sommaire

Introduction aux bases de données NoSQL

- Les bases de données NoSQL, abréviation de "Not Only SQL", sont apparues comme une alternative aux bases de données relationnelles traditionnelles.

- Elles offrent une flexibilité et une évolutivité accrues pour gérer les données non structurées et semi-structurées, fréquemment rencontrées dans les applications modernes.

- Contrairement aux bases de données relationnelles basées sur le modèle entité-association (ERA), les bases de données NoSQL adoptent des modèles de données plus variés, tels que les paires clé-valeur, les documents, les graphes et les colonnes.

Les bases de données orientées colonnes

- Les bases de données orientées colonnes, également appelées "column-stores", stockent les données par colonnes plutôt que par lignes, contrairement aux bases de données relationnelles.

- Cette organisation présente plusieurs avantages, notamment une compression de données plus efficace, un accès plus rapide à des sous-ensembles de colonnes et une meilleure prise en charge des analyses complexes.

- Cassandra, HBase et ScyllaDB sont des exemples populaires de bases de données orientées colonnes.

Architecture des bases de données orientées colonnes

L'architecture des bases de données orientées colonnes se compose en général de plusieurs éléments :

- **Noeuds de données**

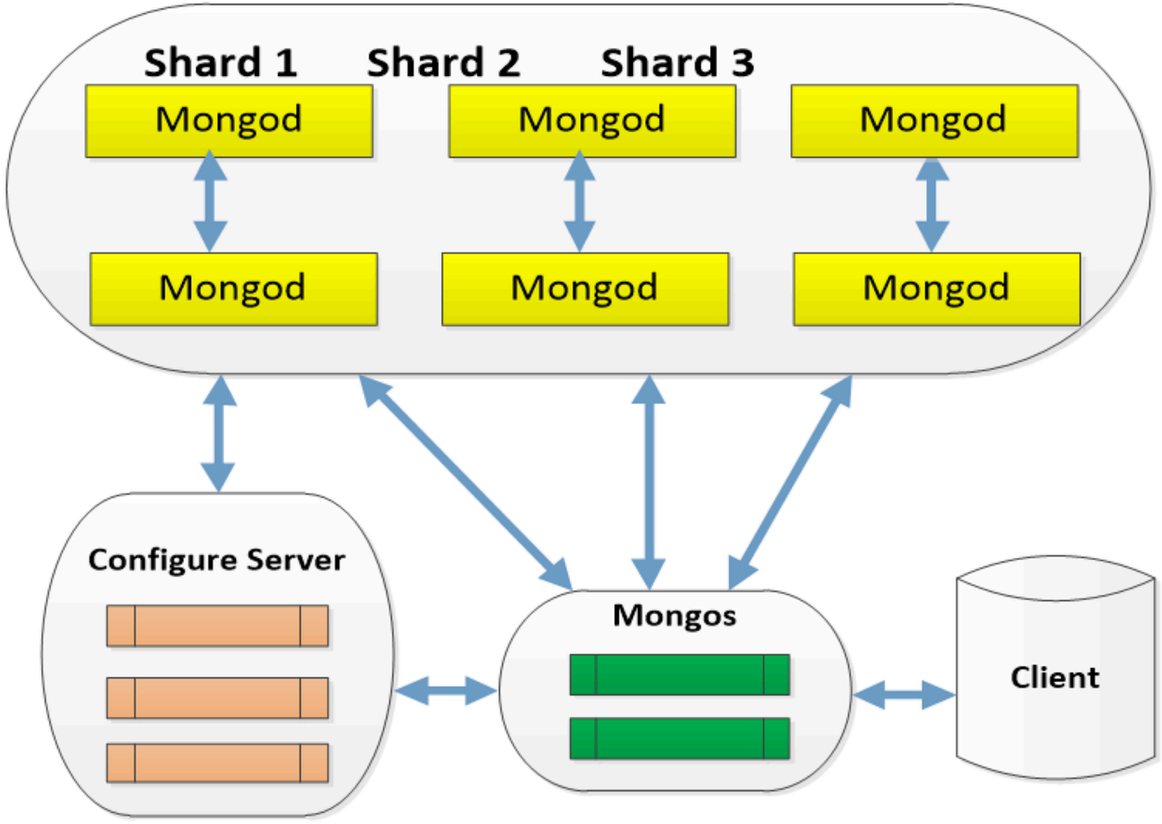
  Ils stockent les données réelles réparties sur plusieurs serveurs.

- **Noeuds de schéma**

  Ils gèrent les métadonnées et la définition du schéma de la base de données.

- **Clients**

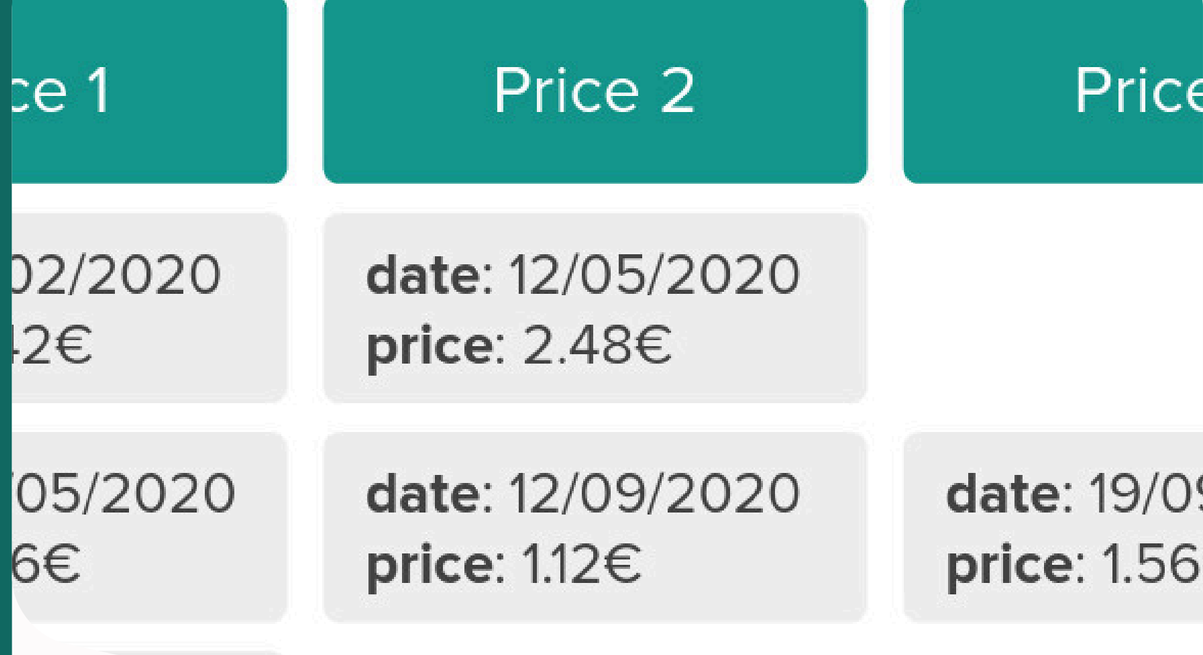
  Il s'agit des applications ou des interfaces utilisateur qui interagissent avec la base de données pour lire, écrire ou interroger des données.



Architecture des bases de données orientées colonnes

- La communication entre ces composants est assurée par des protocoles réseau spécifiques.

- La distribution des données sur plusieurs noeuds permet une évolutivité horizontale, ce qui signifie que l'on peut ajouter des serveurs pour répondre à l'augmentation de la charge de travail ou du volume de données.



Architecture des bases de données orientées colonnes

Domaines d'utilisation des bases de données orientées colonnes

Les bases de données orientées colonnes trouvent leur utilité dans une large gamme d'applications, notamment :

01

Analyse de données volumineuses (Big Data)

Idéales pour traiter de gros volumes de données complexes grâce à leur compression efficace et à leur accès rapide aux sous-ensembles de données.

02

Données temporelles (Time Series)

Structure parfaitement adaptée aux données chronologiques (historique de capteurs, cours de bourse, transactions financières) pour un stockage, une récupération et une analyse facilités.

03

Détection de fraude

Détection rapide des activités frauduleuses en temps réel grâce à l'accès rapide aux données et aux capacités d'analyse des modèles d'utilisation.

04

Internet des Objets (IoT):

Stockage et analyse en temps réel des flux de données provenant d'appareils connectés pour le suivi de l'état des capteurs, l'identification des anomalies et la prise de décision rapide.

Avantages

- Compression des données efficace

  Comme les bases de données orientées colonnes stockent des valeurs similaires ensemble, elles se compressent mieux que les bases de données relationnelles, ce qui réduit l'espace de stockage nécessaire.

- Requêtes analytiques rapides

  L'accès aux sous-ensembles de colonnes est plus rapide, car seules les colonnes requises sont lues, ce qui accélère les analyses de données volumineuses.

- Évolutivité horizontale

  La distribution des données sur plusieurs noeuds permet d'ajouter facilement des serveurs pour répondre aux besoins croissants en stockage et en traitement.

- Gestion efficace des séries chronologiques

  La structure des bases de données orientées colonnes est adaptée aux données temporelles, facilitant le stockage et l'analyse des séries chronologiques.

Inconvénients

01

Mauvaises performances pour les requêtes sur toutes les colonnes :

Accéder à toutes les colonnes d'une table peut être plus lent que dans une base de données relationnelle.

02

Gestion des transactions ACID plus complexe

Garantir l'atomicité, la cohérence, l'isolation et la durabilité (ACID) des transactions peut être plus difficile dans les bases de données orientées colonnes.

03

Apprentissage d'un nouveau modèle de données :

La compréhension et la conception de schémas pour les bases de données orientées colonnes peuvent demander un effort supplémentaire par rapport aux bases relationnelles.

Comparaison avec d'autres catégories de bases de données NoSQL

Caractéristique	Bases de données orientées colonnes	Bases de données clé-valeur	Bases de données documents	Bases de données graphes
Stockage des données	Colonnes	Paires clé-valeur	Documents JSON	Graphes avec nœuds et liens
Accès aux données	Rapide pour les sous-ensembles de colonnes	Rapide pour les accès par clé	Rapide pour les documents entiers	Rapide pour les parcours de graphe
Cas d'utilisation	Analyse de données volumineuses, séries chronologiques	Mise en cache, systèmes de panier d'achat	Stockage de documents semi-structurés, CMS	Réseaux sociaux, recommandations
Avantages	Compression efficace, requêtes analytiques rapides	Simplicité, évolutivité	Flexibilité, structure semi-structurée	Connexions entre entités, requêtes complexes
Inconvénients	Mauvaises performances pour les requêtes sur toutes les colonnes	Nécessité d'un schéma défini	Complexité des requêtes	Apprentissage d'un nouveau modèle de données

Application de Base des Données Orientées Colonnes avec Cassandra

Cassandra : Un choix populaire pour les Bases de Données Orientées Colonnes

Cassandra est un système de gestion de base de données (SGBD) NoSQL open source, orienté colonnes, conçu pour gérer des volumes massifs de données répartis sur plusieurs serveurs.

Il offre une évolutivité horizontale, une haute disponibilité et une tolérance aux pannes, ce qui le rend idéal pour les applications Big Data et les analyses en temps réel.

Cassandra est particulièrement apprécié pour sa capacité à gérer des séries chronologiques, des données non structurées et des charges de travail critiques.

Création d'une application de gestion de bibliothèque avec Cassandra

1

Créer une base de données et des tables

2

Insérer, lire, mettre à jour et supprimer des données

3

Gérer les requêtes avec le langage Cassandra Query Language (CQL)

Scénario de l'application

L'application gère une bibliothèque simple avec des livres et des emprunteurs. Elle permet de:

Ajouter des livres à la bibliothèque (titre, auteur, ISBN, genre)

Consulter les informations des livres

Emprunter des livres à un emprunteur (nom, adresse, email)

Gérer les retours de livres

Afficher la liste des livres empruntés par un emprunteur

Étape 1: Créer un keyspace

Principales caractéristiques d'un keyspace :

1. **Nom du keyspace** : Un nom unique pour identifier le keyspace.
2. **Stratégie de réplication** : Définit comment les données sont répliquées entre les nœuds du cluster. Les stratégies courantes sont SimpleStrategy et NetworkTopologyStrategy.
3. **Facteur de réplication** : Le nombre de réplicas (copies) de chaque donnée dans le cluster.

Création d'un Keyspace

```
cqlsh> drop keyspace if exists bibliotheque;
cqlsh> CREATE KEYSPACE bibliotheque with REPLICATION={'class': 'SimpleStrategy',
 'replication_factor': 1};
```

Utilisation d'un Keyspace

```
cqlsh> USE bibliotheque;
cqlsh:bibliotheque>
```

Étape 2: Création des tables

Créer la table livres:

```
cqlsh:bibliotheque> create table livres(id int primary key, titre text, auteur text,
 idbn text, genre text);
```

Créer la table emprunteurs:

```
cqlsh:bibliotheque> create table emprunteurs (id int primary key, nom text, adresse
text, email text);
```

Créer la table emprunts

```
cqlsh:bibliotheque> create table emprunts( id int primary key, livre_id int, emprunt
eur_id int, date_emprunt date, date_retoutr date);
cqlsh:bibliotheque>
```

Étape 3 : Opérations CRUD

Création (Create)

Ajouter des livres :

```
cqlsh:bibliotheque> INSERT INTO livres( id, titre, auteur, idbn, genre) VALUES(1,
'Le Seigneur des Anneaux', 'J.R.R. Tolkien', '978-2-07', 'Fantasy');
cqlsh:bibliotheque> INSERT INTO livres( id, titre, auteur, idbn, genre) VALUES(2,
'Le Petit Prince', 'Antoine de saint-exupery', '978-2-07', 'Conte');
cqlsh:bibliotheque>
```

Ajouter des emprunteurs :

```
cqlsh:bibliotheque> insert into emprunteurs(id, nom, adresse, email) values(11,
'ahmed', 'rue de la poste', 'ahmed@yahoo.fr');
cqlsh:bibliotheque> insert into emprunteurs(id, nom, adresse, email) values(22,
'amira', 'rue de la poste', 'amira@gmail.com');
cqlsh:bibliotheque>
```

Enregister des emprunts :

```
cqlsh:bibliotheque> insert into emprunts(id, livre_id, emprunteur_id, date_emprunt,
 date_retoutr) values(111, 1, 11, '2024-01-01','2024-02-02');
cqlsh:bibliotheque> insert into emprunts(id, livre_id, emprunteur_id, date_emprunt,
 date_retoutr) values(222, 2, 22 , '2023-02-02', '2023-03-03');
cqlsh:bibliotheque>
```

Étape 3: Opérations CRUD

Lecture (Read)

Lister tous les livres:

```
cqlsh:bibliotheque> select * from livres;

 id | auteur                    | genre   | idbn     | titre
----+---------------------------+---------+----------+-------------------------
  1 |           J.R.R. Tolkien | Fantasy | 978-2-07 | Le Seigneur des Anneaux
  2 | Antoine de saint-exupery |   Conte | 978-2-07 |          Le Petit Prince

(2 rows)
cqlsh:bibliotheque>
```

Lister tous emprunteurs:

```
cqlsh:bibliotheque> select * from emprunteurs ;

 id | adresse          | email             | nom
----+------------------+-------------------+-------
 11 | rue de la poste |   ahmed@yahoo.fr | ahmed
 22 | rue de la poste | amira@gmail.com | amira

(2 rows)
cqlsh:bibliotheque>
```

Étape 3: Opérations CRUD

Lecture (Read)

Lister tous emprunts :

```
cqlsh:bibliotheque> select * from emprunts;

 id  | date_emprunt | date_retoutr | emprunteur_id | livre_id
-----+--------------+--------------+---------------+----------
 111 |   2024-01-01 |   2024-02-02 |            11 |        1
 222 |   2023-02-02 |   2023-03-03 |            22 |        2

(2 rows)
```

Afficher un emprunteur avec id :

```
cqlsh:bibliotheque> select * from emprunteurs where id = 22;

 id | adresse          | email             | nom
----+------------------+-------------------+-------
 22 | rue de la poste | amira@gmail.com | amira

(1 rows)
cqlsh:bibliotheque>
```

Étape 3: Opérations CRUD

Lecture (Read)

Obtenir le nombre de livres par id :

```
cqlsh:bibliotheque> select genre, COUNT(*) as nblivres from livres group by ge
nre ;
InvalidRequest: Error from server: code=2200 [Invalid query] message="Group by
 is currently only supported on the columns of the PRIMARY KEY, got genre"
cqlsh:bibliotheque> select id, COUNT(*) as nblivres from livres group by id ;

 id | nblivres
----+----------
  1 |        1
  2 |        1

(2 rows)
```

Étape 3 : Opérations CRUD

Mise à jour (Update)

Mettre à jour le titre d'un livre:

```
cqlsh:bibliotheque> update livres set titre='le hobbit' where id=1;
cqlsh:bibliotheque> select * from livres where id= 1;

 id | auteur          | genre   | idbn      | titre
----+-----------------+---------+-----------+-----------
  1 | J.R.R. Tolkien | Fantasy | 978-2-07 | le hobbit

(1 rows)
cqlsh:bibliotheque>
```

Mettre à jour l'adresse d'un emprunteur:

```
cqlsh:bibliotheque> update emprunteurs set adresse='7500 paris' where id=22;
cqlsh:bibliotheque> select * from emprunteurs where id=22;

 id | adresse    | email           | nom
----+------------+-----------------+-------
 22 | 7500 paris | amira@gmail.com | amira
```

Étape 3 : Opérations CRUD

Mise à jour (Update)

Enregister le retour d'un livre:

```
cqlsh:bibliotheque> update emprunts set date_retoutr='2024-05-21' where id=111;
cqlsh:bibliotheque> select * from emprunts where id=111;

 id  | date_emprunt | date_retoutr | emprunteur_id | livre_id
-----+--------------+--------------+---------------+----------
 111 |   2024-01-01 |   2024-05-21 |            11 |        1
```

Acti

Mettre à jour le genre d'un livre:

```
cqlsh:bibliotheque> update livres set genre='fantasy' where id=1;
cqlsh:bibliotheque> select * from livres where id=1;

 id | auteur          | genre   | idbn     | titre
----+-----------------+---------+----------+-----------
  1 | J.R.R. Tolkien | fantasy | 978-2-07 | le hobbit

(1 rows)
```

Étape 3: Opérations CRUD

Suppression (Delete)

Supprimer un emprunteur:

```
cqlsh:bibliotheque> delete from emprunteurs where id= 11;
cqlsh:bibliotheque> select * from emprunteurs;

 id | adresse     | email             | nom
----+-------------+-------------------+-------
 22 | 7500 paris | amira@gmail.com | amira
```

Supprimer un emprunt:

```
cqlsh:bibliotheque> delete from emprunts where id=111;
cqlsh:bibliotheque> select * from emprunts ;

 id  | date_emprunt | date_retoutr | emprunteur_id | livre_id
-----+--------------+--------------+---------------+----------
 222 |   2023-02-02 |   2023-03-03 |            22 |        2

(1 rows)
```

Ajout de fonctionnalités à l'application de gestion de bibliothèque

Catégories et sous-catégories

Création de la table categories:

```
cqlsh:bibliotheque> create table categorie( id UUID PRIMARY KEY, nom text);
cqlsh:bibliotheque>
```

Ajout d'une colonne categorie_id à la table livres:

```
cqlsh:bibliotheque> ALTER TABLE livres ADD categorie_id uuid;
cqlsh:bibliotheque> select * from livres;

 id | auteur                    | categorie_id | genre   | idbn     | titre
----+---------------------------+--------------+---------+----------+-----------------
  1 |              J.R.R. Tolkien |         null | fantasy | 978-2-07 |       le hobbit
  2 | Antoine de saint-exupery |         null |   Conte | 978-2-07 | Le Petit Prince

(2 rows)
```

Gestion des utilisateurs et des permissions

Création de la table utilisateurs:

```
cqlsh:bibliotheque> Create table utilisateurs ( id uuid primary key, nom text,
email text, mo_de_passe text, role text);
cqlsh:bibliotheque>
```

Exemple d'ajout d'un utilisateur avec un rôle:

```
                ...
cqlsh:bibliotheque> insert into utilisateurs (id, nom, email, mo_de_passe, rol
e) values(uuid(), 'ahmed', 'ahmed@gmail.com','password123', 'bibliothecaire');

cqlsh:bibliotheque>
```

CONCLUSION

- Les bases de données orientées colonnes constituent une option intéressante pour les applications nécessitant un traitement analytique rapide de gros volumes de données.
- Leur capacité à compresser efficacement les données et à fournir un accès rapide aux sous-ensembles de colonnes en fait un atout majeur pour le Big Data et l'analytique.
- Cependant, il est important de prendre en compte leurs limitations, telles que les performances pour les requêtes complètes et la complexité de la gestion des transactions ACID, avant de les adopter.