Migration d'un catalogue de restaurants de PostgreSQL vers MongoDB

Optimisation via un cache

Serigne MBAYE, Aurel Vehi

BUT Science des Données

8 janvier 2026

Présentation

Dans le cadre de cette SAE, nous avons développé une application pour **exploiter**, **transformer** et **optimiser** l'accès à des données provenant de deux types de stockage. Nous avons utilisé à la fois une base relationnelle et une base NoSQL afin de comparer leurs usages, formats et performances dans un contexte applicatif réel.

Objectifs et démarche

L'objectif principal est de concevoir une architecture permettant la conversion de données relationnelles vers des formats NoSQL, leur exploitation dans une application, l'optimisation des temps d'accès grâce à un mécanisme de cache, et la recherche efficace de restaurants selon différents critères.
Pour cela :

- Les données sont migrées de PostgreSQL vers MongoDB pour bénéficier de la flexibilité du modèle NoSQL.
- L'application permet de rechercher des restaurants par nom, type de cuisine, quartier ou proximité géographique.
- Un cache PostgreSQL optimise les temps de réponse lors de recherches répétées.
- Des scripts Python assurent la conversion, l'intégration et l'exploitation des données.

PostgreSQL : Structure et Schéma

La base de données relationnelle est stockée sous **PostgreSQL** et contient plusieurs tables, notamment `sql_main`, `sql_feedback` et `sql_geo`. Ces tables sont reliées entre elles par des **jointures parfois complexes**, nécessaires pour combiner les données principales, géographiques et les retours utilisateurs. Afin de simplifier l'exploitation des données, ces tables sont regroupées et converties en un format **JSONB** à l'aide d'un script Python (`conversion.py`).

Un script SQL (`insert_queries.sql`) permet de créer et remplir ces tables automatiquement avant la migration.

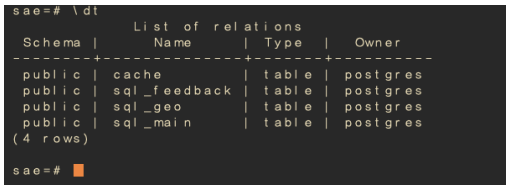
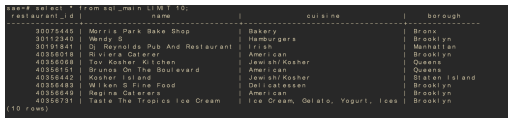
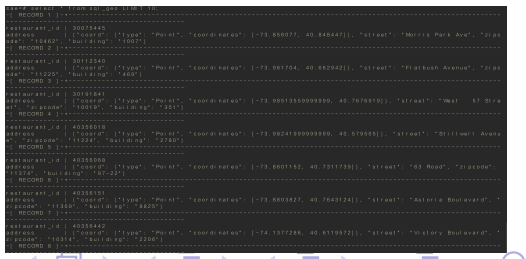


Schéma des tables

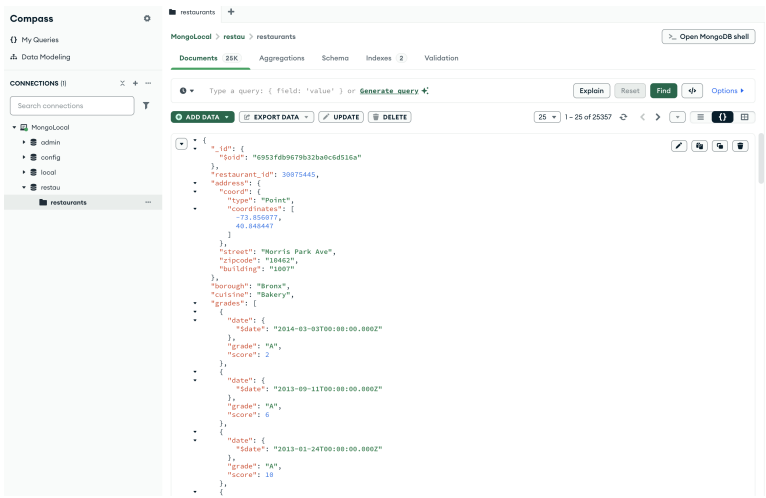


Exemple d'exploitation

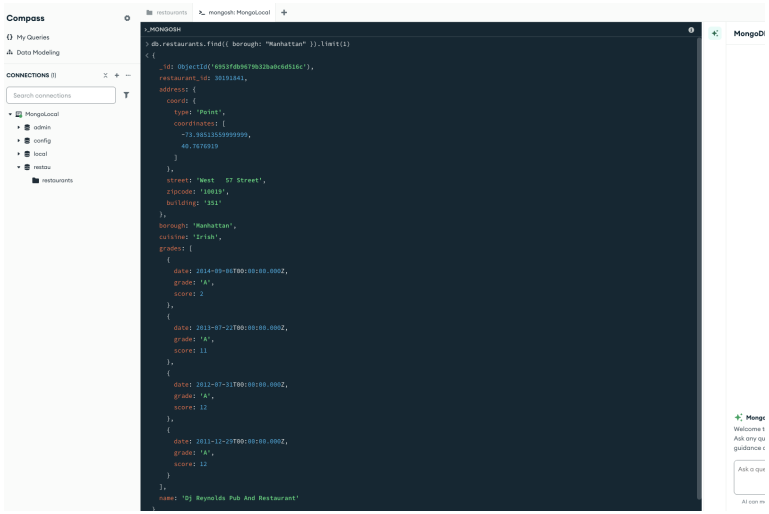


MongoDB : Structure et Exploitation

Après conversion, les données sont importées dans **MongoDB**, une base de données NoSQL orientée documents. Chaque restaurant est stocké sous forme de document JSON, ce qui permet une structure flexible et facilite les traitements analytiques et les requêtes complexes, notamment géospatiales.



Exemple d'exploitation des données MongoDB



Exemple de document stocké dans MongoDB

PostgreSQL vs MongoDB

- Relationnel vs Document
- SQL vs NoSQL
- ACID vs BASE
- Jointures vs Dénormalisation

Script de conversion

- Lecture des données PostgreSQL
- Transformation en documents JSON
- Insertion dans MongoDB
- Validation des données

Calcul de la distance géographique

Pour la recherche géolocalisée des restaurants, la distance entre deux points (latitude, longitude) est calculée à l'aide de la formule de Haversine. Cette méthode permet de déterminer la distance à vol d'oiseau entre deux coordonnées sur la sphère terrestre.

La formule utilisée est :

$$d = 2r \arcsin\left(\sqrt{\sin^2\left(\frac{\Delta\varphi}{2}\right) + \cos(\varphi_1)\cos(\varphi_2)\sin^2\left(\frac{\Delta\lambda}{2}\right)}\right)$$

Paramètres et justification

où :

- d : distance entre les deux points (orthodromique)
- r : rayon de la Terre (environ 6371 km)
- φ_1, φ_2 : latitudes en radians
- $\Delta\varphi$: différence des latitudes
- $\Delta\lambda$: différence des longitudes

Cette méthode est utilisée car elle prend en compte la courbure de la Terre et fournit une estimation précise de la distance réelle entre deux points géographiques, contrairement à un simple calcul euclidien. Elle est donc adaptée pour filtrer efficacement les restaurants à proximité d'un point donné.

Plus d'explications sur la formule de Haversine (Wikipedia)

conversion.py : Conversion des données

Le script `conversion.py` permet de lire les tables `sql_main`, `sql_feedback` et `sql_geo` depuis PostgreSQL, de les regrouper et de les convertir en documents JSONB. Il effectue les jointures nécessaires pour rassembler les informations principales, géographiques et les retours utilisateurs en un seul document par restaurant. Les documents JSONB ainsi créés sont ensuite prêts à être importés dans MongoDB pour une exploitation plus flexible.

Exemple de résultat produit par conversion.py

sae.py : Application finale

Le script `sae.py` constitue le cœur applicatif du projet. Il propose une interface en ligne de commande permettant à l'utilisateur de :

- Saisir une position géographique (latitude/longitude) pour la recherche.
- Filtrer les restaurants par type de cuisine (ou tout afficher).
- Calculer la distance entre la position de l'utilisateur et chaque restaurant grâce à la formule de Haversine.
- Afficher les k restaurants les plus proches selon le filtre choisi.
- Utiliser un système de cache basé sur PostgreSQL pour accélérer les recherches répétées (stockage des résultats pour une même requête).
- Interroger MongoDB pour obtenir les données si le résultat n'est pas en cache.

sae.py : Fonctionnalités principales (suite)

- Gestion de la saisie utilisateur et des coordonnées.
- Recherche et tri par distance.
- Utilisation et mise à jour du cache PostgreSQL.
- Affichage des résultats avec nom, distance et type de cuisine.

```
python3 sae.py

PostgreSQL connection established.
MongoDB connection established.
Latitude Sud-Nord (ex: 41.1): 41.1
Longitude Ouest-Est (ex: -74.0): -74
Type de cuisine recherché (laisser vide pour tout): Italian

Les 5 restaurants les plus proches :
 1. Madison S                          - 22.93 km | Italian
 2. Beccofino                          - 23.20 km | Italian
 3. Patrizia S Of Woodlawn             - 24.80 km | Italian
 4. Linda S Pizza                      - 24.86 km | Italian
 5. Salvatores Of Soho                 - 24.86 km | Italian

Temps d'exécution : 0.130682 secondes

venv ~/Desktop/smartNYCRestaurants git:(main) ±3 +1 -1  Agents ⌘ I
```

Illustration de l'exécution de sae.py (exemple de résultats affichés)

Cache PostgreSQL I

Le système de cache implémenté avec PostgreSQL permet d'optimiser les performances de l'application lors des recherches répétées. Lorsqu'une requête (coordonnées, nombre de résultats, type de cuisine) a déjà été effectuée récemment, le résultat est stocké dans une table dédiée (`cache`). Si la même requête est soumise à nouveau, le résultat est directement récupéré depuis PostgreSQL sans interroger MongoDB ni recalculer les distances.

Fonctionnement du cache :

- Recherche dans la table `cache` si une entrée correspondante existe.
- Si oui, affichage immédiat des résultats stockés (gain de temps).

Cache PostgreSQL II

- Sinon, calcul classique (MongoDB + Haversine), puis insertion du résultat dans le cache.
- **Gestion mémoire :** le cache est limité à 20 entrées pour éviter une surcharge mémoire. Si la limite est atteinte, la table est vidée avant d'insérer de nouveaux résultats.

Ce mécanisme améliore significativement la réactivité de l'application pour les requêtes fréquentes ou répétées tout en maîtrisant l'utilisation de la mémoire côté PostgreSQL.

Cache PostgreSQL III

```
venv ~/Desktop/smartNYCRestaurants git:(main) ±3  2 files changed, 1 insertion(+), 1 deletion(-) (39.093s)
python3 sae.py
PostgreSQL connection established.
MongoDB connection established.
Latitude Sud-Nord (ex: 41.1): 40.2
[ERROR] Latitude hors intervalle [40.5, 41.2]
Latitude Sud-Nord (ex: 41.1): 41.2
Longitude Ouest-Est (ex: -74.0): -74
Type de cuisine recherché (laisser vide pour tout): Afghan

Les 5 restaurants les plus proches :
 1. Balkh Shish Kabab House          - 47.89 km | Afghan
 2. Afghan Kebab House               - 48.15 km | Afghan
 3. Ariana Kebab House               - 48.37 km | Afghan
 4. Afghan Kebab House #1            - 48.48 km | Afghan
 5. Arya Kabob House                 - 51.32 km | Afghan

Temps d'exécution : 0.125225 secondes
```

Première exécution : temps d'exécution ≈ 0.125s (requête non présente dans le cache, calcul complet).

Cache PostgreSQL IV

```
venv ~/Desktop/smartNYCRestaurants git:(main) ±3  2 files changed, 1 insertion(+), 1 deletion(-) (16.335s)
python3 sae.py
PostgreSQL connection established.
MongoDB connection established.
Latitude Sud-Nord (ex: 41.1): 41.2
Longitude Ouest-Est (ex: -74.0): -74
Type de cuisine recherché (laisser vide pour tout): Afghan
[INFO] Résultats depuis le cache PostgreSQL.
 1. Balkh Shish Kabab House          - 47.89 km | Afghan
 2. Afghan Kebab House               - 48.15 km | Afghan
 3. Ariana Kebab House               - 48.37 km | Afghan
 4. Afghan Kebab House #1            - 48.48 km | Afghan
 5. Arya Kabob House                 - 51.32 km | Afghan
Temps d'exécution : 0.003476 secondes

venv ~/Desktop/smartNYCRestaurants git:(main) ±3 +1 -1  Agents ⌘ I
python3 sae.py tab
```

Seconde exécution : temps d'exécution ≈ 0.0034s (résultat récupéré directement depuis le cache PostgreSQL).

Cache PostgreSQL V

Commentaire : On constate que l'utilisation du cache permet de réduire drastiquement le temps de réponse pour une même requête utilisateur. La première exécution nécessite l'accès à MongoDB et le calcul des distances, alors que la seconde lecture est quasi-instantanée grâce au cache.

Conclusion

Ce projet a permis de comparer les approches relationnelles et NoSQL pour le stockage et l'exploitation de données de restaurants. La migration automatisée, l'intégration dans MongoDB et l'optimisation des accès via le cache PostgreSQL ont montré l'intérêt de combiner ces technologies. Des pistes d'amélioration restent possibles, notamment sur la gestion des performances et l'enrichissement des fonctionnalités de l'application.

Références

[1] MongoDB Documentation *MongoDB Manual*
https ://docs.mongodb.com

[2] Psycopg2 Documentation *PostgreSQL adapter for Python*
https ://www.psycopg.org

[3] Projet sur GitHub *smartNYCRestaurants*
https://github.com/JVRLC/smartNYCRestaurants

Merci !

Démos et questions ?