Projet NoSQL

Vous travaillez pour une start-up qui gère une application de recommandations de films. Celle-ci est victime de son succès et l'augmentation du nombre d'utilisateurs rend votre base de données relationnelle difficile et coûteuse à gérer. Vous êtes missionné-e pour étudier les possibilités de migration vers une base de données NoSQL. Vous décidez d'adapter une partie de l'application pour exploiter deux bases de données NoSQL que vous connaissez bien (Neo4j et MongoDB) et de comparer les performances de chacune.

Dans ce projet, vous allez :

- Déployer en local une partie de l'application – partie 1,
- Ré-implémenter les accès à la base de données pour Neo4j et NoSQL – partie 2,
- Construire un benchmark afin de produire un rapport objectif permettant de déterminer quel est le meilleur choix de base de données – partie 3.

Partie 1 : mise en place

La première partie consiste à déployer en local l'application fournie et à la connecter à vos bases de données :

1. Clonez le dépôt github suivant : https://github.com/Camille31/MovieRecommender.
2. Créez dans votre IDE (Eclipse, Netbeans ou autre) un projet Maven à partir du fichier pom.xml.
3. Attachez à ce projet un serveur d'application (de préférence Tomcat).
4. Déployez l'application sur le serveur. Visitez l'url http://localhost:8080/MovieRecommender/hello (à adapter selon la configuration du serveur déployé) pour vous assurer que tout s'est bien déroulé.
5. Pour déployer les bases de données nécessaires à la réalisation de ce projet (MySQL, Neo4j et MongoDB), vous avez deux possibilités :
    a. Installer et déployer sur votre machine (droits administrateur requis) une instance de chaque base de données,
    b. Utiliser la VM fournie contenant les instances préconfigurées et déployées.
6. Peuplez la base de données MySQL en exécutant le programme CsvToMySql.java après avoir modifié les variables `url`, `login` et `password` pour qu'elles correspondent à votre configuration.
7. Vérifiez que l'application est fonctionnelle avec le backend MySQL (il est à nouveau nécessaire de modifier les informations de connexion dans la classe MySqlDatabase). Visitez par exemple l'url http://localhost:8080/MovieRecommender/movies ; la liste des films de la base de données doit apparaître.

Architecture de l'application

La classe MainController définit cinq vues :

- `MainController:showMessage()`

    appelée lorsque l'on visite la page http://localhost:8080/MovieRecommender/hello ou http://localhost:8080/MovieRecommender/hello?name=<NAME>; permet de vérifier

que l'application tourne correctement sans prendre en compte les accès à la base de données.

- `MainController:showMovies()`

  appelée lorsque l'on visite la page http://localhost:8080/MovieRecommender/movies ou http://localhost:8080/MovieRecommender/movies?user_id=<USERID>; affiche la liste de tous les films en base, ou, si un utilisateur est spécifié, la liste des films notés par cet utilisateur.

- `MainController:showMoviesRattings()`

  appelée lorsque l'on visite la page http://localhost:8080/MovieRecommender/movieratings?user_id=<USERID>; affiche les notes laissées par l'utilisateur spécifié et permet d'ajouter ou de modifier des notes.

- `MainController:saveOrUpdateRating()`

  appelée suite à l'indication d'une note pour un film depuis la page http://localhost:8080/MovieRecommender/movieratings?user_id=<USERID> (requête `http POST` suite à la soumission d'un formulaire) ; écrit en base la mise à jour portant sur la note et redirige vers `showMoviesRattings()`.

- `MainController:ProcessRecommendations()`

  appelée lorsque l'on visite la page http://localhost:8080/MovieRecommender/recommendations?user_id=<USERID>; affiche une liste de films suggérés pour l'utilisateur spécifié (non implémentée ; renvoie simplement une liste statique de films).

Partie 2 : ré-implémentation des accès à la base de données

Cette deuxième partie consiste à compléter les accès à la base de données dans le code de l'application web. Celle-ci est déjà fonctionnelle avec une base de données MySQL. De nouveaux accès seront écrits pour Neo4j et MongoDB.

La classe `MainController` accède à la base de données via le champ `AbstractDatabase db` de (actuellement instancié avec la classe `MySqlDatabase`).

Pour chacune des deux bases de données à intégrer ;

1. Ajoutez les données à la base de données cible (utilisez la même procédure des TP précédants),
2. Définissez une classe implémentant l'interface `AbstractDatabase` et instanciez le champ `db` avec cette nouvelle classe,
3. Intégrez au projet les librairies nécessaires à l'établissement d'une connexion entre l'application et la base de données,
4. Implémentez les fonctions définies par l'interface `AbstractDatabase` de façon à ce qu'elles aient le même comportement que celles de `MySqlDatabase`.

Recommandations

Un système de recommandation est une forme spécifique de filtrage de l'information visant à présenter les éléments d'information (films, musique, livres, news, images, pages Web, etc) qui sont susceptibles d'intéresser l'utilisateur. Pour cela, il ordonne les éléments en prédisant pour chacun le

score que l'utilisateur lui donnerait (c'est pourquoi la fonction que vous allez éditer calcule une liste d'objets `Rating`).

Généralement, un système de recommandation permet de comparer le profil d'un utilisateur à certaines caractéristiques de référence, et cherche à prédire l'« avis » que donnerait cet utilisateur. Ces caractéristiques peuvent provenir de :

- l'objet lui-même, on parle « d'approche basée sur le contenu » ou *content-based approach*,
- l'utilisateur,
- l'environnement social, on parle d'approche de filtrage collaboratif ou *collaborative filtering*.

Vous allez implémenter trois variantes de la troisième stratégie. Dans celle-ci, les informations portant sur le comportement de l'utilisateur (achats, notes…) sont utilisée pour établir des similarités avec les autres utilisateurs ; ces similarités sont ensuite exploitées pour recommander les éléments que les utilisateurs similaires ont recommandé dans le passé.

Les appels à la base de données seront écrits pour Neo4j et MongoDB dans la fonction `processRecommendationsForUser()`. Cette méthode est appelée par `MainController` lorsque l'on visite la page http://localhost:8080/MovieRecommender/recommendations?user_id=<USERID>&processing_mode=<PROCESSINGMODE> (l'argument `PROCESSINGMODE` permet de sélectionner la variante utilisée pour calculer les recommandations).

Variante 1 : l'utilisateur le plus proche

Nous définissons u_{sim} l'utilisateur le plus similaire à un utilisateur u comme celui qui a évalué le plus grand nombre de films en commun avec u. Ainsi, les recommandations retournées pour l'utilisateur u seront l'ensemble des films évalués par u_{sim} mais non évalués par u, ordonnées en fonction de la note donnée par u_{sim}.

La requête Neo4j permettant d'obtenir cette liste ordonnée est la suivante :

```
MATCH (target_user:User {id : 1})-[:RATED]->(m:Movie) <-[:RATED]-(other_user:User)
WITH other_user, count(distinct m.title) AS num_common_movies, target_user
ORDER BY num_common_movies DESC
LIMIT 1
MATCH other_user-[rat_other_user:RATED]->(m2:Movie)
WHERE NOT (target_user-[:RATED]->m2)
RETURN m2.title AS rec_movie_title, rat_other_user.note AS rating,
       other_user.id AS watched_by
ORDER BY rat_other_user.note DESC
```

Variante 2: les cinq utilisateurs les plus proches

Dans cette variante sont considérés les cinq utilisateurs les plus proches de u. Notez que dans ce cas, un film peut apparaître plusieurs fois s'il a été noté par plusieurs utilisateurs. Les films doivent être ordonnés par la moyenne des notes qui leur sont attribuées et en cas d'égalité, les films ayant reçus le plus de notes sont retournés en priorité.

Variante 3: prise en compte de la valeur des scores

Une faiblesse des approches précédentes et qu'elles ne prennent en compte, pour calculer la similarité entre deux utilisateurs, que le fait que les films ont été évalués ou non, et non la similarité entre les notes attribuées, c'est-à-dire la similitude entre les goûts des utilisateurs.

Modifiez la méthode pour prendre en compte cet élément. Nous proposons d'utiliser le score de similarité entre utilisateurs suivants :

$$sim(u_1, u_2) = \sum_{m \in scored(e_1) \cap scored(e_2)} 4 - |score(u_1, m) - score(u_2, m)|$$

Cette formule permet d'obtenir une similarité qui est maximale (4) lorsque les utilisateurs ont donné le même score au film considéré et qui baisse au fur et à mesure que les scores divergent (pour atteindre 0 si par exemple le score de u_1 est à 1 et celui de u_2 à 5).

Partie 3 : Benchmark

Vous voulez maintenant choisir définitivement entre Neo4j et MongoDB avant de poursuivre le développement de votre application. Vous allez pour cela simuler l'utilisation de l'application par de nombreux utilisateurs. L'objectif étant de *benchmarker* les deux technologies, notamment en ce qui concerne leur capacité à passer à l'échelle.

La classe `TestGetRecommendation.java` donne un exemple très simple de programme mesurant le temps nécessaire au traitement d'un nombre donné de requêtes. Vous pouvez faire évoluer ce code pour simuler une utilisation plus réaliste du service ou bien utiliser un outil spécialisé comme JMeter.

Vous devez à l'issu de cette troisième partie rédiger un rapport expliquant vos expériences et votre choix. Vous trouverez ci-dessous des suggestions de points à aborder dans vos rapports. Cette liste n'est ni exhaustive, ni obligatoire.

- Justifier les méthodes utilisées pour simuler une utilisation réaliste :
    - plusieurs utilisateurs exploitant l'appli en simultanée.
    - des utilisateurs plus actifs que d'autres (regardent plus ou moins de films, se connectent plus ou moins souvent).
    - Les utilisateurs suivent régulièrement les recommandations qui leur sont faites ; c'est-à-dire qu'ils vont regarder et évaluer un film qui leur a été recommandé peu avant.

  Vous pouvez implémenter des comportements supplémentaires dans le but d'augmenter la fidélité de la simulation. Vous n'êtes pas non plus obligés d'implémenter chacun des exemples donnés ci-dessus ; pensez dans ce cas à justifier pourquoi vous en avez laissé de côté.

- Comment garantir une comparaison des performances toutes choses égales par ailleurs ? Autrement dit, comment peut-on s'assurer que l'on compare bien uniquement les systèmes de base de données entre eux et qu'aucun élément externe ne va modifier les résultats ?
- Deux méthodes essentielles dans l'application finale (la création d'utilisateurs et la création de films) n'ont pas encore été implémentées et ne sont donc pas prises en compte dans le benchmark. Dans quelle mesure cela peut-il fausser les résultats ?
- Enfin, vous pouvez énoncer quelques idées d'évolution de l'appli (concernant les parties implémentées jusque-là) pour améliorer les performances.