**Configuration recommandée dans application.yml**

spring:

datasource:

url: jdbc:postgresql://localhost:5432/your\_database

username: your\_username

password: your\_password

driver-class-name: org.postgresql.Driver

hikari:

maximum-pool-size: 10

minimum-idle: 5

idle-timeout: 300000

max-lifetime: 1800000

connection-timeout: 30000

pool-name: HikariPool-YourApp

auto-commit: true

data-source-properties:

cachePrepStmts: true

prepStmtCacheSize: 250

prepStmtCacheSqlLimit: 2048

useServerPrepStmts: true

rewriteBatchedStatements: true

jpa:

properties:

hibernate:

jdbc:

batch\_size: 50

order\_inserts: true

order\_updates: true

temp:

use\_jdbc\_metadata\_defaults: false

hibernate:

ddl-auto: none

**Explications des paramètres clés**

1. **HikariCP (Pool de connexions)** :
   * maximum-pool-size: 10 : Limite le nombre maximum de connexions dans le pool. Une valeur de 10 à 20 est souvent suffisante pour des charges élevées, mais cela dépend de la capacité de votre base de données et du nombre de threads de l'application. Trop de connexions peut surcharger PostgreSQL.
   * minimum-idle: 5 : Nombre minimum de connexions inactives maintenues dans le pool. Une valeur plus basse réduit la consommation de ressources lorsque l'application est peu sollicitée.
   * idle-timeout: 300000 (5 minutes) : Temps après lequel une connexion inactive est fermée, libérant des ressources.
   * max-lifetime: 1800000 (30 minutes) : Durée de vie maximale d'une connexion dans le pool pour éviter les problèmes de connexions obsolètes.
   * connection-timeout: 30000 (30 secondes) : Temps maximum d'attente pour obtenir une connexion. Une valeur raisonnable évite les blocages prolongés.
2. **Optimisation des requêtes PostgreSQL** :
   * cachePrepStmts: true : Active le cache des prepared statements côté serveur pour réduire la surcharge de parsing des requêtes répétitives.
   * prepStmtCacheSize: 250 : Taille du cache pour les prepared statements. Une valeur plus élevée est utile pour les applications avec de nombreuses requêtes différentes.
   * prepStmtCacheSqlLimit: 2048 : Limite de longueur des requêtes mises en cache. 2048 est une valeur standard pour la plupart des cas.
   * useServerPrepStmts: true : Utilise les prepared statements côté serveur PostgreSQL, réduisant la charge sur le client.
   * rewriteBatchedStatements: true : Optimise les requêtes par lots (batch) en les réécrivant pour améliorer les performances des insertions ou mises à jour massives.
3. **JPA/Hibernate pour données volumineuses** :
   * hibernate.jdbc.batch\_size: 50 : Définit la taille des lots pour les insertions/mises à jour. Une valeur entre 20 et 50 est généralement efficace pour réduire les allers-retours réseau sans surcharger la mémoire.
   * hibernate.order\_inserts: true : Ordonne les insertions pour regrouper les opérations sur la même table, optimisant l'exécution.
   * hibernate.order\_updates: true : Même principe pour les mises à jour.
   * hibernate.ddl-auto: none : Désactive la génération automatique du schéma pour éviter des modifications accidentelles en production.
   * temp.use\_jdbc\_metadata\_defaults: false : Désactive les vérifications inutiles des métadonnées JDBC, améliorant les performances au démarrage.

**Conseils supplémentaires**

* **Tuning PostgreSQL** :

La performance dépend aussi de la configuration de PostgreSQL (ex. : max\_connections, work\_mem, shared\_buffers). Assurez-vous que la base de données peut gérer le nombre de connexions défini dans maximum-pool-size.

* **Monitoring** :

Utilisez des outils comme Micrometer ou Actuator pour surveiller l'utilisation du pool de connexions et ajuster maximum-pool-size ou minimum-idle si nécessaire.

* **Requêtes par lots** :

Si vous traitez des données volumineuses, utilisez JdbcTemplate ou Spring Data JDBC pour des insertions par lots, car JPA peut être moins efficace pour des opérations massives.

* **Indexes et requêtes optimisées** :

Assurez-vous que vos tables ont des index appropriés et que vos requêtes sont optimisées (utilisez EXPLAIN dans PostgreSQL).

**Avertissement**

* **Testez votre configuration** :

Les valeurs suggérées sont des points de départ.

Effectuez des tests de charge pour ajuster les paramètres en fonction de votre cas d'utilisation.

* **Évitez les fuites de connexions** :

Vérifiez que toutes les transactions sont correctement fermées pour éviter d'épuiser le pool.

Si vous avez des détails spécifiques sur votre charge de travail (ex. : nombre d'utilisateurs, volume de données, type de requêtes), je peux affiner davantage la configuration !