Java Database Connectivity

1 Concepts

- Driver
- Connection
- Pooling
- Statement
- PreparedStatement
- read/write
- Batch

2 JDBC : classes et APIs

Interfaces	Classes
– Driver – Connection [i] – Statement [ii] – PreparedStatement [iii] – CallableStatement [iv] – ResultSet [v] – ResultSetMetaData – DatabaseMetaData	– DriverManager – Blob – Clob – Types

3 Connection à la BD

3.1 Simple sans pooling

- Enregistrer la classe du pilote : méthode forName de la classe Class

```
Class.forName("oracle.jdbc.driver.OracleDriver");
```

- Créer une connexion : utiliser une des surcharges suivantes de getConnection de DriverManager

- public static Connection getConnection(String url) throws SQLException
- public static Connection getConnection(String url,String name,String password) throws SQLException

- Fermer la connexion : méthode close de Connection

```
public void close()throws SQLException
```

3.2 Avec pooling

Selon l'implémentation utilisée, il s'agit généralement d'une méthode statique retournant un objet de type Connection. Cet objets devrait être libéré pour permettre au gestionnaire du Pool (cache) de le réaffecter ailleurs. Il faut se fier à la documentation de la librairie utilisée pour les bonnes pratiques et faire attention aux drées de timeouts de cette location de connexion.

Exemples de frameworks : Apache Commons DBCP[vi], HikariCP[vii], C3PO

4 Requêtage de la BD

4.1 Avec requêtes non paramétrées (Statement)

- Créer un objet de type Statement la méthode createStatement() de la classe Connection

```
public Statement createStatement() throws SQLException
```

- Exécuter la requête en utilisant une des méthodes executeQuery ou executeUpdate de Statement selon l'objectif (resp. lecture ou écriture)

- public ResultSet executeQuery(String sql)throws SQLException
- public int executeUpdate(String sql)

- Parcourir les enregistrements retournés par les requêtes select via la méthode next() de ResultSet.
- Récupérer les colonnes de la ligne courante par les méthodes getXXX (remplacer XXX par le type) et en passant comme paramètre l'index de la colonne ou son nom.
- Par défaut le curseur ne se déplace que vers l'avant (statement créé avec paramètre TYPE_SCROLL_INSENSITIVE). Ce comportement peut être changé en spécifiant le statement de type (TYPE_SCROLL_SENSITIVE). Selon la nature du curseur, d'autres méthodes peuvent être utilisées pour le positionnement dans la liste des résultats.
- Pour gérer les transactions, la classe Connection propose les méthodes suivantes :
  - public void setAutoCommit(boolean status): indique si les commits sont fait automatiquement à la suite de chaque requête de type écriture (true par défaut).
  - public void commit(): persiste les modifications depuis le dernier commit/rollback à la BD de manière permanente..
  - public void rollback(): annule toutes les modifications depuis le dernier commit/rollback.

4.2 Avec requêtes paramétrées (PreparedStatement)

- Sous type de Statement.
- Utiles pour les requêtes à forte réutilisation.
- Permet de préparer une requête paramétrée (parfois même précompilée selon le driver) qui ne nécessite que le remplacement de ses paramètres par leurs valeurs effectives pour pouvoir la relancer.
- Dépend fortement de la configuration du cache des requêtes.
- Exemple

```
String sql="insert into MaTable values(?,?,?)";
```

Pour utiliser les prepared statements :

- Utiliser la méthode prepareStatement de Connection.

```
public PreparedStatement prepareStatement(String query)throws SQLException{}
```

- Remplacer les paramètres par leurs valeurs effectives selon leurs positions dans la requête.

```
public void setXXX(int paramIndex, int value) : remplacer XXX par le type du paramètre
```

- Lancer la requête avec executeQuery() ou executeUpdate()

• public ResultSet executeQuery() • public int executeUpdate()

- Exploiter les résultats.
- Réinitialiser les paramètres.

void clearParameters() throws SQLException

- Retour vers l'étape de replacement des paramètres.

4.3 Appel aux procédures stockées

- Utiliser la méthode prepareCall de la connexion pour le CallableStatement nécessaire aux appels aux fonctions/procédures stockées.

```
CallableStatement stmt=con.prepareCall("{call maProcedure(?,?)}");

CallableStatement stmt=con.prepareCall("{?= call maFonction(?,?)}");
```

- Utiliser les méthodes setXXX pour le remplacement des paramètres.
- Utiliser la méthode execute() pour lancer l'exécution de la fonction/procédure.
- Utiliser la méthode getXXX selon le type du résultat.

5 Traitements par lot (Batch)

- Lancement simultané de plusieurs requêtes (généralement d'écriture)
- Réduction des aller/retours à la BD
- Meilleure parallélisation/débit

Pour profiter de ce mode de fonctionnement, utiliser les méthodes suivantes de la classe Statement :

- void addBatch(String query) : ajouter une requête au lot.
- int[] executeBatch() : exécuter le lot de requêtes.
- void clearBatch() : vider le batch.

6 Manipulation des données binaires

- https://www.javatpoint.com/storing-file-in-oracle-database
- https://www.javatpoint.com/storing-image-in-oracle-database
- https://www.javatpoint.com/retrieving-image-from-oracle-database

[i] https://docs.oracle.com/javase/8/docs/api/java/sql/Connection.html
[ii] https://docs.oracle.com/javase/8/docs/api/java/sql/Statement.html
[iii] https://docs.oracle.com/javase/8/docs/api/java/sql/PreparedStatement.html
[iv] https://docs.oracle.com/javase/8/docs/api/java/sql/CallableStatement.html
[v] https://docs.oracle.com/javase/8/docs/api/java/sql/ResultSet.html
[vi] https://commons.apache.org/proper/commons-dbcp/
[vii] https://github.com/brettwooldridge/HikariCP