Database esup-utils-mag

But

Ce package a pour objet de faciliter les connexions aux bases de données et l'exécution des requêtes.

Services proposés

- Gestion des connexion déclarées par le canal
- Encapsulation des ordres SQL : select, insert, delete, update
- Gestion des transactions via l'auto-commit et le rollback
    - Mettre l'auto-commit à false (true par défaut), exécuter les ordres SQL, faire un commit explicite
- Gestion du niveau d'isolation des transactions
    - TRANSACTION_READ_COMMITTED par défaut (n'autorise l'accès à aucune valeur jusqu'à ce qu'elles aient été validées dans la base de données)

Utilisation

Déclaration des connexions d'un canal

Chaque canal déclare dans son fichier de configuration les connexions qu'il est susceptible d'utiliser, et ce de la manière suivante :

```
<pools>
        <pool *name="nom logique du pool"
                *type="JNDI ou JDBC"
                *url="Nom du pool tomcat si JNDI ou URL du driver JDBC"
                driverClassName="classe du driver si JDBC"
                username="nom du user pour la connexion JDBC"
                password="password pour la connexion JDBC"/>
        ......
</pools>

* = obligatoire
```

Il est à noter que les balises "pool" désignent soit un pool Tomcat (type=JNDI), soit une connexion JDBC simple (type=JDBC).

- **Type JNDI** : Fortement recommandé pour des applications volumineuses car ce type s'appuie sur un mécanisme de gestion de pool pour les connexions.
- **Type JDBC** : Ne gère pas de pool de connexion. Une connexion est ouverte puis fermée à chaque exécution d'une requête si ce type est retenu. Ce type est plus approprié lorsque que l'on ne veut pas qu'une connexion sensible (car avec des privilèges importants) ne soit potentiellement mise à la disposition de tous via le fichier server.xml de Tomcat. Cela permet également de savoir précisément ce que l'on fait lors d'un rollback car on travaille sur une connexion propre.
- Attention le nom d'un pool Tomcat ne doit pas comporter la partie **jdbc/** (par exemple, on fera référence au pool tomcat jdbc/Apogee par le nom Apogee)

Ces connexions sont représentées dans le package par la classe Database.

Récupération de la liste des connexions

Pour pouvoir utiliser les connexions, on récupère sous forme de hashMap la liste des connexions définies dans le fichier de configuration du canal. Il s'agit d'une liste d'objet Database.

```
private HashMap connexions = Config.getInstance().getPools();
```

où Config est la classe permettant de parser le fichier de configuration.

Remarque :

Si on utilise le package avec le framework MAG, il n'est pas nécessaire de définir la méthode getPools(). Le framework fournit un comportement par défaut pour la gestion des connexions. Deux autres méthodes sont également disponibles : setPools() et getConnexionDefault().

Exécution d'une requête : objet Query

Deux façons d'instancier un objet Query :

- Soit en utilisant la connexion définie comme "default" (attribut name) dans le fichier de configuration :

```
qry = Config.getInstance().getConnexionDefault();
```

- Soit en utilisant la hashMap de connexions avec comme clé le nom logique du pool

```
qry = new Query((Database) connexions.get("Apogee1"));
```

Il suffit alors d'effectuer son ordre SQL de la façon suivante :

```
try {
        qry = new Query((Database) connexions.get("Apogee1"));
        qry.setSQL("select distinct a.g_n2_cod || '|' || a.g_n3_cod \"CODE\",....
                        "from nw_adm_acces@NABUCOWEB a, g_nv3@NABUCOWEB n " +
                        "where a.login = ? and n.g_n1_cod = a.g_n1_cod .... ");
        // on passe les paramètres à la requête
        qry.getStmt().setString(1, login);
        // on exécute l'ordre SQL
        qry.select();
        while (qry.getRs().next()){
           <traitement du resultset>
        }
}
catch (SQLException e) {
        LogService.log(LogService.ERROR ,<msg d'erreur>);
}
finally {
        qry.close();
}
```

Il est important de noter que qry.close() ferme le statement, le resultset et la connexion si besoin est (ie si on ne travaille pas avec un pool Tomcat).

Remarques :

- Si l'on veut effectuer plusieurs requêtes SQL indépendantes, on n'utilise qu'un seul objet Query mais on fait plusieurs qry.setSQL(). En effet, la méthode setSQL() ferme le resultset et le statement en cours à chaque appel.
- Pour le cas des requêtes imbriquées, il faut utiliser plusieurs objets Query et veiller à bien fermer les connexions ainsi ouvertes.