

JDBC

JDBC : Introduction

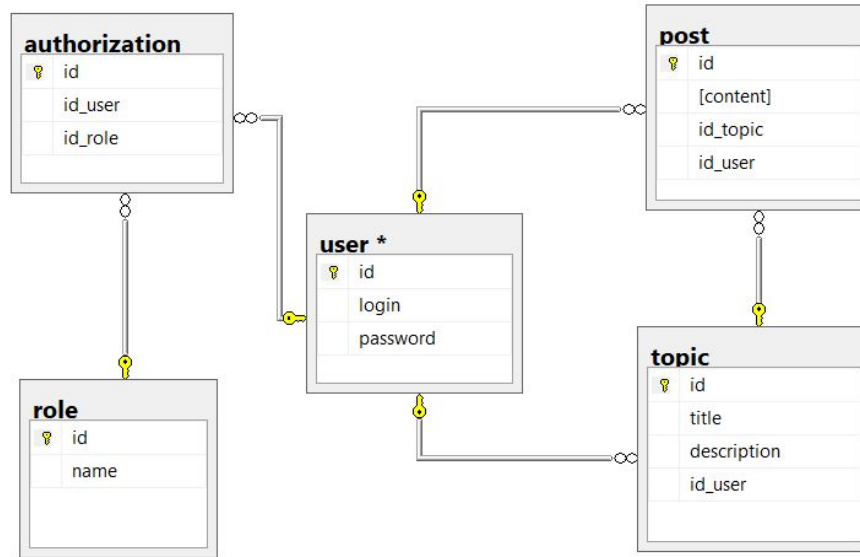
- Java DataBase Connectivity.
- Interface de programmation (API) développée par SUN.
- Permet d'accéder à des bases de données relationnelles (à condition d'avoir la librairie correspondante).
- Accessible depuis JAVA 1.1 (dernière modification en JAVA 1.4).

JDBC : Fonctionnalités

- Établir une connexion avec le SGBD.
- Envoyer une requête SQL au SGBD.
- Récupérer les données retournées par le SGBD.

Présentation du cas d'exemple

Dans ces slides, nous ferons la connexion vers la DB (en postgresql) d'un forum assez simple.



JDBC : Le driver

- Installer le driver (.jar) permettant de faire la connexion le SGBD (chacun a son propre driver).
 - Soit on l'inclut dans le projet même dans le CLASSPATH.
=> à faire si l'on souhaite partager notre projet.
 - Soit on l'inclut dans le dossier lib/ext de notre IDE (ex : `C:\Program Files\Java\jdk1.8.0_121\jre\lib\ext`).
=> à faire si on ne veut faire l'opération qu'une fois pour tous les projets.

JDBC : La connexion

- Établir une connexion avec la base de données
 - La première ligne n'est plus obligatoire depuis JDBC 4.
 - Il faut avoir créé un user au préalable dans le SGBD (avec les droits de connexion).
 - Il faut importer Connection et DriverManager de java.sql.

```
try {  
    Class.forName("org.postgresql.Driver"); // PAS OBLIGATOIRE !!!  
    System.out.println("Driver O.K.");  
  
    String url = "jdbc:postgresql://localhost:5432/forum";  
    String user = "postgres";  
    String passwd = "postgres";  
  
    Connection co = DriverManager.getConnection(url, user, passwd);  
    System.out.println("Connexion effective !");  
} catch (Exception e) {  
    e.printStackTrace();  
}
```

JDBC : Les requêtes

Émettre une requête (=query), récupérer l'information et la traiter.

```
String query = "SELECT id, name FROM topic";
Statement statement = co.createStatement();
ResultSet resultSet = statement.executeQuery(query);

while(resultSet.next())
{
    int id = resultSet.getInt("id");
    String nom = resultSet.getString("name");
    System.out.println(id + " " + nom);
}
resultSet.close(); // on ferme le resultSet
statement.close(); // on ferme le statement
co.close(); // on ferme la connexion
```

Il est préférable de fermer les objets dans la clause finally.

JDBC : Le Statement

- L'interface "Statement" permet l'envoi d'instruction SQL au SGBD.
- Nous récupérons le "Statement" grâce à la méthode "createStatement" de la connexion créée au préalable.
- Il existe d'autre type d'exécution de requête.
 - "execute" : retourne un booléen.
 - "executeUpdate" : retourne un entier (utilisée pour les ordres DDL), provoque une erreur si la requête retourne des données.
 - "executeQuery" : retourne un "ResultSet".

JDBC : Le ResultSet

- L'interface ResultSet permet de récupérer les résultats d'une query (select).
- Un objet ResultSet contient toutes les lignes (row) qui correspondent aux résultats récupérés ainsi que des colonnes (column) et un curseur (cursor).
- Offre des méthodes afin de parcourir les données ("next", "first", "last", ...).
- Offre des méthodes afin de récupérer les valeurs de la ligne où est placé le curseur ("getString", "getInt", "getDouble", ...).

JDBC : Le ResultSet

Vous pouvez mettre à jour vos données via le ResultSet grâce aux méthodes “updateInt”, “updateString”, ...

Cependant, il faut que notre Statement l’autorise et pour cela, il faut l’instancier via un autre constructeurs (une surcharge).

=> `createStatement(int resultSetType, int resultSetConcurrency)`

```
Statement statement = co.createStatement(ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
```

`ResultSet.CONCUR_UPDATABLE` signifie que l’on peut mettre à jour les données, à l’inverse de `ResultSet.CONCUR_READ_ONLY`.

JDBC : Le ResultSet

resultSetType : `ResultSet.TYPE_SCROLL_INSENSITIVE` signifie que les données sont insensible à des modifications ayant eu lieu entre la récupération et le traitement.
Exemple : Vous récupérez tous les users à 08h00 et à 08h10 vous souhaitez récupérer le login du dernier user qui, à 08h05, a changé son login. Vous aurez toujours l'ancienne valeur.

`ResultSet.TYPE_SCROLL_SENSITIVE` permet d'avoir les nouvelles valeurs.

Une fois la valeur modifiée, il faut sauvegarder via la méthode "updateRow".

On peut annuler les modifications avec la méthode "cancelRowUpdates".

JDBC : exemple update ResultSet

```
String query = "SELECT name FROM topic";
Statement statement = co.createStatement( ResultSet.TYPE_SCROLL_SENSITIVE , ResultSet.CONCUR_READ_ONLY );

ResultSet resultSet = statement.executeQuery(query);

while (resultSet.next()) {
    String value = resultSet.getString("name");
    if(value.equals("JDBC est ennuyant")){
        resultSet.updateString("name", "JDBC est génial !!!");
        resultSet.updateRow();
    }
}
```

JDBC : Les MetaData

Les metadatas sont des données servant à décrire une structure (par exemple : le nom de la table, nombre de colonne, type de champs, ...).

Voici un exemple affichant le nom des colonnes, leur type JAVA ainsi que leur type en SQL.

```
for(int i = 1; i <= metaData.getColumnCount(); i++){  
    System.out.print(metaData.getColumnName(i) + "\t");  
    System.out.print(metaData.getColumnClassName(i) + "\t");  
    System.out.print(metaData.getColumnTypeName(i) + "\t");  
    System.out.println();  
}
```

Attention, l'indice commence à 1 !!!

JDBC : Utilisation du ResultSet + MetaData

Imaginons que nous ne sachions pas ce que nous allons récupérer, grâce aux metaData, nous pouvons faire ceci :

```
String query = "SELECT * FROM topic";
Statement statement = co.createStatement();
ResultSet resultSet = statement.executeQuery(query);
ResultSetMetaData metaData = resultSet.getMetaData();

for (int i = 1; i <= metaData.getColumnCount(); i++) {
    String columnName = metaData.getColumnName(i);
    System.out.print(columnName + "\t");
}
System.out.println();

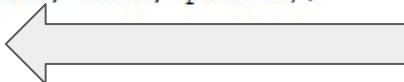
while (resultSet.next()) {
    for (int i = 1; i <= metaData.getColumnCount(); i++) {
        String columnName = metaData.getColumnName(i);
        Object value = resultSet.getObject(columnName);
        System.out.print(value + "\t");
    }
    System.out.println();
}
```

On utilise
.getObject
car on ne sait pas ce
que l'on va recevoir

Try-with-resources statement

Java 1.7 (= JAVA 7) nous offre les “*try-with-resources statement*” qui vont fermer les connection à notre place. **Attention à la syntaxe !**

```
try (  
    Connection connection = DriverManager.getConnection(url, user, passwd);  
    Statement statement = connection.createStatement()  
)  
{  
    try (ResultSet resultSet = statement.executeQuery("Votre requête")) {  
        // Faites ce que vous voulez avec le resultSet  
    }  
    try (ResultSet resultSet = statement.executeQuery("Votre seconde requête")) {  
        // Faites ce que vous voulez avec le resultSet  
    }  
} catch (SQLException e) {  
    // TODO Auto-generated catch block  
    e.printStackTrace();  
}
```



pas de ;
car c'est
le dernier

JDBC : Requête avec données de l'utilisateur

Demandons à l'utilisateur de se connecter.

```
// Demander le login et mot de passe
Scanner sc = new Scanner(System.in);
System.out.println("Entrez votre login");
String login = sc.nextLine();
System.out.println("Entrez votre password");
String password = sc.nextLine();

String query = "SELECT id FROM user WHERE login = '" + login + "' ";
query += "AND password = '" + password + "'";

Statement statement = co.createStatement();
ResultSet resultSet = statement.executeQuery(query);
```

Notez les apostrophes avant de concaténer le login/password.

Tous les champs ont été récupérés en tant que "String" dans ce cas-ci.

JDBC : L'injection SQL

Il y a une énorme faille de sécurité, que se passerait-il si l'utilisateur rentrait n'importe quel login et comme mot de passe : “ OR ' ' = ' “ ?

On peut imaginer pire, il pourrait avoir accès à toute votre DB, supprimer des infos, ...

=> **Injection SQL !!!**

Voyons comment résoudre ce problème.

JDBC : Le PreparedStatement

Afin d'éviter l'injection SQL, nous allons utiliser des "PreparedStatement" au lieu de "Statement" => + performant et + sécurisé.

```
String query = "SELECT id FROM users WHERE login = ? AND password = ?";  
PreparedStatement pst = co.prepareStatement(query);  
pst.setString(1, login);  
pst.setString(2, password);  
  
ResultSet resultSet = pst.executeQuery();
```

Notez l'absence des apostrophes par rapport à la dernière fois et l'apparition des "?". Les "?" vont être remplacés par les valeurs affectées par la suite.

Il existe aussi un "executeUpdate" et un "execute".

Attention, le premier "?" commence à 1 et non 0 !!!

JDBC : avancé

JDBC offre encore plus de fonctionnalités comme :

- Pouvoir appeler des procédures stockées via “CallableStatement”.
- Gérer les transactions (regroupement de requêtes pouvant être annulées si l’une d’elle échoue).

Veillez à bien vérifier les possibilités de JDBC par rapport au SGBD que vous utilisez.