Connectivité d'une application à une BD (JDBC)

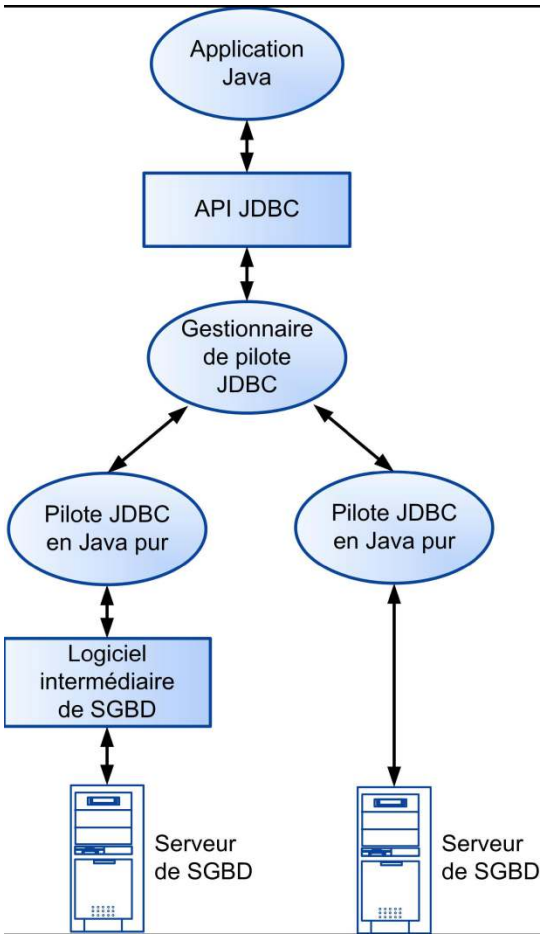
Comment interroger et modifier le contenu d'une BD dans un programme Java ?

Introduction JDBC

- JDBC (Java DataBase Connectivity) est une API Java utilisé pour se connecter à une BD
  - Le SGBD doit posséder une interface JDBC
  - La base doit être déclarée accessible (point de connexion)
- JDBC est composé de deux interfaces
  - Rédacteur d'applications
  - Rédacteur de pilotes
- C'est le cœur des API de plus haut niveau
  - + : accès standardisé au SGBD acceptant des fonctionnalités SQL
  - - : très proche du modèle relationnel, la persistance des objets d'applications n'est pas transparente → commandes SQL

Architecture JDBC

- Plusieurs types de pilotes
  - Pilote JDBC en Java pur + connexion directe (très performant)
  - Pilote JDBC en Java pur via middleware (déploiement sur SGBD distribués)
  - Pilote JDBC adHoc (conversion JDBC vers API client)
  - Pont JDBC-ODBC (peu performant)



Interfaces JDBC… de base

- DriverManager: gestion des pilotes disponibles
- Connection: établissement d'une connexion avec une base
- Statement: exécution d'une requête SQL à travers une connexion
- ResultSet: gestion des résultats d'une requête sous forme d'une liste d'éléments.
- SQLException: encapsule les erreurs lors des accès à la base.
- Canevas d'utilisation des interfaces:
  - importer le package java.sql (import java.sql.*)
  - Enregistrement du pilote JDBC correspondant,
  - Connexion au point d'entrée JDBC de la base,
  - Création d'un descripteur de requête,
  - Exécution d'une requête SQL,
  - Parcours de la liste des résultats et traitement par l'application,
  - Terminer (fermer) la connexion.

Exemple d'utilisation 1/5

- Soit la table Fichier(NumEtudiant, Nom, Prenom, Annee, Poursuite) :
  - NumEtudiant : numéro étudiant (clef primaire)
  - Nom : nom de l'étudiant (chaîne de caractère),
  - Prenom : prénom de l'étudiant (chaîne de caractère),
  - Annee : promotion (chaîne de caractère),
  - Poursuite : situation à la sortie de la formation (chaîne de caractère).

- La table est accessible sur la base ufrima de im2ag-oracle. Un spécialiste a déclaré cette base dans le SGBD via la chaine de connexion:

jdbc:oracle:thin:@im2ag-oracle.e.ujf-grenoble.fr:1521:ufrima

Exemple d'utilisation 2/5

- On commence par définir le pilote JDBC qui permet l'accès à la base.
  - On crée un objet du pilote correspondant ici Oracle.
  - On enregistre ensuite ce pilote dans le gestionnaire JDBC.

Ex : DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());

- On peut ensuite se connecter à la BD :
  - On crée un descripteur sur la BD de type `Connection`,
  - On utilise la méthode `getConnection(String, String, String)`
    - 1er paramètre : chaine de connexion (à connaître),
    - 2eme paramètre : nom d'utilisateur
    - 3eme paramètre : mot de passe de l'utilisateur

Ex : Connection base = DriverManager.getConnection(
"jdbc:oracle:thin:@im2ag-oracle.e.ujf-grenoble.fr:1521:ufrima", "jouanotf",
"bd2009");

Exemple d'utilisation 3/5

- Pour poser une requête, il faut créer un descripteur de requête :
  - On crée un objet de type `Statement`,
  - On utilise la méthode `createStatement()` de l'objet de type `Connection`

Ex : Statement requete = base.createStatement();

- Pour envoyer une requête sur la base (sous la forme d'une chaine de caractère) et récupérer les résultats, on utilise la méthode `executeQuery(String )` de l'objet `Statement`

```
Ex : ResultSet resultat = requete.executeQuery(
                "SELECT Nom, Prenom, Poursuite " +
                "FROM fichier " +
                "WHERE Annee = '1997'");
```

Exemple d'utilisation 4/5

- Exploitation des résultats d'une requête
  - Les méthodes `next()` et `previous()` appelées sur un objet de type ResultSet permettent de parcourir les éléments de la liste.
  - La méthode `getString(String )` retourne la valeur de l'attribut dont le nom est passé en paramètre

```
Ex : while(resultat.next()) {
        System.out.println("Nom = " + resultat.getString("Nom")
        + ", Prenom = " + resultat.getString("Prenom")
        + ", Travail = " + resultat.getString("Poursuite"));
   }
```

- Libération des ressources à l'aide de la méthode `close()`.

Exemple d'utilisation 5/5

```
import java.sql.*;

public class Travail {
   public static void main(String[] arg) {
      try {
         DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver()); // Chargement du pilote
         Connection base = DriverManager.getConnection ("jdbc:oracle:thin:@hopper.e.ujf-grenoble.fr:1521:ufrima",
      "jouanotf", "bd2009"); // connexion
         Statement requete = base.createStatement(); // création du descripteur de requête
         ResultSet resultat = requete.executeQuery( // exécution d'une requête
                  "SELECT Nom, Prenom, Poursuite " +
                  "FROM fichier " +
                  "WHERE Annee = '1997'");
         while(resultat.next()) { // récupération des résultats
            System.out.println("Nom = " + resultat.getString("Nom")
                     + ", Prenom = " + resultat.getString("Prenom")
                     + ", Travail = " + resultat.getString("Poursuite"));
         }
         requete.close();
         resultat.close();
         base.close(); // fermeture de la connexion
      } catch (Exception err) { System.out.println("Une erreur, Oh Oh!"); } // Attention il faut capturer les
      exceptions !
   }
}
```

Gestion des transactions en JDBC

- Changer la gestion de l'auto-commit
  - Méthode `setAutoCommit(true | false)` de la classe `Connection`.
- Définir une transaction
  - Méthode `commit()` de `Connection`
  - Méthode `rollback()` de `Connection`
- Niveaux d'isolation
  - Méthode `setTransactionIsolation(READ_COMMITED | SERIALIZABLE)` de Connection
  - Méthode `getTransactionIsolation()` retourne le niveau d'isolation courant.