Java DataBase Connectivity

Master TIIL & ILIADE 1ère année

Eric Cariou

Université de Bretagne Occidentale
UFR Sciences et Techniques – Département Informatique

Eric.Cariou@univ-brest.fr

Octobre 2016

Introduction

- JDBC : Java DataBase Connectivity
- Framework permettant l'accès aux bases de données relationnelles dans un programme Java
  - Indépendamment du type de la base utilisée (MySQL, Oracle, Postgres ...)
    - Seule la phase de connexion au SGBDR change
  - Permet de faire tout type de requêtes SQL
    - Sélection de données dans des tables
    - Création de tables et insertion d'éléments dans les tables
    - Gestion des transactions
    - Récupérer des informations sur la structure de la BDD
- Packages : `java.sql` et `javax.sql`

Principes généraux d'accès à une BDD

- Première étape
  - Préciser le type de driver que l'on veut utiliser
    - Driver permet de gérer l'accès à un type particulier de SGBD
- Deuxième étape
  - Récupérer un objet « Connection » en s'identifiant auprès du SGBD et en précisant la base utilisée
- Étapes suivantes
  - A partir de la connexion, créer un « statement » correspondant à une requête particulière (définie par une chaîne de caractères)
  - Exécuter ce statement au niveau du SGBD
    - Si exécute un SELECT, on récupère un tableau de résultats
  - Fermer le statement
- Dernière étape
  - Se déconnecter de la base en fermant la connexion

Exemple BDD

- Exemple de gestion de sports et de disciplines
  - Un sport se compose de plusieurs disciplines
  - Deux tables définies dans un schéma nommé « sports »
    - sport (*code_sport*_, intitule)
      discipline (*code_discipline*, intitule, code_sport)
- SGDB considéré
  - Serveur MySQL
  - En local sur le port 3306
- En salles de TP
  - Serveur Maria DB (MySQL version open source)

Exemple BDD

- Contenu des tables pour l'exemple
  - Table sport
    -
```
code  |
sport | intitule
------+-----------
    1 | athlétisme
    2 | ski
    3 | natation
```
  - Table discipline
    -
```
code       |                     | code
discipline | intitule            | sport
-----------+---------------------+------
         1 | 100 mètres          | 1
         2 | 200 mètres          | 1
         3 | saut en hauteur     | 1
         4 | saut en longueur    | 1
         5 | 200m 4 nages        | 3
         6 | 100m papillon       | 3
         7 | marathon            | 1
```

Exception SQLException

- Toutes les méthodes d'accès au SGBD peuvent lever l'exception SQLException
  - Exception générique lors d'un problème d'accès à la base lors de la connexion, d'une requête ...
    - Plusieurs spécialisations sont définies (voir API)
  - Opérations possibles sur cette exception
    - int getErrorCode() : le code de l'erreur renvoyé par le SGBD (et dépendant du type du SGBD)
    - SQLException getNextException() : si plusieurs exceptions sont chaînées entre elles, retourne la suivante ou null s'il n'y en a pas
    - String getSQLState() : retourne « l'état SQL » associé à l'exception

Connexion au SGBD

- Classe java.sql.DriverManager
  - Gestion du contrôle et de la connexion au SGBD
- Méthodes principales
  - static void registerDriver(Driver driver)
    - Enregistre le driver pour un type de SGBD particulier
      - Le driver est dépendant du SGBD utilisé
    - Facultatif : peut retrouver le driver dans JAR associés au projet
  - static Connection getConnection(String url, String user, String password)
    - Crée une connexion permettant d'utiliser une base
    - url : identification de la base considérée sur le SGBD
      - Format de l'URL est dépendant du SGGB utilisé
    - user : nom de l'utilisateur qui se connecte à la base
    - password : mot de passe de l'utilisateur

Exemple connexion

```
public class AccesBDDSports {

private Connection conn = null;

public void connexionSGBD() {
        try {
                // enregistrement du driver spécifique à mySQL
                DriverManager.registerDriver(new com.mysql.jdbc.Driver());

                // connexion au schéma « sports » avec utilisateur « eric »
                conn = DriverManager.getConnection(
                        "jdbc:mysql://localhost:3306/sports", "eric", "eric");
        }
        catch (SQLException ex) {
                System.err.println(" Problème de connexion : "+ex) ;
        }
}
```

Exécution requêtes SQL

- 3 types d'instructions exécutables sur la BDD
  - Instruction simple : classe Statement
    - On exécute directement et une fois l'action sur la base
  - Instruction paramétrée : classe PreparedStatement
    - L'instruction est générique, des champs sont non remplis
    - Permet une pré-compilation de l'instruction optimisant les performances
    - Pour chaque exécution, on précise les champs manquants
  - Appel d'une procédure ou fonction stockée dans la BDD : classe CallableStatement
  - Pour ces instructions, 2 types d'ordres possibles
    - Update : mise à jour du contenu de la base
    - Query : consultation (avec un select) des données de la base

Exécution requêtes SQL

- Méthodes de Connection
  - Statement createStatement()
    - Retourne un objet permettant de réaliser une instruction simple
  - PreparedStatement prepareStatement(String ordre)
    - Retourne un objet permettant de réaliser une instruction paramétrée et pré-compilée pour optimiser les performances
      - Si on a besoin d'exécuter plusieurs fois la même requête avec des valeurs différentes
    - Dans l'ordre passé en paramètre de la méthode, des champs libres (au nombre quelconque) sont précisés par des « ? »
      - Avant l'exécution de l'ordre, on précisera la valeur des champs
  - void close()
    - Ferme la connexion avec le SGBD

Instruction simple

- Classe Statement
  - ResultSet executeQuery(String ordre)
    - Exécute un ordre de type SELECT sur la base
    - Retourne un objet de type ResultSet contenant tous les résultats de la requête
  - int executeUpdate(String ordre)
    - Exécute un ordre de type INSERT, UPDATE, ou DELETE
    - Retourne le nombre de lignes ajoutées/modifiées
  - void close()
    - Ferme l'instruction

Instruction paramétrée

- Classe PreparedStatement
  - Avant d'exécuter l'ordre, on remplit les champs avec
    - void set[Type](int index, [Type] val)
      - Remplit le champ en i^{ème} position définie par index avec la valeur val de type [Type]
      - [Type] peut être : String, int, float, long …
      - Ex : void setString(int index, String val)
  - ResultSet executeQuery()
    - Exécute un ordre de type SELECT sur la base
    - Retourne un objet de type ResultSet contenant tous les résultats de la requête
  - int executeUpdate()
    - Exécute un ordre de type INSERT, UPDATE, ou DELETE
    - Retourne le nombre de lignes ajoutées/modifiées

Lecture des résultats

- Classe ResultSet
  - Contient les résultats d'une requête SELECT
    - Plusieurs lignes contenant plusieurs colonnes
    - On y accède ligne par ligne puis colonne par colonne dans la ligne
  - Changements de ligne
    - boolean next()
      - Se place à la ligne suivante s'il y en a une
      - Retourne true si le déplacement a été fait, false s'il n'y avait pas d'autre ligne
    - boolean previous()
      - Se place à la ligne précédente s'il y en a une
      - Retourne true si le déplacement a été fait, false s'il n'y avait pas de ligne précédente
    - boolean absolute(int index)
      - Se place à la ligne numérotée index
      - Retourne true si le déplacement a été fait, false sinon

Lecture des résultats

- Classe ResultSet
  - Accès aux colonnes/données dans une ligne
    - Soit par la position de la colonne, soit par son nom
  - [type] get[Type](int colPosition)
    [type] get[Type](String colName)
    - Retourne le contenu de la colonne en position colPosition ou de nom colName dont l'élément est de type [type] avec [type] pouvant être String, int, float, boolean ...
      - Ex : String getString(int colPosition) ou

        String getString(String colName)
  - Fermeture du ResultSet
    - void close()

Correspondances types Java – SQL

- Chaque type SQL à une correspondance en Java
  - Existera méthodes get/set[Type] associées

Type SQL	Type Java
CHAR, VARCHAR	String
NUMERIC, DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT, DOUBLE	double
BINARY, VARBINARY, LONGVARBINARY	byte[ ]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Exemple : instruction simple

- Exécution d'une instruction simple de type SELECT
  - Afficher tous les sports dans la base

```
// création du statement via la connexion au SGBD
Statement req = conn.createStatement();

// exécution de la requête SELECT
ResultSet res = req.executeQuery("SELECT * FROM SPORT") ;

// parcourt les lignes de résultat une par une
while (res.next()) {
        // colonne « intitule » de type VARCHAR donc String coté Java
        String intitule = res.getString("intitule");
        // colonne « code_sport » de type INTEGER donc int coté Java
        int code = res.getInt("code_sport");
        System.out.println(code + " - " + intitule);
}
```

Exemple : instruction simple

- Résultat affiché pour notre base

  1 – athlétisme
  2 – ski
  3 – natation

- Variantes pour récupérer le contenu des colonnes
  - « code_sport » est la première colonne de la table
  - « intitule » est la deuxième colonne de la table
  - La numérotation des colonnes commence à 1

```
while (res.next()) {
        String intitule = res.getString(2);
        int code = res.getInt(1);
        System.out.println(code + " - " + intitule);
}
```

Exemple : instruction paramétrée

- Instruction paramétrée de type SELECT
  - Afficher toutes les disciplines d'un certain sport passé en paramètre de la méthode

```
public void afficherDisciplinesSport(String sport) throws SQLException {

        // prépare la requête SQL en laissant un paramètre
        PreparedStatement reqParam = conn.prepareStatement("
            SELECT DISCIPLINE.INTITULE FROM DISCIPLINE, SPORT
            WHERE SPORT.INTITULE = ? AND
             SPORT.CODE_SPORT=DISCIPLINE.CODE_SPORT");

        // remplit le paramètre avec le sport passé en paramètre
        reqParam.setString(1, sport);

        //exécute la requête et affiche les résultats
        ResultSet res = reqParam.executeQuery();
        System.out.println(" Les disciplines du sport "+sport);
        while (res.next()) System.out.println(" -> "+res.getString(1));
}
```

Exemple : instruction paramétrée

- Exécution de

```
AccesSportsJDBC acces = new AccesSportsJDBC();
acces.connexionSGBD();
acces.afficherDisciplinesSport("athlétisme");
```

- Donne affichage suivant

Les disciplines du sport athlétisme
-> 100 mètres
-> 200 mètres
-> saut en hauteur
-> saut en longueur
-> marathon

Exemple : séquence d'instructions

- Ajout d'une discipline pour un sport donné
  - Entrées
    - Intitulé de la discipline et intitulé du sport
  - Étapes
    1. Récupérer la clé primaire du sport
    2. Récupérer la valeur max de la clé primaire dans la table discipline pour définir une clé primaire unique
    3. Insérer le triplet de la discipline dans la table

```
public void ajouterDiscipline(String intitule, String sport) throws SQLException {

        Statement req = conn.createStatement();
        ResultSet res = req.executeQuery("SELECT CODE_SPORT FROM
                                        SPORT WHERE INTITULE = '"+sport+"'");
        boolean trouve = res.next();
        if (!trouve) {
                // lève une exception pour signaler que le sport n'existe pas
                throw new SQLException("Sport "+sport+ "non trouvé");
        }
        ...
```

Exemple : séquence d'instructions

```
    ...
    // récupère le code du sport
    int codeSport = res.getInt("CODE_SPORT");

    // calcul de la clé primaire de la nouvelle discipline
    req = conn.createStatement();
    res = req.executeQuery("
              SELECT MAX(CODE_DISCIPLINE) FROM DISCIPLINE");
    res.next();
    int codeDisc = res.getInt(1);
    codeDisc++;

    // insertion de la nouvelle discipline
    PreparedStatement reqParam = conn.prepareStatement("
                              INSERT INTO DISCIPLINE VALUES (?,?,?)") ;
    reqParam.setInt(1, codeDisc);
    reqParam.setString(2, intitule);
    reqParam.setInt(3, codeSport);
    int nb = reqParam.executeUpdate();
    System.out.println(" Nb de disciplines insérées : " + nb);
}
```

Exemple : séquence d'instructions

- Exécution de

```
acces.ajouterDiscipline("descente", "ski");
acces.ajouterDiscipline("slalom", "ski");
acces.afficherDisciplinesSport("ski");
```

- Donne affichage suivant

Nb de disciplines insérées : 1
Nb de disciplines insérées : 1
Les disciplines du sport ski
-> descente
-> slalom

Transaction

- Principe
  - Exécution d'une action ou d'une séquence d'actions
    - Ici des requêtes de modification de bases de données
  - Soit par un seul élément / processus
  - Soit par plusieurs
    - Cas des transactions distribuées
- Exemple de séquence d'action
  - Transfert d'argent d'un compte vers un autre
    - Requière un débit *puis* un crédit
    - Il faut faire les 2 actions ou aucune sinon on se retrouve dans un état incohérent
    - Begin Transaction
      Debiter (#1244, 1000€)
      Crediter (#8812, 1000€)
      End Transaction

Transaction

- Propriétés d'une transaction
  - Propriétés ACID [Härder & Reuter, 83]
  - Atomicité
    - Tout ou rien : l'action de la transaction est entièrement réalisée ou pas du tout, pas d'intermédiaire à moitié fait
  - Cohérence
    - L'exécution d'une transaction fait passer le système d'un état cohérent à un autre
  - Isolation
    - Les transactions n'interfèrent pas entre elles
  - Durabilité
    - Les effets de la transaction sont enregistrés de manière permanente

Transaction

- En JDBC par défaut
  - Toute requête est directement exécutée et validée sur la base
- Pour réaliser une transaction en JDBC, à partir de l'objet de connexion sur la BDD
  1. Désactiver la validation automatique
     - conn.setAutoCommit(false)
  2. Exécuter une série de requêtes
  3. Si pas de problème, valider (« commit »), sinon revenir sur les modifications (« roll back »)
     - conn.commit() ou conn.rollback()
  4. Réactiver la validation automatique
     - conn.setAutoCommit(true)

Transaction : exemple

- Double ajout de disciplines en mode transactionnel

```
try {
        // désactivation de l'auto validation
        conn.setAutoCommit(false);

        // modifications sur la base
        this.ajouterDiscipline("descente", "ski");
        this.ajouterDiscipline("slalom", "ski");

        // validation des modifications
        conn.commit();
}
catch (Exception ex) {
        // en cas de problème, on annule les modifications
        try { conn.rollback(); }
        catch (SQLException sqlEx) {
                System.err.println(" Pb avec rollback : " + sqlEx); }
        }
}

// réactive la validation automatique
try { conn.setAutoCommit(true); }
catch (SQLException ex) { System.err.println(" Pb avec auto commit : " + ex); }
```

Transaction : points de sauvegarde

- Par défaut, un « roll back » annule toutes les modifications non encore validées
- On peut placer dans la séquence d'instructions des points de sauvegarde et valider des modifications jusqu'à un point donné
- Classe Connection
  - Placement d'un point de sauvegarde avec un nom optionnel
    - Savepoint setSavepoint(String name)
      Savepoint setSavepoint()
  - Variante de la méthode rollback pour valider jusqu'à un certain point de sauvegarde
    - rollback(Savepoint sp)

Transaction : ex. points sauvegarde

```
Savepoint s1=null, s2=null;
try {
        conn.setAutoCommit(false);

        this.ajouterDiscipline("biathlon", "ski");
        s1 = conn.setSavepoint("Biathlon");

        this.ajouterDiscipline("slalom", "ski");
        s2 = conn.setSavepoint();

        this.ajouterDiscipline("contre la montre", "cyclisme");

        conn.commit();
}
catch (Exception e) {
        try { conn.rollback(s1); }
        catch (SQLException sqlEx) { System.err.println(" Pb avec rollback : " + sqlEx); }
}

try { conn.setAutoCommit(true); }
catch (SQLException ex) { System.err.println(" Pb avec auto commit : " + ex); }
```

Transaction : ex. points sauvegarde

- Ajouts de trois disciplines avec un point de sauvegarde entre chaque
  1. Ajout du « biathlon » pour le « ski »
  2. Point de sauvegarde 1
  3. Ajout du « slalom » pour le « ski »
  4. Point de sauvegarde 2
  5. Ajout du « contre la montre » pour le « cyclisme »
  6. Validation des modifications
- Le sport « cyclisme » n'existe pas dans la base
  - Exception levée lors de l'ajout du « contre la montre »
  - Exécution du roll back sur le premier point de sauvegarde
  - Le « biathlon » sera inséré dans la base mais pas le « slalom » dont l'insertion se fait après ce point de sauvegarde

Méta-Données

- Méta-données
  - Données sur les données
- On peut récupérer en JDBC
  - Des informations générales sur un SGBD
    - Produit, version, driver
    - Utilisateur connecté
    - Fonctionnalités supportées
    - …
  - Des informations sur les schémas/tables du SGBD
    - Description des tables, des clés primaires et étrangères
    - …
  - Voir API de la classe DatabaseMetaData pour la liste exhaustive des fonctionnalités

Méta-Données : exemple

- Exemple : info sur le serveur

```
DatabaseMetaData meta = conn.getMetaData();
System.out.println("Produit : " + meta.getDatabaseProductName()
                                  + " " + meta.getDatabaseProductVersion());
System.out.println("Utilisateur : " + meta.getUserName());
System.out.println("Driver JDBC : " + meta.getDriverName()
                                        + " " + meta.getDriverVersion());
System.out.println("URL de connexion : " + meta.getURL());
System.out.println("Base en lecture seulement : " + meta.isReadOnly());
```

- Résultat de l'exécution

Produit : MySQL 5.7.14-log
Utilisateur : eric@localhost
Driver JDBC : MySQL Connector Java mysql-connector-java-5.1.39 (Revision: 3289a357af6d09ecc1a10fd3c26e95183e5790ad)URL de connexion : jdbc:mysql://localhost:3306/sports
Base en lecture seulement : false

Méta-Données : exemple

- Code suivant affiche la structure des tables

```
DatabaseMetaData meta = conn.getMetaData();
ResultSet tables = meta.getTables(null, null, null, null);
while (tables.next()) {
        String nomTable = tables.getString("TABLE_NAME");
        System.out.print(nomTable + " ( ");
        ResultSet colonnes = meta.getColumns(null, null, nomTable, null);
        boolean premier = true;
        while (colonnes.next()) {
                String nomColonne = colonnes.getString("COLUMN_NAME");
                int type = colonnes.getInt("DATA_TYPE");
                String nomType = JDBCType.valueOf(type).getName();
                if (!premier) System.out.print(" , ");
                premier = false;
                System.out.print(nomType);
                if (type == Types.VARCHAR)
                        System.out.print("[" + colonnes.getInt("COLUMN_SIZE") + "]");
                System.out.print(" " + nomColonne);
        }
        System.out.println(" )");
}
```

Méta-Données : exemple

- Résultat affiché

  discipline (INTEGER code_discipline , VARCHAR[45] intitule , INTEGER code_sport)
  sport (INTEGER code_sport , VARCHAR[45] intitule)

- Fonctionnement général
  1. Récupérer l'ensemble des tables : un ResultSet
  2. Parcourir l'ensemble des tables
     1. Récupérer le nom de la table courante
     2. Récupérer l'ensemble des colonnes de la table courante
     3. Afficher le détail de chaque colonne une par une
- Voir API pour le détail des signatures des différentes méthodes