TME JDBC

Introduction:

L'objectif de ce TME est savoir accéder à un ou plusieurs SGBD depuis une application java, en utilisant l'interface **JDBC.** Savoir parcourir le résultat d'une requête, définir une requête paramétrée puis l'invoquer, présenter le résultat d'une requête, interroger le dictionnaire (méta-données) du SGBD. Savoir mettre en œuvre l'évaluation de requêtes réparties en utilisant JDBC : combiner les résultats de plusieurs requêtes posées sur différents SGBD.

Le serveur de données (Oracle sur la machine db-oracle), est un SGDB relationnel supportant l'interface JDBC. Il contient la base de données tennis dont le schéma est :

Joueur2 (NuJoueur, Nom, Prenom, AnNaiss, Nationalite)

Gain2 (NuJoueur, LieuTournoi, Annee, Prime, Sponsor**)**

Rencontre2 (NuGagnant, NuPerdant, LieuTournoi, Annee, Score**)**

Les attributs NuGagnant, NuPerdant et NuJoueur sont définis sur le même domaine. Les clés des relations sont soulignées.

Accès à une base de données avec JDBC

JDBC permet d'accéder au SGBD depuis une application écrite en langage java. Afficher la documentation en ligne du TME (lien TME JDBC depuis la page d'accueil), puis afficher la documentation java (lien API).

Utilisation des bibliothèques Java

Les bibliothèques java sont regroupées en packages. Un package contient plusieurs interfaces et plusieurs classes. Une classe (et une interface) contient des méthodes et des variables. La documentation en ligne permet de naviguer dans les bibliothèques pour déterminer les classes, interfaces et méthodes à utiliser pour construire une application java. Les classes et interfaces JDBC du package ***java.sql*** permettent d'accéder à une base de données. Naviguer dans la documentation du package java.sql pour répondre aux questions suivantes :

a) Donner le nom et le type des paramètres des méthodes getConnection de la classe DriverManager.

b) A quoi sert la méthode *next* de l'interface ResultSet ?

c) A quoi sert la méthode *setMaxRows* de l'interface Statement ?

d) Quelles sont les sous-interfaces de l'interface Statement ?

1 Première connexion à une base de données

Installer l'environnement logiciel (la procédure d'installation est détaillée sur la fiche technique, ci-après) puis exécuter le programme Joueur.
1.1) Etudier le programme Joueur.java. (voir en annexe). Commenter brièvement les lignes importantes pour expliquer chaque étape du programme.
1.2) Sur une feuille (à faire chez soi), représenter sous la forme d'un graphe, les scénarios d'accès à une base de donnée en utilisant les interfaces, classes et les méthodes de JDBC. Le graphe est défini comme suit :

- un **nœud** (rectangle) représente une interface ou une classe.
- un **arc** orienté (flèche) représente une méthode. L'arc est tel que :
- son origine est l'interface dans laquelle la méthode est définie,
- sa destination est l'interface du résultat de la méthode.

Le graphe doit contenir les classes, interfaces et méthodes du package java.sql qui sont utilisées dans *Joueur.java*, ainsi que les éléments suivants : PreparedStatement, ResultSetMedatata, DatabaseMetadata, executeUpdate, int, String et getMetaData.

2 Accès paramétré à la base de données

2.1) Exécuter le programme MaxPrime (saisir une année, par ex. 1992). Editer le fichier MaxPrime.java. Compléter l'en-tête avec vos nom et prénom. Compléter tous les commentaires pour expliquer ce que fait le programme.
2.2) Copier et modifier le programme précédent pour obtenir le programme *MaxPrime2.java* qui exécute la même requête **en boucle,** en demandant à chaque itération une nouvelle valeur à l'utilisateur. Le programme MaxPrime2 se termine lorsque la saisie de l'utilisateur est vide. Pour améliorer les performances du programme, deux conditions sont requises :

- la connexion vers le SGBD est ouverte une seule fois pendant toute la durée du programme (*i.e.,* réutiliser le même objet de type Connection à chaque itération).

- le SGBD analyse la requête une seule fois pendant toute la durée du programme (*i.e.,* utiliser l'interface PreparedStatement pour définir une requête paramétrée ; utiliser la méthode setInt pour affecter une valeur à un paramètre de la requête).

Remarque : ne pas oublier d'appeler le constructeur de MaxPrime2. (*ie*., remplacer new MaxPrime() par new MaxPrime2().

3 Requête générique

3.1) Quel est le rôle de l'interface ResultSetMetaData et de sa méthode getColumnCount ?

3.2) Compléter le programme *Generique.java* pour traiter une requête SQL quelconque passée en paramètre, et afficher les valeurs des tuples du résultat sous la forme « val_1 val_2 … val_n » (un tuple par ligne).

Tester l'exécution avec la requête donnant tous les Joueurs nés en 1972 :

```
java Generique ''select * from Joueur2 where annaiss = 1972''
```

3.3) Compléter le programme pour afficher en en-tête le **nom des attributs** du résultat.

Exemple d'exécution (ne pas chercher à tabuler le résultat)

```
NUJOUEUR    NOM         PRENOM      ANNAISS     NATIONALITE
-------------------------------------------------------------
1           MARTINEZ    Conchita    1972        Espagne
14          SAMPRAS     Pete        1972        Etats-Unis
```

3.4) Ecrire le programme *GeneriqueXHTML.java* qui affiche le résultat d'une requête quelconque dans un tableau formaté en XHTML. Tester l'exécution en stockant le résultat dans un fichier :

```
java GeneriqueXHTML ''select * from Joueur2'' > resultat.html
```

Exemple de résultat :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-
transitional.dtd">
<html>
  <head ><title>Résultat</title></head>
<body>
  <h3>La requete est : </h3> select * from Joueur2
  <h3>le resultat est : </h3>
  <table border="2">
   <tr><th>NUJOUEUR</th><th>NOM</th><th>PRENOM</th><th>ANNAISS</th><th>NATIONALITE</th></tr>
   <tr><td>1</td><td>MARTINEZ</td><td>Conchita</td><td>1972</td><td>Espagne</td></tr>
   <tr><td>2</td><td>NAVRATILOVA</td><td>Martina</td><td>1957</td><td>Etats-Unis</td></tr>
   …
   <tr><td>14</td><td>SAMPRAS</td><td>Pete</td><td>1972</td><td>Etats-Unis</td></tr>
  </table>
</body>
</html>
```

Visualiser graphiquement le résultat dans un navigateur avec la commande :

```
netscape resultat.html (ou mozilla resultat.html)
```

4 Schéma d'une relation

4.1) Quel est le rôle de l'interface DatabaseMetaData et comment obtenir une instance de ce type à partir d'une connexion ? Expliquer la méthode getColumns de l'interface DatabaseMetaData. Dans un SGBD les relations sont généralement organisées en hiérarchie à 2 niveaux (catalogue et schéma). Ainsi, une relation appartient à un schéma lui-même appartenant à un catalogue. Dans Oracle, l'organisation n'a qu'un seul niveau : une relation appartient à un schéma égal au nom de celui qui a créé la relation, il n'y a pas de niveau catalogue.

4.2) Créer le programme Schema.java qui affiche le nom et le type des attributs d'une relation passée en paramètre.

Exemple d'exécution

```
java Schema JOUEUR2                    // écrire le nom de la relation en MAJUSCULE
Le schéma de JOUEUR2 est :             // résultat affiché
NOM          TYPE
------------------------------
NUJOUEUR           NUMBER
NOM                VARCHAR2
PRENOM             VARCHAR2
ANNAISS            NUMBER
NATIONALITE        VARCHAR2
```

5 Jointure inter-bases

Soit une deuxième base de données (située dans un autre SGBD différent du premier), contenant la relation

SPONSOR (NOM, NATIONALITE) // nationalité des sponsors

La deuxième base de données est accessible seulement en lecture, par une connexion à la base oracle en tant qu'utilisateur «anonyme» avec le mot de passe «anonyme».La relation Sponsor contient **100 000** tuples. Soit la requête R1 :«Donner le nom et la nationalité des joueurs avec le nom et la nationalité de leurs sponsors, dans l'ordre des noms de joueur».

5.1) Quelle est la durée approximative d'une lecture séquentielle de la relation Sponsor ? Répondre en écrivant le programme *Generique2.java* (obtenu en modifiant *Generique.java*), puis en utilisant la commande *time* :

```
time java Generique2 "requete"…        // mesure le temps de réponse de la requête
```

Veiller à omettre l'affichage du résultat dans le terminal pour éviter de mesurer le temps d'affichage au lieu du temps de lecture des données.

5.2) Ecrire R1 en SQL.

5.3) Pourquoi ne peut-on pas exécuter cette requête R1 avec une seule connexion au SGBD ?

5.4) On veut obtenir la cardinalité du résultat de R1 ? Donner en SQL une requête R2 telle :

R2 est une sous-expression de R1, et R2 peut être exécutée entièrement sur le premier SGBD

la cardinalité de R2 est égale à celle de R1, *i.e.,* card(R2) = card(R1)

Exécuter R2 et donner la valeur de card(R1).

5.5) Compléter le programme Sponsor.java pour traiter R1. Pendant tout le traitement de la requête R1, combien d'instances de type Connection et Statement sont créées ? Combien de sous-requêtes sont exécutées ? Combien de fois la relation Sponsor est-elle lue ? Quels sont les nuplets transférés depuis le SGBD vers l'application java ?

Exemple d'exécution :

```
java Sponsor                           //commande
CONNORS, Etats-Unis, Dunlop, Ecosse
CONNORS, Etats-Unis, Lacoste, France
EDBERG, Suede, Dunlop, Ecosse
…
SAMPRAS, Etats-Unis, Reebok, Angleterre
WILANDER, Suede, Dunlop, Ecosse
WILANDER, Suede, Kennex, USA
```

5.6) Mesurer le temps moyen d'exécution du programme avec la commande time :

```
time java Sponsor                              //commande
…
real 0m3.957s          user 0m1.530s           // affiche le temps de réponse
```

5.7) Ecrire le programme SponsorTF.java pour traiter la requête R2 : «Donner le nom et la nationalité des joueurs avec le nom et la nationalité de leurs sponsors, dans l'ordre des **noms de sponsor**» en utilisant l'algorithme de jointure par tri fusion. Combien d'instances de type Connection et Statement sont créées pendant le traitement de la requête ?

5.8) Comparer les temps de réponse de R1 et R2. Mesurer (en pourcentage) l'écart entre le temps de réponse de Sponsor et SponsorTF. Interpréter le résultat. Proposer une solution Sponsor2.java pour améliorer le temps de réponse de la requête R2.

5.9) Détailler une solution pour traiter R1 en transférant les clés (voir cours : semi-jointure)

5.10) (facultatif) Proposer une implémentation pour d'autres algorithmes de jointure (grace join, hybrid hash join).

Fiche Technique pour les TME

Installer l'environnement logiciel (Java, JDBC)

```
cd                                  // aller dans le répertoire $HOME
tar zxvf $BD_TOOL/jdbc-etu.tgz      // installer l'archive
cd jdbc-etu                         // aller dans le répertoire de travail
```

Dans la suite du TME, sauvegarder tous vos programmes dans le répertoire de travail jdbc-etu

1.3) Editeur de texte:

Editer les fichiers avec emacs. Activer les options suivantes :

Programme en couleur : Menu Option > Syntax Highlighting

Repérage de la structure du programme : Menu Option > Paren Match Highlighting

1.4) Environnement java

Pour tester la compilation du code source en bytecode, compiler le programme de test Bonjour.java (cela crée le fichier Bonjour.class)

```
javac Bonjour.java
```

Pour tester la machine virtuelle lancer l'exécution du fichier Bonjour.class :

```
java Bonjour
```

Fichier Joueur.java

```
1  import java.sql.*;
2  import java.io.*;
3
4    public class Joueur {
5      String server = "frelon";
6      String port = "1521";
7      String database = "oracle";
8      String user = "p6lip000";
9      String password = "p6lip000";
10     String requete = "select nom, prenom from Joueur";
11     Connection connexion = null;
12     …
13     /** La méthode traiteRequete */
14     public void traiteRequete() {
15         try {
16            String url = "jdbc:oracle:thin:@" +server + ":" +port + ":"
17 +database;
18            connexion = DriverManager.getConnection(url, user, password);
19            Statement lecture =  connexion.createStatement();
20            ResultSet resultat = lecture.executeQuery(requete);
21            while (resultat.next()) {
22              String tuple = resultat.getString(1) + " " +
23 resultat.getString(2);
24              out.println(tuple);
25            }
26            resultat.close();
27            lecture.close();
28            connexion.close();
29         }
           catch(Exception e){ … }
       } }
```