TME - Architecture Web

JDBC - JSP

Introduction:

L'objectif de ce TD/TME est de réaliser une application de type web qui accède à une base de données. Cette application repose sur l'architecture client-serveur à 3 couches (*cf.* cours sur l'architecture C/S 3-tiers) : le serveur de données, le serveur web et le navigateur client. Nous construisons l'application en trois étapes:

Etape1 : passerelle d'accès au serveur de données avec JDBC,

Etape 2 : maquette de l'interface utilisateur en HTML

Etape 3 : application web avec JSP. (hors programme BDR)

Les étapes 2 et 3 ne sont pas au programme BDR.

Le serveur de données (Oracle sur la machine frelon), 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.

Etape 1 : Accès à une base de données avec JDBC

Le package 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 bibliothèque JDBC).

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, 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 préparer 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 a 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                  // nom de 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 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 Generique 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 peut être exécutée par un 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 tuples 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, ...).

Etapes 2 et 3 : Interface utilisateur en HTML et application JSP

Sauvegarder tous vos fichiers HTML et JSP dans le répertoire ~/tomcat-etudiant/webapps/tp

1 Présentation de JSP

L'échange d'information entre l'utilisateur et l'application s'effectue au moyen de formulaires HTML et de balises JSP (*cf.* cours).

Le formulaire est interprété par le navigateur pour présenter une interface de saisie à l'utilisateur.

Lorsque l'utilisateur valide sa saisie, les données saisies sont passées en paramètre à l'application.

L'application produit un résultat au format HTML en fonction des données saisies par l'utilisateur. L'application délègue au SGBD une partie des traitements de données.

Pour les 3 applications suivantes (voir le code source en annexe) : décrire le rôle des diverses balises HTML et

JSP :

- Hello.jsp : balises form, input, <% %> et <%= %>.
- Joueur.jsp : balises dbOpen, dbQuery, dbClose.
- Iteration.jsp : balises dbNextRow, dbCloseQuery.

2 Application générique

) Créer le formulaire HTML *inforelation.html* pour demander à l'utilisateur de saisir un nom de relation et un nombre entier N. Créer l'application inforelation.jsp qui affiche :

- le schéma de la relation en précisant quels attributs composent la clé primaire.

- et les N premiers tuples de la relation, triés par valeur de croissante de clé primaire.

) Ecrire le formulaire *tournoi.html* et l'application *tournoi.jsp* pour demander à l'utilisateur de choisir un nom de joueur dans une liste déroulante, puis afficher tous les tournois auxquels le joueur a participé.

3 Présentation hiérarchisée de données relationnelles

) Ecrire une application *gagnant.jsp* qui présente (avec des listes imbriquées) pour chaque année puis pour chaque tournoi la liste des matchs avec les noms du gagnant puis du perdant triés par ordre alphabétique du gagnant.

) Même question mais les matchs sont triés par ordre d'importance (*ie.* finale, demi finales, …).

4 Mise à jour de données

(*) Créer une application *maj.jsp* pour que l'utilisateur puisse choisir une relation qu'il a précédemment créée, pour y ajouter, supprimer ou modifier des tuples.

5 Mise à jour de données

(*) Ecrire une application plan.jsp pour visualiser graphiquement un plan d'exécution. Utiliser des listes imbriquées pour représenter le structure en arbre du plan. Pour chaque opérateur, afficher le nom de l'algorithme choisi et l'estimation de la cardinalité du résultat et du coût.

Remarque : Les étapes marquées d'une (*) sont facultatives.

Fiche Technique pour les TME

Installer l'environnement logiciel (Java, JDBC)

1.1)Répertoire de travail.

Pour faciliter l'utilisation des divers outils, le nom des répertoires et de vos fichiers de travail doivent être identiques pour tous les binômes. Pour créer l'arborescence de travail, sauvegarder l'archive *jdbc.tar* dans votre **répertoire racine** et extraire son contenu.

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

Dans la suite du TME, sauvegarder tous vos programmes dans le répertoire ~/jdbc.

1.2)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

1.3)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
```

Uniquement pour les étapes 2 et 3 du TME : Installer le serveur web

Répertoire de travail. Pour créer l'arborescence de travail, sauvegarder l'archive *tomcat-etudiant.tar* dans votre **répertoire racine** et extraire son contenu.

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

- Dans une fenêtre de terminal, démarrer le serveur web avec la commande :

```
web-start
```

- Dans une **autre** fenêtre de terminal, stopper le serveur web avec la commande :

```
web-stop
```

- Lancer le client web (netscape) et ouvrir une connexion vers le serveur web :

```
netscape http://localhost:60001/tp/
```

Attention, en fin de séance, **stopper le serveur web** avant de quitter le poste de travail avec la commande web-

stop.

Remarque concernant le développement des applications JSP : si le client web (netscape) est sur une machine différente du serveur web, s'assurer (avec la commande **date**) que les horloges du client et du serveur sont à la même heure. Si l'horloge du serveur est 'trop' en avance par rapport à celle du client, tomcat ne recompile pas les fichiers JSP que vous avez modifiés, car la date de la demande d'exécution (horloge du client) reste inférieure à la date de modification du fichier JSP (horloge du serveur).

TSVP

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";
1      String requete = "select nom, prenom from Joueur";
0      Connection connexion = null;
1      &
1      /** La méthode traiteRequete */
1      public void traiteRequete() {
2          try {
1             String url = "jdbc:oracle:thin:@" +server + ":" +port + ":"
3  +database;
1             connexion = DriverManager.getConnection(url, user, password);
4             Statement lecture =  connexion.createStatement();
1             ResultSet resultat = lecture.executeQuery(requete);
5             while (resultat.next()) {
1               String tuple = resultat.getString(1) + " " + resultat.getString(2);
6               out.println(tuple);
1             }
7             resultat.close();
1             lecture.close();
8             connexion.close();
1          }
9          catch(Exception e){ & }
2      }
0  }
2
1
2
2
2
3
2
4
2
5
2
6
2
7
2
8
2
9
3
0
```

Fichier Hello.jsp

```
1  <%@page session="false" %>
```

```
<jsp:useBean id="name" class="oracle.jsp.jml.JmlString" scope="request" >
        <jsp:setProperty name="name" property="value" param="newName" />
</jsp:useBean>

<HTML>
  <HEAD><TITLE>Hello User</TITLE></HEAD>
  <BODY>
    <% if (!name.isEmpty()) { %>
      <H3>Hello <%= name.getValue() %></H3>
    <% } %>
    <P>
    Quel est votre prénom:
    <FORM METHOD=get>
      <INPUT TYPE=TEXT name=newName size = 20><br>
      <INPUT TYPE=SUBMIT VALUE="Envoyer">
    </FORM>
  </BODY>
</HTML>
```

Fichier Iteration.jsp

```
<%@ taglib uri="/WEB-INF/sqltaglib.tld" prefix="jml" %>
   <HTML>
       <HEAD>
           <TITLE>Result Set Iteration Sample </TITLE>
       </HEAD>
       <BODY BGCOLOR="#FFFFFF">
            <jml:dbOpen URL="jdbc:oracle:thin:@frelon:1521:oracle"
                        user="p6lip548" password="p6lip548" connId="c1">
            </jml:dbOpen>
            <jml:dbQuery connId="c1" output="jdbc" queryId="myquery">
               select nom from Joueur
            </jml:dbQuery>
            <jml:dbNextRow queryId="myquery">
                <%= myquery.getString(1) %> <br>
            </jml:dbNextRow>
            <jml:dbCloseQuery queryId="myquery" />
            <jml:dbClose connId="con1" />
       </BODY>
  </HTML>
```

Fichier Joueur.jsp

```
  <%@ taglib uri="/WEB-INF/sqltaglib.tld" prefix="jml" %>
   <HTML>
       <HEAD>
```

```
 4            <TITLE>Requete simple: Tableau des Joueurs </TITLE>
 5        </HEAD>
 6        <BODY BGCOLOR="#FFFFFF">
 7             <jml:dbOpen URL="jdbc:oracle:thin:@frelon:1521:oracle"
 8              user="p6lip548" password="p6lip548" connId="c1">
 9             </jml:dbOpen>
10             <jml:dbQuery connId="c1">
11                select * from Joueur
12             </jml:dbQuery>
13             <jml:dbClose connId="c1" />
14        </BODY>
15   </HTML>
```