JAVA ET LES BASES DE DONNÉES

Notions : JDBC, pont JDBC-ODBC, Curseurs, Métadonnées, Persistance

INTRODUCTION

Java, comme tout bon langage de programmation, permet l'accès aux bases de données pour stocker les informations. Pour ce faire, *Sun*, l'éditeur historique du langage, a créé la fonctionnalité ***JDBC*** (*Java DataBase Connectivity*), qui permet d'établir une interface standardisée entre n'importe quel SGBD et le langage.

Pour rester compatible avec les pilotes *ODBC* du monde Windows, java implémente un pont JDBC-ODBC.

Par ailleurs, une autre technique consiste à enregistrer sur un support physique les objets créés en mémoire (fichier). On parle de ***persistance*** en langage objet, et de **sérialisation** en Java. Cette technique est aussi utilisée lorsque l'on souhaite faire transiter des objets à travers le réseau.

I CONFIGURATION DE JAVA

1 - PONT JDBC-ODBC

La première étape est de demander à Java d'établir le lien avec l'outil *ODBC*. La manière la plus simple de le réaliser est d'écrire la ligne suivante, où les termes en gras sont les mots clés de java (attention aux majuscules/minuscules) :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  //installe le pont JDBC-ODBC
```

Cette ligne est à ajouter dans chaque classe où une connexion aux bases de données est requise.

Remarque : il est possible de réaliser des connexions directes en *JDBC*, sans passer par *ODBC*. Il faudra alors trouver les pilotes *JDBC* appropriés à la base de données à laquelle on se connecte.

2 - ÉTABLISSEMENT DE CONNEXION

Il faut ensuite faire le lien avec le pilote *ODBC* particulier créé précédemment. Il faudra avoir importé les outils SQL de java :

```
import java.sql.*     //à placer en tête du fichier .java (par exemple une applet)
…
Public class ___{…
…
//code à positionner dans une méthode
```

```
Class.forName( "sun.jdbc.odbc.JdbcOdbcDriver");
Connection cnx = DriverManager.getConnection ("jdbc:odbc:GQCM","","");
//crée une connection avec un pilote  ODBC

cnx.close();            //ferme la connexion au pilote en fin d'application
```

3 LES BASES ET LA GESTION D'ERREUR

Toute interaction avec une base de données est susceptible de générer des erreurs : Pilote erroné, table vide, requête mal formée, dépassement de la fin d'un curseur, etc.

Java impose donc d'encadrer toute action (ou toute suite d'actions) sur une table par un ***try {}*** et d'intercepter les erreurs (*Exception*) dans un ***catch{}***. Le code doit donc être écrit de la sorte :

```
//code à positionner dans une méthode
try {
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver")
Connection cnx = DriverManager.getConnection("jdbc:odbc:GQCM","","");
//crée une connection avec un pilote  ODBC
}
catch (Exception e) {e.printStackTrace();}
…
try {cnx.close();            //ferme la connection au pilote}
catch (Exception e) {e.printStackTrace();}
```

II REQUÊTES

Pour exécuter ensuite des requêtes de création, de modification ou de sélection, il faudra manipuler les objets *Statement* (qui sont les **assertions** SQL) et *ResultSet* (*RecordSet* en *VB*).

1 L'OBJET STATEMENT

Il s'agit de l'objet contenant les requêtes SQL à faire exécuter sur la base de données. Il peut s'agir de requêtes de sélection, qui retourneront alors un objet *ResultSet*, ou d'action (*INSERT, UPDATE* et *DELETE*) qui retourneront un booléen.

Exemple d'utilisation

```
    //Déclare un objet Statement appelé stmt
Statement stmt  ;
        //crée à partir de la connexion un nouvel objet Statement
stmt = cnx.createStatement( );
        //exécute la requête et stocke le résultat dans une variable i
int i = stmt.executeUpdate("Insert into Propositions values (5,'Autre
choix',False, False);")  ;
        //effectue une validation des commandes opérées
i = stmt.executeUpdate("commit;") ;
```

```
        //exécute la requête sélection et stocke le résultat dans une
variable Resultset
ResultSet rs = stmt.executeQuery("Select * from propositions");
```

2 L'OBJET RESULTSET

C'est un curseur permettant de parcourir les résultats obtenus lors d'une interrogation de base de données.

Exemple d'utilisation (penser à mettre les *try* et *catch* nécessaires)

```
Connection con = DriverManager.getConnection("jdbc:odbc:GQCM","","");
//crée une connection avec un pilote  ODBC
        //Déclare un objet Statement appelé stmt
Statement stmt  ;
        //crée à partir de la connexion un nouvel objet Statement
stmt = con.createStatement( );
        //exécute la requête et stocke le résultat dans une variable i
int i = stmt.executeUpdate("Insert into Propositions values (5,'Autre
choix',False, False);")  ;
        //effectue une validation des commandes opérées
i = stmt.executeUpdate("commit;") ;
        //exécute la requête sélection et stocke le résultat dans une
variable Resultset
ResultSet rs = stmt.executeQuery("Select * from propositions");

        //positionne le ResultSet sur la première ligne
rs.next() ;
        //récupère le champ numéro 0 (Identifiant de type numérique)
int leNum = rs.getInt(0) ;
        //récupère le champ Libelle
String leLibel = rs.getString("propLibelle") ;

//-----------------Exemple d'utilisation pour une interface----------------
-----
        //Manipule un objet JcheckBox (case à cocher) de nom ivjchkRep
ivjchkRep.setText(leLibel);         //affecte le libellé avec la données de
la base
ivjchkRep.setSelected(monRecordSet.getBoolean("propReponse"));    //idem
pour la propriété Reponse
//------------
        //passe à l'enregistrement suivant et manipule un autre bouton
rs.next();
String s = rs.getString("propLibelle");
ivjchkAttendu.setText(s);
ivjchkAttendu.setSelected(monRecordSet.getBoolean("propValide"));
```

PARCOURS D'UN RESULTSET

Lorsque l'on souhaite traiter l'intégralité des données d'un resultset, on utilisera la boucle suivante :

```
ResultSet rs = stmt.executeQuery("Select * from propositions");
While rs.next() {
……
String libellé = rs.getString("propLibelle")       //donne le contenu du champ String propLibelle
….
}
```

PRINCIPALES MÉTHODES ASSOCIÉES À L'OBJET RESULTSET

Descriptif

type_retourné	nom_méthode(paramètres)	Explications
Accès aux champs de l'enregistrement		
Récupère la valeur d'une colonne dans l'enregistrement courant au format :		
boolean	getBoolean(int NuméroColonne)	booléen
boolean	getBoolean(String NomColonne)	booléen
double	getDouble(int NuméroColonne)	double
double	getDouble(String NomColonne)	double
float	getFloat(int NuméroColonne)	float
float	getFloat(String NomColonne)	float
int	getInt(int NuméroColonne)	int
int	getInt(String NomColonne)	int
long	getLong(int NuméroColonne)	long
long	getLong(String NomColonne)	long
Object	getObject(int NuméroColonne)	Object

Descriptif

type_retourné	nom_méthode(paramètres)	Explications
Object	getObject(String NomColonne)	Object
short	vgetShort(int NuméroColonne)	short
short	vgetShort(String NomColonne)	short
String	getString(int NuméroColonne)	String
String	getString(String NomColonne)	String
Accès aux champs de l'enregistrement		
Récupère la valeur d'une colonne dans l'enregistrement courant en tant qu'objet		
Date	getDate(int NuméroColonne)	java.sql.Date
Date	getDate(int NuméroColonne, Calendar cal)	java.sql.Date
Date	getDate(String NomColonne)	java.sql.Date
Date	getDate(String NomColonne, Calendar cal)	java.sql.Date
Time	getTime(int NuméroColonne)	java.sql.Time

type_retourné	nom_méthode(paramètres)	Explications
Time	getTime(int NuméroColonne, Calendar cal)	java.sql.Time
Time	getTime(String NomColonne)	java.sql.Time
Time	getTime(String NomColonne, Calendar cal)	java.sql.Time
Déplacements et positionnements		
boolean	isAfterLast()	
boolean	isBeforeFirst()	
boolean	isFirst()	
boolean	isLast()	
boolean	last()	
Boolean	next()	
boolean	previous()	
int	getRow()	Indique le numéro d'enregistrement courant
boolean	relative(int rows)	Déplace le curseur d'un certain nombre de lignes en avant (rows > 0) ou en arrière (rows <0)
void	refreshRow()	Rafraîchit l'enregistrement courant
void	updateRow()	JDBC 2.0 met à jour la base de données avec le nouveau contenu de l'enregistrement courant
Tests et Requête		
Statement	getStatement()	JDBC 2.0 retourne la requête qui a produit cet objet Resultset
boolean	rowDeleted()	JDBC 2.0 Indique si un enregistrement a été effacé
boolean	rowInserted()	JDBC 2.0 Indique si l'enregistrement courant a été inseré
boolean	rowUpdated()	JDBC 2.0 Indique si l'enregistrement courant a été mis à jour.

3 L'OBJET RESULTSETMETADATA

Il s'agit de pouvoir récupérer des informations sur la structure de la table associée à un resultSet : nom et type des champs, nombre de champs par exemple.

Exemple d'utilisation :

```
Statement stmt = con.createStatement();
ResultSet rs=stmt.executeQuery("select * from proposition");
   ResultSetMetaData metaRS = rs.getMetaData();
   int nbCols = metaRS.getColumnCount();
String nomChamps[ ] = new String [nbCols] ;
//-----------------------------------------------------------------
…..
for (int i=1; i<=nbCols;i++) {
     nomChamps[i]=rsmd.getColumnName(i)
      }
```

Principales méthodes associées à l'objet ResultSetMetaData

Descriptif

type_retourné	nom_méthode(paramètres)	Explications
String	getColumnClassName(int column)	Retourne le nom de la classe dont la colonne est une instance
int	getColumnCount()	Retourne le nombre de colonne dans le resultset
int	getColumnDisplaySize(int column)	Donne la taille maximum de la colonne en nombre de caractère
String	getColumnLabel(int column)	Donne le nom d'affichage de la colonne
String	getColumnName(int column)	Retourne le nom de la colonne désignée
int	getColumnType(int column)	Donne le type SQL de la colonne (CHAR, INTEGER…)
String	getColumnTypeName(int column)	Donne le type de la colonne dans l'application SGBD
int	getPrecision(int column)	Donne le nombre de chiffres de la colonne
int	getScale(int column)	Donne le nombre de chiffre après la virgule de la colonne
String	getTableName(int column)	
Boolean	isAutoIncrement(int column)	Indique si la colonne est une numérotation automatique (auquel cas elle sera en lecture seule)
Boolean	isCaseSensitive(int column)	Indique si la colonne est sensible à la casse
Boolean	isCurrency(int column)	Indique si la colonne est de type monétaire
Boolean	isDefinitelyWritable(int column)	Indique si l'écriture est possible et assurée sur cette colonne
int	isNullable(int column)	Indique si la colonne accepte les valeurs nulles
Boolean	isReadOnly(int column)	Indique si la colonne est en lecture seule
Boolean	isSearchable(int column)	Indique si la colonne peut être utilisée dans une clause Where
Boolean	isSigned(int column)	Indique si la colonne est en numérique signée (positif et négatif)
Boolean	isWritable(int column)	Indique s'il est possible d'écrire dans la colonne

III PERSISTANCE

En natif dans la technologie, Sun propose la possibilité de faire durer les objets au-delà de la mémoire de la machine qui l'exécute. Une implémentation de la classe Serializable permettra d'enregistrer les instances d'objet dans un flux (fichier ou réseau).

Soit la classe Enchere décrite comme suit :

```
public class Enchere implements Serializable {…….}
```

Soit une classe ayant pour propriété :

```
Enchere tabEncheres ;
```

L'enregistrement de l'ensemble des données du tableau tabEncheres peut être réalisé par la fonction :

```
public void externalise()
{//try et catch pour prendre en compte la gestion d'erreurs
try {ObjectOutputStream os ;                        //pour effectuer la sortie
os = new ObjectOutputStream (new FileOutputStream("c:\\essai.sortie"));
//génère un fichier, attention au double \
for (i=1;i<=max; i++) {
     os.writeObject(tabE[]); }
os.close();
}
     catch (Exception e) {e.printStackTrace();}
```

Le code pour permettre la récupération aura la forme :

```
try{
ObjectInputStream is = new ObjectInputStream (new
FileInputStream("c:\\essai.sortie")); //déclare et ouvre le fichier en
lecture
for (i=1;i<=max; i++) {
  tabEncheres[i]= (Enchere) is.readObject();    //on retransforme (parse)
l'information lue au format de l'objet instancié
}}     catch (Exception e) {e.printStackTrace();}
```