Exemple

Exécution d'une instruction simple de type UPDATE

Ajouter une catégorie « céréales » de code 3 dans la table catégories

```
Statement st = c.createStatement();
int nb = st.executeUpdate(''insert into categorie values (3, 'cereales')'');
System.out.println( '' nombre de lignes modifiées : ''+nb);
st.close();
```

Exemple

Instruction paramétrée de type SELECT: Retourne tous les produits d'une catégorie

```
String libelle;
int code;
req = c.prepareStatement("select codeprod, nomprod " +
"from categorie c, produit p where c.codecat=p.codecat and libellecat = ?");
req.setString(1, "Fruits");
ResultSet res = req.executeQuery();
while(res.next()) {
code = res.getInt(1);
libelle = res.getString(2);
System.out.println(" produit : "+code +","+ libelle); }
req.close();
```

Exemple

Instruction paramétrée de type UPDATE

Ajout de 2 nouveaux produits dans la table produit

```
req = connec.prepareStatement("insert into produit values (?,?,?)");
req.setInt(1, 12);
req.setString(2, "Fraise");
req.setInt(3, 1);
nb = req.executeUpdate();
req.setInt(1, 13);
req.setString(2, "Carotte");
req.setInt(3, 2);
nb = req.executeUpdate();
req.close();
```

Depuis JDBC version 2.1

Pour créer différents Statement qui donneront des ResultSet ayant des propriétés différentes :

Statement createStatement ( ) throws SQLException;

Statement createStatement (int rsType, int rsConcurrency) throws SQLException;

permet de définir:

- le type du ResultSet (rsType), sa « navigabilité »
- le fait qu'il permette ou non des mises à jour (rsConcurrency)

Différents types de ResultSet

Navigabilité :

TYPE_FORWARD_ONLY (défaut) : navigation « en avant » uniquement;

TYPE _SCROLL_ INSENSITIVE :

dans tous les sens, et où on veut, mais pas d'accès aux modifications sur la source de données depuis l'ouverture du ResultSet;

TYPE_ SCROLL_ SENSITIVE :navigation + accès aux modifications;

Mise à jour :

CONCUR_READ_ONLY (défaut) : pas de modification

CONCUR_UPDATABLE : modifications possibles

Mouvements dans un ResultSet

Les méthodes

Lors de la création : pointe « avant » la première ligne.

Si de type TYPE FORWARD ONLY, que :

next() :passe à la ligne suivante. retourne true si elle existe, false sinon (après la dernière ligne).

sinon :

previous() : ligne précédente

first() : sur la première ligne, retourne true si elle existe, false sinon (resultSet vide)

last() : sur la dernière;

beforeFirst() : avant la première (comme à l'ouverture)

afterLast () : après la dernière

Mouvements dans un ResultSet

relative(int rows) : rows lignes après la position courante, revient en arrière si rows est négatif;

absolute(int row) : se place à la row-ième ligne. Si row est égal à 1 : sur la première, si row est négatif, sur la dernière. (si 0, sur la première aussi)

Exemple

```
Statement stm =
c.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE);
ResultSet rs=stm.executeQuery("SELECT * FROM produit");
rs.absolute(3); // positionne sur la 3ème ligne;
rs.relative(5); // positionne sur la 8ème ligne
```

La classe ResultSet

Si ouvert en mode CONCUR_UPDATABLE, permet la mise à jour de la base par le biais du ResultSet (sinon on le fera par execute ou executeUpdate)

Mise à jour d'une ligne

En 2 étapes :

mise à jour de la nouvelle valeur de la colonne : **updateXXX**

changements affectés à la ligne concernée (alors seulement la base sera mise à jour) : **updateRow()**

Pour réaliser une mise à jour dans la ligne courante désignée par le curseur, il faut :

1-utiliser une des méthodes updateXXX() sur chacun des champs à modifier,

2- Une fois toutes les modifications faites dans une ligne, il faut appeler la méthode updateRow() pour reporter ces modifications dans la base de données car les méthodes updateXXX() ne font des mises à jour que dans le jeu de résultats.

- Les mises à jour sont perdues si un changement de ligne intervient avant l'appel à la méthode updateRow().

Exemple

```
Statement st = c.createStatement(ResultSet.TYPE_FORWARD_ONLY,

ResultSet.CONCUR_UPDATABLE);

ResultSet res = st.executeQuery("select codecat, libellecat from categorie where
 libellecat='Fruits' ");

    res.next(); // positionne sur la 1ère ligne;

    res.updateString(2, "Frutti");

// ou  res.updateString("libellecat", " Frutti" );

//modifie l'attribut libellecat

    res.updateRow(); // effectue la modification de la ligne
```

La classe ResultSet

C'est une classe qui représente une abstraction d'une table qui se compose de plusieurs enregistrements constitués de colonnes qui contiennent les données. Les principales méthodes pour obtenir des données sont :

Méthode	Rôle
getInt(int)	retourne sous forme d'entier le contenu de la colonne dont le numéro est passé en paramètre.
getInt(String)	retourne sous forme d'entier le contenu de la colonne dont le nom est passé en paramètre.
getFloat(int)	retourne sous forme d'un nombre flottant le contenu de la colonne dont le numéro est passé en paramètre.
getFloat(String)	retourne sous forme d'un nombre flottant le contenu de la colonne dont le nom est passé en paramètre.
getDate(int)	retourne sous forme de date le contenu de la colonne dont le numéro est passé en paramètre.
getDate(String)	retourne sous forme de date le contenu de la colonne dont le nom est passé en paramètre.

La classe ResultSet

Méthode	Rôle
next()	se déplace sur le prochain enregistrement : retourne false si la fin est atteinte
close()	ferme le ResultSet
getMetaData()	retourne un objet de type ResultSetMetaData associé au ResultSet.

- Il existe deux formes de ces méthodes : indiquer le numéro de la colonne en paramètre (en commençant par 1) ou indiquer le nom de la colonne en paramètre. La première méthode est plus efficace mais peut générer plus d'erreurs à l'exécution notamment si la structure de la table évolue.
- Attention : il est important de noter que ce numéro de colonne fourni en paramètre fait référence au numéro de colonne de l'objet resultSet (celui correspondant dans l'ordre SELECT) et non au numéro de colonne de la table.
- La méthode getString() permet d'obtenir la valeur d'un champ de n'importe quel type.

La classe ResultSet

Durant le parcours d'un ResultSet, il est possible d'effectuer des mises à jour sur la ligne courante du curseur. Il faut déclarer l'objet ResultSet comme acceptant les mises à jour. Avec les versions précédentes de JDBC, il fallait utiliser la méthode executeUpdate() avec une requête SQL. Maintenant pour réaliser ces mises à jour, JDBC 2.0 propose de les réaliser via des appels de méthodes plutôt que d'utiliser des requêtes SQL.

Méthode	Rôle
updateXXX(String, XXX)	permet de mettre à jour la colonne dont le nom est fourni en paramètre. Le type Java de cette colonne est XXX
updateXXX(int, XXX)	permet de mettre à jour la colonne dont l'index est fourni en paramètre. Le type Java de cette colonne est XXX
updateRow()	permet d'actualiser les modifications réalisées avec des appels à updateXXX()
boolean rowsUpdated()	indique si la ligne courante a été modifiée
deleteRow()	supprime la ligne courante
rowDeleted()	indique si la ligne courante est supprimée
moveToInsertRow()	permet de créer une nouvelle ligne dans l'ensemble de résultat
insertRow()	permet de valider la création de la ligne

La classe ResultSet

La méthode cancelRowUpdates() permet d'annuler toutes les modifications faites dans la ligne. L'appel à cette méthode doit être effectué avant l'appel à la méthode updateRow().

Pour insérer une nouvelle ligne dans le jeu de résultat, il faut tout d'abord appeler la méthode moveToInsertRow(). Cette méthode déplace le curseur vers un buffer dédié à la création d'une nouvelle ligne. Il faut alimenter chacun des champs nécessaires dans cette nouvelle ligne. Pour valider la création de cette nouvelle ligne, il faut appeler la méthode insertRow().

Exp :

```
Statement st =
 conn.createStatement(ResultSet.TYPE_FORWARD_ONLY,
 ResultSet.CONCUR_UPDATABLE);
ResultSet res = st.executeQuery("select codecat, libellecat
 from categorie");

// Move the cursor to the insert row
res.moveToInsertRow();
res.updateString("codecat", "5") ;
res.updateString("libellecat", "cereale");
// Store the insert into database
res.insertRow();
```

Pour supprimer la ligne courante, il faut appeler la méthode **deleteRow()**. Cette méthode agit sur le jeu de résultats et sur la base de données. A la création du ResultSet, le curseur est positionné avant la première occurrence à traiter. Pour se déplacer dans l'ensemble des occurrences, il y a toujours la méthode next() pour se déplacer sur le suivant mais aussi plusieurs autres méthodes pour permettre le parcours des occurrences en fonctions du mode utilisé dont les principales sont :

Méthode	Rôle
boolean isBeforeFirst()	Renvoyer un booléen qui indique si la position courante du curseur se trouve avant la première ligne
boolean isAfterLast()	Renvoyer un booléen qui indique si la position courante du curseur se trouve après la dernière ligne
boolean isFirst()	Renvoyer un booléen qui indique si le curseur est positionné sur la première ligne
boolean isLast()	Renvoyer un booléen qui indique si le curseur est positionné sur la dernière ligne
boolean first()	Déplacer le curseur sur la première ligne
boolean last()	Déplacer le curseur sur la dernière ligne

Méthode	Rôle
boolean absolute(int)	Déplacer le curseur sur la ligne dont le numéro est fourni en paramètre à partir du début s'il est positif et à partir de la fin s'il est négatif. 1 déplace sur la première ligne, -1 sur la dernière, -2 sur l'avant dernière ...
boolean relative(int)	Déplacer le curseur du nombre de lignes fourni en paramètre par rapport à la position courante du curseur. Le paramètre doit être négatif pour se déplacer vers le début et positif pour se déplacer vers la fin. Avant l'appel de cette méthode, il faut obligatoirement que le curseur soit positionné sur une ligne.
boolean previous()	Déplacer le curseur sur la ligne précédente. Le booléen indique si la première occurrence est dépassée.
void afterLast()	Déplacer le curseur après la dernière ligne
void beforeFirst()	Déplacer le curseur avant la première ligne
int getRow()	Renvoyer le numéro de la ligne courante

```
public class ExpJdbc {
static Connection maConnection() throws SQLException {
    final String url = "jdbc:mysql://localhost/db_jeu";
    Connection conn = DriverManager.getConnection(url, "root", "");
    return conn;}
public static void afficherInfoCurseur(ResultSet resultat)
     throws SQLException{
String curseur = "row = "+resultat.getRow()+") ";
if (resultat.isBeforeFirst()){  curseur = "Debut";}
else if (resultat.isAfterLast()){  curseur = "Fin";}
else  curseur+=resultat.getString(1)+ " - " +resultat.getString(3) ;System.out.println(curseur);
}

public static void main(String[] args) {
Connection bd=null;
try {
bd = maConnection();
Statement st = bd.createStatement(ResultSet.TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE);
String sql = "SELECT  nom, prenom,sexe  FROM joueur WHERE nbressai > 5";
ResultSet res = st.executeQuery(sql);
System.out.println("Before first: row= "+ res.getRow());
while(res.next())
{System.out.println(res.getString(1)+" - " +res.getString(2) ) ; }
System.out.println("After last: row= "+ res.getRow());
afficherInfoCurseur(res);
res.absolute(2);
afficherInfoCurseur(res);
res.relative(-2);
afficherInfoCurseur(res);
} catch (SQLException e) {
if(bd==null)System.out.println("la connection a echoué");
else System.out.println("une SQLexception est levée");
}
finally
{try {if (bd != null ) {bd.close();  System.out.println("connection fermée");}}
catch (SQLException e)
{System.out.println("Impossible de fermer la connection.") ;}
}  }  }
```

pseudo	nom	prenom	datnais	sexe	nbressai
deda	denguir	delida	2018-09-06	Feminin	8
Yos	Hannachi	yosra	2018-08-30	Feminin	5
Dragon	Beji	Ali	2018-08-30	Masculin	2
Roi	Ben Saleh	mohamed	2018-08-30	Masculin	6
hihi	haha	hoho	2018-09-06	Feminin	6

Affichage

Before first: row= 0

denguir - delida

Ben Saleh - mohamed

haha – hoho

After last: row= 0

Fin

row = 2) Ben Saleh - Masculin

Debut

connection fermée