TP8 Java EE

Utilisation des Bases de données -- JDBC

Classe : 3 IM AU : 2021/2022

Objectifs

Se familiariser avec

1. La connexion à une base de données
2. L'utilisation de l'API JDBC
3. Introduire les concepts fondamentaux suivants : pilote, établissement d'une connexion, interrogation de la base par le biais de requêtes SQL, objet résultat de type ResultSet
4. La réalisation des requêtes et l'exploitation des résultats

Introduction

Supposons que nous disposons déjà d'une base de données intitulée "base_de_produits" contenant deux tables : "lesutilisateurs" et "produits" réalisés avec les requêtes SQL suivantes

```
CREATE DATABASE IF NOTEXISTS `base_de_produits` DEFAULTCHARACTERSET latin1 COLLATE latin1_swedish_ci; USE base_de_produits;

CREATETABLE IF NOTEXISTS `lesutilisateurs` (
  `lelogin` varchar(20) NOTNULL,
  `lepassword` varchar(10) NOTNULL,
PRIMARYKEY (`lelogin`)
) TYPE=MyISAM;

INSERTINTO `lesutilisateurs` (`lelogin`, `lepassword`) VALUES
('admin', 'admin'),
('user1', 'user1'),
('user2', 'user2'),
('administrateur', 'a');

CREATETABLE IF NOTEXISTS `produit` (
  `nom` varchar(20) NOTNULL,
  `category` varchar(20) NOTNULL,
  `quantite` int(11) NOTNULL,
  `prixunit` doubleNOTNULL,
PRIMARYKEY (`nom`)
) TYPE=MyISAM;

INSERTINTO `produit` (`nom`, `category`, `quantite`, `prixunit`) VALUES
('produit1', 'cat1', 52, 10.5),
('produit2', 'cat1', 10, 20),
('produit3', 'cat5', 3, 50),
('produit4', 'cat3', 1, 15) ;
```

Interrogation d'une base de donnée

Avant de pouvoir interroger la base de données, nous devons :

- choisir le bon pilote ;
- établir une "connexion" avec la base concernée.

Choix du pilote

Dans un premier temps, il va falloir que le programme puisse utiliser convenablement le pilote voulu. On pourrait penser qu'il suffit de l'appeler de façon usuelle, en ayant pris soin d'importer les bonnes classes. Mais Java utilise une démarche moins directe, à savoir qu'il existe un objet, dit "gestionnaire de pilotes", instance de la classe **DriverManager**, chargé de gérer les différents pilotes de bases de données existants. Pour rendre disponible le "bon pilote", on peut recourir à la méthode **forName** de la classe **Class** en lui fournissant la référence du pilote concerné. Cet appel provoque :

- le chargement en mémoire de la classe correspondante (elle implémente l'interface Driver) qui devient accessible à la machine virtuelle ;
- l'instanciation d'un objet de cette classe et appel de son constructeur qui enregistre la classe auprès du gestionnaire de pilotes, de sorte que le pilote concerné deviendra effectivement utilisable.

Dans notre cas (base créée avec MySQL), voici comment nous procéderons :

```
Class.forName("com.mysql.jdbc.Driver");
```

Le nom "com.mysql.jdbc.Driver" correspond à un pilote permettant d'accéder à une base de données MySQL sur laquelle s'exécutera le programme Java. S'il s'agissait d'une autre base utilisant un SGBDR différent, le pilote porterait un nom différent.

La méthode *forName* provoque une exception ClassNotFoundException si le pilote voulu n'a pu être obtenu. Ainsi nous devons entourer le code par un bloc try/catch

```
try{
	Class.forName("com.mysql.jdbc.Driver");
}catch (ClassNotFoundException e) {
	e.printStackTrace();
}
```

Établissement d'une connexion

Dans un deuxième temps, il va falloir établir ce que l'on nomme une "connexion" avec la base de données. Pour établir cette connexion, on utilisera la méthode statique **getConnection** de la classe **DriverManager**, en lui fournissant la référence de la base. Généralement cette référence contient l'identification du SGBBR concerné (pour nous jdbc:mysql), ainsi que l'identification de la base elle-même.

En définitive, l'instruction :

```
conn = DriverManager.getConnection("jdbc:mysql://localhost/base_de_produits");
```

nous fournit :

- soit un objet de type **Connection** contenant toutes les informations nécessaires à l'utilisation ultérieure de la base, si l'établissement de la connexion a pu s'opérer ;
- soit la valeur **null** si la connexion n'a pu être établie.

Pour se connecter en fournissant un login et mot de passe, on peut utiliser la méthode **getConnection** disposant de deux arguments supplémentaires permettant de préciser un nom d'utilisateur (login) et un

mot de passe (password):

```
conn=DriverManager.getConnection("jdbc:mysql://localhost/base_de_produits","iduser","passuser");
```

« **iduser** » et « **passuser** » forment respectivement le login et le mot de passe d'un utilisateur ayant le droit d'accéder à la base de donnée ***base_de_produits***

Interrogation de la base

Une fois la connexion établie, on peut dialoguer avec le SGBDR, par le biais de requêtes SQL, ce qui permet, notamment :

- d'accéder à certains champs (ou à leur ensemble) de tout ou partie des enregistrements d'une table ;
- de réaliser des mises à jour d'une table : insertion, suppression ou modification d'enregistrements.

Pour récupérer les différents champs de la table produit, nous utilisant la requête suivante :

```
"select * from base_de_produits.produit";
```

Pour transmettre une telle requête au SGDBR, il faut :

- créer un objet dont la classe implémente l'interface **Statement**, à l'aide de la méthode **createStatement** de la classe **Connection**, qu'on applique à l'objet représentant la connexion avec la base (ici **conn**) :

```
Statement statement = conn.createStatement();
```

- appliquer à cet objet **statement**, la méthode **executeQuery** à laquelle on fournit la requête SQL en argument :

```
statement.executeQuery("select * from base_de_produits.produit");
```

Celle-ci fournit alors en résultat un objet de type **ResultSet** contenant les informations sélectionnées.

En résumé, voici comment procéder pour interroger la base (ici, pour faciliter les choses, nous avons placé la requête SQL dans une chaîne) :

```
String requete = ""select * from base_de_produits.produit";
ResultSet res ;
Statement statement = connec.createStatement() ;
res statement.executeQuery(requete);
```

On notera bien que la requête SQL n'est interprétée qu'après l'appel de la méthode **executeQuery**. Ce n'est qu'à ce moment que le SGBD pourra détecter une éventuelle faute (syntaxe, nom de champ incorrect...), ce qui provoquera alors une exception de type **SQLException**.

Exploitation du résultat

L'objet, de type **ResultSet**, fourni par **executeQuery**, dispose de méthodes permettant de le parcourir enregistrement par enregistrement, à l'aide d'un "curseur". Plus précisément :

- la méthode **next** fait progresser le curseur d'un enregistrement au suivant, en fournissant la valeur **null** s'il n'y a plus d'enregistrement ;
- au départ le curseur est positionné avant le premier enregistrement ; il faut donc effectuer un appel à **next** pour qu'il soit bien positionné sur le premier enregistrement (s'il existe) de l'objet résultat.

Pour obtenir l'information d'un champ donné de l'enregistrement "courant" (désigné par le curseur), on dispose de méthodes dont le nom dépend du type de l'information concernée, par exemple **getString** pour

une chaîne de caractères, **getInt** pour une valeur de type entier. Ainsi, ceci nécessite d'établir une correspondance entre les types SQL et les types Java. Voici comment parcourir notre résultat et en afficher les valeurs :

```
while (resultSet.next()) {
	String nom = resultSet.getString("nom");
	String category = resultSet.getString("category");
	int quantite = resultSet.getInt("quantite");
	double prixunite = resultSet.getDouble("prixunit");
	System.out.println("nom="+nom+",categorie="+category+",quantite="+quantite+",prix
unitaire="+prixunite);
}
```

Lorsque l'on extrait une information de l'objet résultat par une méthode de la forme getXXX (XXX désigne un type Java), celle-ci convertit l'information SQL dans le type mentionné. Ainsi, pour utiliser convenablement la bonne méthode, il faut savoir quel est le type Java susceptible de représenter le mieux possible une valeur du type SQL concerné.

Type SQL	Description	Type Java conseillé	Méthode Java
BIT	1bit	boolean	getBoolean
TINYINT	entier 8 bits	byte	getByte
SMALLINT	entier 16 bits	short	getShort
INTEGER	entier 32 bits	int	getInt
BIGINT	entier 64 bits	long	getLong
REAL	flottant 32 bits	float	getFloat
DOUBLE	flottant 64 bits	double	getDouble
CHAR(n)	chaîne d'exactement *n* caractères	String	getString
VARCHAR(n)	chaîne d'au plus *n* caractères	String	getString
DATE	date	Java.sql.Date	getDate
TIME	heure	Java.sql.Time	getTime
TIMESTAMP	date et heure	Java.sql.Timestamp	getTimestamp
DECIMAL (c, d)	nombre décimal de c chiffres dont *d* décimales représenté de façon exacte	java.math.BigDecimal	getBigDecimal

Libération des ressources

En théorie, l'objet représentant la connexion sera automatiquement détruit par le ramasse- miettes dès qu'il ne sera plus utilisé. Cependant, cette destruction peut n'avoir lieu qu'après la fin du programme. Dans des applications réelles, où plusieurs utilisateurs peuvent accéder à une même base de données, il sera préférable de détruire cet objet dès que possible :

```
conn.close();
```

Notez que l'objet résultat (de type **ResultSet**) n'est plus accessible lorsque la connexion est fermée. Il faut donc éviter de fermer la connexion trop tôt.

Résumé

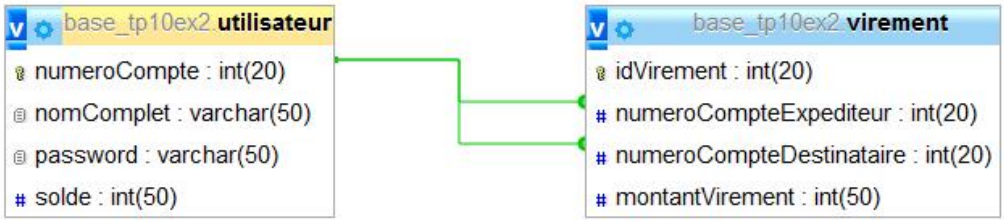
Voici un exemple récaptulatif des étapes précédentes :

```
// connexion à la base dedonnée
Connection conn = null;
try {
```

```
        Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException e) {
        e.printStackTrace();
}
try {
conn = DriverManager.getConnection("jdbc:mysql://localhost/base_de_produits", "root", "");
} catch (SQLException e) {
        e.printStackTrace();
}
System.out.println("!connection successful");
// requête
String req = "select * from base_de_produits.produit";
try {
Statement statement =conn.createStatement();//cetobjetpermetdetransmettrelarequête
ResultSet resultSet = statement.executeQuery(req);//obtentiondurésultatdansunobjet   // de
type ResultSet
while (resultSet.next()) { // (resultSet.next() renvoie null s'il n'y a plus
d'enregistrement
String nom = resultSet.getString("nom");
String category = resultSet.getString("category");
int quantite = resultSet.getInt("quantite");
double prixunite = resultSet.getDouble("prixunit");
System.out.println("nom="+nom+" ,categorie="+category+", quantite="+quantite+", prix
unitaire="+prixunite);
}
} catch (SQLException e) {
e.printStackTrace();
}
```

Exercice

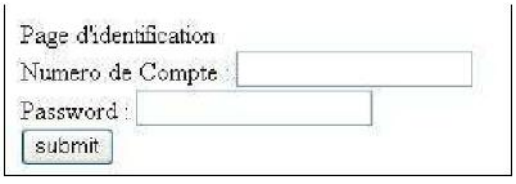
Le schéma de la base de données est le suivant:



Travail demandé: On désire développer une application Web pour la gestion d'un compte bancaire.

Partie 1 : Authentification

1. Créer la page "*identification.html*" permettant à l'utilisateur de s'authentifier en saisissant son login (numéro de compte) et son mot de passe.



2. Définir la page "*identification.html*" comme page d'accueil dans le descripteur de déploiement Web.xml.
3. Créer une servlet nommée *Identification* qui vérifie l'authenticité de l'utilisateur en vérifiant le couple (numeroCompte, password) dans la table *utilisateur* de la base :

- Si la combinaison numéro de Compte et mot de passe n'existe pas, l'utilisateur est redirigé vers la page "*identification.html*".
- Sinon (utilisateur authentifié)
  - Créer une nouvelle session.
  - Rajouter les attributs "*nomComplet*" et "*numeroCompte*" à la session.
  - Renvoyer vers **la servlet** "*ListeVirements*".

Partie 2 : Consultation du compte

1. Créer une servlet nommée *ListeVirements* qui :
   a. Récupère le numéro de compte de l'utilisateur actif de la session déjà créée dans la servletIdentification.
   b. Intérroge la base de données pour récupérer la liste des virements où l'utilisateur actif est l'expéditeur.
   c. Intérroge la table utilisateur pour récupérer le solde de l'utilisateur(utiliser un autre objet de type Statement).
   d. Renvoie les résultats des deux requêtes précédentes à la pagelisteVirements.jsp .
2. Créer la page *listeVirements.jsp* qui :
   a. affiche la liste des virements et le solde de l'utilisateur déjà envoyés par la servlet ListeVirements.
   b. affiche le nom de l'utilisateur récupéré de la session.
   c. c. contient un lien vers la page nouveauVirement.html.

Bonjour Mohamed Ben Ali
Vous avez 500 dinars

numéro bénéficiaire	montant
5678	200
5678	630
9111	630

Nouveau Virement

Partie 3 : Gestion des virements

1. Créer la page nouveauVirement.html contenant un formulaire où l'utilisateur peut saisir le numéro de compte du bénéficiaire du virement ainsi que le montant à virer (voir figure 4). Les données saisies seront envoyées à la servlet Virement.

Nouveau virement
Numéro bénéficiaire :
montant :
submit

2. Créer la servlet Virement qui :
   a. récupère le numéro de compte de l'utilisateur courant,
   b. récupère le montant du virement et le numéro du compte bénéficiaire du formulaire virement précédent.
   c. insère le nouveau virement dans la table virement (tous les attributs sauf idVirement qui est en auto incrémentation).
   d. met à jour le solde du compte expéditeur (celui de l'utilisateur connecté) en soustrayant le montant du virement.
   e. faire la redirection nécessaire permettant d'afficher la nouvelle liste de virements.

Annexe sur l'exécution des requêtes

Pour exécuter une requête SQL, l'interface ***Statement*** dispose de trois méthodes (elles reçoivent toutes les trois la requête en argument) :

- ***executeQuery*** s'applique à une requête de sélection et fournit en résultat un objet de type *ResultSet ;*
- ***executeUpdate*** s'applique à une requête de mise à jour d'une table ou de gestion de la base ; elle fournit en résultat (de type *int),* le nombre d'enregistrements modifiés dans le premier cas ou la valeur -1 dans le second (gestion) ;
- ***execute*** s'applique à n'importe quelle requête. Elle fournit en résultat un booléen valant *true* si la requête SQL fournit des résultats (sous forme d'un objet de type *ResultSet)* et *false* sinon. Il faut alors, suivant le cas, utiliser l'une des deux méthodes *getResultSet* pour obtenir l'objet résultat ou *getUpdateCount* pour obtenir le nombre d'enregistrements modifiés. Cette dernière méthode s'avère surtout utile lorsque l'on doit exécuter une requête de nature inconnue ; c'est ce qui peut se produire avec un programme qui exécute des requêtes SQL fournies en données, par exemple dans un fichier texte.