IUT DE CRETEIL-VITRY **Durée 3 heures**
DPT INFO

TP 2 : R3.07

Configuration de l'environnement Java (JDBC) - MySQL :

MySQL :

La Configuration de l'environnement permettra de garantir que votre application Java puisse se connecter et interagir avec la base de données MySQL en toute transparence. Voici les étapes à suivre :

1. Installation de Java : Assurez-vous que Java Development Kit (JDK) est installé sur votre système. Vous aurez besoin d'une version de JDK compatible avec votre version de JDBC.
2. Installation de MySQL : Si ce n'est pas déjà fait, installez et configurez MySQL Server sur votre système. Assurez-vous que le serveur MySQL est en cours d'exécution.
3. Téléchargement du pilote JDBC : Pour MySQL, vous devrez télécharger le pilote JDBC MySQL approprié depuis le site officiel de MySQL (généralement appelé MySQL Connector/J). Ce pilote est nécessaire pour établir une connexion entre votre application Java et la base de données MySQL.
4. Configuration de la base de données : Une fois connecté à MySQL, créez une nouvelle base de données, par exemple : *CREATE DATABASE TP2_JDBC;*
   Définissez ensuite cette base de donnée comme base de données courante avec la commande : *USE TP2_JDBC;*
5. Créer les tables nécessaires pour la suite de ce TP et y insérer les données (exécution des scripts SQL sur EPREL pour la création des tables et l'insertion des données.
6. Configuration de la connexion : Vous devrez connaître les détails de connexion à votre base de données, tels que l'URL de connexion (String url = "jdbc:mysql://localhost:3306/*TP2_JDBC*"), le nom d'utilisateur et le mot de passe. Ces informations seront utilisées pour établir une connexion JDBC.

JDBC :

Importation des bibliothèques JDBC dans le projet :

Pour utiliser JDBC dans votre projet Java, vous devez importer les bibliothèques JDBC nécessaires.

1. Téléchargement du pilote JDBC : Comme mentionné précédemment, téléchargez le pilote JDBC MySQL (https://dev.mysql.com/downloads/connector/j/).
2. Importation des bibliothèques : Ajoutez le fichier JAR du pilote JDBC téléchargé à votre projet Java. Cela peut généralement être fait en copiant le fichier JAR dans le répertoire du projet et en l'important dans votre environnement de développement (par exemple, dans Eclipse, vous pouvez le faire via les propriétés du projet et la section "Bibliothèques").

3. Utilisation des classes JDBC : Dans votre code Java, importez les classes JDBC nécessaires en utilisant des déclarations d'importation. Par exemple :
   import java.sql.Connection;
   import java.sql.DriverManager;
   import java.sql.SQLException;

L'importation des bibliothèques JDBC permet à votre application Java d'accéder aux fonctionnalités de JDBC, y compris la création de connexions à la base de données, l'exécution de requêtes SQL, et la récupération des résultats.

Préliminaires JDBC :

JDBC (Java Database Connectivity) est une API Java qui permet aux développeurs d'interagir avec des bases de données relationnelles à partir de programmes Java. Il joue un rôle essentiel en facilitant la connexion, l'envoi de requêtes SQL, la récupération de résultats et la gestion des données entre une application Java et une base de données.
L'API JDBC est composée de plusieurs parties principales :

Driver JDBC : Les drivers JDBC sont des composants spécifiques au fournisseur de base de données qui permettent à JDBC de communiquer avec une base de données particulière.

Interface DriverManager : Cette interface gère une liste de pilotes de base de données. Elle permet d'établir des connexions avec la base de données en sélectionnant le driver approprié en fonction de l'URL de connexion.

Interface Connection : Une fois une connexion établie, l'interface Connection représente cette connexion à la base de données. Elle permet d'exécuter des requêtes SQL, de gérer les transactions et de récupérer les objets Statement et PreparedStatement.

Interface Statement : L'interface Statement est utilisée pour exécuter des requêtes SQL simples. Cependant, elle est sujette aux injections SQL et n'est pas recommandée pour les requêtes dynamiques. Un objet "Statement" est utilisé pour exécuter des requêtes SQL simples et statiques sans paramètres.
Exemple:

```
Statement statement = connexion.createStatement();
String selectQuery = "SELECT nom, prenom FROM Etudiants WHERE age > 20";
ResultSet resultSet = statement.executeQuery(selectQuery);
```

Interface PreparedStatement : L'interface PreparedStatement est utilisée pour exécuter des requêtes SQL paramétrées, ce qui les rend plus sûres et plus efficaces. Un objet "PreparedStatement" est utilisé pour exécuter des requêtes SQL préparées avec des paramètres.
Exemple :

```
String updateQuery = "UPDATE Etudiants SET age = ? WHERE nom = ?";
PreparedStatement preparedStatement = connexion.prepareStatement(updateQuery);
preparedStatement.setInt(1, 26); // Nouvel âge
preparedStatement.setString(2, "Smith"); // Nom de l'étudiant à mettre à jour
int rowsUpdated = preparedStatement.executeUpdate();
```

Interface ResultSet : Lorsqu'une requête SQL est exécutée, les résultats sont stockés dans un objet ResultSet. L'interface ResultSet permet de parcourir et de récupérer les données de ces résultats. Un objet ResultSet est utilisé pour récupérer les résultats d'une requête SQL et parcourir les lignes de données résultantes.
Exemple :

```
String selectQuery = "SELECT nom, prenom FROM Etudiants";
ResultSet resultSet = statement.executeQuery(selectQuery);
while (resultSet.next()) {
    String nom = resultSet.getString("nom");
    String prenom = resultSet.getString("prenom");
    System.out.println("Nom : " + nom + ", Prénom : " + prenom);
}
```

Exercice 1: Gestion des erreurs, connexion et fermeture de ressources sous JDBC

Créez un programme Java qui gère la connexion à la base de données MySQL en utilisant JDBC. En cas d'échec de la connexion, gérez l'erreur en affichant un message d'erreur approprié. Si la connexion réussit, affichez un message de succès. Assurez-vous également de fermer correctement toutes les ressources de connexion à la fin de l'exécution.
P.S. afin de charger le pilote JDBC d'une façon explicite, on peut utiliser la commande `"Class.forName("com.mysql.cj.jdbc.Driver");"` dans le bloc try-catch.

Exercice 2 : Création et exécution d'une requête SQL avec JDBC

Créez une requête SQL à l'aide de JDBC pour récupérer toutes les locations effectuées par le client dont l'ID_Client est 3, puis affichez les détails de ces locations (ID_Location, ID_Véhicule, ID_Employé, Date_Début, Date_Fin, Coût_Total). Donner une première version du code sans l'utilisation de ***PreparedStatement*** puis une deuxième version du code avec l'utilisation de ***PreparedStatement.***

Exercice 3 –Insertion de données avec paramètres sécurisés

Donner le code Java pour insérer un nouveau client dans la table "Clients" en utilisant JDBC avec des paramètres sécurisés. Assurez-vous d'afficher un message indiquant si l'insertion a réussi ou échoué.

Exercice 4 – suppression de données

Donner le code Java pour supprimer toutes les locations ayant un cout total supérieur 275 Euros de la table "Locations".

Exercice 5 – Utilisation de procédures stockées avec JDBC

Dans cet exercice, nous allons créer une procédure stockée pour mettre à jour le prix de location de tous les véhicules d'une certaine marque dans la table "Vehicules". La procédure stockée doit prendre en entrée le nom de la marque à mettre à jour et le nouveau prix de location.

- Créez une procédure stockée dans MySQL nommée "MiseAJourPrixVehicules" qui prend deux paramètres en entrée : le nom de la marque (VARCHAR) et le nouveau prix de location (DECIMAL).
- Dans la procédure stockée, utilisez une instruction SQL pour mettre à jour tous les enregistrements dans la table "Véhicules" qui correspondent à la marque spécifiée avec le nouveau prix de location.
- Donnez le code Java avec JDBC qui fait appel à cette procédure stockée en fournissant la marque à mettre à jour et le nouveau prix en tant que paramètres.
- Affichez un message pour indiquer si la mise à jour a réussi ou échoué.

Exercice 6 – Gestion de transactions avec JDBC

Vous avez les tables "Comptes" et "Transactions" avec des données préalablement insérées (fichier Compte_banque.sql). L'objectif de cet exercice est de simuler le transfert d'argent d'un compte source vers un compte cible en utilisant une transaction. Donner le code Java avec JDBC qui effectue les actions suivantes dans une transaction :

- Retirez une certaine somme d'argent du compte source.
- Déposez la même somme d'argent sur le compte cible.
- Ajoutez une entrée dans la table "Transactions" pour enregistrer la transaction.
- Utilisez une transaction pour garantir que les trois actions sont réalisées de manière atomique. Assurez-vous d'utiliser commit pour valider la transaction en cas de succès, et rollback pour l'annuler en cas d'échec.
- Affichez un message pour indiquer si la transaction a été réussie ou annulée.

Exercice 7 – Gestion de transactions avec JDBC – gestion des erreurs

Adapter le code de l'exercice 6 pour simuler les deux cas suivants :

- Le solde du compte source est insuffisant pour effectuer le retrait
- Le compte source n'existe pas