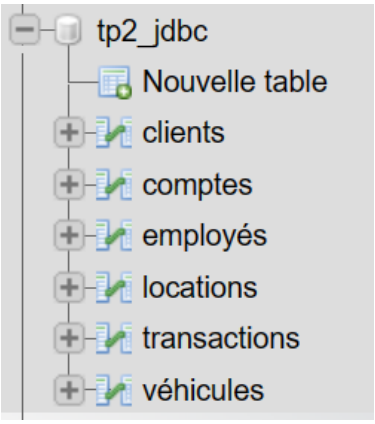
TP2 - SQL :

Création de la base de données et importation des tables dans celle-ci :



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.

→ **Réponse :**

Code :

```
import java.sql.*;

public class ConnectDB {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/tp2_jdbc";
        String username = "root";
        String password = "";

       // Définir la connexion en dehors du bloc try pour qu'elle soit accessible
dans le bloc finally
        Connection connection = null;

        try {
            // Charger le pilote JDBC
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Établir une connexion à la base de données
            connection = DriverManager.getConnection(url, username, password);
```

```
            // Création d'une instruction SQL :
            //PreparedStatement prepStatement = connection.prepareStatement();

            if (connection != null) {
                System.out.println("Connexion à la base de données réussie !");

        } catch (ClassNotFoundException e) {
             System.out.println("Erreur lors du chargement du pilote JDBC : " +
e.getMessage());
        } catch (SQLException e) {
            System.out.println("Erreur lors de la connexion à la base de données
: " + e.getMessage());
        } finally {
            try {
                if (connection != null) {
                    connection.close();
                    System.out.println("Connexion fermée avec succès.");
                }
            } catch (SQLException e) {
                System.out.println("Erreur lors de la fermeture de la connexion
: " + e.getMessage());
            }
        }
    }
}
```

Exécution :

```
\bin\java.exe' '@C:\Users\neroj\AppData\Local\Temp\cp
Connexion à la base de données réussie !
Connexion fermée avec succès.
```

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.***

→ **Réponse :**

1ère exemple sans commandes préparées :

Code :

```
  // Première exemple sans utiliser PreparedStatement :
```

```
  // Créer une instruction SQL
  Statement statement = connection.createStatement();
  String selectQuery = "SELECT * FROM locations WHERE ID_client =  " + id_client
;
  ResultSet resultSet = statement.executeQuery(selectQuery);
```

2e exemple avec la commande préparée :

```
// Afficher le résultat :
                int id_client = 3;
                String selectionSQL = "SELECT * FROM locations WHERE ID_client
=  " + id_client ;

                // Deuxième exemple utilisation de PreparedStatement :
                                        PreparedStatement    prepStatement    =
connection.prepareStatement(selectionSQL);

                ResultSet resultSet = prepStatement.executeQuery();

                while (resultSet.next()){
                     // Récupérer les donnéesQ
                        int ID_location = resultSet.getInt("ID_location");
                        int ID_Véhicule = resultSet.getInt("ID_Véhicule");
                        int ID_Employé = resultSet.getInt("ID_Employé");
                        Date Date_Début = resultSet.getDate("Date_Début");
                        Date Date_Fin = resultSet.getDate("Date_Fin");
                        int Coût_Total = resultSet.getInt("Coût_Total");
                         System.out.println("Le client avec l'id :" + id_client
+ " a effectué une location ( ID_location : " + ID_location + " ) de véhicule (
ID_Véhicule : " + ID_Véhicule +
                         ") du " + Date_Début + " au " + Date_Fin + " et ça lui
a coûté " + Coût_Total + " avec un employé d'id : " + ID_Employé);
                        }
                    }
```

Exécution :

```
Le client avec l'id :3 a effectué une location ( ID_location : 3 ) de véhicule ( ID_Véhicule : 2) du 2023-09-12 au 2023-09-17 et ça lui a coûté 275 avec un employé d'id : 3
Connexion fermée avec succès.
```

Commentaire :

Les deux moyens d'affichage permettent d'afficher le même contenu cependant utiliser une interface ***PreparedStated*** est plus préconisé car elle permet de récupérer les données de manière plus sécurisées et efficaces.

Exercice 3 – Insertion de données :

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.

→ **Réponse :**

Code :

```
// Exercice 3 - Insertion de données :

// Requête pour insérer les données :
String insertSQL = "INSERT INTO Clients (Nom,Prénom,Adresse,Numéro_Téléphone)
VALUE ('Kamil','Leo','1 rue perdu 23449 Perdu', '09-34-34-34-34')";
// Préparer la commande :
PreparedStatement preparedStatement = connection.prepareStatement(insertSQL);
// Exécuter la commande :
int rowsInserted = preparedStatement.executeUpdate();

// Si l'insertion est un succès :
if (rowsInserted > 0) {
   System.out.println("Les données ont été insérées avec succès !");
} else {
   System.out.println("Les données n'ont pas été insérées avec succès.");
                }

            }
```

Exécution :

```
mk98jo23nsdwjx711tfn.argfile' 'ConnectDB'
Connexion à la base de données réussie !
Les données ont été insérées avec succès !
Connexion fermée avec succès.
```

Vue LocalHost :

Éditer	Copier	Supprimer	9	Kamil	Leo	1 rue perdu 23449 Perdu	09-34-34-34-34

Commentaire :

J'ai gardé la même démarche que pour l'affichage tout en apportant les modifications nécessaires pour l'insertion. Je me suis aidée du cours pour écrire le code java correspondant à l'insertion des données dans la table. Puis j'ai ajouté une condition if pour vérifier que l'insertion a été effective.

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".

Code :

```
// Exercice 4 - Suppression de données :

                // Requête sql pour supprpimer les données :
                String delSQL = "DELETE FROM Locations WHERE Coût_total > 275";
                // Préparer la commande :
                                        PreparedStatement    prepStatement    =
connection.prepareStatement(delSQL);
                // Exécuter la commande :
                int rowsDeleted = prepStatement.executeUpdate();

                // Vérifier si la suppression a été faite :
                if (rowsDeleted != 0) {
                      System.out.println("Les données ont été supprimées avec
succès !");
                } else {
                     System.out.println("Les données n'ont pas été supprimées
...");
                }
```

Exécution :

```
mk98jo23nsdwjx711tfn.argfile' 'ConnectDB'
Connexion à la base de données réussie !
Les données ont été supprimées avec succès !
Connexion fermée avec succès.
```

Vue LocalHost :

ID_Location	ID_Client	ID_Véhicule	ID_Employé	Date_Début	Date_Fin	Coût_Total
ner 1	1	1	1	2023-09-10	2023-09-15	250.00
ner 2	2	3	2	2023-09-11	2023-09-14	180.00
ner 3	3	2	3	2023-09-12	2023-09-17	275.00
ner 4	4	5	4	2023-09-13	2023-09-16	220.00
ner 5	5	4	5	2023-09-14	2023-09-19	260.00
ner 6	6	7	6	2023-09-15	2023-09-18	240.00

Commentaire :

J’ai utilisé la même structure pour effectuer une suppression j’ai juste apporté des modifications pour le nom des variables et évidemment pour la requête SQL.

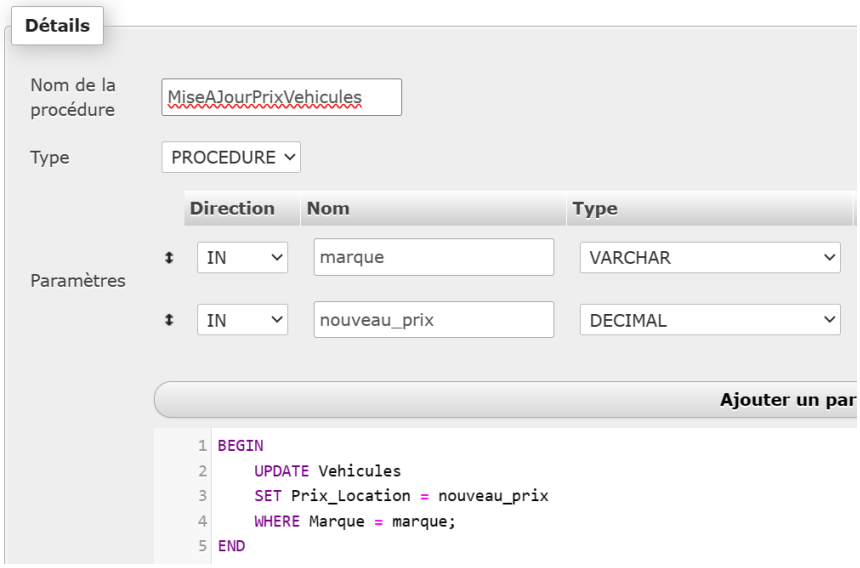
Exercice 5 - Utilisation de procédures stockées avec JDBC :

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.

Création dans localhost :



- 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.

Code :

```
// Exercice 5 - Exécution d'une procédure :

            // Nom de la marque à mettre à jour et nouveau prix
            String marque = "Ford"; // Remplacez par la marque souhaitée
           double nouveauPrix = 22.00; // Remplacez par le nouveau prix souhaité

            // Appel de la procédure stockée
            String call = "{call MiseAJourPrixVehicules(?, ?)}";
```

```
            callableStatement = connection.prepareCall(call);
            callableStatement.setString(1, marque);
            callableStatement.setDouble(2, nouveauPrix);

            // Exécute la procédure stockée
            callableStatement.execute();

            System.out.println("Mise à jour réussie.");
```

Exécution :

```
\bin\java.exe' '@C:\Users\nero
Mise à jour réussie.
```

Vue LocalHost :

3	Ford	Focus	2021	IJ 789 KL	22.00

Commentaire :

Pour arriver à terme de cet exercice, j'ai rencontré des problématiques, notamment le problématique concernant l'interface ***CallableStatement*** que je n'arrivais pas exécuter dans mon fichier courant donc j'ai créé un autre fichier java pour exécuter alors que le code était le même. Également, il y avait des erreurs écriture dans ma procédure ce qui a généré la modification de toutes les données de la table.

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.

Code :

```
// Importation des interfaces nécessaires :
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
```

```java
public class transaction {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/tp2_jdbc";
        String username = "root";
        String password = "";

        // Définir la connexion en dehors du bloc try pour qu'elle soit
accessible dans le bloc finally
        Connection connection = null;

        try {
            // Charger le pilote JDBC
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Établir une connexion à la base de données
            connection = DriverManager.getConnection(url, username, password);

            // Exercice 6 - Transaction :
            // Désactivation du mode commit automatique :
            connection.setAutoCommit(false);

            withdraw(connection, 100001, 500);

            // Effectuer un dépôt dans le compte 2 :
            deposit(connection, 100010, 500);

            // Valider la transaction :
            connection.commit();
            System.out.println("Transaction réussie !");
        } catch (SQLException e) {
            System.out.println("Transaction échouée ! : " + e);
            if (connection != null) {
                try {
                    connection.rollback(); // En cas d'erreur, annule la
transaction
                } catch (SQLException rollbackException) {
                    System.err.println("Erreur lors de l'annulation de la
transaction : " + rollbackException);
                }
            }
        } catch (ClassNotFoundException e) {
            // Gestion de l'exception ClassNotFoundException
            e.printStackTrace();
        } finally {
            if (connection != null) {
                try {
                    connection.setAutoCommit(true); // Réactive le mode commit
automatique
```

```
                    connection.close(); // Ferme la connexion
                } catch (SQLException closeException) {
                    System.err.println("Erreur lors de la fermeture de la
connexion : " + closeException);
                }
            }
        }
    }

    // Méthode withdraw :
    private static void withdraw(Connection connection, int Numéro_Compte,
double valeur) throws SQLException {
        String sql = "UPDATE comptes SET Solde = Solde - ? WHERE Numéro_Compte
= ?";
        // Préparation de la requête :
        try (PreparedStatement preparedStatement =
connection.prepareStatement(sql)) {
            // on vient associer les valeurs :
            preparedStatement.setDouble(1, valeur);
            preparedStatement.setInt(2, Numéro_Compte);
            preparedStatement.executeUpdate();
        }
    }

    // Méthode deposit :
    private static void deposit(Connection connection, int Numéro_Compte,
double valeur) throws SQLException {
        String sql = "UPDATE comptes SET Solde = Solde + ? WHERE Numéro_Compte
= ?";
        // Préparation de la requête :
        try (PreparedStatement preparedStatement =
connection.prepareStatement(sql)) {
            // on vient associer les valeurs :
            preparedStatement.setDouble(1, valeur);
            preparedStatement.setInt(2, Numéro_Compte);
            preparedStatement.executeUpdate();
        }
    }
}
```

Exécution :

```
tfn.argfile' 'transaction
Transaction réussie !
```

Vue LocalHost :

Numéro_Compte	Titulaire	Solde
100001	John Doe	4500.00
100002	Jane Smith	3500.00
100003	Alice Johnson	7500.00
100004	Bob Williams	6200.00
100005	Eva Davis	4300.00
100006	Michael Brown	8800.00
100007	Olivia Lee	3000.00
100008	David Wilson	4100.00
100009	Sophia Miller	6500.00
100010	Liam Anderson	8200.00

Commentaire :

Pour réaliser cet exercice, je me suis aidée du cours, j'ai suivi l'exemple donnée concernant la transaction en java et cela m'a aidé à réussir l'exercice.

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

Code :

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.sql.ResultSet;


public class transaction {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/tp2_jdbc";
        String username = "root";
        String password = "";

        // Définir la connexion en dehors du bloc try pour qu'elle soit
accessible dans le bloc finally
        Connection connection = null;

        try {
```

```
            // Charger le pilote JDBC
            Class.forName("com.mysql.cj.jdbc.Driver");

            // Établir une connexion à la base de données
            connection = DriverManager.getConnection(url, username, password);

            // Exercice 6 - Transaction :
            // Désactivation du mode commit automatique :
            connection.setAutoCommit(false);

            withdraw(connection, 100001, 500);

            // Effectuer un dépôt dans le compte 2 :
            deposit(connection, 100010, 500);

            // Valider la transaction :
            connection.commit();
            System.out.println("Transaction réussie !");
        } catch (SQLException e) {
            System.out.println("Transaction échouée ! : " + e);
            if (connection != null) {
                try {
                    connection.rollback(); // En cas d'erreur, annule la
transaction
                } catch (SQLException rollbackException) {
                    System.err.println("Erreur lors de l'annulation de la
transaction : " + rollbackException);
                }
            }
        } catch (ClassNotFoundException e) {
            // Gestion de l'exception ClassNotFoundException
            e.printStackTrace();
        } finally {
            if (connection != null) {
                try {
                    connection.setAutoCommit(true); // Réactive le mode commit
automatique
                    connection.close(); // Ferme la connexion
                } catch (SQLException closeException) {
                    System.err.println("Erreur lors de la fermeture de la
connexion : " + closeException);
                }
            }
        }
    }

    // Méthode withdraw :
    private static void withdraw(Connection connection, int Numéro_Compte,
double valeur) throws SQLException {
```

```
        // Obtenir le solde actuel du compte
        double soldeActuel = getSolde(connection, Numéro_Compte);

        // Vérifier si le solde est suffisant pour le retrait
        if (soldeActuel >= valeur) {
            String sql = "UPDATE comptes SET Solde = Solde - ? WHERE
Numéro_Compte = ?";
            // Préparation de la requête :
            try (PreparedStatement preparedStatement =
connection.prepareStatement(sql)) {
                // Associer les valeurs :
                preparedStatement.setDouble(1, valeur);
                preparedStatement.setInt(2, Numéro_Compte);
                preparedStatement.executeUpdate();
            }
        } else {
            System.out.println("Solde insuffisant pour effectuer le
retrait.");
        }
    }

    // Méthode pour obtenir le solde actuel du compte
    private static double getSolde(Connection connection, int Numéro_Compte)
throws SQLException {
        String sql = "SELECT Solde FROM comptes WHERE Numéro_Compte = ?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(sql)) {
            preparedStatement.setInt(1, Numéro_Compte);
            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                return resultSet.getDouble("Solde");
            }
            return 0.0; // Compte inexistant
    }
}


    // Méthode deposit :
    private static void deposit(Connection connection, int Numéro_Compte,
double valeur) throws SQLException {
        // Vérifier si le compte existe
        if (getCompte(connection, Numéro_Compte)) {
            String sql = "UPDATE comptes SET Solde = Solde + ? WHERE
Numéro_Compte = ?";
            // Préparation de la requête :
            try (PreparedStatement preparedStatement =
connection.prepareStatement(sql)) {
                // Associer les valeurs :
                preparedStatement.setDouble(1, valeur);
```

```
                preparedStatement.setInt(2, Numéro_Compte);
                preparedStatement.executeUpdate();
            }
        } else {
            System.out.println("Le compte source n'existe pas.");
        }
    }

    // Méthode pour vérifier si le compte existe
    private static boolean getCompte(Connection connection, int Numéro_Compte)
throws SQLException {
        String sql = "SELECT COUNT(*) FROM comptes WHERE Numéro_Compte = ?";
        try (PreparedStatement preparedStatement =
connection.prepareStatement(sql)) {
            preparedStatement.setInt(1, Numéro_Compte);
            ResultSet resultSet = preparedStatement.executeQuery();
            resultSet.next();
            return resultSet.getInt(1) == 1;
        }
    }
}
```

Exécution :

```
mk98jo23nsdwjx711tfn.argf
Transaction réussie !
```

Le cas où le compte existe et le montant est valide :

Avant - Après pour withdraw :

Supprimer	100001	John Doe	4500.00

→

Numéro_Compte	Titulaire	Solde
100001	John Doe	4000.00

Le cas où le compte n'existe pas :

```
withdraw(connection, Numéro_Compte:101, valeur:500);

// Effectuer un dépôt dans le compte 2 :
deposit(connection, Numéro_Compte:10, valeur:500);
```

```
Le compte source n'existe pas.
```

Le cas où le solde n'est pas suffisant :

```
withdraw(connection, Numéro_Compte:100001, valeur:400000);
```

```
Solde insuffisant pour effectuer le retrait.
```