TP5 L'accès aux bases de données : Consultation et mise à jour

OBJECTIF DU TP

- Accéder à une base de données dans un programme java.

Objectifs spécifiques de ce TP

- Installer le driver d'un SGBD (Oracle, MySQL, Access….).
- Tester la connexion au serveur de données.
- Se connecter à une base de données
- Exécuter une requête de mise à jour et de consultation.

Travail à faire

1. Créer la base de données bdGestion avec MySQL.

Description de la base de données bdGestion :

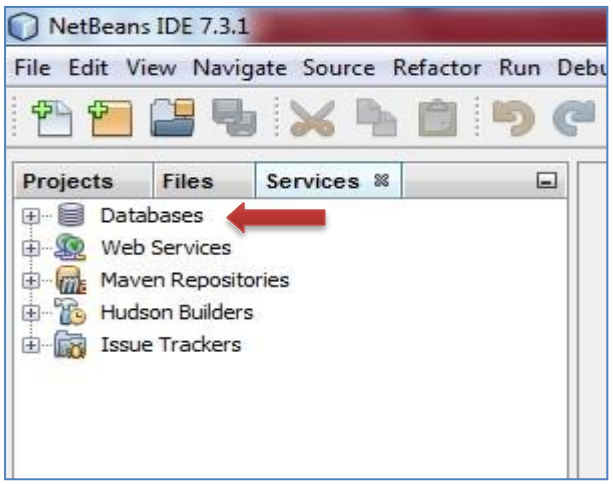
La table : **Client**

Colonne	Type	Null
codeCli	Varchar(20)	Non
nomCli	Varchar(20)	Non
adresseCli	Varchar(20)	Non
emailCli	Varchar(10)	Non

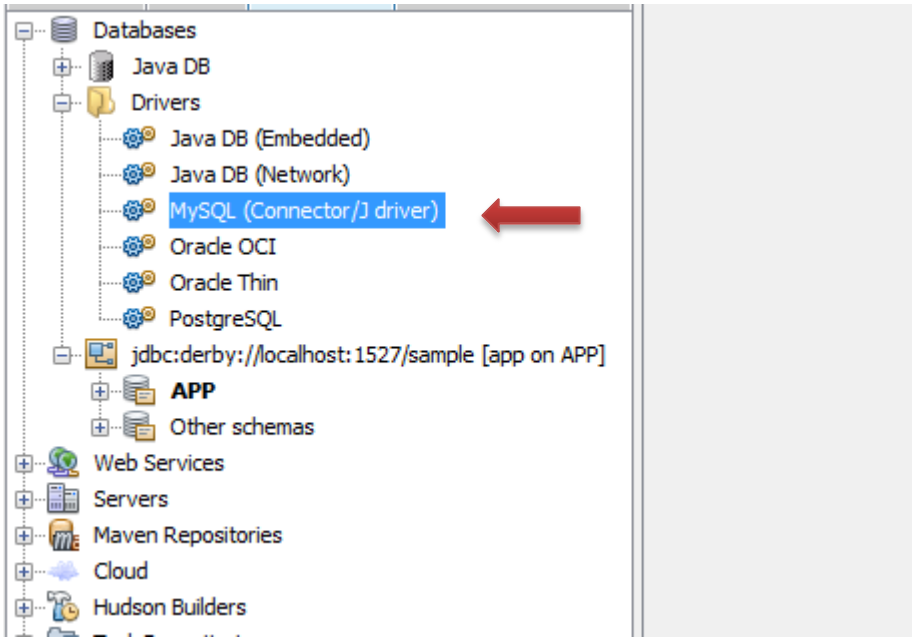
La table : **Facture**

Colonne	Type	Null
numFacture	Varchar(30)	Non
dateFacture	Date	Non
total	Float	Non
refCli	Varchar(20)	Non

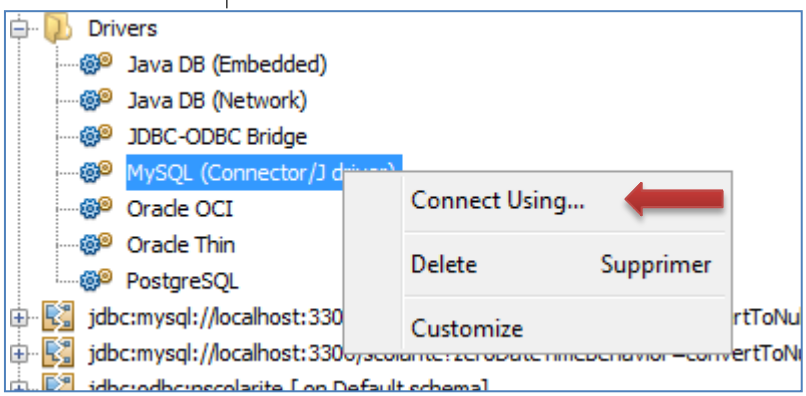
2. Testez la connexion à votre base de données à partir de votre EDI NetBeans.



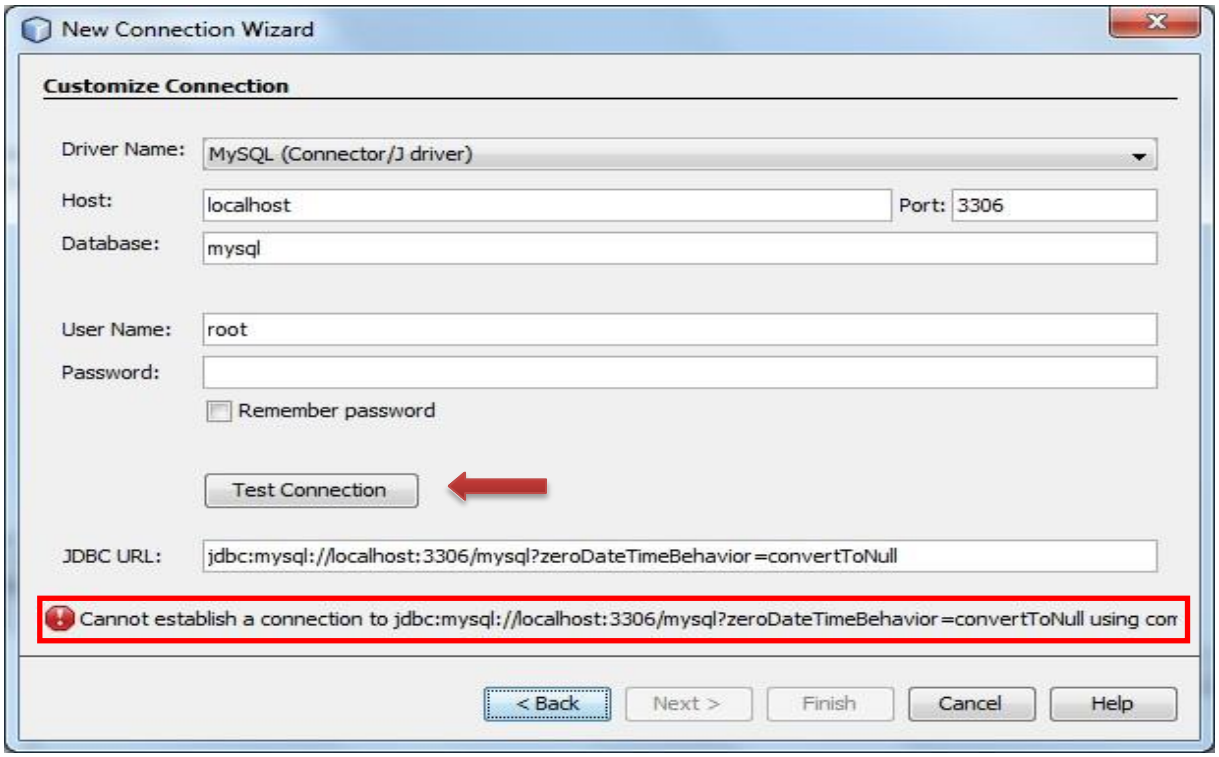
✓ Développez l'arborescence DataBases.



✓ Développez l'arborescence Drivers.
✓ Avec le driver qui correspond à la base de données, testez la connexion avec connect using.

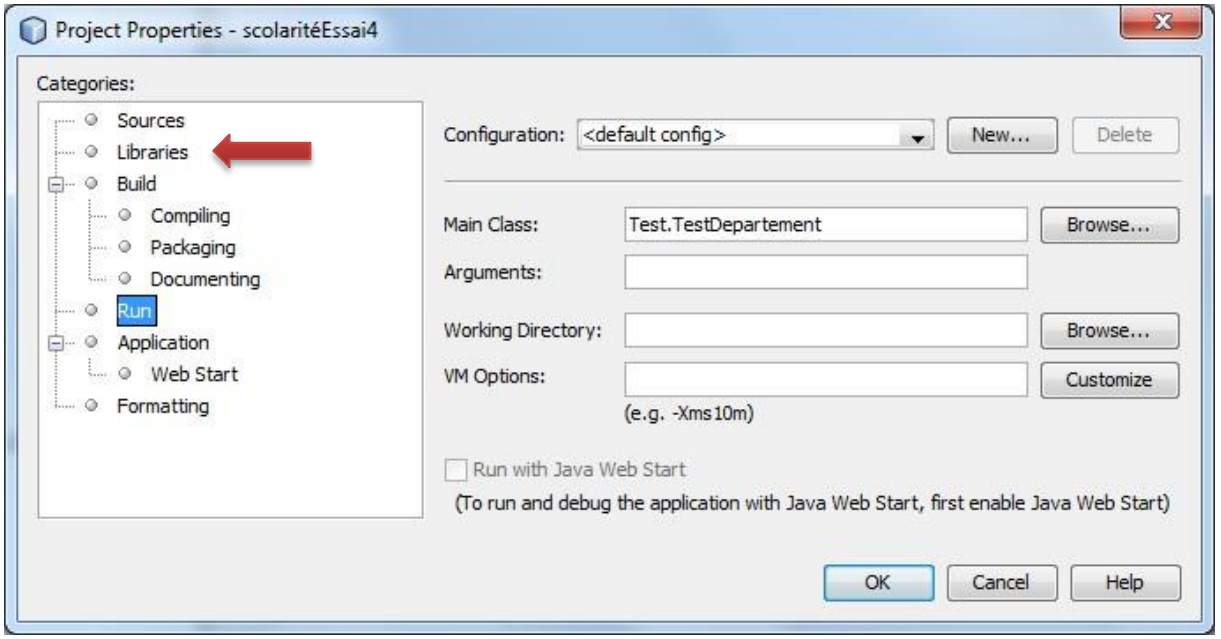


✓ La fenêtre suivante s'ouvre :

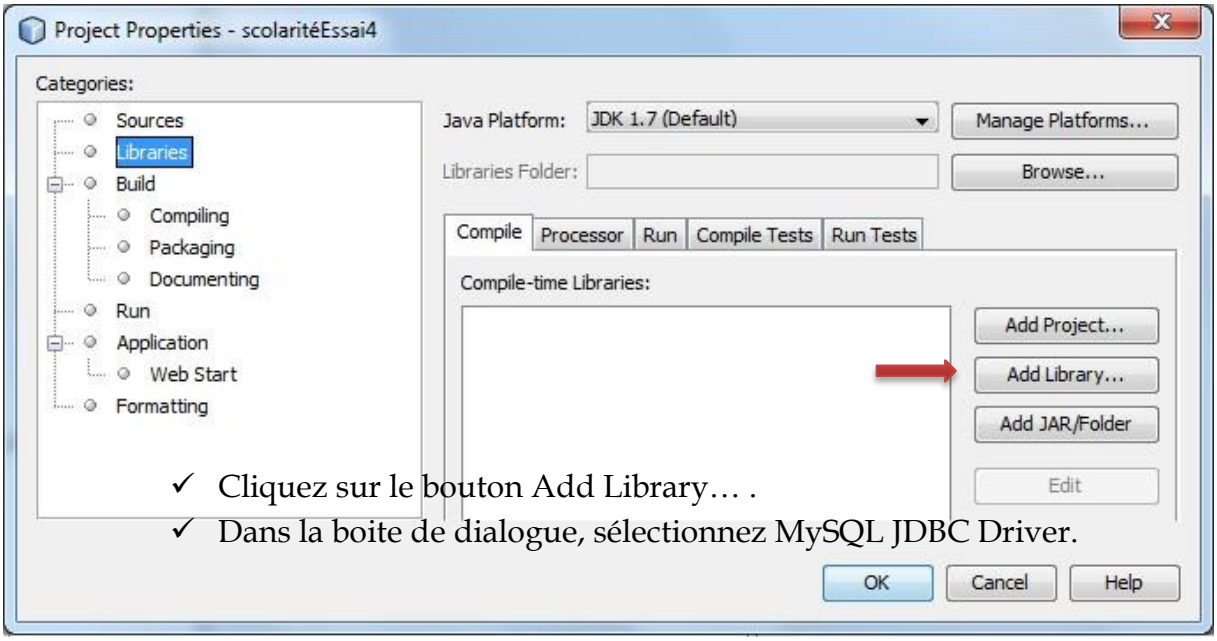


✓ Ça ne marche pas pour divers raisons :
✓ Le serveur de base de données n'est pas démarré.
✓ Le driver du SGBD n'est pas ajouté au projet.
✓ …

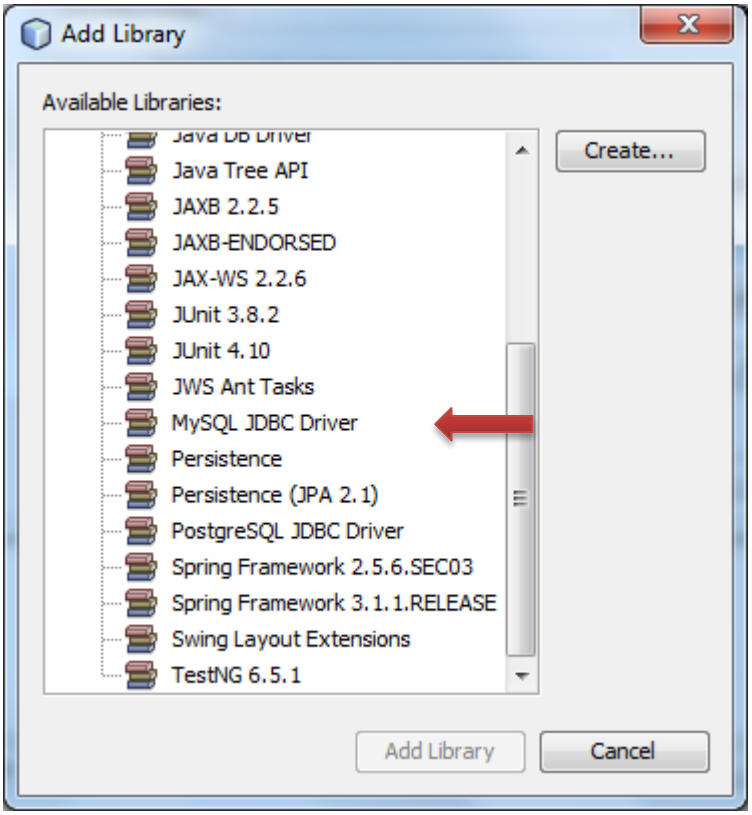
3. Ajoutez le drivers JDBC (MySQL) avec le menu : Run – Set Project Configuration – Customize.



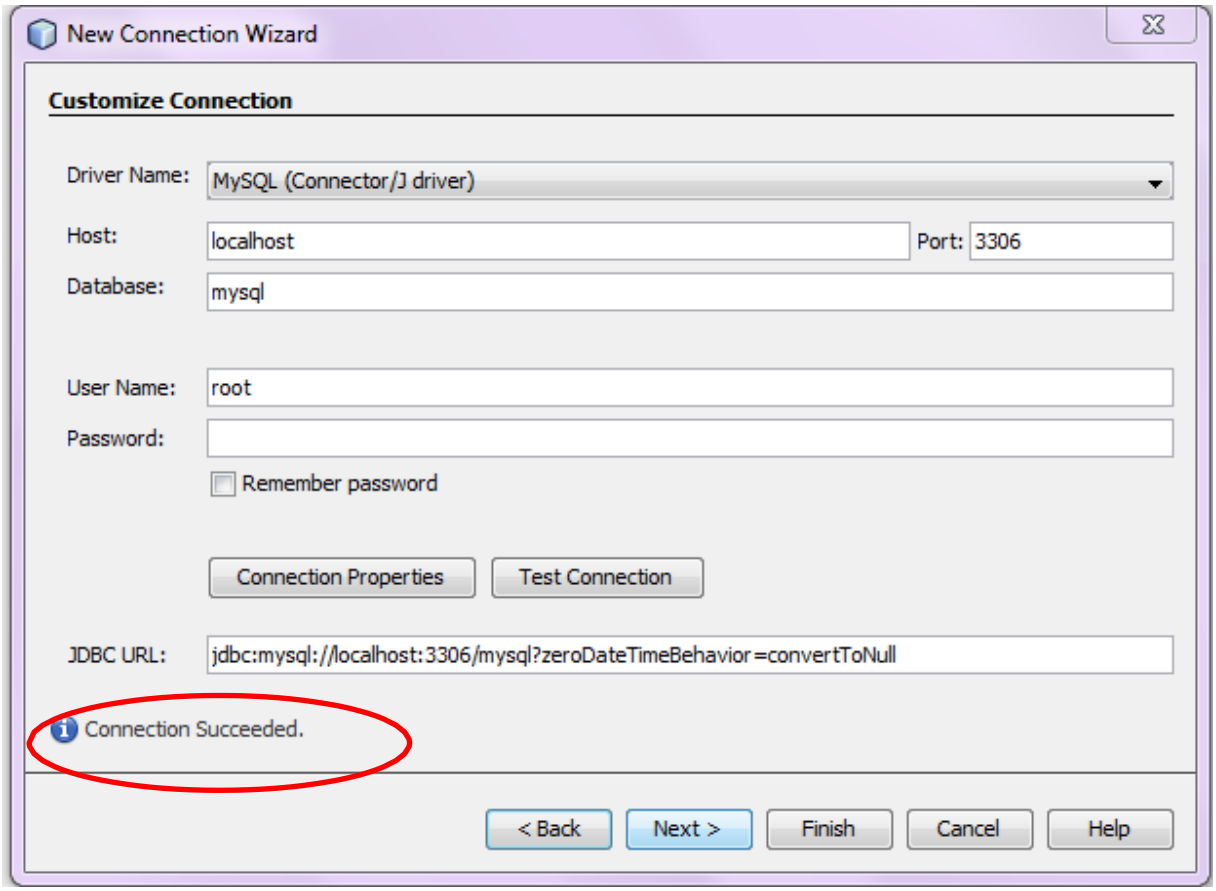
- ✓ Dans l'arborescence, choisissez librairies.



- ✓ Cliquez sur le bouton Add Library… .
- ✓ Dans la boite de dialogue, sélectionnez MySQL JDBC Driver.



- ✓ et testez la connexion.



I. Connexion à une Base de données

- **Les classes de l'API JDBC sont :**

l y a la classe **DriverManager** et les 3 interfaces **Connection**, **Statement** (et **PreparedStatement**), et ResultSet, chacune correspondant à une étape de l'accès aux données :

Classe/Interface	**Rôle**
DriverManager	Charger et configurer le driver de la base de données.
Connection	Réaliser la connexion et l'authentification à la base de données.
Statement (et PreparedStatement)	Contenir la requête SQL et la transmettre à la base de données.
ResultSet	Parcourir les informations retournées par la base de données dans le cas d'une sélection de données

- Créez une classe **Connexion** :
  - ✓ Créez un package DAO
  - ✓ Créez une nouvelle classe : **LaConnexion**

LaConnexion
-con : Connection -pass :chaine de caratcères -uselog :chaine de caractères
+ seConnecter () : Connection

Classe LaConnexion

```
package DAO;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class LaConnexion {
   private  static Connection con;
   private  static String user;
   private static String passWord;

   public  static Connection seConnecter(){
      if (con==null)// pas de connexion
      { try{
            //Charger un driver avec la méthode statique forName de la classe Class.
            Class.forName("com.mysql.jdbc.Driver");
            //D'établir une connexion vers la base de données Gestion, avec la méthode statique
getConnection de la classe DriverManager

con=DriverManager.getConnection("jdbc:mysql://localhost:3306/bdgestion",user,passWord);
            System.out.println("connexion établie");

      }catch(ClassNotFoundException e)
      {System.out.println("driver non trouvé"+e.getMessage());
}catch(SQLException ex){
      System.out.println("bd non trouvé ou problème d'identification "+ex.getMessage());
   }
   }
   return con;   }
public void setUser(String user) {
   this.user = user; }

public void setPassWord(String passWord) {
   this.passWord = passWord;
}}
```

- Créer une classe java main nommée **TestDAO** et tester la classe LaConnexion en ajoutant ce code :

```
public class TestDAO {
   public static void main(String[] args) {
        LaConnexion c=new LaConnexion();
        c.setUser("root");
        c.setPassWord("");
        c.seConnecter();
        }
```

II. Partie Consultation de la base de données

2.1 Consultation avec l'interface Statement

Une fois la connexion établie, il est possible d'exécuter des ordres SQL. Les objets qui peuvent être utilisés pour obtenir des informations sur la base de données sont :

Classe	Rôle
DatabaseMetaData	informations à propos de la base de données : nom des tables, index, version ...
ResultSet	résultat d'une requête et information sur une table. L'accès se fait enregistrement par enregistrement.
ResultSetMetaData	informations sur les colonnes (nom et type) d'un ResultSet

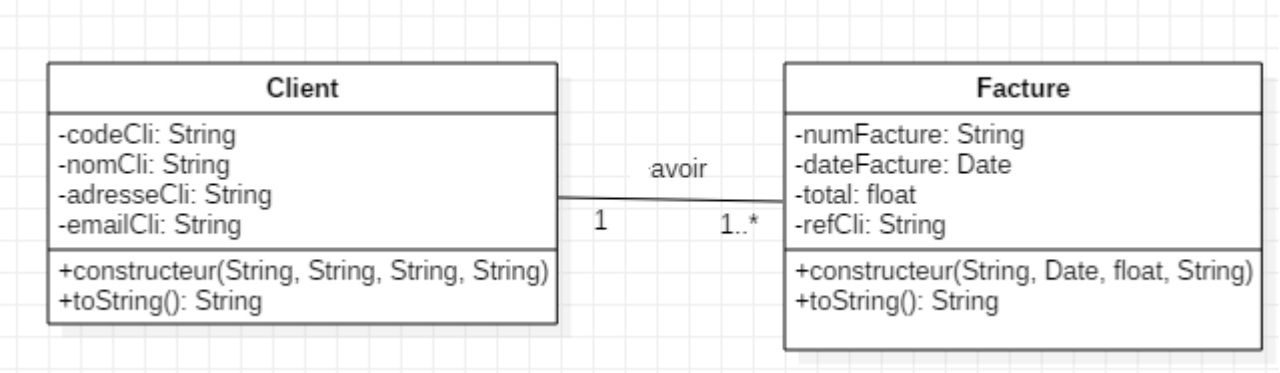
La classe ResultSet

- ✓ 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
int **getInt**(int columnIndex) throws SQLException	retourne le contenu de la colonne dont le numéro est passé en paramètre dans la ligne actuelle de cet objet ResultSet en tant qu'entier.
float **getFloat**(int columnIndex) throws SQLException	retourne le contenu de la colonne dont le numéro est passé en paramètre dans la ligne actuelle de cet objet ResultSet en tant que float
String **getString**(int columnIndex) throws SQLException	retourne le contenu de la colonne dont le numéro est passé en paramètre dans la ligne actuelle de cet objet ResultSet en tant que String
Date **getDate**(int columnIndex) throws SQLException	retourne le contenu de la colonne dont le numéro est passé en paramètre dans la ligne actuelle de cet objet ResultSet en tant qu'objet de la classe java.sql.Date

boolean **next**()throws SQLException	se déplace sur le prochain enregistrement : retourne false si la fin est atteinte
void **close**()throws SQLException	ferme le ResultSet
ResultSetMetaData **getMetaData**() throws SQLException	retourne un objet de type ResultSetMetaData qui contient le nombre, les types et les propriétés des colonnes de cet objet ResultSet.

- Ajouter un package nommé Gestion qui contient les deux classes Client et Facture suivantes :



- Dans le package DAO Créez une classe **DAOClient**:

DAOClient
+lister() :ArrayList<Client>[]

+lister() :ArrayList<Client>[]

Retourne tous les clients enregistrés dans la base de données dans un ArrayList en utilisant l'objet Statement, pour cela il faut :

- Se connecter à la **bdgestion**
- Crée une **requête select** pour extraire tous les clients se trouvant dans la table Client. La requête est une chaine de caractères.
- Crée un objet **Statement** : st sur la connexion établit avec la méthode createStatement.
- Envoie la requête SQl avec st.
- Exécuter la requête : **executeQuery**
- Placer le résultat dans un objet **ResultSet**
- Parcourir le résultat (ResultSet) pour remplir une collection ArrayList (avec next).

A chaque appel de **next** le ResultSet pointe un nouveau enregistrement, pour accéder aux champs il faut utiliser les get selon le type des valeurs par exemple getString, getInt.

En paramètre pour les get on peut soit donner le numéro du champ ou son nom :

email = rs.getString(4); //l'email est le 4e champ

codeCli= rs.getInt("CodeCli");

Code de la méthode lister()

```
public static ArrayList<Client> lister(){
    ArrayList<Client> cl= new ArrayList<>();
    Connection cn=LaConnexion.seConnecter();
    String requete="select * from `client`;";
    String codeCl; String nomCl,adrCl,emailCl;
    Client c;
    try{
       Statement st=cn.createStatement();
       ResultSet rs=st.executeQuery(requete);

      while (rs.next()) {
       // extraire les données de l'enregistrement en cours
        codeCl=rs.getString(1);
        nomCl=rs.getString(2);
        adrCl=rs.getString(3);
        emailCl=rs.getString(4);
        c=new Client(codeCl,nomCl,adrCl,emailCl);
       cl.add(c);
  }
  System.out.println(" consultation ok");
} catch (SQLException e) {
  //traitement de l'exception
       System.out.println("pbleme de consultation");
}
  return cl; }
```

Test relatif à cette méthode

```
public class TestDAO {
....
ArrayList<Client> col=DAOClient.lister();
  System.out.println(col.toString());
}
```

2.1 Consultation avec l'interface PreparedStatement

L'interface PreparedStatement :

Elle définit les méthodes pour un objet qui va encapsuler une requête précompilée. Ce type de requête est particulièrement adapté pour une exécution répétée d'une même requête avec des paramètres différents.

Lors de l'utilisation d'un objet de type **PreparedStatement**, la requête est envoyée au moteur de la base de données pour que celui-ci prépare son exécution.

Un objet qui implémente l'interface **PreparedStatement** est obtenu en utilisant la méthode **prepareStatement**() d'un objet de type Connection. Cette méthode attend en paramètre une chaîne de caractères contenant la requête SQL. Dans cette chaîne, chaque paramètre est représenté par un caractère ?.

Un ensemble de méthode setXXX() (ou XXX représente un type primitif ou certains objets tels que String, Date, Object, ...) permet de fournir les valeurs de chaque paramètre défini dans la requête. Le premier paramètre de ces méthodes précise le numéro du paramètre dont la méthode va fournir la valeur. Le second paramètre précise cette valeur.

- Ajoutez dans la classe **DAOClient** les méthodes statiques suivantes :

  +**trouver**(val :chaine) :**TreeMap**<Integer, Client>

  Retourne tous les clients (sous forme de Map dont la clé est le code du client et la valeur est l'objet Client) qui ont le nom qui commence par la chaine passée en paramètre.

  +**listerParVille**(a :String) : **TreeSet** <Clients>

  Retourne tous les clients enregistrés dans la base qui habitent dans la ville passée en paramètre et le résultat sous forme d'une collection **TreeSet**.

  Utiliser la classe **PreparedStatement**.

  +**chercher**(CodeCli :chaine) : **Client**

  Retourne le client qui a le code passé en paramètre

Code de la méthode trouver(val : chaine)

```
public static  TreeMap< String,Client> trouver(String val)
  {TreeMap<String,Client> res=new TreeMap<String,Client>();
     Connection cn=LaConnexion.seConnecter();
     String requete = "select * from `client`where `nomCli`like ? ";

      try{ PreparedStatement pst=cn.prepareStatement(requete);
        pst.setString(1,val+"%");
        ResultSet rs=pst.executeQuery();
        String code;
        String nom,adresse,email;
        Client c;
        if(rs!=null)
        {while (rs.next()){
          code=rs.getString("codeCli");
           nom=rs.getString("nomCli");
           adresse=rs.getString("adressCli");
           email=rs.getString("emailCli");
           c=new Client(code,nom,adresse,email);
           res.put(code, c);
        }}}catch(SQLException ex)
        {System.err.println("probleme de req select"+ex.getMessage()); }
      return res;
       }
```

Test de cette méthode

```
TreeMap<String,Client> colm=DAOClient.trouver("s");
System.out.println(colm.toString());
```

III. Mise à jour des tables

3.1 Première méthode de mise à jour avec l'interface Statement :

- Dans le package DAO, définissez la classe **DAOFacture**

DAOFacture
+ ajouter (f :Facture) : booléen

- La méthode **ajouter**, permet d'ajouter une nouvelle facture dans la base de données :
  - Se connecte à la base
  - Crée une **requête insert** avec les valeurs se trouvant dans l'objet facture (paramètre de la méthode ajouter). La requête est une chaine de caractères.
  - Crée un objet **Statement** : st sur la connexion établit avec la méthode createStatement.
  - Envoie la requête SQl insert (contenant toutes les informations de la nouvelle facture) avec st.

Classe DAOFacture

```
package DAO;
import Gestion.Client;
import Gestion.Facture;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;

public class DAOFacture {
public static boolean ajouter(Facture f)
   {
      Connection cn=LaConnexion.seConnecter();
      String requete= "insert into `facture` values ('"+f.getNumFact()+"',
'"+f.getDateFact()+"','"+f.getTotalFact()+"','"+f.getRefCli()+"');";

     try{ Statement st=cn.createStatement();
        int n=st.executeUpdate(requete);
        if (n>=1)
             System.out.println("ajout réussi");
        return true;
    }catch(SQLException ex)
     { System.out.println("problème de requête"+ex.getMessage()); }
     return false;
   }}
```

- tester avec la classe **TestDAO** la classe **DAOFacture** en ajoutant ce code :

```
public class TestDAO {

   public static void main(String[] args) {
        ……
        Facture f =new Facture("F51","5/4/2020",500,2);
         DAOFacture.ajouter(f);
}
```

3.2 Deuxième méthode de mise à jour avec l'interface PreaparedStatement

❖ Ajouter à la classe **DAOClient** les méthodes suivantes :

DAOClient
+ ajouter (c :Client) : booléen +changerAdresse(Client e, String adr) +supprimer(Client c)

✓ La méthode ajouter, permet d'ajouter un nouveau client dans la base de données.

✓ Au lieu d'utiliser L'interface **Statement** pour la requête nous allons utiliser l'interface **PreparedStatement**, cette dernière permet de paramétrer une requête dans les valeurs paramétrées seront précisées après. Cela nous évitera de construire la requête avec les concaténations nécessaires avec les valeurs.

La classe DAOClient (partie1) : l'ajout

```
public static boolean ajouter(Client c)
  {  Connection cn=LaConnexion.seConnecter();
    String requete = "insert into `client` values(?,?,?,?)";
     try{PreparedStatement pst=cn.prepareStatement(requete);
       pst.setString(1, c.getCodeCli());
       pst.setString(2,c.getNomCli());
       pst.setString(3,c.getAdresseCli());
       pst.setString(4,c.getEmailCli());
       //permet de mettre l'attribut email à Null
      // pst.setNull(4, java.sql.Types.VARCHAR);
        int n=pst.executeUpdate();
        if (n>=1)
            System.out.println("ajout réussi");
        return true;
     }catch(SQLException ex)
     { System.out.println("problème de requête"+ex.getMessage());  }
     return false; }
```

- ✓ La méthode **ChangerAdresse** permet de changer pour le client dont le numéro est donné à une nouvelle adresse.
  - Se connecter à la base
  - Créer une requête update avec les valeurs se trouvant dans l'objet Client et la valeur de l'adresse passés comme paramètres.
  - Envoie une requête SQl update (contenant toutes les informations nécéssaires)

La Classe DAOClient(la modification en utilisant l'interface PreparedStatement)

```
public static boolean changerAdresse(Client c,String adr)
   {Connection cn=LaConnexion.seConnecter();
     String requete = "update `client` set `adressCli`=? where`client`.`codeCli`=?;";
      try{
         PreparedStatement pst=cn.prepareStatement(requete);
        pst.setString(1, adr);
        pst.setString(2,c.getCodeCli());
         int n=pst.executeUpdate();
          if (n>=1)
              System.out.println("Modif réussi");
         return true;

      }catch(SQLException ex)
      { System.out.println("problème de requête Modif"+ex.getMessage());
      }
      return false;
    }
```

Tester la classe DAOClient dans la classe main : TestDAO en ajoutant ce code

Classe TestDAO :

```
package DAO;
import Gestion.Client;
import Gestion.Facture;
import java.util.Date;
public class TestDAO {
   public static void main(String[] args) {
      ……..
     Client cl= new Client(14,"hamdi","Kef","h");
     DAOClient.ajouter(cl);
      DAOClient.changerAdresse(cl, "Kélibia");}}
```

- ✓ La méthode **supprimer** permet de supprimer un client dont le numéro est donné il faut alors :
  - Se connecter à la base
  - Créer une requête update avec la valeur se trouvant dans l'objet Client passé comme paramètre.
  - Envoie une requête SQl update (contenant toutes les informations nécéssaires)

La Classe DAOClient(la suppression en utilisant l'interface PreparedStatement)

```
public static boolean supprimer(Client c)
   {Connection cn=LaConnexion.seConnecter();
  //"update `client` set `AdresseClient`=? where`Client`.`CodeClient`=?;";
     String requete = "delete from `client` where`client`.`codeCli`=?;";
      try{
         PreparedStatement pst=cn.prepareStatement(requete);
        pst.setString(1,c.getCodeCli());

         int n=pst.executeUpdate();
          if (n>=1)
              System.out.println("suppression réussie");
          return true;

      }catch(SQLException ex)
      {  System.out.println("problème de requête de suppression"+ex.getMessage());
      }
      return false;
    }}
```

Tester la classe **DAOClient**(méthode **supprimer**) dans la classe main : TestDAO en ajoutant ce code

Classe TestDAO :

```
public class TestDAO {
   public static void main(String[] args) {
       ……..
     Client cl= new Client(15,"hamdi","Sfax","h@gmail.com");
     DAOClient.ajouter(cl);
     DAOClient.supprimer(cl);
}}
```