Objectifs

Comprendre la manière dont fonctionne JDBC. Utiliser les méthodes principales de cette API. Utiliser l'un des drivers PostgreSQL de type 4 : `org.postgresql.Driver`.

1 Installation du driver JDBC - PostgreSQL

Dans ce qui suit, le package du driver est noté `ledriver.jar`. Ce nom doit évidemment être adapté en fonction de la version actuelle (par ex `pg74.215.jdbc3.jar` ou un lien de type `postgresql.jar`)

1. Le package correspondant au pilote a été précédemment installé dans le répertoire `/usr/share/java`. Ce package est simplement un ensemble de classes regroupant les implémentations des interfaces de l'API `java.sql`. Vérifiez la présence et le nom de la classe Driver par la commande :
   ```
   jar tvf /usr/share/java/ledriver.jar | grep Driver
   jar tvf /usr/share/java/ledriver.jar | grep Result
   ```

2. Mettre à jour le `CLASSPATH` pour avoir accès à ce fichier :
   `setenv CLASSPATH /usr/share/java/ledriver.jar:.` en csh
   `export CLASSPATH=/usr/share/java/ledriver.jar:.` en bash.

   Vous constatez que le driver se nomme `org.postgresql.Driver`, on rappelle que la connexion à la base `mabase` à partir de votre machine se fait à l'URL `jdbc:postgresql://sqlserver/mabase`
3. Testez la connection avec un programme rudimentaire (`TestDB.java`) qui effectue simplement une connexion à la base puis la referme aussitot. **Gardez le toujours quelque part dans un coin !**

2 Programmes de base

Tous les programmes de ce TP seront écrits en utilisant le driver `org.postgresql.Driver`. Il faut donc avoir préalablement installé PostgreSQL, le JDK ainsi que ce driver JDBC.

1. Ecrire un programme Java qui effectue une création de table
2. Ecrire un programme Java qui effectue une insertion de table
3. Ecrire un programme Java qui effectue une sélection de table
4. Ecrire un programme Java qui affiche le nombre de tuples d'une table.
5. Ecrire un programme Java qui affiche la structure d'une table (nombre, noms et types des colonnes).

3 Les properties

Afin que les programmes soient les plus portables possible, il est préférable d'externaliser toutes les constantes utilisées. L'usage d'un objet `Properties` du package `java.util` favorise ce type de travail

1. Modifiez le programme `TestDB` précédent pour que `driver, url, base, schema, user, password` soient chargés à partir d'un fichier `Properties` et non pas "en dur" dans le code.

4 Un objet générique

Ecrire une classe `BddTools` utilisant vos `Properties` et permettant d'interroger n'importe quelle table de la base `Postgres`. Cette classe contiendra notamment les méthodes suivantes :

1. `connecter(String base)` permettant de se connecter à une base de notre serveur. Elle sera notamment appelée par le constructeur.
2. `int nbLines(String table)` permettant de récupérer le nombre de tuples d'une table.
3. `ArrayList getRequest(String requete)` permettant de récupérer sous forme d'une ArrayList d'ArrayLists tous les tuples d'une requête. La premiere ArrayList contientra les noms de colonnes.
4. `String describe(String table)` permettant de récupérer la description de la structure d'une table.
5. `fermer()`, pour clore la connexion.
6. Vous créerez ensuite une classe `TestBdd` permettant de tester la classe précédente.

```
public class TestBdd
{
  public static void main(String args[])
  {
       BddTools bdd = new BddTools("mabase");
       System.out.println(bdd.nbLines("Clients"));
       System.out.println(bdd.decribe("Clients");)
       System.out.println(bdd.getRequest("select * from Clients"));
       essai.close();
    }
}
```