JAVA + SECURITE

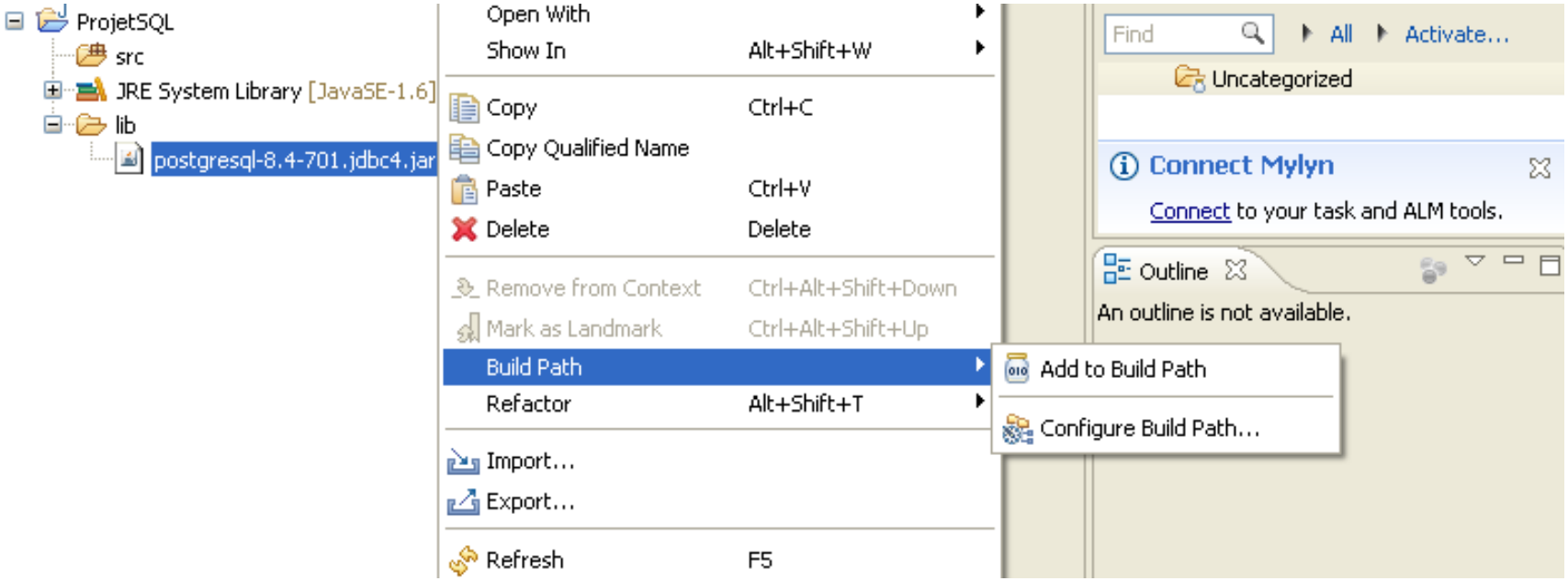
JDBC

- **Comment écrire une application Java qui utilise un serveur PostgreSQL pour la persistance des données ?**
  - En utilisant la librairie JDBC, librairie java pour utiliser du sql
    - Indépendante du serveur BD
    - import java.sql.*
- **Pour PostgreSQL :**
  - http://jdbc.postgresql.org/
  - postgresql-9.3-1102.jdbc41.jar

DRIVER

Dans Eclipse, on crée un répertoire lib au niveau du projet dans lequel on place le driver.

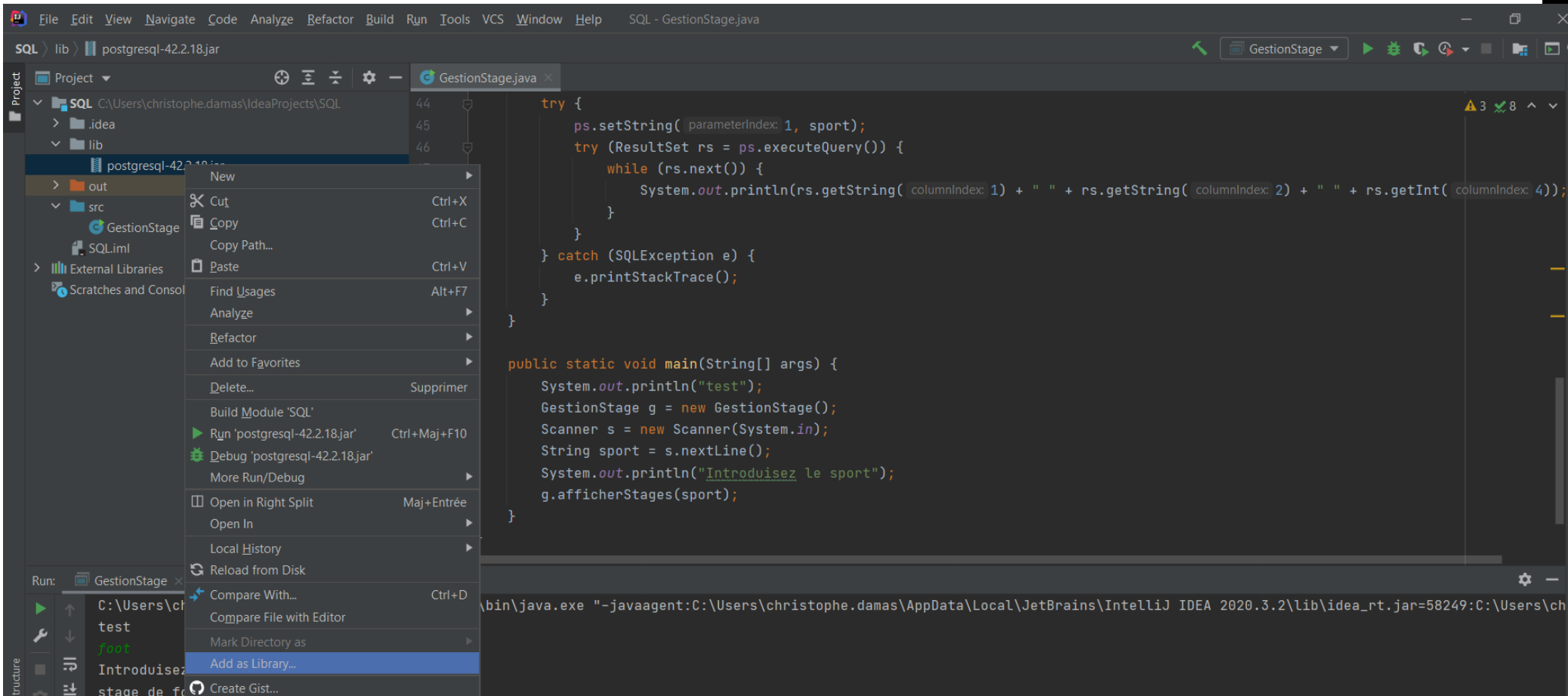
- On ajoute le driver au Build Path.



DRIVER

Dans IntelliJ, on crée un répertoire lib au niveau du projet dans lequel on place le driver.

- Add as Library



DRIVER

Une fois le driver ajouté au Build Path, il faut forcer son chargement à l'exécution :

```
try {
   Class.forName("org.postgresql.Driver");
} catch (ClassNotFoundException e) {
   System.out.println("Driver PostgreSQL manquant !");
   System.exit(1);
}
```

CONNEXION

Il faut ensuite obtenir une connexion entre l'application Java et le serveur de BD

```
String url="jdbc:postgresql://172.24.2.6:5432/dbcdamas14“;
Connection conn=null;
try {
   conn=DriverManager.getConnection(url,”cdamas14”,”azerty”);
} catch (SQLException e) {
   System.out.println("Impossible de joindre le server !");
   System.exit(1);
}
```

CONNEXION

- **Chaque échange entre Java et la BD se fait via une connexion.**
- **On peut ouvrir autant de connexions qu'on le souhaite.**
  - A la même BD ou à d'autres.
- **L'ouverture d'une connexion prend du temps.**
  - On garde donc la connexion active tant qu'on en a besoin.
- **Pour les applications simples : connexion ouverte au démarrage de l'application, et fermée à la fin.**

STATEMENT

- **Statement = instruction(s) à envoyer à la BD.**
  - On peut réutiliser un statement plusieurs fois.

```
try {
   Statement s = conn.createStatement();
   s.executeUpdate("INSERT INTO exercice.utilisateurs "+
        "VALUES (DEFAULT, 'Damas', 'Christophe');");
} catch (SQLException se) {
   System.out.println("Erreur lors de l'insertion !");
   se.printStackTrace();
   System.exit(1);
}
```

RESULTSET

ResultSet = Résultat d'un SELECT

```
try {
   Statement s = conn.createStatement();
   try(ResultSet rs= s.executeQuery("SELECT nom"+
                        "FROM exercice.utilisateurs;"){
         while(rs.next()) {
                  System.out.println(rs.getString(1));
         }
   }
} catch (SQLException se) {
   se.printStackTrace();
   System.exit(1);
}
```

RESULTSET

- **Résultat par curseur (un tuple à la fois)**
  - `next()`
    - avance le curseur
    - retourne faux quand arrivé à la fin, vrai sinon
  - `getString(int c), getTime(int c), getInt(int c), getDouble(int c), ...`
    - Retourne la valeur de la colonne c du tuple en cours
    - En utilisant le type demandé
- **Attention on compte à partir de 1 (pas de 0)**

Cfr Javadoc

TRY AVEC RESSOURCE

```
try(ResultSet rs= s.executeQuery("SELECT nom"+
                        "FROM exercice.utilisateurs;"){
        while(rs.next()) {
                System.out.println(rs.getString(1));
        }
   }
```

Ceci n'est pas un try...catch...finally !

C'est un try avec ressource

=> la ressource sera fermée à la fin du bloc

=> équivalent à un finally {rs.close();}

RESULTSET: METADATA

`getMetaData()` (sur un ResultSet)

- Retourne un objet metadata
  - `getColumnCount()` : nombre de colonnes
  - `getColumnName(int c)` : nom de la colonne
  - `getColumnType(int c)` : type SQL
  - Cfr. Javadoc

EXEMPLE

```
System.out.println("Entrez vos nom et prénom:");
String nom=scanner.nextLine();
String prenom=scanner.nextLine();
try {
   Statement s = conn.createStatement();
   s.executeUpdate("INSERT INTO exercice.utilisateurs "+
        "VALUES (DEFAULT, '"+nom+"', '"+prenom+"');");
} catch (SQLException se) {
   System.out.println("Erreur lors de l'insertion !");
   se.printStackTrace();
   System.exit(1);
}
```

TESTONS…

```
Entrez vos nom et prénom:
Damas
Christophe
```

et le code exécute réellement ceci en SQL :

```
INSERT INTO exercice.utilisateurs VALUES (DEFAULT,
'Damas', 'Christophe');
```

TESTONS ENCORE...

```
Entrez vos nom et prénom:

Damas

Christophe'); DROP TABLE exercice.utilisateurs; SELECT ('A
```

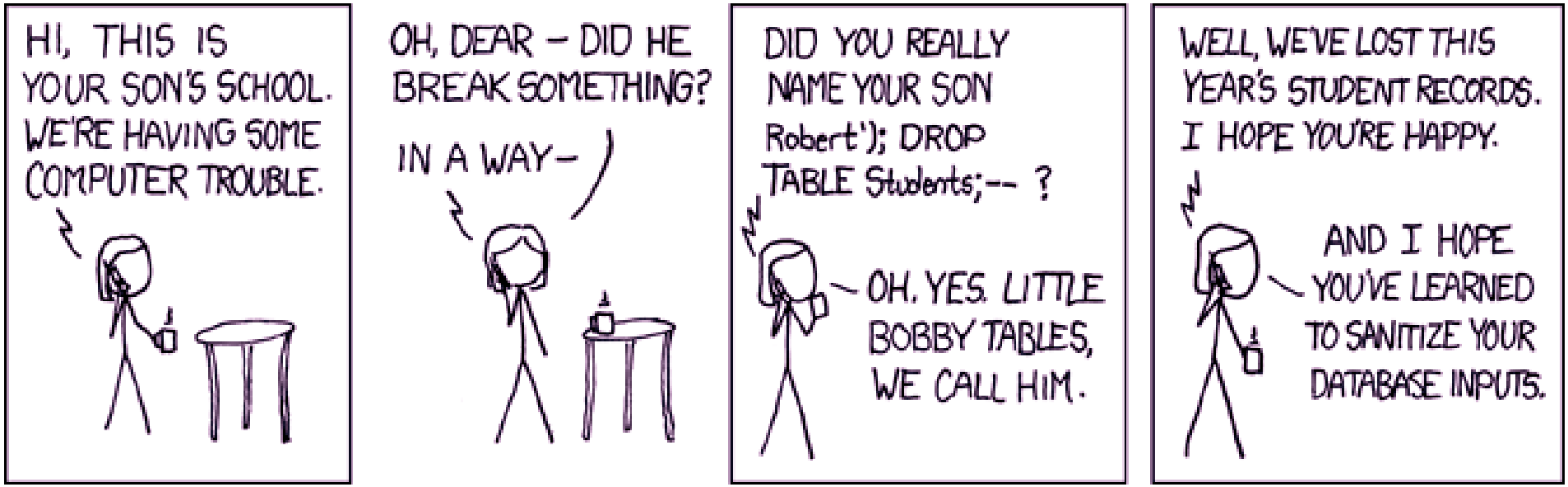
et le code exécute réellement ceci en SQL :

```
INSERT INTO exercice.utilisateurs VALUES (DEFAULT,

'Damas', 'Christophe'); DROP TABLE exercice.utilisateurs;
   SELECT('A');
```

On vient juste de perdre la table utilisateurs...

http://xkcd.com/327/



SQL INJECTION

- **Attaque permettant d'injecter des requêtes SQL non prévues.**
- **Faille de sécurité très fréquente dans le web !**
- **Solution générale : PreparedStatement**

PREPAREDSTATEMENT

- **PreparedStatement = une instruction paramétrée à envoyer à la BD.**
  - On peut réutiliser un PreparedStatement plusieurs fois, avec des valeurs de paramètres différents.

```
try {
    PreparedStatement ps = conn.prepareStatement("INSERT INTO"+          "
                                        exercice.utilisateurs VALUES (DEFAULT, ?, ?);");
    ps.setString(1,"Damas");
    ps.setString(2,"Christophe");
    ps.executeUpdate();
    ps.setString(1,"Ferneeuw");
    ps.setString(2,"Stéphanie");
    ps.executeUpdate();
} catch (SQLException se) {
    System.out.println("Erreur lors de l'insertion !");
    se.printStackTrace();
    System.exit(1);
}
```

BONNE PRATIQUE

- **On n’utilise jamais de Statement:**
  - PreparedStatement évite les SQL injections
- **On crée les PreparedStatement une seule fois:**
  - au démarrage ou au premier usage
  - l’instruction SQL est compilée une seule fois => exécution multiple + rapide

OÙ PLACER LE CODE ?

Manipulation des données de la BD

- Soit en pl/pgSQL au serveur
- Soit en Java/JDBC au client

COMPARONS

Serveur Intelligent	Client Intelligent
+ cohérence gérée en un seul endroit	- chaque client doit gérer la cohérence
- code séparé entre deux outils, plus compliqué à comprendre	+ code en un seul langage
+ bénéficie des capacités de la BD (vues, ...)	- doit redévelopper ces capacités

Si on veut être capable de migrer facilement d'une BD à une autre, alors on place aussi peu de code que possible du côté serveur.

Si on veut profiter autant que possible des capacités de la BD, alors on place autant de code que possible du côté serveur.

POUR CE COURS

- **Apprentissage du SQL, donc :**
  - On favorise le serveur autant que possible.
- **Au niveau Java**
  - Uniquement des PreparedStatement
  - Uniquement des SELECT que sur une seule chose à la fois.
    - SELECT * /des champs FROM une table/une vue/une procédure WHERE des conditions
  - Attention aux performances: gestion connexion, PreparedStatement préparé une seule fois, …

SECURITE

POURQUOI PROTÉGER SES MDP ?

- **Empêcher d'utiliser illicitement votre application.**
- **Mais aussi souvent on utilise une combinaison email/password.**
  - Les utilisateurs recherchent la facilité : cette combinaison fonctionnera probablement ailleurs.
- **Il existe un réel marché pour vendre ces informations.**
- **Il existe un réel risque que les utilisateurs perdent de l'argent/du temps suite à une compromission de leur mdp.**

1ÈRE IDÉE

- **On ne va pas retenir les mots de passe en clair dans la base de données**
- **On va utiliser une fonction de hashage:**
  - A la création, on retient le hash du mdp
  - Pour authentifier, on hash le mdp proposé et on compare avec le contenu de la DB.

-> Solution imparfaite

  - Les mêmes mots de passe ont le même hash
    - possibilité de créer un dictionnaire inverse et d'obtenir tous les mots de passe

ETAT DE L'ART: SALAGE

- **On va empêcher le dictionnaire inverse en bidouillant le mdp encrypté**

  ⇒Utilisation de sel

  - Pour chaque mot de passe, on va générer un sel (par exemple « TYUIYTI »). Au lieu de hacher le mdp « toto », on va hacher le mot de passe «TYUIYTItoto».
- **Pour chaque mot de passe, la DB stocke**
  - Le sel
  - Le hachage du sel+mdp

POUR LE PROJET: BCRYPT

- `public static String gensalt();`
- `public static String hashpw(String password, String salt);`
  - Retourne un String contenant le sel concaténé avec le hachage du sel+mdp
  - Il y a donc un seul champ à stocker en DB
- `public static boolean checkpw(String plaintext, String hashed)`

EXEMPLE

```
public static void main(String[] args) {
   String sel=BCrypt.gensalt();
   // sel = $2a$10$yxXt/Hg.Bp38/U3M461Bfe
   String aStockerDansLaDB =
   BCrypt.hashpw("azerty123", sel);
   // aStockerDansLaDB =
   $2a$10$yxXt/Hg.Bp38/U3M461BfenWdYtQQ1PPRrX94yA
   PXB8cNk4oPvo7m
   System.out.println(BCrypt.checkpw("azerty12",
   aStockerDansLaDB));
   // imprime faux
}
```