

Accès à une BD depuis un programme JAVA

Java Database Connectivity

Erick STATTNER

Maitre de Conférences en Informatique

Université des Antilles

erick.stattner@univ-antilles.fr

www.erickstattner.com



Sommaire

1. Introduction JDBC
2. Accès à MySQL depuis une application JAVA
3. Pour aller plus loin
4. Injection SQL

I. Introduction JDBC

Depuis un programme JAVA

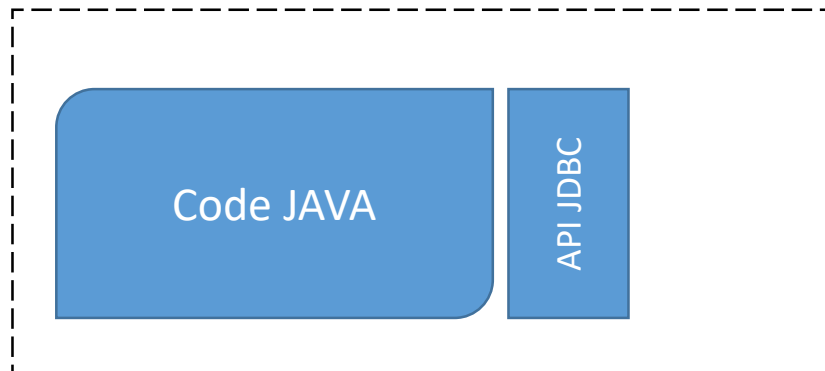
- JDBC: Java DataBase Connectivity
- API qui permet l'accès aux bases de données relationnelle dans un programme JAVA
 - Indépendant du type de base utilisée (MySQL, Postgres, Oracle, ...)
 - Permet de réaliser toutes les opérations SQL de type CRUD (Create, Read, Update, Delete)
 - S'appuie sur la notion de *driver* (ou *pilote*)
- JDBC fait partie du JDK depuis JAVA1.1
 - Toutes les classes sont regroupées dans les packages: *java.sql* et *javax.sql*

I. Introduction JDBC

Accès à la base de données

- API JDBC
- Ensemble de fonctions pour
 - Se connecter à la BD
 - Manipuler les données via SQL

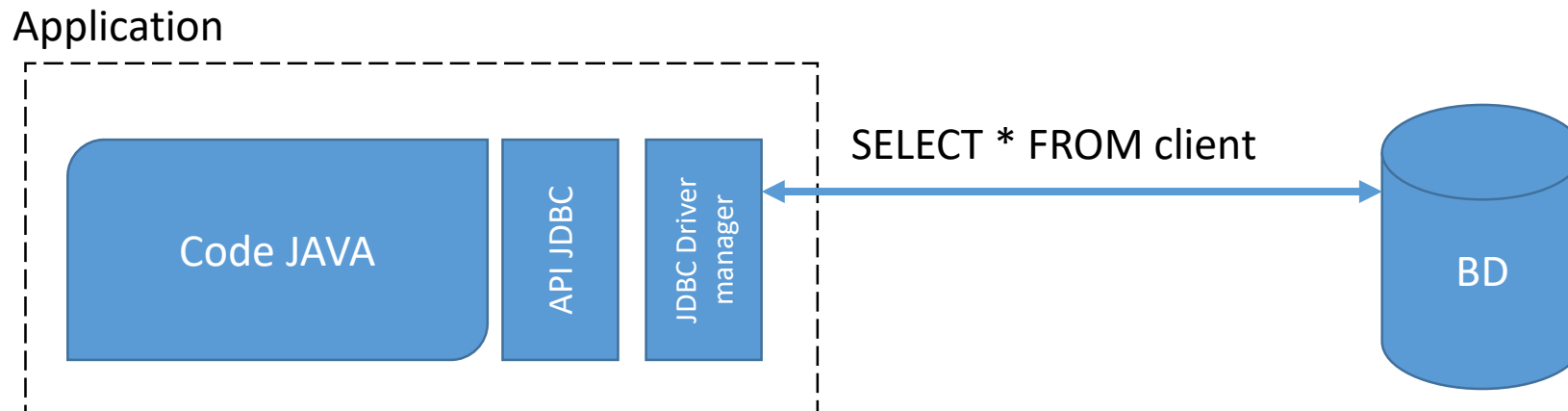
Application



I. Introduction JDBC

Indépendance de JDBC

- Garantie par la notion de *driver*
- Gere l'accès à un type particulier de base
 - Chaque type de base possède son driver dédié
- Convertit automatiquement les instructions JDBC pour la BD ciblée
- En cas d'évolution de la BD, seul le driver est a modifier !



I. Introduction JDBC

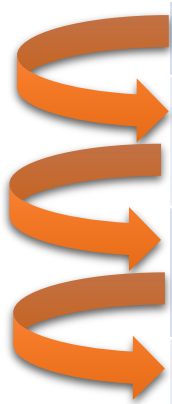
Etapes d'accès à une BD via JDBC

1. Définir le pilote
2. Se connecter à la base de données
3. Créer un *statement* (état) pour un ordre à effectuer
4. Traiter les résultats
5. Se déconnecter

I. Introduction JDBC

Pour réaliser ces étapes, JDBC propose 4 classes principales

- Dont la construction est séquentielle



Classe	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

II. Accès à MySQL depuis une application Java

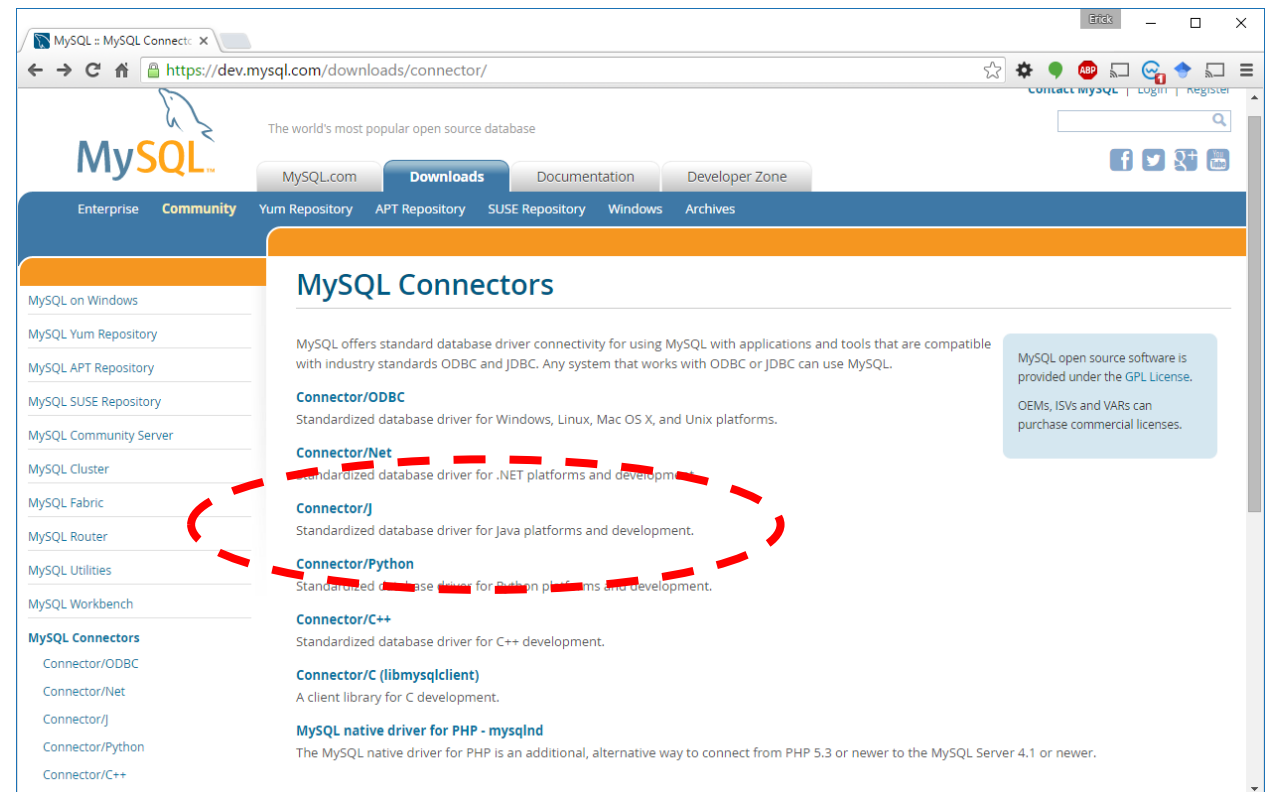
Etape 1) Définir le pilote

- Préciser le driver que l'on souhaite utiliser
- Chaque type de BD a son driver
- Il permet de communiquer avec le SGBD dans un langage qui lui est compréhensible
- Présenter sous forme de classes JAVA
- Des pilotes pour: MySQL, MariaDB, PostGres, Oracle, etc.

II. Accès à MySQL depuis une application Java

Etape 1) Exemple driver MySQL

- Le télécharger sur le site officiel
 - Distribué sous la forme de .jar



II. Accès à MySQL depuis une application Java

Etape 1) Charger le driver

- Permet de charger le pilote avec son nom de classe
- Sans créer explicitement d'objet de type driver
- Cette méthode peut lever une `ClassNotFoundException` en cas de problème

```
try{  
    Class.forName("org.gjt.mm.mysql.Driver");  
}  
catch(Exception e){  
    System.out.println("Impossible de charger le driver")  
}
```

II. Accès à MySQL depuis une application Java

Etape 1) Définir le pilote

- Le nom de classe à utiliser est fourni dans la documentation du driver

Exemple de définition de pilote

- Pour se connecter via ODBC
`Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");`
- Pour se connecter à une base Oracle
`Class.forName("oracle.jdbc.driver.OracleDriver");`
- Pour se connecter à une base PostGres
`Class.forName("postgresql.Driver");`
- Pour se connecter à une base MySQL
`Class.forName("org.gjt.mm.mysql.Driver");`
- Pour se connecter à une base dont le driver est passé en paramètre au lancement du programme
`Class.forName(args[0]);`

II. Accès à MySQL depuis une application Java

Etape 2) Se connecter à la BD

- Nécessite que le pilote soit correctement chargé
- Récupérer un objet *Connection* à l'aide de la méthode *getConnection* de la classe *DriverManager*
- La méthode *getConnection* permet de s'authentifier auprès de la BD
- 2 possibilités

Méthodes	Signature et description
static <u>Connection</u>	<u>getConnection</u> (<u>String</u> url) Attempts to establish a connection to the given database URL.
static <u>Connection</u>	<u>getConnection</u> (<u>String</u> url, <u>String</u> user, <u>String</u> password) Attempts to establish a connection to the given database URL.

II. Accès à MySQL depuis une application Java

Etape 2) Se connecter à la BD

- User

Nom de l'utilisateur autorisé à se connecter

- Password

Mot de passe de l'utilisateur

- Url

Identification de la base sur le SGBD

L'url peut encapsuler plusieurs paramètres

- Syntaxe générale:

`jdbc:<sous-protocol>:<compléments>`

- Sous-protocole: permet de distinguer le type de driver (indiqué dans la documentation)
 - Complément: peut inclure le nom de la base, le port, l'identifiant, le mot de passe, etc.

- Ex. `String url = "jdbc:odbc:maBD"`

II. Accès à MySQL depuis une application Java

Etape 2) Exemple de connexion à la BD

- A une base de données Films sur MySQL

```
String urlBD = "jdbc:mysql://localhost/films";
```

```
String user = "root"
```

```
String mdp = "root"
```

```
Connection con = DriverManager.getConnection(urlBD , user, mdp);
```

- Variante:

```
Connection con =
```

```
DriverManager.getConnection("jdbc:mysql://localhost/films?user=root&password=root");
```

- Lire la documentation du driver pour voir toutes les variantes possibles

<https://goo.gl/VMdksG>

- La méthode peut lever une `java.SQLException` en cas de problème

II. Accès à MySQL depuis une application Java

Etape 3) Créer un ordre à effectuer

- Une fois la connexion établie, on peut exécuter des ordres SQL
- Les requêtes sont exécutées à travers un objet `Statement` obtenu à partir de l'objet `Connection`
`Statement stmt = con.createStatement();`
- Le `statement` permet ensuite d'envoyer des requêtes SQL à la base à travers les méthodes

Méthodes	Instructions SQL concernées	Type de retour	Sémantique
<code>executeQuery(String requete)</code>	SELECT	ResultSet	Tableau de résultats
<code>executeUpdate(String requete)</code>	UPDATE, INSERT, DELETE	int	Nombre de lignes modifiées
<code>execute(String requete)</code>	AUTRES (create, drop, etc.)	boolean	faux si erreur

II. Accès à MySQL depuis une application Java

Etape 3) Créer un ordre à effectuer

- Exemple 1
 - `String req = " SELECT * FROM acteurs"`
`Statement stmt = con.createStatement();`
`ResultSet rslt = stmt.executeQuery(req);`
 - Un tableau contenant tous les acteurs est renvoyé dans la variable rslt de type ResultSet
- Exemple 2
 - `String req = « UPDATE acteurs SET nom='alain ' WHERE id=2"`
`Statement stmt = con.createStatement();`
`int nb = stmt.executeUpdate(req);`
`System.out.println(« ligne modifiée »+nb);`
 - Le nombre d'enregistrement mis à jour est retournée par la méthode executeUpdate

II. Accès à MySQL depuis une application Java

Etape 3) Créer un ordre à effectuer

- Si une erreur se produit lors de la requête, l'exception *SQLException* est levée
- **ATTENTION**
Si l'on utilise *executeQuery()* pour exécuter une requête SQL ne contenant pas d'ordre SELECT, une exception de type *SQLException* est levée.
MAIS la requête est tout de même effectuée !
- Il n'est pas nécessaire de définir un objet *Statement* pour chaque ordre SQL
 - On peut en définir un et le réutiliser

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- Lors d'une requête de type SELECT, les résultats sont renvoyés dans un objet de type `ResultSet`
- Tableau contenant plusieurs lignes et plusieurs colonnes
- Parcourir les lignes:
 - On accède à une ligne avec les méthodes `next`, `previous` ou `absolute`
 - `next()` se place sur la ligne suivante si elle existe
renvoie `true` si le déplacement réussi, `false` sinon
 - `previous()` se place sur la ligne précédente si elle existe
renvoie `true` si le déplacement réussi, `false` sinon
 - `Absolute(int index)` se place sur la ligne dont le numéro est `index` si elle existe
renvoie `true` si le déplacement réussi, `false` sinon
 - Le premier appel à `next()` positionne sur la première ligne
 - Les appels successifs à `next()` permettent de parcourir tous les résultats

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- Qqs méthodes de de la classe [ResultSet](#)

Méthode	Rôle
boolean isBeforeFirst()	Renvoyer un booléen qui indique si la position courante du curseur se trouve avant la première ligne
boolean isAfterLast()	Renvoyer un booléen qui indique si la position courante du curseur se trouve après la dernière ligne
boolean isFirst()	Renvoyer un booléen qui indique si le curseur est positionné sur la première ligne
boolean isLast()	Renvoyer un booléen qui indique si le curseur est positionné sur la dernière ligne
boolean first()	Déplacer le curseur sur la première ligne
boolean last()	Déplacer le curseur sur la dernière ligne
boolean absolute(int)	Déplacer le curseur sur la ligne dont le numéro est fourni en paramètre à partir du début s'il est positif et à partir de la fin s'il est négatif. 1 déplace sur la première ligne, -1 sur la dernière, -2 sur l'avant dernière ...
boolean relative(int)	Déplacer le curseur du nombre de lignes fourni en paramètre par rapport à la position courante du curseur. Le paramètre doit être négatif pour se déplacer vers le début et positif pour se déplacer vers la fin. Avant l'appel de cette méthode, il faut obligatoirement que le curseur soit positionné sur une ligne.
boolean previous()	Déplacer le curseur sur la ligne précédente. Le boolean indique si la première occurrence est dépassée.
int getRow()	Renvoyer le numéro de la ligne courante

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- Parcourir les colonnes
 - Accès aux colonnes avec une méthode `getXXX` qui prend en paramètre soit le nom de l'attribut, ou le rang dans la requête sous forme d'entier.
 - Le premier attribut à le rang 1

Méthode	Rôle
<code>getInt(int)</code>	retourne sous forme d'entier le contenu de la colonne dont le numéro est passé en paramètre.
<code>getInt(String)</code>	retourne sous forme d'entier le contenu de la colonne dont le nom est passé en paramètre.
<code>getFloat(int)</code>	retourne sous forme d'un nombre flottant le contenu de la colonne dont le numéro est passé en paramètre.
<code>getFloat(String)</code>	retourne sous forme d'un nombre flottant le contenu de la colonne dont le nom est passé en paramètre.
<code>getDate(int)</code>	retourne sous forme de date le contenu de la colonne dont le numéro est passé en paramètre.
<code>getDate(String)</code>	retourne sous forme de date le contenu de la colonne dont le nom est passé en paramètre.

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- Correspondances de types

Type SQL	Type à utiliser en JAVA
Char, Varchar	String
Bit	boolean
Tinyint	byte
Smallint	short
Integer	int
Float, Double	double
Date	java.sql.Date
Time	java.sql.Time
Timestamp	java.sql.Timestamp

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- Attention aux erreurs de conversion, qui lèvent des `SQLException`
 - Tous les attributs peuvent être récupérées à l'aide de `getString()`
- Les méthodes qui renvoient la colonne selon le rang sont utiles:
 - Pour les attributs calculés
Ex. `SELECT max(salaire) FROM Employe`
 - Les noms ne sont pas nécessairement connus
Ex. `SELECT * FROM clients`
 - Le rang correspond au rang dans le `SELECT`
- **En revanche, attention aux modifications de la base, qui peuvent entraîner des variations dans l'ordre des attributs !**

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- Exemple:

Acteurs(id, nom, prénom, nationalité)

```
Statement stmt = con.createStatement();
String req      = « SELECT id, nom, prenom FROM Acteurs WHERE nationalité='fr' »
ResultSet rslt  = stmt.executeQuery(req);
while( rslt.next () ){
    System.out.println(rslt.getInt(1) + " - " + rslt.getInt(2) + " - " + rslt.getString(3));
}
```

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- **Problème:** certaines valeurs peuvent être **NULL** dans une table
 - Les méthodes qui renvoient des objets, renvoient **NULL**
ex. `getString()`, `getDate()`, etc.
 - Les méthodes qui renvoient des types primitifs renvoient 0
ex. `getInt()`, `getDouble()`, etc.
- Comment les reconnaître en JAVA?
 - Utiliser la méthode `wasNull()` de l'objet *ResultSet*
 - Permet de tester si la dernière colonne lue avait une valeur **NULL** dans la base de données

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

- La classe *ResultSetMetaData* apporte des informations complémentaires sur le résultat d'une requête, c'est-à-dire sur un *ResultSet*
- La méthode *getMetaData()* appelé sur un objet *ResultSet* renvoie un objet de type *ResultSetMetaData*

```
ResultSet rslt           = stmt.executeQuery(req);  
ResultSetMetaData info   = rslt.getMetaData();
```

- Fonctions utiles

Méthode	Rôle
<code>int getColumnCount()</code>	Retourner le nombre de colonnes du <i>ResultSet</i>
<code>String getColumnName(int)</code>	Retourner le nom de la colonne dont le numéro est donné
<code>String getColumnLabel(int)</code>	Retourner le libellé de la colonne donnée
<code>boolean isCurrency(int)</code>	Retourner true si la colonne contient un nombre au format monétaire
<code>boolean isAutoIncrement(int)</code>	Retourner true si la colonne est auto incrémentée

II. Accès à MySQL depuis une application Java

Etape 4) Traiter les résultats

Exemple: Afficher toutes les colonnes de la table résultat

Acteurs(id, nom, prénom, nationalité)

```
Statement stmt          = con.createStatement();
String req              = « SELECT * FROM Acteurs WHERE nationalité='fr' »
ResultSet rslt          = stmt.executeQuery(req);
ResultSetMetaData info  = rslt.getMetaData()
while( rslt.next () ){
    for(int i = 1; i <= info.getColumnCount(); i++){
        System.out.println(rslt.getString(i);
    }
}
```

II. Accès à MySQL depuis une application Java

Etape 5) Se déconnecter

- Appeler la méthode `close()` sur
 - L'objet `ResultSet`
 - L'objet `Statement`
 - L'objet `Connection`

III. Pour aller plus loin

Gestion des erreurs

- Toutes les méthodes présentées sont susceptibles de lever une [SQLException](#)
- Exception générique qui survient lors
 - D'un problème d'accès à la BD
 - D'une requête
 - etc.
- Plusieurs spécialisations existent pour traiter finement le problème [SyntaxErrorException](#), [SQLDataException](#), [SQLClientInfoException](#), etc.
- Autrement, [SQLException](#) offre de nombreuses méthodes

Type de retour	Description
int	getErrorCode() Retrieves the vendor-specific exception code for this SQLException object.
SQLException	getNextException() Retrieves the exception chained to this SQLException object by <code>setNextException(SQLException ex)</code> .
String	getSQLState() Retrieves the SQLState for this SQLException object (standard X/Open and SQL99)

III. Pour aller plus loin

Pour aller plus loin JAVADOC: <https://goo.gl/XNX8sj>

- DriverManager
- DataBaseMetaData
- Connection
- Statement
- PreparedStatement
- ResultSet
- ResultSetMetaData
- SQLException

IV. Injection SQL



IV. Injection SQL

Alimenter une base de données avec les informations saisies par l'utilisateur

soulève un certain nombre de problèmes:

- Champs vides
- Données incohérentes / erreurs de frappe
- Utilisateurs malveillants

S'il ne sont pas traités affecte:

- La base de données
- Le programme

IV. Injection SQL

Champs vides

- Forcer l'utilisateur à saisir les données attendus
- Ex. Le champ nom est obligatoire pour continuer

Données incohérentes / erreurs de frappe

- Vérifier les données avant l'enregistrement en base
 - Vérification valeur numérique
 - Utiliser des expressions régulières
 - Faire appel à des services WEB
- Ex. age > 0 et adrMail est correcte

Conseils

- Au moment de redemander la saisie à l'utilisateur, prenez soin d'y replacer les valeurs déjà saisie
- Signaler explicitement les problèmes rencontrés

IV. Injection SQL

Injection SQL

- Type d'attaque qui cible les applications interagissant avec une BD
- Vise à insérer du code SQL lors des interactions avec la BD dans le but de modifier le comportement des requêtes
 - Enchaîner plusieurs requêtes
 - Ignorer une partie de la requête
 - Modifier son comportement
- Type d'attaque le plus répandu et facile à mettre en oeuvre

IV. Injection SQL

Conséquence

- Contournement formulaire d'authentification
- Vol d'informations dans la base ou Dump de la totalité de la BD
- Compromettre l'intégrité de la base
- Exécution de code malveillant
- Planter l'application
- Modifier l'affichage

IV. Injection SQL

Exemple 1: Perturber bon fonctionnement

- Par exemple, dans une page WEB, la saisie de balises HTML peut conduire à un affichage erroné
 - Exemple: un utilisateur s'inscrit sur un site en remplissant les champs:

Nom :	<input type="text" value="<h1>TOTO</h1>"/>
Prenom :	<input type="text" value="<h1>Alain</h1>"/>

- Si les données ne sont pas traitées, les champs sont enregistrés tels quels dans la BD Et interprétés par le navigateur lors de la lecture.
 - Conséquence: l'affichage des membres est perturbé
Le nom et le prénom de cet utilisateurs affichés plus gros
- La saisie d'un script pourrait permettre d'exécuter du code malveillant
[https://fr.wikipedia.org/wiki/Injection de code dans les applications web](https://fr.wikipedia.org/wiki/Injection_de_code_dans_les_applications_web)

IV. Injection SQL

Exemple 2: Contourner authentication

- Considérons une requête d'authentification simple

String login = champLogin.getText()

String mdp = champMdp.getText()

String req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

IV. Injection SQL

Exemple 2: Contourner authentication

- Considérons une requête d'authentification simple

Login:	Root'; --
Mdp:	Lol je t'ai eu

- Que devient la requête précédente

String login = champLogin.getText()

String mdp = champMdp.getText()

String req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

IV. Injection SQL

Exemple 2: Contourner authentication

- Considérons une requête d'authentification simple

Login:	Root'; --
Mdp:	Lol je t'ai eu

- Que devient la requête précédente

String login = champLogin.getText()

String mdp = champMdp.getText()

String req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

- **Probleme:**

L'utilisateur root n'existe peut etre pas !

IV. Injection SQL

Exemple 2: Contourner authentication

- Que devient la requête

String login = champLogin.getText()

String mdp = champMdp.getText()

String req = "SELECT id, login, mdp FROM membre
WHERE login = ' " + login + " ' AND mdp = ' " + mdp + " ' ";

- Si l'utilisateur saisit:

Login: ' OR 1=1; --

Mdp: Je t'ai encore eu !

Login: toto

Mdp: ' or 1=1; --

Login: ' OR 1=1

Mdp: ' OR 1=1; --

IV. Injection SQL

Exemple 3: Modifier le comportement d'une requête

- `String produit` `= champProduit.getText()`
`String req` `= "SELECT numero, libelle, description FROM produits`
`WHERE libelle like ' " + produit + ' ";`

Produit cherché:

IV. Injection SQL

Exemple 3: Modifier le comportement d'une requête

- `String produit` `= champProduit.getText()`
`String req` `= "SELECT numero, libelle, description FROM produits`
`WHERE libelle like ' " + produit + ' ";`

- Que devient la requête précédente après la saisie de

Produit cherché:

```
' UNION SELECT id as numero, login as libelle, mdp as  
description FROM membres; --
```

- **Beaucoup de possibilité:**
 - Connaitre la structure des tables
 - Exécuter des scripts
 - etc.
- **Des exemples d'injection SQL avancés:**
<http://www.bases-hacking.org/injections-sql-avancees.html>
<http://php.net/manual/fr/security.database.sql-injection.php>

IV. Injection SQL



IV. Injection SQL

Solution générale:

- Conserver les variables d'authentification dans des fichiers séparés et protégés
- Vérifier le format des données saisies et notamment la présence de caractères spéciaux
- Limiter la taille des champs
- Ne pas afficher dans les messages d'erreurs
 - Une partie de la requête
 - Des informations sur la structure des bases de données
- Ne pas conserver les utilisateurs par défaut
- Restreindre au minimum les privilèges des comptes utilisés
- Ne pas stocker directement les mot de passe dans la base, mais un hash

IV. Injection SQL

Dans un programme JAVA

- Vérifier la présence de caractères spéciaux dans les chaines
- Ne pas utiliser la concaténation pour construire la requête
- Recours aux requêtes paramétrées

Preventing SQL Injection in Java

[https://www.owasp.org/index.php/Preventing SQL Injection in Java](https://www.owasp.org/index.php/Preventing_SQL_Injection_in_Java)