

# 9. Accès aux bases de données

## 9.1. Injection SQL

## Comment passer outre une étape d'identification?

Soit le formulaire d'identification:

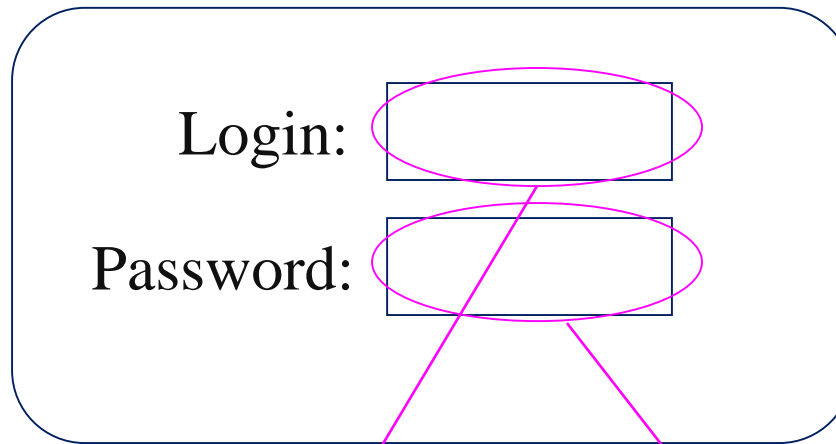


Diagram of a login form with two input fields:

- Login: [ ]
- Password: [ ]

The input fields are highlighted with pink ovals. Arrows point from these ovals to the placeholders in the SQL query below.

Soit la requête de vérification:

```
Select * from UserTable  
where login = ' ... ' and password = ' ... ' ;
```

Login: ' or '1' = '1

Password: ' or '1' = '1

# Select \* from UserTable

where login = ' ' or '1' = '1 ' and password = ' ' (or '1' = '1') ;

3

Exemple d'injection SQL (en mysql):

Login:	<input type="text" value="' /*"/>
Password:	<input type="text" value="' */ or '1' = '1"/>

Résultat de la requête de vérification:

Select \* from UserTable

where login = ' ' /\* and password = ' ' \*/ or '1' = '1'

Vrai

## Accéder en écriture à une base de données par injection SQL

### Exemple

Login:	<input type="text" value="'; insert into .../*"/>
Password:	<input type="text"/>

Résultat de la requête de vérification:

Select \* from UserTable

where login = ' '; insert into ... /\*

Insertion d'une ligne dans une table

## Accéder en lecture à une base de données par injection SQL

### Exemple

Critère de recherche: ' or '1' = '1' union select ... /\*

Résultat:

Résultat de la requête de vérification:

Select \* from MyTable

where critereRecherche = ' ' or '1' = '1' union select ... /\*

Récupérer des lignes dans une table

## Comment éviter les injections SQL?

Empêcher la mauvaise interprétation des '

Exemples:

- Utiliser des fonctions qui encapsulent les ' (ex: addslashes( ))
- Utiliser la classe PreparedStatement <> Statement

# 9. Accès aux bases de données

- 9.1. Injection SQL

- 9.2. Accès aux bases de données à partir d'un programme Java



couche **Data Access**

## **import**

```
import java.util.* ;  
import java.sql.*;
```

## ① Connexion à la BD

### Créer (et ouvrir) une connexion

Cf labo

## Principe

*Déléguer au serveur d'applications la gestion des occurrences de connexion*

⇒ Création d'un **pool de connexions** géré par GlassFish:

*lorsqu'un accès à la BD est nécessaire, on demande une occurrence de connexion.*

Récupération de la ressource (ex: une connexion BD) dans le code Java via

**JNDI**

*Java Naming and Directory Interface*

service de nommage et de répertoires :

lien entre un nom et une information



permet de retrouver une ressource à partir de son nom

## Créer un pool de connexions et une source de données

- Sélectionner un projet

- Créer un nouveau fichier :

Category : *GlassFish*

File Type : *JDBC Resource*

- General Attributes :

Create New JDBC Connection Pool :

**JNDI Name** : *jdbc/myDerby* (*par exemple*)

Object Type : *user*

Enabled : *true*

- Choose Database Connection :

JDBC Connection Pool Name : *myDerbyPool* (*par exemple*)

Extract from Existing Connection : *jdbc:derby://localhost: 1527/...*

*(en choisir une existante)*

- Add Connection Pool Properties :

Datasource Classname : *org.apache.derby.jdbc.ClientDataSource*

Resource Type : *javax.sql.ConnectionPoolDataSource*

## Utilisation du pool de connexions dans les classes Java via JNDI

```
import java.sql.Connection;
```

```
import javax.sql.DataSource;
```

```
import javax.naming.InitialContext;
```

```
import javax.naming.Context;
```

```
try
```

```
    { Context ctx = new InitialContext();
```

```
        DataSource source = (DataSource) ctx.lookup("jdbc/myDerby");
```

```
        Connection connexion = source.getConnection();
```

```
    }
```

```
catch (Exception ex)
```

```
    {...}
```

## Fermer une connexion

```
Connection connexion = ... ;  
connexion.close( );
```

↳ *throws SQLException*

Soit la table Book dans la base de données

Book
<u>Isbn</u>
NbPages <u>[0..1]</u>
Title <u>[0..1]</u>
EditionDate <u>[0..1]</u>

chaîne de caractères

entier

chaîne de caractères

date



## *Dans la couche Model*

class **Book**

{ private String isbn;

private Integer nbPages;

private ~~int~~ nbPages;

private String title;

private GregorianCalendar editionDate;

...

}



Prévoir des variables d'instance de type référence  
et non de type primitif,  
afin de gérer les attributs/colonnes facultatifs

## ② Exécuter une instruction SQL autre qu'une requête

Connection **connexion** = ... ;

String *instructionSQL* = "insert ...values ( ? , ? , ? ... )";  
*/\* contient l'instruction SQL d'insertion à exécuter \*/*

**PreparedStatement** **prepStat** = **connexion**.prepareStatement(*instructionSQL* );

**prepStat.setString(1, "blabla");**      *// si colonne1 est de type chaîne de caractères*

**prepStat.setInt(2, 100);**      *// si colonne2 est de type entier*

*java.sql.Date sqlDate* ← ...;

**prepStat.setDate(3, sqlDate);**      *// si colonne3 est de type date*

...

```
int nbIns = prepStat.executeUpdate( );
```



↳ throws *SQLException*

*retourne le nombre de lignes modifiées*

*NB.*

*L'utilisation de la classe **PreparedStatement** empêche les injections SQL*



*La classe **Statement***

Attention aux **valeurs inconnues** éventuelles (cfr valeur **null** en SQL) lors d'insertion dans les **colonnes facultatives** !

## Version 1

### ① **insert** pour les colonnes obligatoires

```
Book book = ... ;
```

```
// insertion dans la table d'une ligne avec des valeurs dans toutes les colonnes
```

```
// obligatoires
```

```
String instructionSQL = "insert into Book (Isbn) values (?);"
```

```
PreparedStatement prepStat = connexion.prepareStatement(instructionSQL);
```

```
prepStat.setString(1, book.getIsbn( ));
```

```
prepStat.executeUpdate( );
```

## ② **update** pour les colonnes facultatives

...

// valeur à insérer dans une colonne **facultative** de type numérique

```
if (book.getNbPages( ) != null)  
{ instructionSQL =  
    " update Book set NbPages = ? where Isbn = ' " + book.getIsbn() + " ' ";  
    prepStat = connexion.prepareStatement(instructionSQL);  
    prepStat.setInt(1, book.getNbPages( ));  
    prepStat.executeUpdate( );  
}
```

// valeur à insérer dans une colonne **facultative** de type **date**

```
if (book.getEditionDate() != null)
{
    instructionSQL =
        " update Book set EditionDate = ? where Isbn = '" + book.getIsbn() + "' ";
    prepStat = connexion.prepareStatement(instructionSQL);
    prepStat.setDate(1, new java.sql.Date(book.getEditionDate().getTimeInMillis()));
    prepStat.executeUpdate();
}
```

*De type **GregorianCalendar***

Insertion de **valeurs inconnues (null en SQL)** dans les **colonnes facultatives !**

## Version 2

① **insert** de valeurs null (inconnues) dans les colonnes facultatives

*Exemple: insertion d'un livre avec un nombre de pages, un titre et une date d'édition inconnus*

```
Book book = ... ;
```

```
String instructionSQL = "insert into Book (Isbn, NbPages, Title, EditionDate)  
values (?, ?, ?, ?)";
```

```
PreparedStatement prepStat = connexion.prepareStatement(instructionSQL);
```

```
prepStat.setString(1, book.getIsbn( ));
```

```
prepStat.setNull(2, Types.INTEGER);
```

```
prepStat.setNull(3, Types.VARCHAR);
```

```
prepStat.setNull(4, Types.TIMESTAMP);
```

```
prepStat.executeUpdate( );
```

### ③ Confirmer (cfr notion de transaction SQL : commit)

par défaut : connexion.**setAutoCommit**(*true*) ; ↗

*modifications permanentes dans la BD (automatiquement)*

si connexion.**setAutoCommit**(*false*) ;

⇒ `prepStat.executeUpdate(instructionSQL);`

...

connexion.**commit**( );

⇒ *modifications permanentes dans la BD*

↗ *throws **SQLException***



## ④ Exécuter une requête SQL et créer une collection d'objets

*requête SQL = accès en lecture aux données*

### Exécuter une requête SQL

Connection `connexion` = ... ;

String *requeteSQL* =

“select ...where colonne1 = ? and colonne2 = ? and colonne3 = ? ...”;

*/\* contient la requête SQL à exécuter\*/*

**PreparedStatement** `prepStat` = `connexion.prepareStatement(requeteSQL)` ;

`prepStat.setString(1, “blabla”);` *// si colonne1 est de type chaîne de caractères*

`prepStat.setInt(2, 100);` *// si colonne2 est de type entier*

`java.sql.Date sqlDate = ...;`

`prepStat.setDate(3, sqlDate);` *// si colonne3 est de type date*

...

**ResultSet** donnees = **prepStat.executeQuery( );**

↳ throws *SQLException*

*//un ResultSet contient les lignes de résultat de la requête*

**ResultSetMetaData** meta = donnees.**getMetaData( );**

↳ throws *SQLException*

*// un ResultSetMetaData contient des meta données sur le résultat de la requête*

Meta données =

informations/renseignements sur les données

Ex:

ResultSetMetaData **meta** = ... ;

**meta**.getColumnCount( ) // le nombre de colonnes

**meta**.getColumnName(i) // le nom de la colonne i

**meta**.getColumnType(i) // le type de la colonne i

## Créer une collection d'objets à partir d'un ResultSet

*Boucler sur les lignes de résultats de la requête (ResultSet):*

```
ResultSet donnees = ... ;
```

```
while (donnees.next())
```

```
    { recupérer la ligne courante du ResultSet }
```


## *Récupérer les valeurs de la ligne courante du ResultSet:*

```
ResultSet donnees = ... ;
```

```
while (donnees.next( ))
```

```
{ ...
```

```
  donnees .
```

 { getString(...)  
 getInt(...)  
 getDate(...)  
 ...

*Indice de la colonne ou  
nom de la colonne*

```
  ...
```

```
}
```

## Type de colonne // classes en Java

Type de colonne	Constante	Valeur	get	⇒	type de retour
Texte	Types. <b>VARCHAR</b>	12	<b>getString()</b>	⇒	<b><u>S</u>tring</b>
Octet	Types. <b>TINYINT</b>	-6	<b>getInt()</b>	⇒	int
Entier	Types. <b>SMALLINT</b>	5	<b>getInt()</b>	⇒	int
Entier long	Types. <b>INTEGER</b>	4	<b>getInt()</b>	⇒	int
Réel simple	Types. <b>REAL</b>	7	<b>getDouble()</b>	⇒	<b><u>d</u>ouble</b>
Réel double	Types. <b>DOUBLE</b>	8	<b>getDouble()</b>	⇒	<b><u>d</u>ouble</b>
Décimal	Types. <b>NUMERIC</b>	2	<b>getDouble()</b>	⇒	<b><u>d</u>ouble</b>
Oui/Non	Types. <b>BIT</b>	-7	<b>getBoolean()</b>	⇒	<b><u>b</u>oolean</b>
Date/Heure	Types. <b>TIMESTAMP</b>	93	<b>getDate()</b>	⇒	java.sql. <b><u>D</u>ate</b>
Mémo	Types. <b>LONGVARCHAR</b>	-1	<b>getString()</b>	⇒	<b><u>S</u>tring</b>

## Conversion `GregorianCalendar` ⇔ `java.sql.Date` :

`GregorianCalendar` `gregC` = ... ;

`java.sql.Date` `sqlD` = new `java.sql.Date`(`gregC.getTimeInMillis()`);

## Conversion `java.sql.Date` ⇔ `GregorianCalendar` :

`java.sql.Date` `sqlD` = ... ;

`GregorianCalendar` `gregC` = new `GregorianCalendar`();

`gregC.setTime`(`sqlD`);

ResultSet **donnees** = ... ; // contient le résultat de la requête “select \* from livre;”

Book book; int pages; String title; java.sql.Date dateEd;

**ArrayList<Book> allBooks** = new ArrayList <Book>();

while (**donnees.next()**)

{ **book** = new Book (**donnees.getString**(“Isbn”));

Colonne obligatoire

pages = **donnees.getInt**(“NbPages”);

if (**donnees.isNull()** == false)

{ book.setNbPages(pages);}

Colonnes facultatives

title = **donnees.getString**(“Title”);

if (**donnees.isNull()** == false)

{ book.setTitle(title);}



...

*// récupération d'un objet GregorianCalendar à partir d'une colonne de type date en BD*

```
dateEd = donnees.getDate("EditionDate") ;
```

*De type java.sql.Date*

```
if (donnees.wasNull( ) == false)
```

```
    { GregorianCalendar gc = new GregorianCalendar();
```

```
      gc.setTime(dateEd);
```

```
      book.setEditionDate(gc);
```

```
    }
```

```
allBook.add(book);
```

```
}
```

couche **User Interface**

## ⑤ Créer un modèle de données (TableModel) à partir d'une collection d'objets

*Nécessaire pour pouvoir afficher le résultat d'une requête dans un composant Swing (JTable)*

```
import javax.swing.table.*;  
import java.util.*;           classe abstraite  
                               ↑
```

Créer une sous-classe de **AbstractTableModel**  
qui contient 2 variables d'instance:

- La liste des noms de colonnes
- La liste d'objets correspondant aux données

Attention:

Absolument redéfinir les méthodes:

**getColumnCount( )**

**getRowCount( )**

**getColumnName(int col)**

**getValueAt(int row, int col)**

**getColumnClass(int col)**

*Car appelées implicitement par Java pour afficher correctement les données dans une JTable*

```

public class AllBooksModel extends AbstractTableModel

{ private ArrayList<String> columnNames = new ArrayList<String>( );
  private ArrayList<Book> contents = new ArrayList<Book>( );

  public AllBooksModel (ArrayList<Book> books)
      { contents = books;
        columnNames.add("Isbn");
        columnNames.add("Titre");
        columnNames.add("Date d'édition");
        columnNames.add("Nombre de pages");
      }

  public int getColumnCount( ) {return columnNames.size( );}

  public int getRowCount( ) {return contents.size( );}

  public String getColumnName(int col) {return columnNames.get(col); }

```

```
public Object getValueAt(int row, int col)
{
    Book book = contents.get(row);
    switch(col)
    {
        case 0: return book.getIsbn();
        case 1: return book.getTitle();
        case 2: {if (book.getEditionDate() != null)
                    return book.getEditionDate().getTime();
                else
                    return null;
                }
        case 3: return book.getNbPages();
        default: return null;
    }
}
```

```
public Class getColumnClass (int col)
{
    Class c;
    switch (col)
    {
        case 0: c = String.class;
                break;
        case 1: c = String.class;
                break;
        case 2: c = Date.class;
                break;
        case 3: c = Integer.class;
                break;
        default: c = String.class;
    }
    return c;
}
```

## User Interface

MainJFrame

NewBookPanel

AllBooksPanel

**AllBooksModel**

## Model

## Controller

ApplicationController

Book

## Business Logic

BookManager

AddBookException

## Data Access

BookDBAccess

AllBooksException



## ⑥ Afficher un modèle de données via une JTable

*Créer une JTable:*

```
AllBooksModel model = ... ;
```

```
JTable table = new JTable (model) ;
```

## Afficher une JTable:

JTable **table** = ...

```
// TableColumn col = table.getColumnModel( ).getColumn(1);
```

```
// col.setPreferredWidth(200);
```

*/\* imposer la largeur d'une colonne \*/*

```
// table.setAutoResizeMode(JTable.AUTO_RESIZE_OFF);
```

```
JScrollPane defilant = new JScrollPane (table) ;
```

↳ à ajouter au container

## ⑦ Récupérer l'indice d'une ligne sélectionnée par l'utilisateur dans une JTable

### 1. A la création de la JTable

```
JTable table = ...;
```

```
table.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
```

```
ListSelectionModel listSelect = table.getSelectionModel( );
```

```
...
```

### 2. Dans la gestion d'événement (ex: si clic sur bouton)

```
int indiceLigneSelectionnee = listSelect.getMinSelectionIndex( );
```

```
...    // récupération des valeurs de la ligne sélectionnée
```