

# JAVA

---

JDBC

# Package JDBC

---

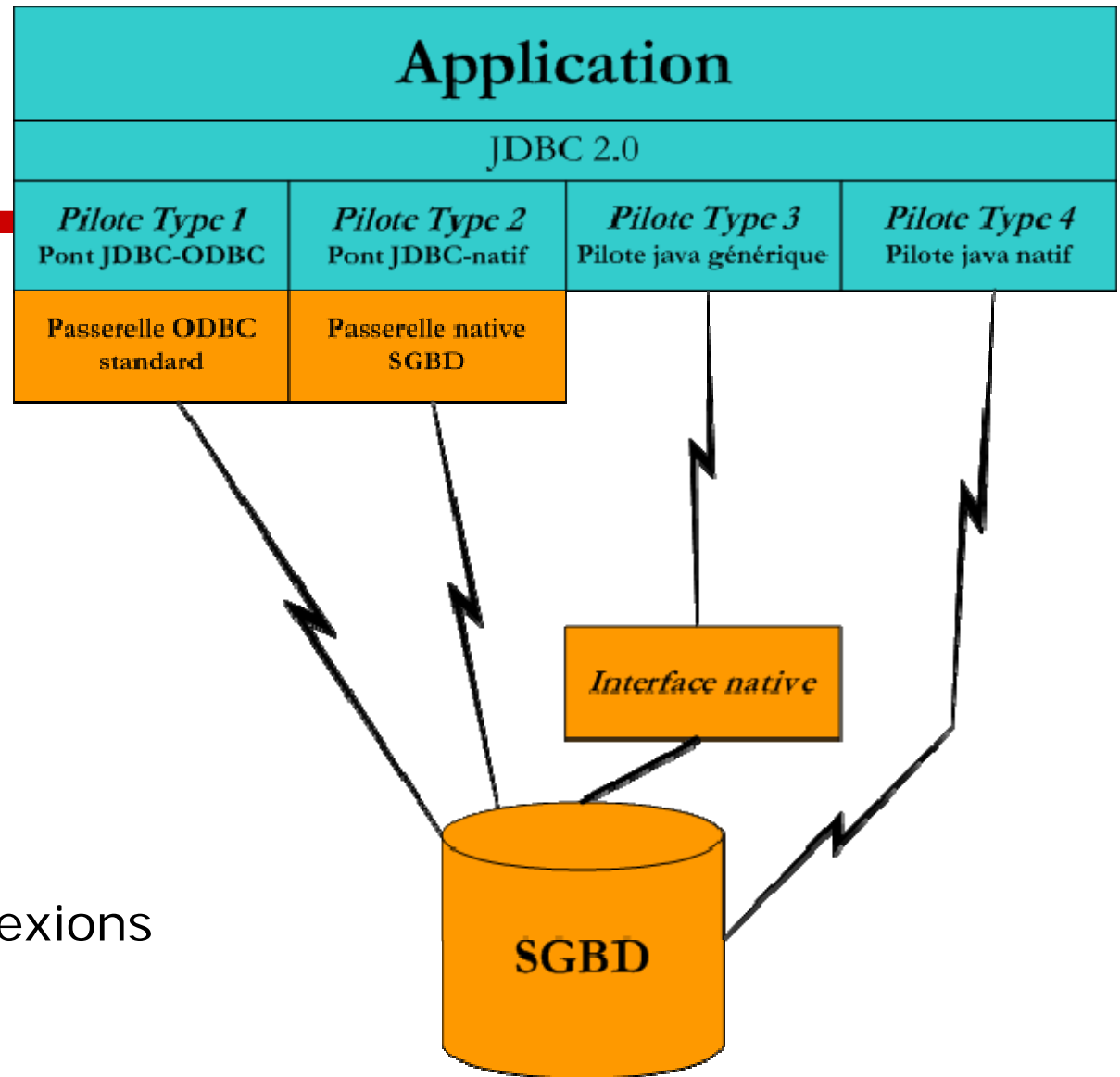
- `import java.sql.*;`
  - Modèles d'interconnexion
  - Opérations de base
  - Connexion
  - Interrogation d'une base
  - Mise à jour d'une base
  - Les ensembles résultats
    - Parcours d'un ensemble résultat
    - Modification via un ensemble résultat
  - Les requêtes paramétrées
  - Transactions
-

# Le JDBC

---

- ❑ Accès homogène aux SGBDR
  - ❑ Abstraction des SGBDR cibles
  - ❑ Supporte SQL (évidemment!)
  - ❑ Simple à mettre en oeuvre
-

# Le JDBC



□ Modèles d'interconnexions

# Le JDBC

---

- Manipuler une BD se résume aux opérations suivantes :
    - Charger le *driver*
    - Connecter l'application à la BD
    - Créer une requête
    - Exécuter la requête
    - Lire et traiter les résultats (données ou code d'erreur)
-

# Le JDBC

---

□ Charger le driver :

```
Class.forName("Nom_de_classe_driver");
```

- Charge la classe correspondant au driver souhaité
- Le driver est enregistré dans la classe **DriverManager**
- L'application peut désormais se connecter via ce driver
- Il peut y avoir plusieurs drivers par application
- Exemples :

```
Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
Class.forName("org.postgresql.Driver");  
Class.forName("oracle.jdbc.driver.OracleDriver");
```

---

# Le JDBC

---

□ Connecter l'application à la BD :

Connection BD =

```
DriverManager.getConnection(URL, login, passwd);
```

BD : nom de l'objet dans l'application

URL : chaîne de caractères composée de la manière suivante: "**jdbc:sousprotocole:NomBD**"

Login : chaîne de car. contenant l'identifiant de l'utilisateur

Passwd : chaîne de car. contenant son mot de passe

---

# Le JDBC

---

- Exemples correspondants aux drivers précédents :

```
Connection base = DriverManager.getConnection( "jdbc:odbc:ComptesTitre",  
                                              "admin",  
                                              "h5Fz9X");
```

```
Connection base = DriverManager.getConnection( "jdbc:postgresql:ComptesTitre",  
                                              "admin",  
                                              "h5Fz9X");
```

```
Connection base = DriverManager.getConnection( "jdbc:oracle:thin:@194.57.140.75:1521:ComptesTitre",  
                                              "admin",  
                                              "h5Fz9X");
```

- Déconnexion de la base de données : **base.close();**
-



# Le JDBC

---

## □ Les requêtes simples

- Requête = **Statement**

- C'est l'objet **connection** qui en assure la création par **createStatement()**

- Exemple :

```
/**  
 * création d'une requête simple.  
 */  
Statement requete = base.createStatement();
```

---

# Le JDBC

---

## □ Requête de sélection (SELECT)

```
ResultSet Res = requete.executeQuery("SELECT...");
```

- **Res** est un objet contenant les données sélectionnées
- **Res** ne peut être parcouru que vers l'avant

## □ Exemple:

```
ResultSet Res = requete.executeQuery(  
    "SELECT * FROM client WHERE nom_client='SKYWALKER'  
");
```

---

# Le JDBC

---

- Navigation simple dans un **ResultSet**
    - Un élément (row) est composé des champs décrits dans la clause « SELECT »
    - Au départ, le « curseur » est positionné avant le premier élément
    - **boolean next()**: avance d'un élément. Renvoie **false** s'il n'y en a plus.
    - On ne peut ni reculer ni revenir au début
    - On ne peut que consulter les éléments
-

# Le JDBC

---

## □ Navigation simple dans un **ResultSet**

- **xxx** `getXXX(int i)` : renvoie la valeur du champ `i` ( $i \geq 1$ ) de l'élément courant. Le champ doit être de type `XXX`. On ne peut lire qu'une seule fois le champs!!!
- **xxx** `getXXX(String N)` : renvoie la valeur du champ nommé « `N` ». Le champ doit être de type `XXX`. Même remarque.

## □ Exemple :

```
while (Res.next())
    System.out.println( Res.getInt(1)+" "+
                        Res.getString(2)+" "+
                        Res.getString(3));
```

# Le JDBC

---

- Requête de mise à jour (INSERT, DELETE, UPDATE)

```
int n = requete.executeUpdate("...code SQL...");
```

**n** est le nombre d'enregistrements mis à jour

---

# Le JDBC

---

□ Exemple d'insertion :

```
System.out.print("Nom      : ");
String N = in.readLine();
System.out.print("Adresse  : ");
String A = in.readLine();
requete.executeUpdate(
    "INSERT INTO client (nom_client, adr_client)" +
    "VALUES ('"+N+"', '"+A+"')"
);
// INSERT INTO client (nom_client, adr_client)
// VALUES ('SKYWALKER', 'Tatouine')
```

---

# Le JDBC

---

## □ Les ensembles résultats avancés

- `ResultSet` par défaut trop restrictifs d'où :

```
statement.createStatement(int TypeNavigation, int TypeAcces)
```

TypeNavigation : `ResultSet.TYPE_FORWARD_ONLY`  
`ResultSet.TYPE_SCROLL_INSENSITIVE`  
`ResultSet.TYPE_SCROLL_SENSITIVE`

TypeAcces : `ResultSet.CONCUR_READ_ONLY`  
`ResultSet.CONCUR_UPDATABLE`

---

# Le JDBC

---

## □ Navigation bi-directionnelle:

`beforeFirst()`

`afterLast()`

`first()`

`last()`

`next()`

`previous()`

`absolute(int)`

`relative(int)`

Etc.

---



# Le JDBC

---

## □ Modification des éléments (dans la BD)

- Modifier les champs avec

```
updateXXX(int numcol,XXX newval)
```

```
updateXXX(String nomcol, XXX newval)
```

- Répercuter les modifications dans la base

```
updateRow()
```

- Annuler les modifications dans la base

```
cancelRowUpdates()
```

- Ajouter une nouvelle ligne

```
moveToInsertRow() // crée une ligne vide
```

```
insertRow() // ajoute la ligne
```

Attention : pas forcément supporté par le driver

---

# Le JDBC

---

## □ Exemple :

```
Statement requeteCpt = base.createStatement(  
    ResultSet.TYPE_SCROLL_SENSITIVE,  
    ResultSet.CONCUR_UPDATABLE  
);
```

```
ResultSet cpt = requeteCpt.executeQuery(  
    "SELECT num_cpt, solde_cpt "+  
    "FROM Compte,Client "+  
    "WHERE Client.nom_client='LANUEL' AND "+  
    "Compte.num_client=Client.num_client"  
);
```

```
while (cpt.next())  
{  
    double d = cpt.getDouble(2)+1000;  
    cpt.updateDouble(2,d);  
    cpt.updateRow();  
}
```

---

# Le JDBC

---

## □ SQL Injection

### ■ Exemple :

```
string N = in.readLine();
```

```
string R = "SELECT * "+  
"FROM client "+  
"WHERE nom_client='"+N+"'";
```

```
ResultSet Res = requete.executeUpdateQuery(R);
```

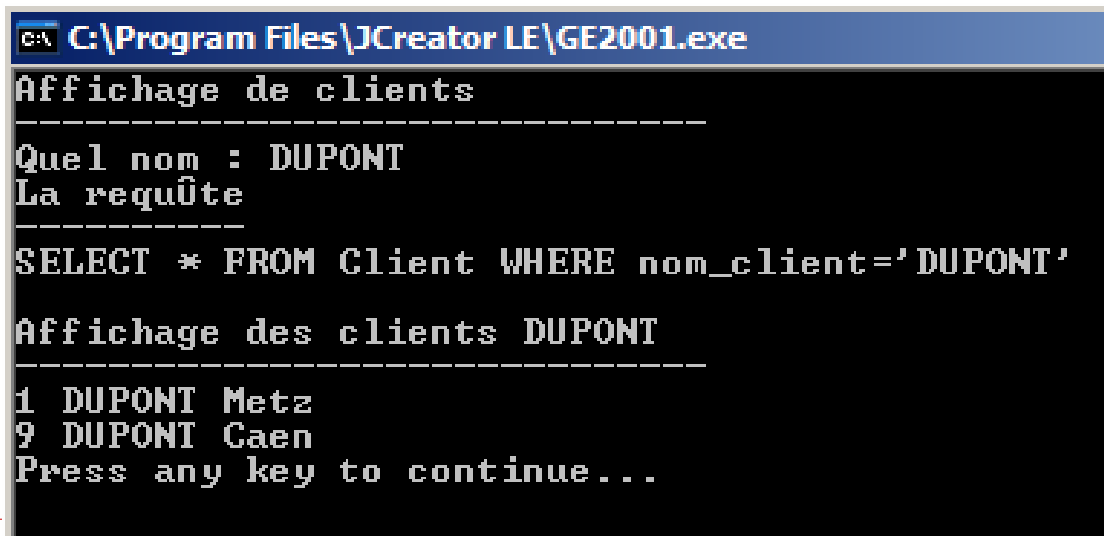
- Si l'utilisateur saisit **DUPONT** la requête construite est :
-

# Le JDBC

---

```
SELECT *  
FROM Client  
WHERE nom_client='DUPONT'
```

- Le résultat est :



```
C:\Program Files\JCreator LE\GE2001.exe  
Affichage de clients  
-----  
Quel nom : DUPONT  
La requête  
-----  
SELECT * FROM Client WHERE nom_client='DUPONT'  
Affichage des clients DUPONT  
-----  
1 DUPONT Metz  
9 DUPONT Caen  
Press any key to continue...
```

# Le JDBC

---

- Si l'utilisateur saisit ` OR `1'='1`
- La requête devient :

```
SELECT *  
FROM Client  
WHERE nom_client='' OR '1'='1'
```

- Le résultat est :

```
SELECT * FROM Client WHERE nom_client='' OR '1'='1'  
Affichage des clients ' OR '1'='1'  
-----  
1 DUPONT Metz  
9 DUPONT Caen  
10 DURAND Limoges  
11 TAPIS Marseille  
12 SKYWALKER Tatouine  
Press any key to continue..._
```

# Le JDBC

---

## □ D'où :

- Sécuriser tous les composants de la BD
  - Utiliser le moins possible les requêtes avec substitution de texte
  - Utiliser le plus possible des requêtes paramétrées
-

# Le JDBC

---

- Principe des requêtes paramétrées :
    - « compilation » de la requête
    - Les valeurs sont données séparément et utilisées au moment de l'exécution
    - Plus de substitution de texte
-

# Le JDBC

---

- ❑ Formuler la requête où les paramètres sont symboliser par '?'
- ❑ Exemple:

```
/**
 * Formulation de la requête avec paramètre
 */
String R = "SELECT * "+
           "FROM Client "+
           "WHERE nom_client=?";
```

- ❑ Créer une requête paramétrée :
-



# Le JDBC

---

- ❑ Créer la requête paramétrée :

```
/**  
 * Création d'une requête paramétrée.  
 */  
PreparedStatement requete = base.prepareStatement(R);
```

- ❑ Fixer les valeurs des paramètres :

- Accesseurs de la classe PreparedStatement :

**setXXX(num\_p, val)**

- ❑ **xxx** est le type du paramètre
  - ❑ **num\_p** est le numéro du paramètre (commence à 1)
  - ❑ **val** est la valeur passée en paramètre
-

# Le JDBC

---

- Exemple:

```
requete.setString(1,N);
```

- Exécution d'une requête paramétrée :
  - `executeQuery()` pour les sélections
  - `executeUpdate()` pour les mises à jour
  - Exemple :

```
ResultSet Res = requete.executeQuery();
```

---

# Le JDBC

---

## □ Transactions

- Transaction = suite de requêtes
  - JDBC est en mode « auto-commit »
  - Chaque requête est une seule transaction
  - D'où : `LaConnexion.setAutoCommit(false);`
  - Exécuter une transaction : `LaConnexion.commit()`
    - Exécute toutes les requêtes depuis le dernier commit/rollback
  - Annuler une transaction : `LaConnexion.rollback()`
    - Annule toutes les requêtes depuis le dernier commit/rollback
-

# Le JDBC

---

- Métadonnées d'une requête
    - ResultSetMetaData **getMetaData()**
  
  - Quelques fonctions de ResultSetMetaData
    - int **getColumnCount()**
    - String **getColumnName(int column)**
    - int **getColumnType(int column)**
      - Voir la classe `java.sql.Types` pour les constantes
    - String **getTableName(int column)**
    - Int **isNullable(int column)**
-

# Le JDBC

---

## □ Exemple

```
public static void AfficherRequete(ResultSet R) throws Exception
{
    /**
     * Affichage des noms des champs
     */
    ResultSetMetaData rmd = R.getMetaData();
    int nbCol = rmd.getColumnCount();
    for (int i=1; i<=nbCol; i++)
        System.out.print(rmd.getColumnName(i)+"\t");
    System.out.println();
}
```

---

# Le JDBC

---

```
/**
 * Affichage des données
 */
while (R.next())
{
    for (int c=1; c<=nbCol; c++)
    {
        if (rmd.getColumnType(c)==Types.INTEGER)
            System.out.print(R.getInt(c));
        else if (rmd.getColumnType(c)==Types.BOOLEAN)
            System.out.print(R.getBoolean(c));
        else
            System.out.print(R.getString(c));
        System.out.print("\t");
    }
    System.out.println();
}
}
```

# Le JDBC

---

```
public static void main(String[] args) throws Exception
{
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    Connection base = DriverManager.getConnection(
        "jdbc:odbc:Musique",
        "",
        "");

    Statement S = base.createStatement();
    ResultSet R = S.executeQuery("SELECT * FROM \"Auteur\"");
    AfficherRequete(R);
    R = S.executeQuery("SELECT * FROM Genre");
    AfficherRequete(R);
}
```

---

# Le JDBC

---

## □ Métadonnées d'un SGBD

### ■ Dans la classe **Connection**

□ DatabaseMetaData **getMetaData()**

□ String **getCatalog()**

### ■ Dans la classe **DatabaseMetaData**

□ ResultSet **getColumns**(String catalog,  
String schemaPattern,  
String tableNamePattern,  
String columnNamePattern)

---



# Le JDBC

---

- Gestion des erreurs = gestion des exceptions
    - SQLException
    - SQLWarning
  
  - Contiennent les codes d'erreurs natives de la BD
  - Voir la documentation de la BD pour un traitement précis
  
  - Voir le chapitre sur les exceptions
-