

# JDBC

Java DataBase Connectivity

---

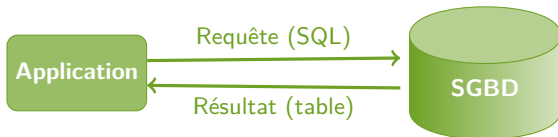
Projet de bases de données – Ensimag 2A

# Architecture client / serveur

---

- ▶ Nous n'utilisons pas ici le SGBD directement, mais en mode **client / serveur**.
- ▶ Le SGBD est vu comme le **serveur de requêtes**.
- ▶ L'application est vue comme le **client**.

- ▶ Nous n'utilisons pas ici le SGBD directement, mais en mode **client / serveur**.
- ▶ Le SGBD est vu comme le **serveur de requêtes**.
- ▶ L'application est vue comme le **client**.

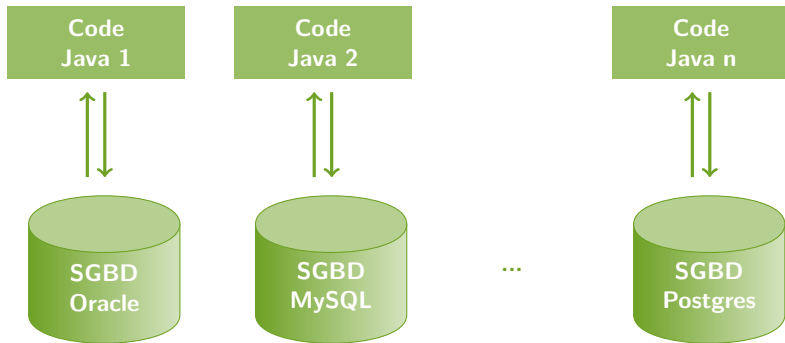


# À quoi sert JDBC ?

---

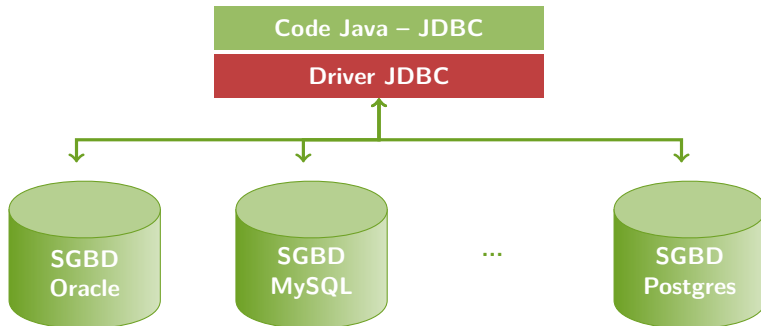
- ▶ API (ensemble de classes et d'interfaces) permettant à des applications Java d'accéder à des SGBD de manière **uniforme**.

- ▶ API (ensemble de classes et d'interfaces) permettant à des applications Java d'accéder à des SGBD de manière **uniforme**.



# À quoi sert JDBC ?

- API (ensemble de classes et d'interfaces) permettant à des applications Java d'accéder à des SGBD de manière **uniforme**.



# À quoi sert JDBC ?

---

- ▶ API (ensemble de classes et d'interfaces) permettant à des applications Java d'accéder à des SGBD de manière **uniforme**.

Avec JDBC, on peut...

- ▶ ...se connecter à un ou plusieurs SGBD (par le biais de *drivers*)
- ▶ ...envoyer des requêtes SQL (interrogation, mise à jour, transactions)
- ▶ ...récupérer le résultat sous forme structurée.

# Exemple

---

## Requêtes sur la base employés

On veut réaliser une application permettant :

- ▶ de récupérer la liste des employés selon leur nom (potentiellement incomplet) ;
- ▶ d'afficher le salaire de ces employés.



L'utilisation de JDBC requiert plusieurs étapes :

1. enregistrement d'un **pilote** (*driver*) JDBC ;
2. **connexion** avec la base de données ;
3. description et **création** d'une **requête** ;
4. **exécution** de la requête ;
5. **récupération** du **résultat** et traitement des données ;
6. **fermeture** de la connexion.

## Code

```
try {  
    DriverManager.registerDriver(  
        new oracle.jdbc.driver.OracleDriver()  
    );  
    ...  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

- ▶ Une connexion est représentée par une instance de la classe `java.sql.Connexion`.
- ▶ Une application peut avoir plusieurs connexions sur une ou plusieurs bases.
- ▶ Une connexion nécessite :
  - ▶ une URL : `jdbc:<sous-protocole>:<URL de la base>`
  - ▶ un nom de connexion
  - ▶ un mot de passe

## Code

```
try {  
    String url =  
        "jdbc:oracle:thin:@oracle1.ensimag.fr:1521:oracle1";  
    String user = "codd";  
    String passwd = "*****";  
  
    Connection connection =  
        DriverManager.getConnection(url, user, passwd);  
    ...  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

- ▶ Dans JDBC, la notion de requête simple est représentée par l'interface `java.sql.Statement`.
- ▶ La classe `java.sql.Connection` permet de créer une requête simple grâce à la méthode de classe `createStatement()`.
- ▶ L'exécution d'une requête se fait :
  - ▶ **interrogation** : par l'utilisation de la méthode `executeQuery(String query)` qui renvoie une instance de `java.sql.ResultSet`,
  - ▶ **mise-à-jour** (insert, update, delete, create, alter ou drop) : par l'utilisation de la méthode `executeUpdate(String query)` qui renvoie le nombre de tuples mis-à-jour.

## Code

```
try {  
    ...  
    Statement stmt = Connection.createStatement();  
    stmt.executeQuery("SELECT * FROM emp;");  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

- Pour lancer plusieurs fois la même requête avec des paramètres différents, on peut utiliser des **requêtes paramétrées** (permet au SGBD d'optimiser l'exécution), représentées en JDBC par l'interface `java.sql.PreparedStatement`.
- La classe `java.sql.Connection` permet de créer une requête paramétrée grâce à la méthode de classe `prepareStatement(String query)`.
- La spécification de paramètres se fait en utilisant `'?'`.
- On peut instancier ces paramètres avec les méthodes `setString(int nb, String s)`, `setInt(int nb, int i)`... `nb` est le numéro du paramètre.

```
try {  
    ...  
    PreparedStatement statement =  
        connection.prepareStatement(  
            "SELECT * FROM emp WHERE ename LIKE ?");  
  
    System.out.print("Nom de l'employe : ");  
    Scanner scan = new Scanner(System.in);  
    String empName = scan.next();  
    scan.nextLine();  
  
    statement.setString(1, empName);  
    ResultSet res = statement.executeQuery();  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```



- ▶ Le résultat d'une requête d'interrogation est donné sous la forme d'une instance de `java.sql.ResultSet`.
- ▶ On peut parcourir l'ensemble des tuples avec les méthodes `next()` et `previous()`.
- ▶ On peut récupérer les valeurs du tuple courant :
  - ▶ en donnant le nom d'une colonne : avec `getInt(String s)`, `getString(String s)`, ...
  - ▶ en donnant le numéro d'une colonne : avec `getInt(int i)`, `getString(int i)`, ...

## code

```
try {  
    ...  
    while (res.next()) {  
        System.out.println(  
            "Employe " + res.getString("ename")  
            + " -> salaire : " + res.getInt("sal")  
        );  
    }  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

# Fermeture de la connexion

---

- Il vaut toujours mieux fermer derrière soi (pour libérer les ressources du SGBD par exemple)...
- Pour cela il faut utiliser la méthode `close()` de la classe `java.sql.Connection`.

## code

```
try {  
    ...  
    connection.close();  
} catch (SQLException e) {  
    e.printStackTrace();  
}
```

- ▶ À l'ouverture d'une connexion, une transaction démarre.
- ▶ Par défaut, JDBC est en **autocommit**
  - ▶ **Attention ! Il vaut mieux changer cette politique par défaut !**
  - ▶ Pour changer, utiliser la méthode `setAutoCommit(boolean ac)` de l'interface `java.sql.Connection`.
- ▶ Les méthodes `commit()` et `rollback()` de l'interface `java.sql.Connection` permettent respectivement de valider et d'annuler la transaction en cours.

# Transactions : points de sauvegarde

---

- **Création d'un point de sauvegarde :**  
setSavePoint(String name) de l'interface  
java.sql.Connection, qui renvoie une instance de  
java.sql.SavePoint.
- **Abandon partiel :** rollBack(java.sql.SavePoint sp) de  
l'interface java.sql.Connection qui revient au point de  
sauvegarde passé en paramètre.

- ▶ On peut récupérer des informations sur le schéma de la base de données (métadonnées) :
  - ▶ pour le résultat d'une requête particulière, avec la méthode `getMetaData()` de l'interface `java.sql.ResultSet`, qui renvoie une instance de `java.sql.ResultSetMetaData`.
  - ▶ pour toute la base, avec la méthode `getMetaData()` de l'interface `java.sql.Connection`, qui renvoie une instance de `java.sql.DatabaseMetaData`.
- ▶ Ces informations concernent par exemple le nom des colonnes, leur type, etc.