



JAVA DATABASE CONNECTIVITY
(JDBC)

Java et les BDD

JDBC (Java Database Connectivity)



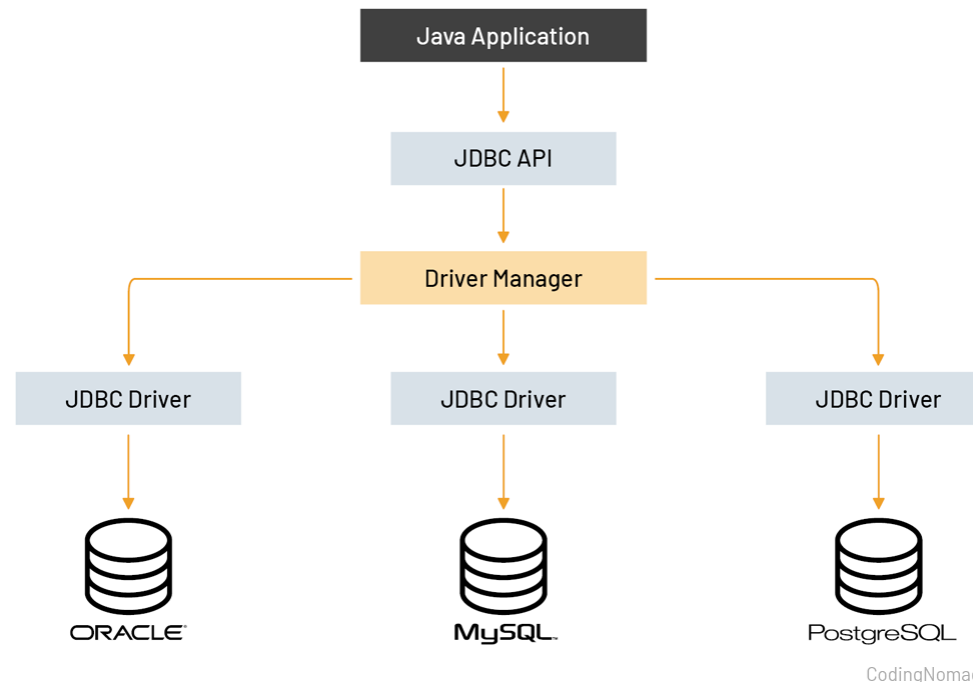
Qu'est ce que JDBC ?



JAVA DATABASE CONNECTIVITY
(JDBC)

JDBC (Java DataBase Connectivity) est un moyen d'émettre des requêtes SQL vers les serveurs, et de récupérer le résultat de cette requête dans du code Java.

On considère aujourd'hui qu'environ 80% des produits logiciels développés ont besoin d'accéder à une base de données. Dès la conception du langage Java, l'accès aux bases de données a donc été une priorité des concepteurs. L'API JDBC (Java Database Connectivity) a été la première API





Qu'est ce que JDBC ?



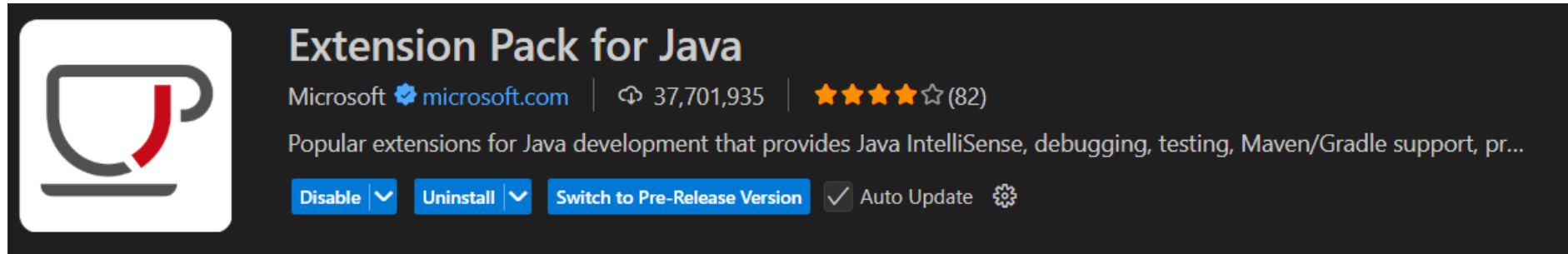
JAVA DATABASE CONNECTIVITY
(JDBC)

Objectifs de JDBC

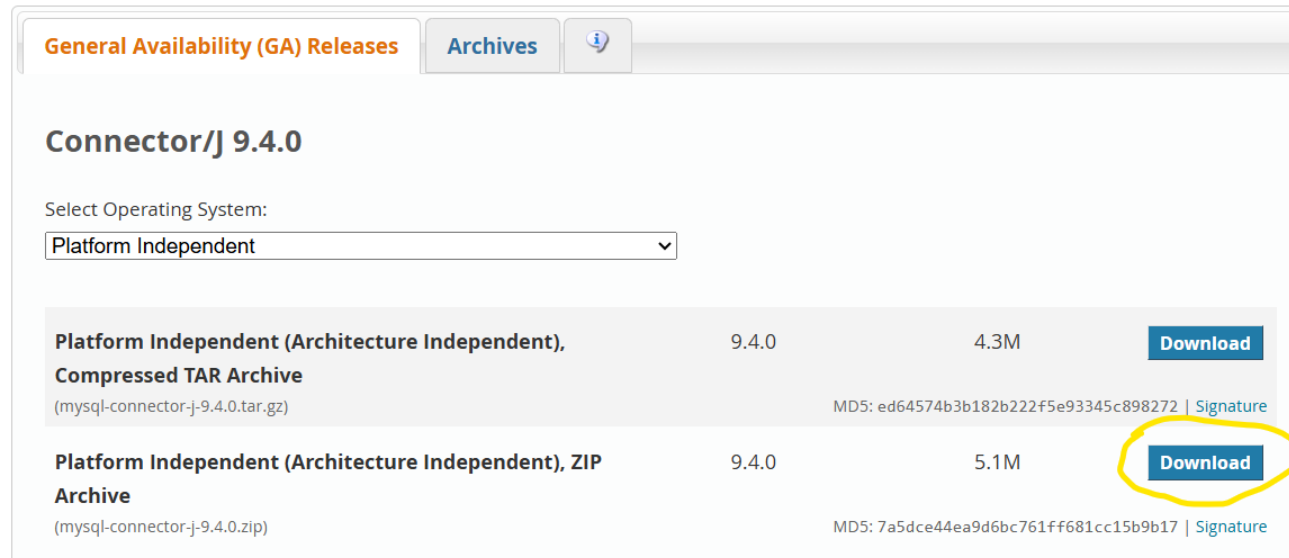
- Permettre aux programmeurs Java d'écrire un code indépendant de la base de données et du moyen de connexion utilisé
- API JDBC (Java DataBase Connectivity) 4.3
- interface uniforme permettant un accès homogène aux SGBD
- simple à mettre en œuvre
- indépendant du SGBD support
- supportant les fonctionnalités de base du langage SQL
- Liés a Java :
 - portabilité sur de nombreux OS et sur de nombreux SGBDR (Oracle, Informix, Sybase, ..)
 - uniformité du langage de description des applications, des applets et des accès aux bases de données
 - liberté totale vis-à-vis des constructeurs.

Installations

1. Dans Vscode : installer l'extension Extension Pack for Java

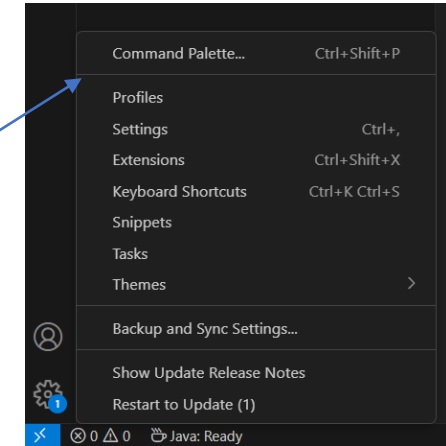
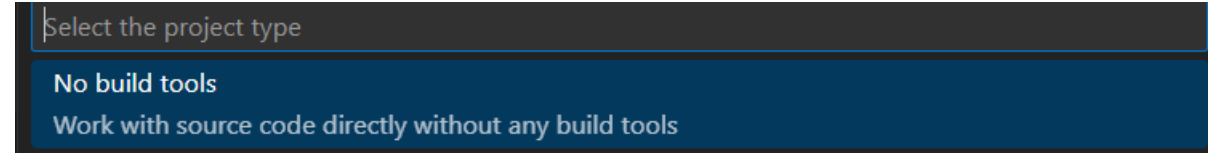
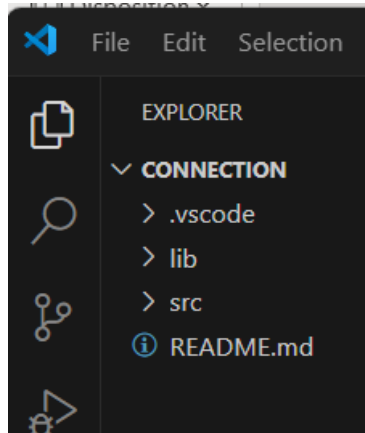


2. Télécharger le Driver JDBC pour MySQL : <https://dev.mysql.com/downloads/connector/j/>



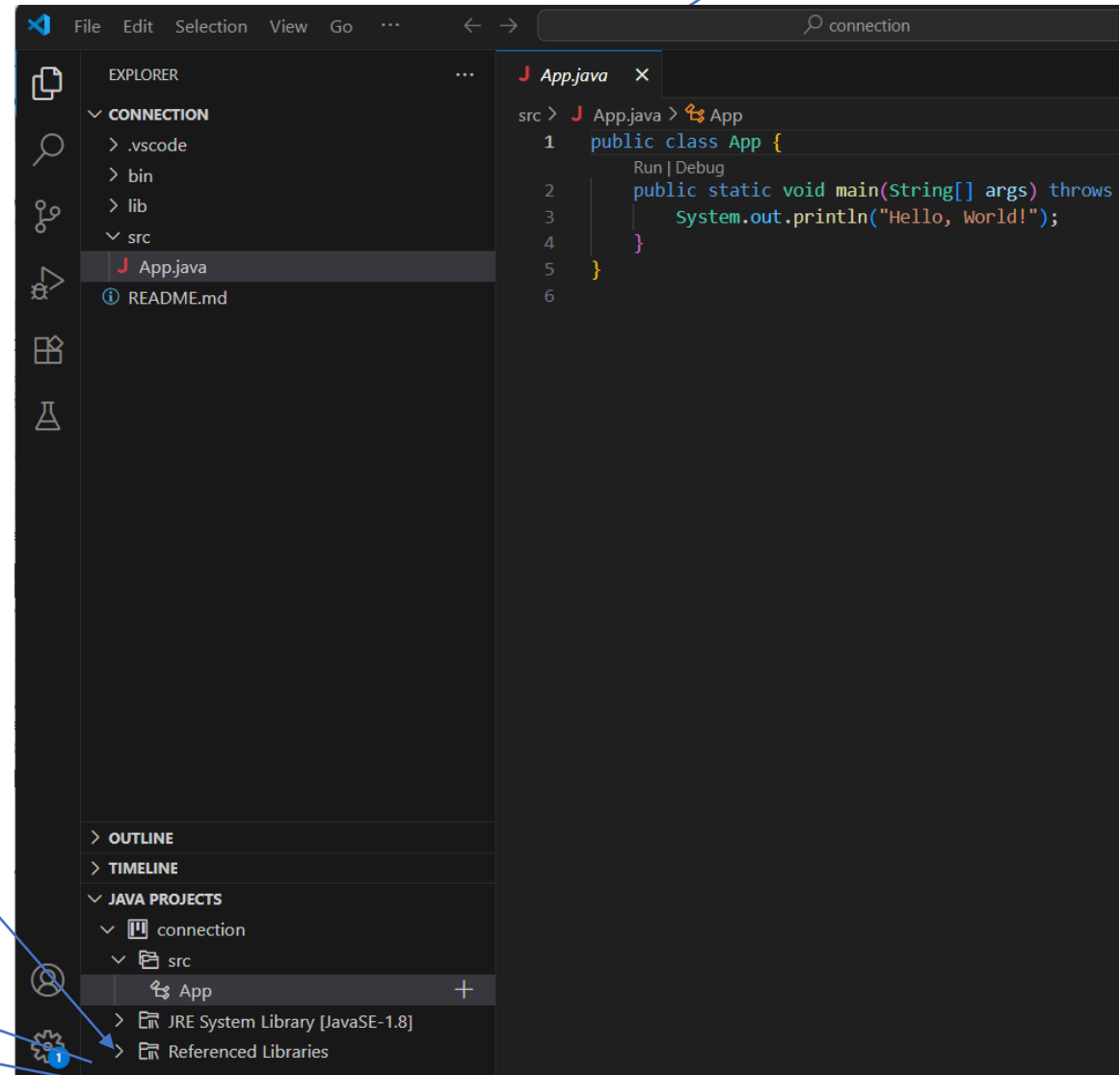
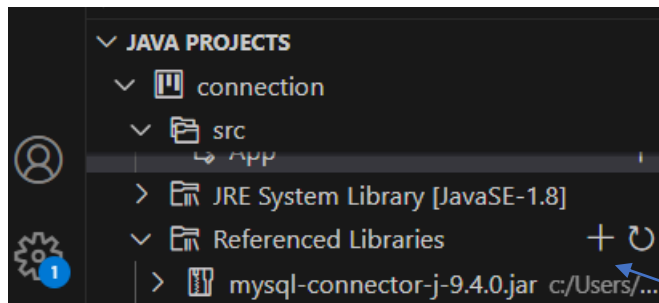
Installations

3. Ouvrir un dossier jdbctests avec VSCode, ouvrir la command palette
Et taper *create java project*, sélectionner *No build tools*, sélectionner le dossier *jdbctests*, puis nommer *connection*



Installations

4. Déclarer le fichier mysql-connector-j-9.4.0.jar comme librairie



Création d'une base test sur Wamp

1- Créer une base de données *vo/s* dans Wamp

2- Créer un table vol :

```
CREATE TABLE vol (  
    id_vol      INT AUTO_INCREMENT PRIMARY KEY,  
    numero_vol  VARCHAR(10) NOT NULL,  
    ville_depart VARCHAR(100) NOT NULL,  
    ville_arrivee VARCHAR(100) NOT NULL,  
    date_depart DATETIME NOT NULL,  
    date_arrivee DATETIME NOT NULL,  
    id_avion    INT,  
    id_pilote   INT,  
    statut      VARCHAR(20)  
);
```

3- Remplir la table avec des données fictives :

```
INSERT INTO vol (numero_vol, ville_depart, ville_arrivee, date_depart, date_arrivee, id_avion, id_pilote, statut) VALUES  
('AF123', 'Paris', 'New York', '2025-09-01 10:30:00', '2025-09-01 13:00:00', 1, 1, 'prévu'),  
('LH456', 'Francfort', 'Tokyo', '2025-09-02 14:00:00', '2025-09-03 07:30:00', 2, 2, 'retardé'),  
('BA789', 'Londres', 'Los Angeles', '2025-09-03 16:15:00', '2025-09-03 19:45:00', 3, 3, 'annulé'),  
('DL234', 'Atlanta', 'Rio de Janeiro', '2025-09-04 09:00:00', '2025-09-04 17:20:00', 4, 4, 'prévu'),  
('EK777', 'Dubai', 'Sydney', '2025-09-05 23:55:00', '2025-09-06 20:10:00', 5, 5, 'prévu'),  
('QF001', 'Sydney', 'Singapour', '2025-09-06 08:30:00', '2025-09-06 14:00:00', 6, 6, 'prévu'),  
('AC890', 'Toronto', 'Paris', '2025-09-07 18:45:00', '2025-09-08 07:15:00', 7, 7, 'retardé'),  
('IB321', 'Madrid', 'Buenos Aires', '2025-09-08 22:00:00', '2025-09-09 07:00:00', 8, 8, 'prévu'),  
('AZ654', 'Rome', 'Athènes', '2025-09-09 12:00:00', '2025-09-09 14:00:00', 9, 9, 'prévu'),  
('KL987', 'Amsterdam', 'Cape Town', '2025-09-10 20:15:00', '2025-09-11 06:30:00', 10, 10, 'annulé');
```

Connection à la BDD grâce à JDBC

Dans App.java

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class App {
    public static void main(String[] args) throws Exception {
        String url = "jdbc:mysql://localhost:3306/vols"; // nom de ta BDD ici vols
        String user = "root"; // ton utilisateur
        String password = ""; // ton mot de passe

        try (Connection conn = DriverManager.getConnection(url, user, password)) {
            System.out.println("Connexion réussie !");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

```
PS C:\Users\villaume4\Documents\transfert\InfoSQL\jdbc test\connection> c:; cd 'c:\Users\villaume4\Documents\transfert\InfoSQL\jdbc test\connection'; & 'C:\Program Files\Java\jre1.8.0_271\bin\java.exe' '-cp' 'C:\Users\VILLAU~1\AppData\Local\Temp\cp_59pgltszh93vk2\ufcb5nulig.jar' 'App'
○ Connexion réussie !
```


Envoi d'une requête en SELECT et récupération du résultat

Dans Vol.java, on crée un modèle Vol pour simplifier la récupération du résultat

```
public class Vol {  
  
    private int id;  
    private String numero;  
    private String villeDepart;  
    private String villeArrivee;  
    private String dateDepart;  
    private String dateArrivee;  
    private String statut;  
  
    public Vol(int id, String numero, String villeDepart, String villeArrivee,  
              String dateDepart, String dateArrivee, String statut) {  
        this.id = id;  
        this.numero = numero;  
        this.villeDepart = villeDepart;  
        this.villeArrivee = villeArrivee;  
        this.dateDepart = dateDepart;  
        this.dateArrivee = dateArrivee;  
        this.statut = statut;  
    }  
  
    // Getters / setters / toString()  
    @Override  
    public String toString() {  
        return id + " - " + numero + " (" + villeDepart + " ->" + villeArrivee + ") " + statut;  
    }  
}
```

Envoi d'une requête en SELECT et récupération du résultat

Dans App.java, on complète pour envoyer une requête SELECT

```
System.out.println("Connexion réussie !");
//requête SQL à exécuter
String sql = "SELECT * FROM vol";
//prépare un objet `Statement` pour que vous puissiez ensuite lui donner une requête SQL à exécuter.
Statement stmt = conn.createStatement();
//envoi de la requête et récupération du résultat
ResultSet rs = stmt.executeQuery(sql);
//on parcourt le résultat de la requete en SELECT tant qu'il y a une ligne suivante
while (rs.next()) {
    Vol v = new Vol(
        rs.getInt("id_vol"),
        rs.getString("numero_vol"),
        rs.getString("ville_depart"),
        rs.getString("ville_arrivee"),
        rs.getString("date_depart"),
        rs.getString("date_arrivee"),
        rs.getString("statut")
    );
    System.out.println(v.toString()+"\n");
}
```

Envoi d'une requête en SELECT et récupération du résultat

Résultat :

```
Connexion réussie !
1 - AF123 (Paris ->New York) prévu
2 - LH456 (Francfort ->Tokyo) retardé
3 - BA789 (Londres ->Los Angeles) annulé
4 - DL234 (Atlanta ->Rio de Janeiro) prévu
5 - EK777 (Dubai ->Sydney) prévu
6 - QF001 (Sydney ->Singapour) prévu
7 - AC890 (Toronto ->Paris) retardé
8 - IB321 (Madrid ->Buenos Aires) prévu
9 - AZ654 (Rome ->Athènes) prévu
10 - KL987 (Amsterdam ->Cape Town) annulé
```

Envoi d'une requête en INSERT grâce à PreparedStatement

```
System.out.println("Connexion réussie !");
//insertion d'un nouveau vol
//Préparation de la requête avec des ? au niveau des valeurs pour plus de sécurité et possibilité de réutiliser
String sqlInsert = "INSERT INTO vol (numero_vol, ville_depart, ville_arrivee, date_depart, date_arrivee, statut) VALUES (?, ?, ?, ?, ?, ?)";


try (PreparedStatement pstmt = conn.prepareStatement(sqlInsert)) {
    //on complète les valeurs en précisant leur type
    pstmt.setString(1, "AF502");
    pstmt.setString(2, "Paris");
    pstmt.setString(3, "London");
    // Assurez-vous que le format de date correspond à ce que votre BDD attend (ex: YYYY-MM-DD HH:MI:SS)
    pstmt.setString(4, "2025-09-21 07:00:00");
    pstmt.setString(5, "2025-09-21 09:00:00");
    pstmt.setString(6, "prévu");
    //on récupère le nombre de lignes insérées (normalement 1) après avoir exécuter la requête
    int affectedRows = pstmt.executeUpdate();
    if (affectedRows > 0) {
        System.out.println("Vol inséré avec succès !");
    }
} catch (SQLException e) {
    System.out.println("Erreur lors de l'insertion du vol.");
    e.printStackTrace();
}

System.out.println("\n--- Liste de tous les vols ---");//on poursuit avec le SELECT pour vérifier
```

Envoi d'une requête en INSERT et récupération du résultat

Résultat :

```
Connexion réussie !  
Vol AF502 inséré avec succès !  
  
--- Liste de tous les vols ---  
1 - AF123 (Paris ->New York) prévu  
  
2 - LH456 (Francfort ->Tokyo) retardé  
  
3 - BA789 (Londres ->Los Angeles) annulé  
  
4 - DL234 (Atlanta ->Rio de Janeiro) prévu  
  
5 - EK777 (Dubai ->Sydney) prévu  
  
6 - QF001 (Sydney ->Singapour) prévu  
  
7 - AC890 (Toronto ->Paris) retardé  
  
8 - IB321 (Madrid ->Buenos Aires) prévu  
  
9 - AZ654 (Rome ->Athènes) prévu  
  
10 - KL987 (Amsterdam ->Cape Town) annulé  
  
11 - AF502 (Paris ->London) prévu
```



Envoi d'une requête en UPDATE grâce à PreparedStatement

```
String sqlupdate = "UPDATE vol SET statut = ? WHERE id_vol = ?";  
try (PreparedStatement stmtupdate = conn.prepareStatement(sqlupdate)) {  
    stmtupdate.setString(1, "retardé");  
    stmtupdate.setInt(2, 10);  
    stmtupdate.executeUpdate();  
}
```

Créez vos tests pour tester ce UPDATE ...

Envoi d'une requête en DELETE grâce à PreparedStatement

```
String sqldel = "DELETE FROM vol WHERE id_vol = ?";  
try (PreparedStatement stmtdel = conn.prepareStatement(sqldel)) {  
    stmtdel.setInt(1, id);  
    stmtdel.executeUpdate();  
}
```

Créez vos tests pour tester ce DELETE ...

Intérêt du PreparedStatement au lieu du Statement (pour insert update et delete)

Avec Statement :

```
Statement stmt = conn.createStatement();  
String sql = "INSERT INTO vol (numero_vol, ville_depart)  
VALUES ("  
    + numero + ", " + villeDepart + ")";  
stmt.executeUpdate(sql);
```

Avec PreparedStatement :

```
String sql = "INSERT INTO vol (numero_vol, ville_depart) VALUES  
(?, ?)";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(1, numero);  
pstmt.setString(2, villeDepart);  
pstmt.executeUpdate();
```

1- Sécurité : protection contre l'injection SQL

- Exemple d'attaque :

```
String numero = "AF123'); DROP TABLE vol; --";
```

- Avec un Statement, ça exécute vraiment DROP TABLE vol.
- Avec un PreparedStatement, le contenu est traité comme une **valeur de paramètre**, pas comme du SQL → donc pas d'attaque possible.

Intérêt du PreparedStatement au lieu du Statement (pour insert update et delete)

Avec Statement :

```
Statement stmt = conn.createStatement();  
String sql = "INSERT INTO vol (numero_vol, ville_depart)  
VALUES ("  
    + numero + ", " + villeDepart + ")";  
stmt.executeUpdate(sql);
```

Avec PreparedStatement :

```
String sql = "INSERT INTO vol (numero_vol, ville_depart) VALUES  
(?, ?)";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(1, numero);  
pstmt.setString(2, villeDepart);  
pstmt.executeUpdate();
```

2- Performance (surtout quand la requête est exécutée plusieurs fois)

La requête SQL avec ? est **précompilée une seule fois** par le serveur.

- Ensuite, seules les valeurs changent.
- Si tu fais beaucoup d'inserts/updates, ça va plus vite.

Intérêt du PreparedStatement au lieu du Statement (pour insert update et delete)

Avec Statement :

```
Statement stmt = conn.createStatement();  
String sql = "INSERT INTO vol (numero_vol, ville_depart)  
VALUES ("  
    + numero + ", " + villeDepart + ")";  
stmt.executeUpdate(sql);
```

Avec PreparedStatement :

```
String sql = "INSERT INTO vol (numero_vol, ville_depart) VALUES  
(?, ?)";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(1, numero);  
pstmt.setString(2, villeDepart);  
pstmt.executeUpdate();
```

3- Lisibilité et maintenance

- La requête est plus claire (pas de concaténation avec des guillemets).
- Tu sé pares la logique SQL du contenu des données.

Intérêt du PreparedStatement au lieu du Statement (pour insert update et delete)

Avec Statement :

```
Statement stmt = conn.createStatement();  
String sql = "INSERT INTO vol (numero_vol, ville_depart)  
VALUES ("  
        + numero + ", " + villeDepart + ")";  
stmt.executeUpdate(sql);
```

Avec PreparedStatement :

```
String sql = "INSERT INTO vol (numero_vol, ville_depart) VALUES  
(?, ?)";  
PreparedStatement pstmt = conn.prepareStatement(sql);  
pstmt.setString(1, numero);  
pstmt.setString(2, villeDepart);  
pstmt.executeUpdate();
```

4- Gestion automatique des types

- setString, setInt, setDate... → le driver fait la conversion.
- Pas besoin de t'embêter avec ' autour des strings, ni avec le format des dates.

Tests non sécurisé avec Statement

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

public class StatementInjectionSQL {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/vols?serverTimezone=UTC";
        String user = "root";
        String password = ""; // adapte ton mot de passe

        try (Connection conn = DriverManager.getConnection(url, user, password);
            Scanner sc = new Scanner(System.in)) {

            System.out.println("=== DEMO AVEC Statement (non sécurisé) ===");
            System.out.print("Entrez la ville d'arrivée : ");
            String arrivee = sc.nextLine();

            String sql = "SELECT * FROM vol WHERE ville_arrivee = '" + arrivee + "'";
            System.out.println("Requête exécutée : " + sql);

            try (Statement stmt = conn.createStatement();
                ResultSet rs = stmt.executeQuery(sql)) {
                while (rs.next()) {
                    System.out.println(rs.getInt("id_vol") + " - " +
                                        rs.getString("numero_vol") + " → " +
                                        rs.getString("ville_arrivee"));
                }
            } catch (SQLException e) {
                e.printStackTrace();
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Tests sécurisé avec PreparedStatement

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

public class PreparedStatementSecure {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/vols?serverTimezone=UTC";
        String user = "root";
        String password = ""; // adapte ton mot de passe

        try (Connection conn = DriverManager.getConnection(url, user, password);
            Scanner sc = new Scanner(System.in)) {

            System.out.println("=== DEMO AVEC PreparedStatement (sécurisé) ===");
            System.out.print("Entrez la ville d'arrivée : ");
            String arrivee = sc.nextLine();

            String sql = "SELECT * FROM vol WHERE ville_arrivee = ?";
            System.out.println("Requête préparée (sécurisée).");

            try (PreparedStatement pstmt = conn.prepareStatement(sql)) {
                pstmt.setString(1, arrivee);
                ResultSet rs = pstmt.executeQuery();
                while (rs.next()) {
                    System.out.println(rs.getInt("id_vol") + " - " +
                                        rs.getString("numero_vol") + " → " +
                                        rs.getString("ville_arrivee"));
                }
            }

        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```