# Using JDBC (Java Database Connectivity) Driver

To use a JDBC (Java Database Connectivity) driver in Java, you need to follow a series of steps to establish a connection to a database, execute queries, and handle results. Below, I'll walk you through the process step-by-step in a clear and practical way.

### Step 1: Add the JDBC Driver to Your Project

First, you need the appropriate JDBC driver for the database you're working with (e.g., MySQL, PostgreSQL, Oracle, etc.). Each database vendor provides its own driver.

- **For Maven Projects**: Add the dependency to your `pom.xml`. For example, for MySQL:

  ```xml
  <dependency>
      <groupId>mysql</groupId>
      <artifactId>mysql-connector-java</artifactId>
      <version>8.0.33</version> <!-- Check for the latest version -->
  </dependency>
  ```

  For PostgreSQL:

  ```xml
  <dependency>
      <groupId>org.postgresql</groupId>
      <artifactId>postgresql</artifactId>
      <version>42.7.3</version> <!-- Check for the latest version -->
  </dependency>
  ```

- **For Non-Maven Projects**: Download the JDBC driver JAR file from the vendor's website (e.g., MySQL Connector/J from mysql.com) and add it to your project's classpath manually.

### Step 2: Load the JDBC Driver

In older versions of Java (pre-JDBC 4.0), you had to explicitly load the driver using `Class.forName()`. With modern JDBC drivers (JDBC 4.0+), this step is optional as the driver is auto-registered via the Service Provider mechanism. However, for clarity, here's how it works:

```java
try {
    Class.forName("com.mysql.cj.jdbc.Driver"); // MySQL example
    // For PostgreSQL: Class.forName("org.postgresql.Driver");
} catch (ClassNotFoundException e) {
    e.printStackTrace();
}
```

### Step 3: Establish a Connection

Use the `DriverManager` class to connect to the database by providing a connection URL, username, and password. The URL format depends on the database.

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

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase"; // MySQL example
        String username = "your_username";
        String password = "your_password";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            System.out.println("Connected to the database!");
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

- **URL Examples**:
  - MySQL: `jdbc:mysql://localhost:3306/mydatabase?useSSL=false`
  - PostgreSQL: `jdbc:postgresql://localhost:5432/mydatabase`
  - Oracle: `jdbc:oracle:thin:@localhost:1521:xe`

### Step 4: Create a Statement and Execute Queries

Once connected, you can execute SQL queries using `Statement`, `PreparedStatement` (preferred for parameterized queries), or `CallableStatement` (for stored procedures).

### Example with a Simple Query:

```java
import java.sql.*;

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase";
        String username = "your_username";
        String password = "your_password";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            Statement statement = connection.createStatement();
            ResultSet resultSet = statement.executeQuery("SELECT * FROM users");

            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"))
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

**Example with a PreparedStatement (Safer for User Input):**

```
try (Connection connection = DriverManager.getConnection(url, username, password)) {
    String sql = "SELECT * FROM users WHERE id = ?";
    PreparedStatement preparedStatement = connection.prepareStatement(sql);
    preparedStatement.setInt(1, 5); // Set the parameter value
    ResultSet resultSet = preparedStatement.executeQuery();

    while (resultSet.next()) {
        System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"));
    }
} catch (SQLException e) {
    e.printStackTrace();
}
```

**Step 5: Close Resources**

Always close your `ResultSet`, `Statement`, and `Connection` objects to free up resources. Using try-with-resources (as shown above) automatically handles this for you.

**Full Working Example (MySQL)**

Here's a complete example assuming a `users` table with `id` (int) and `name` (varchar) columns:

```java
import java.sql.*;

public class JdbcExample {
    public static void main(String[] args) {
        String url = "jdbc:mysql://localhost:3306/mydatabase?useSSL=false";
        String username = "root";
        String password = "your_password";

        try (Connection connection = DriverManager.getConnection(url, username, password)) {
            String sql = "SELECT * FROM users WHERE id = ?";
            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setInt(1, 1);
            ResultSet resultSet = preparedStatement.executeQuery();

            while (resultSet.next()) {
                System.out.println("ID: " + resultSet.getInt("id") + ", Name: " + resultSet.getString("name"))
            }
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

### Key Tips

- **Error Handling**: Always catch `SQLException` to handle database-related errors.
- **Dependencies**: Ensure the JDBC driver matches your database version.
- **Security**: Use `PreparedStatement` for queries with user input to prevent SQL injection.
- **Connection Pooling**: For production apps, consider using a connection pool (e.g., HikariCP) instead of `DriverManager`.

Let me know if you need help with a specific database or more advanced use cases!