JDBC

Asist.dr. Limboi Sergiu

Ce este JDBC?

- **JDBC** (Java Database Connectivity) este o **interfață standard** din Java care permite aplicațiilor să se conecteze la baze de date relaționale (PostgreSQL, MySQL, Oracle, SQLite etc.) și să execute comenzi SQL.
- Gândește-te la JDBC ca la un **translator între Java și SQL**:
  - Java vorbește "Java"
  - Baza de date vorbește "SQL"
  - JDBC face legătura dintre ele

Componentele principale JDBC

Componentă	Descriere	Exemple
DriverManager	Gestionează driver-ele JDBC instalate și creează conexiuni la baza de date.	`DriverManager.getConnection(...)`
Connection	Reprezintă o conexiune activă la baza de date.	`Connection conn = ...`
Statement / PreparedStatement	Execută comenzi SQL (INSERT, SELECT, UPDATE etc.).	`PreparedStatement ps = conn.prepareStatement("SELECT * FROM users")`
ResultSet	Conține rezultatele unei interogări SQL (ca un tabel în memorie).	`ResultSet rs = ps.executeQuery()`
SQLException	Clasă pentru gestionarea erorilor legate de bază de date.	`catch (SQLException e) { ... }`

Adaugi dependența în proiectul Gradle

build.gradle

```
plugins { id 'application' }
repositories { mavenCentral() }

dependencies {
    implementation 'org.postgresql:postgresql:42.7.4' // driver JDBC
    testImplementation 'org.junit.jupiter:junit-jupiter:5.11.0'
}

application { mainClass = 'com.example.Main' }
```

Cum funcționeaza JDBC

- **Încarci driverul JDBC** (Se face automat din Java 6+, dacă dependența este în classpath.)
- **Creezi conexiunea** către baza de date Connection conn = DriverManager.getConnection(url, user, pass);
- **Creezi un obiect Statement sau PreparedStatement**
- **Executi comanda SQL**
  - executeUpdate() → pentru INSERT, UPDATE, DELETE
  - executeQuery() → pentru SELECT
- **Procesezi rezultatele**
- **Închizi resursele**

Obținerea conexiunii la baza de date

```java
package com.example;

import java.sql.*;

public class Main {
    public static void main(String[] args) {
        String url  = "jdbc:postgresql://localhost:5432/demo_db";
        String user = "postgres";
        String pass = "parola_ta";

        try (Connection conn = DriverManager.getConnection(url, user, pass)) {
            System.out.println("Conectat!");
            // aici poți apela metodele de insert & select (mai jos)
        } catch (SQLException e) {
            e.printStackTrace();
        }
    }
}
```

Tipuri de instrucțiuni JDBC

- Statement – pentru comenzi fixe

```
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("SELECT * FROM users");
```

- Prepared Statement- pentru comenzi parametrizate

```
PreparedStatement ps = conn.prepareStatement("INSERT INTO users(lastname, firstname) VALUES (?, ?
ps.setString(1, "Ion");
ps.setString(2, "Maria");
ps.executeUpdate();
```

Tipuri de rezultate

- Un **ResultSet** este ca un tabel cu rânduri și coloane.
- rs.next(); // trece la următorul rând
- rs.getInt("id"); // citește o coloană (prin nume)
- rs.getString(2); // citește o coloană (prin index)
- rs.wasNull(); // verifică dacă ultima coloană a fost NULL

Exemplu INSERT

```
static void insertUser(Connection conn, String last, String first) throws SQLException {
    String sql = "INSERT INTO users(lastname, firstname) VALUES (?, ?)";
    try (PreparedStatement ps = conn.prepareStatement(sql)) {
        ps.setString(1, last);
        ps.setString(2, first);
        int rows = ps.executeUpdate();
        System.out.println("Inserate: " + rows);
    }
}
```

Exemplu SELECT

```
static void listUsers(Connection conn) throws SQLException {
    String sql = "SELECT id, lastname, firstname FROM users ORDER BY id";
    try (PreparedStatement ps = conn.prepareStatement(sql);
         ResultSet rs = ps.executeQuery()) {

        while (rs.next()) {
            long id = rs.getLong("id");
            String last = rs.getString("lastname");
            String first = rs.getString("firstname");
            System.out.printf("%d | %s | %s%n", id, last, first);
        }
    }
}
```

Avantaje JDBC

Independență față de baza de date – același cod funcționează și cu MySQL, și cu PostgreSQL.

- Control total asupra SQL-ului – vezi exact ce interogări se trimit.
- Interfață standard – poți trece la alte librării (Spring JDBC, JPA) fără să schimbi baza.
- Stabilitate – este parte din Java SE standard.

Dezavantaje JDBC

- Mult cod boilerplate (multe try/catch, închideri de resurse).
- Nu are mapare obiect-relațională automată (ca în Hibernate sau JPA)
- Necesită cunoașterea SQL-ului nativ.