

Arquitetura N-Tier e JDBC

- Arquitetura N-Tier
- JDBC

Programação Distribuída / José Marinho

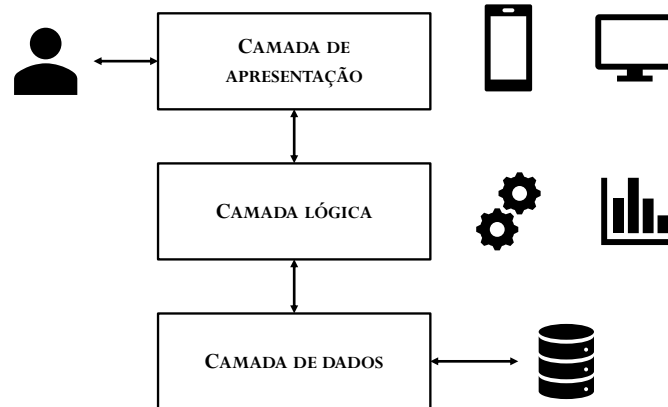
Arquitetura N-Tier

- Arquitetura de *software* N-Tier ou Multi-Tier
- Separação lógica e física das diversas funções em várias camadas (1-Tier: centralizado e 2-Tier: cliente-servidor)
- Cada camada pode ser constituída por um ou mais componentes (subcamadas)
- A arquitetura 3-Tier é a mais habitual
 - Apresentação
 - Lógica (de negócio)
 - Dados

2

Programação Distribuída / José Marinho

Arquitetura N-Tier



3

Arquitetura N-Tier

- N-Tier vs. Padrão MVC (Modelo-Vista-Controlador)
 - Princípios de base comparáveis
 - Separação física (e não apenas lógica) dos diversos componentes
 - Modelo no MVC → Lógica + Dados
 - A camada de apresentação de uma arquitetura N-Tier pode, por sua vez, ser estruturada de acordo com o padrão MVC, Observável/Observador, *Publisher/Subscriber*, etc.

4

Arquitetura N-Tier

- Vantagens
 - Possibilidade de acrescentar, eliminar e modificar componentes sem implicações na globalidade da aplicação
 - Gestão, manutenção e reutilização do *software* facilitados
 - Escalabilidade e flexibilidade
 - Aumento de desempenho e de resiliência potenciado pela distribuição dos componentes por diversas máquinas
 - Aumento de segurança através da omissão de dados sensíveis na camada de apresentação

5

Arquitetura N-Tier

- Possibilidade de tornar o processo de desenvolvimento mais eficiente, distribuindo-o por diversas equipas com áreas de competência distintas
- Aspectos críticos
 - Desempenho dos canais de comunicação
 - Desempenho do *hardware*
 - O número de camadas não deve ir além do mínimo necessário para não aumentar a complexidade da arquitetura e o esforço de desenvolvimento

6

Arquitetura N-Tier

- Interação dos componentes através de mensagens trocadas através de redes de dados
- Diversas soluções de *middleware* oferecem paradigmas de desenvolvimento baseados em objetos remotos e serviços web: abstração em relação à troca subjacente de mensagens protocolares do nível de aplicação via TCP e UDP

7

JDBC

- *Java Database Connectivity*
- API (*Application Programming Interface*) para acesso a dados armazenados, essencialmente, em bases de dados relacionais
- Três tipos de operações
 - Ligação a fontes de dados
 - Submissão de comandos SQL (*Structured Query Language*)
 - Devolução e processamento de respostas a pedidos

8

JDBC

- Exemplo elementar

```
Connection con = DriverManager.getConnection(
    "jdbc:mysql:myDatabase",
    username,
    password);

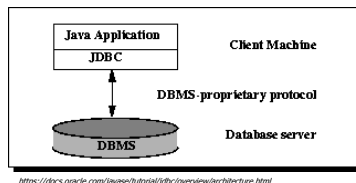
Statement stmt = con.createStatement();
ResultSet rs = stmt.executeQuery(
    "SELECT a, b, c FROM Table1");

while (rs.next()) {
    int x = rs.getInt("a");
    String s = rs.getString("b");
    float f = rs.getFloat("c");
}
```

9

JDBC

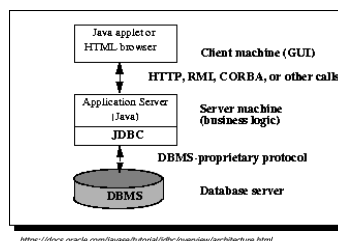
- Suporta a arquitetura de acesso 2-Tier
 - Abordagem cliente/servidor
 - A aplicação terminal interage diretamente com a fonte de dados através de um *driver* JDBC específico
 - A aplicação e a fonte de dados podem estar localizadas em máquinas distintas



10

JDBC

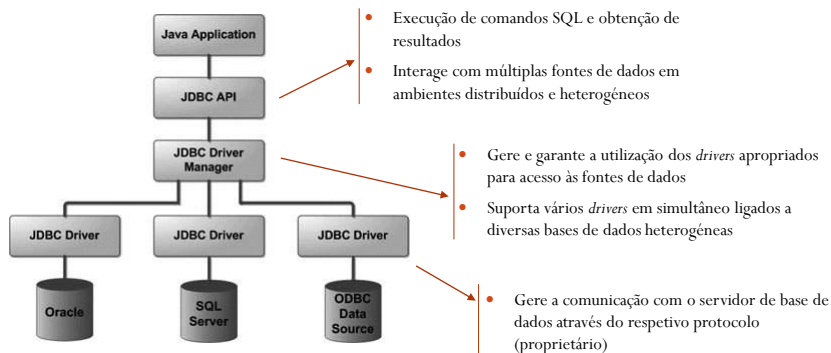
- Suporta a arquitetura de acesso 3-Tier
 - A aplicação terminal interage com a camada intermédia (lógica) que, por sua vez, interage com a fonte de dados
 - Permite controlar o acesso aos dados
 - Pode facilitar o desenvolvimento de aplicações e permitir o aumento de desempenho



11

JDBC

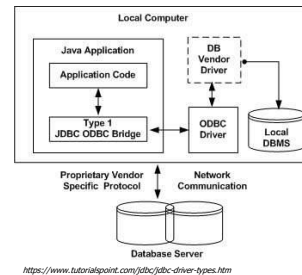
- Componentes principais



12

JDBC - Drivers

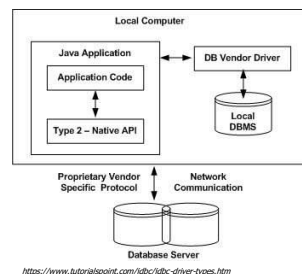
- Tipo 1
 - Mapeamento entre a API JDBC e outra API de acesso a dados (e.g., JDBC-ODBC *Bridge*)
 - Geralmente suportado por uma biblioteca nativa → portabilidade limitada
 - Não é habitualmente usado



13

JDBC - Drivers

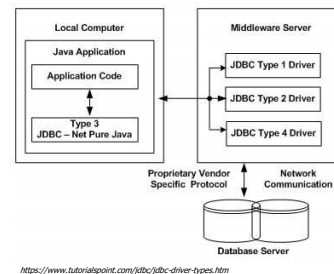
- Tipo 2
 - Parcialmente desenvolvidos em Java e em código nativo
 - API JDBC → API C/C++ específico de cada SGBD (Sistema de Gestão de Bases de Dados)
 - Portabilidade limitada
 - Usado quando não existem soluções de tipo 3 e 4 disponíveis



14

JDBC - Drivers

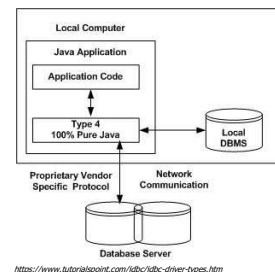
- Tipo 3
 - Desenvolvido em Java
 - Permite aceder a vários SGBD em simultâneo (preferido quando existe esta necessidade)
 - Comunica com um servidor intermédio (*middleware*) através de um protocolo de comunicação independente de qualquer SGBD
 - O servidor intermédio é que comunica com a fonte de dados



15

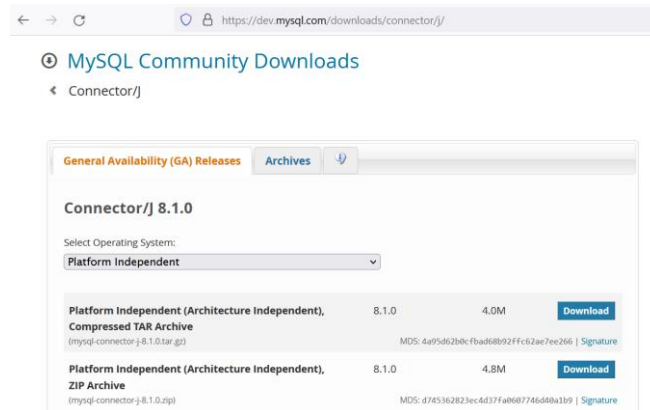
JDBC - Drivers

- Tipo 4
 - Desenvolvidos em Java e preferido quando se acede a um único SGBD
 - A aplicação comunica diretamente com a fonte de dados através do respetivo protocolo proprietário
 - Exemplo: Connector/J do MySQL
 - Protocolos de comunicação com suporte para as fases de conexão, encriptação, compressão e troca de comandos e respostas



16

JDBC - Drivers



17

JDBC - Drivers

O método `forName()` devolve uma instância da classe `Class` associada à classe ou interface com o nome indicado. Neste caso, `com.mysql.jdbc.Driver` refere-se à classe que implementa a interface `java.sql.Driver` (driver JDBC). Desta forma, o driver fica registado no `DriverManager` que o usa para tentar estabelecer ligações. Já não é necessário desde o Java 1.6.

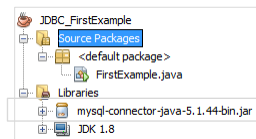
```
Class.forName("com.mysql.jdbc.Driver");
```

```
System.out.println("Connecting to database...");
conn = DriverManager.getConnection("jdbc:mysql://localhost/world", USER, PASS);

System.out.println("Creating statement...");
stmt = conn.createStatement();
String sql;
sql = "SELECT * FROM City WHERE CountryCode='FRA'";
ResultSet rs = stmt.executeQuery(sql);

while(rs.next()){
    //Retrieve by column name
    int id = rs.getInt("id");
    String name = rs.getString("name");
    int population = rs.getInt("population");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Name: " + name);
    System.out.println(", Population: " + population);
}
```



```
ID: 2974, Name: Paris, Population: 2125246
ID: 2975, Name: Marseille, Population: 798430
ID: 2976, Name: Lyon, Population: 445452
ID: 2977, Name: Toulouse, Population: 390350
ID: 2978, Name: Nice, Population: 342738
ID: 2979, Name: Nantes, Population: 270251
ID: 2980, Name: Strasbourg, Population: 264115
ID: 2981, Name: Montpellier, Population: 225392
ID: 2982, Name: Bordeaux, Population: 215363
ID: 2983, Name: Rennes, Population: 206229
ID: 2984, Name: Le Havre, Population: 190905
ID: 2985, Name: Reims, Population: 187306
ID: 2986, Name: Lille, Population: 184657
ID: 2987, Name: St-Mtienne, Population: 180210
ID: 2988, Name: Toulon, Population: 160639
ID: 2989, Name: Grenoble, Population: 153317
ID: 2990, Name: Angers, Population: 151279
ID: 2991, Name: Dijon, Population: 149867
ID: 2992, Name: Brest, Population: 149634
ID: 2993, Name: ... Population: 146305
```

18

JDBC - Classes e interfaces

- **Package java.sql**
 - **Driver**: interface com métodos para gerir a comunicação com o SGBD
 - **DriverManager**: classe para gerir um lista de drivers
 - **Connection**: interface para contactar o SGBD
 - **Statement**: interface para submeter pedidos SQL
 - **ResultSet**: interface para aceder às respostas aos pedidos
 - **SQLException**: classe para gerir erros que surjam em aplicações de BD

19

JDBC – Exemplo 1

```
//STEP 1. Import required packages
import java.sql.*;

public class FirstExample {
    // JDBC driver name and database URL
    static final String JDBC_DRIVER = "com.mysql.jdbc.Driver";
    static final String DB_URL = "jdbc:mysql://localhost/EMP";

    // Database credentials
    static final String USER = "username";
    static final String PASS = "password";

    public static void main(String[] args) {
        Connection conn = null;
        Statement stmt = null;
    }
}
```

20

JDBC – Exemplo 1

```
ResultSet rs = null;
try{
    //STEP 2: Register JDBC driver
    //Class.forName(JDBC_DRIVER);

    //STEP 3: Open a connection
    System.out.println("Connecting to database...");
    conn = DriverManager.getConnection(DB_URL,USER,PASS);

    //STEP 4: Execute a query
    System.out.println("Creating statement...");
    stmt = conn.createStatement();
    String sql;
    sql = "SELECT id, first, last, age FROM Employees";
    rs = stmt.executeQuery(sql);
```

21

JDBC – Exemplo 1

```
//STEP 5: Extract data from result set
while(rs.next()){
    //Retrieve by column name
    int id = rs.getInt("id");
    int age = rs.getInt("age");
    String first = rs.getString("first");
    String last = rs.getString("last");

    //Display values
    System.out.print("ID: " + id);
    System.out.print(", Age: " + age);
    System.out.print(", First: " + first);
    System.out.println(", Last: " + last);
}
}catch(SQLException se){
    System.out.println(se);
```

22

JDBC – Exemplo 1


```
}catch(Exception e){
    System.out.println(e);
}finally{
    //STEP 6: Clean-up environment
    try{
        if(stmt!=null) stmt.close();
    }catch(SQLException se){}
    try{
        if(conn!=null) conn.close();
    }catch(SQLException se){}
    try{
        if(rs!=null) rs.close();
    }catch(SQLException se){}
    }//end try
} //end main
} //end FirstExample
```

23

JDBC – Exemplo 2

- Ligação a uma base de dados SQLite
- www.sqlitetutorial.net/sqlite-java/
- mvnrepository.com/artifact/org.xerial/sqlite-jdbc

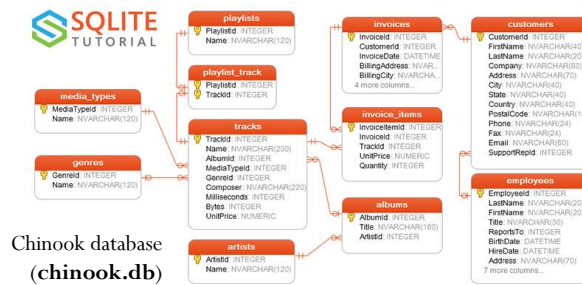
Home » org.xerial » sqlite-jdbc » 3.43.0.0

	SQLite JDBC » 3.43.0.0
SQLite JDBC is a library for accessing and creating SQLite database files in Java (it includes native libraries)	
License	Apache 2.0
Categories	JDBC Drivers
Tags	sqlite database sql jdbc driver
HomePage	https://github.com/xerial/sqlite-jdbc
Date	Aug 29, 2023
Files	pom (17 KB) jar (12.6 MB) View All
Repositories	Central EEA SIK
Ranking	#383 in MavenRepository (See Top Artifacts) #6 in JDBC Drivers
Used By	1,200 artifacts

24

JDBC – Exemplo 2

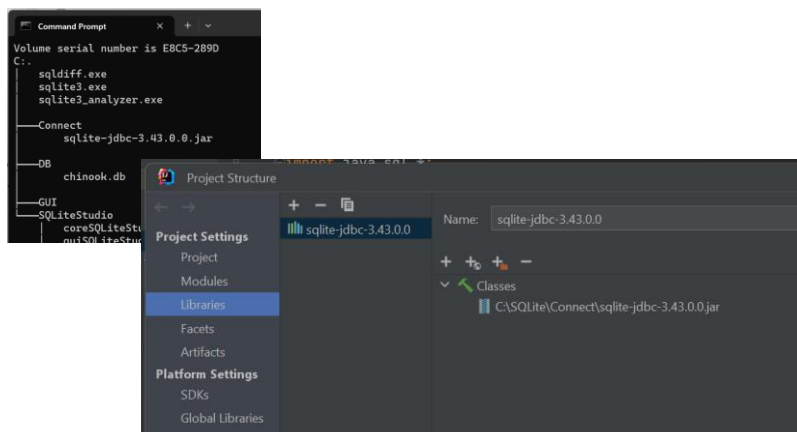
- www.sqlitetutorial.net/sqlite-sample-database/



Chinook database
(chinook.db)

25

JDBC – Exemplo 2



26

JDBC – Exemplo 2

```
package net.sqlitetutorial;
import java.sql.*;

public class Connect {
    public static void connect() {
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;

        try {
            String url = "jdbc:sqlite:C:/sqlite/db/chinook.db";
            conn = DriverManager.getConnection(url);
            stmt = conn.createStatement();
            String query = "SELECT EmployeeId,FirstName,LastName FROM Employees";
            rs = stmt.executeQuery(query);

            while(rs.next()) {
                int id = rs.getInt("EmployeeId");
                String first = rs.getString("FirstName");
                String last = rs.getString("LastName");
            }
        }
    }
}
```

27

JDBC – Exemplo 2

```
        System.out.print("ID: " + id);
        System.out.print(", First: " + first);
        System.out.println(", Last: " + last);
    }
} catch (SQLException ex) {
    System.out.println(ex.getMessage());
} finally {
    try { if (conn != null) conn.close(); } catch (SQLException ex) {}
    try { if (stmt != null) stmt.close(); } catch (SQLException ex) {}
    try { if (rs != null) rs.close(); } catch (SQLException ex) {}
}
}

public static void main(String[] args) {
    connect();
}
}
```

28

JDBC – Exemplo 2 (*try-with-resources*)

```
public class Connect {  
  
    public static void connect() {  
  
        String url = "jdbc:sqlite:C:/sqlite/db/chinook.db";  
        String query = "SELECT EmployeeId, FirstName, LastName FROM Employees";  
  
        try(Connection conn = DriverManager.getConnection(url);  
            Statement stmt = conn.createStatement();  
            ResultSet rs = stmt.executeQuery(query)){  
  
            while(rs.next()){  
                int id = rs.getInt("EmployeeId");  
                String first = rs.getString("FirstName");  
                String last = rs.getString("LastName");  
  
                System.out.print("ID: " + id);;  
                System.out.print(", First: " + first);  
                System.out.println(", Last: " + last);  
  
            } catch (SQLException ex) {  
                System.out.println(ex.getMessage());  
            }  
        }  
  
        public static void main(String[] args) {  
            connect();  
        }  
    }  
}
```

29

JDBC – Exemplo 2 (*try-with-resources*)

```
        System.out.print("ID: " + id);;  
        System.out.print(", First: " + first);  
        System.out.println(", Last: " + last);  
  
    }  
  
    } catch (SQLException ex) {  
        System.out.println(ex.getMessage());  
    }  
}  
  
public static void main(String[] args) {  
    connect();  
}  
}
```

30

JDBC – *executeUpdate()*

- `int executeUpdate(String sql)`
- Execução de instruções SQL de INSERT, UPDATE e DELETE (*Data Manipulation Language* - DML)
- Execução de instruções SQL que não retornam qualquer tipo de informação (criação, modificação e eliminação de tabelas, definição das permissões dos utilizadores, etc.)
- Devolve o número de linhas afetadas no caso de instruções SQL de DML ou 0 para as instruções sem valores de retorno

31

JDBC – *executeUpdate()*

```
stmt.executeUpdate("DELETE FROM pi_workers WHERE address = '192.92.92.92' AND
port = 5001;");

if(stmt.executeUpdate("UPDATE pi_workers SET timestamp=CURRENT_TIMESTAMP WHERE
address LIKE '%192.92.92.92%' AND port=5001;") < 1){

    System.out.println("Entry update failed");

    if(stmt.executeUpdate("INSERT INTO pi_workers (address,port) VALUES
('192.92.92.92',5001;") < 1){

        System.out.println("Entry insertion failed");
    }
}

stmt.executeUpdate("CREATE DATABASE IF NOT EXISTS registoPresencas;");
stmt.executeUpdate("USE testSQLtable;");
stmt.executeUpdate("DROP TABLE IF EXISTS utilizador;");
```

32