INE5646 – Programação para Web

Unidade III – Seção V

Bancos de Dados

Prof. Frank Siqueira – Turma A

Prof. Leandro Komosinski – Turma B

Conteúdo

- Bancos de Dados na Web
- Bibliotecas e APIs
  - JDBC
- Bibliotecas de *Tags*
  - JSTL SQL
- *Frameworks* de mapeamento O-R

Bancos de Dados na Web

- Justificativa
  - Os dados utilizados por grande parte das aplicações Web precisam ser persistidos
  - Dados mantidos na memória de aplicações são voláteis, ou seja, se perdem quando o servidor cai
  - Em geral, há mais espaço disponível em disco do que em memória RAM
- Estratégia
  - Manter os dados em um banco de dados
  - Trazer para a memória somente o que for necessário no momento para executar a aplicação

Bancos de Dados na Web

- Modelos de Dados
  - Em geral, as aplicações Web utilizam bancos de dados baseados no modelo relacional (tabelas)
  - Uma linguagem de consulta (SQL, por exemplo) podem ser usada por aplicações Web para obter dados a partir de tabelas do BD
- Reuso
  - Muitas vezes o banco utilizado por uma aplicação Web já foi criado previamente, e é usado por outras aplicações corporativas (Web ou *desktop*)
  - Deve-se evitar duplicidade de dados, que pode resultar em inconsistências (dados divergentes)

Bancos de Dados na Web

- O acesso ao BD a partir da aplicação Web pode ser feito através de:
  - Bibliotecas e APIs: fornecem rotinas ou classes para acesso ao BD
  - Bibliotecas de *tags* para acesso a BDs: instruções de acesso ao BD (consultas, atualizações, etc.) são executadas inserindo *tags* em páginas dinâmicas
  - *Frameworks* de mapeamento de dados: dados contidos no BD são mapeados de/para estruturas da linguagem de programação utilizada

Bibliotecas e APIs

- Suporte Nativo da Linguagem de Programação
  - Grande parte das linguagens de programação fornece suporte nativo para acesso a bancos de dados na forma de bibliotecas/APIs
  - Exemplos:
    - PHP: módulo dbx
    - Python: DB-API interface
    - Perl: módulo DBI
    - Java: JDBC API

Bibliotecas e APIs

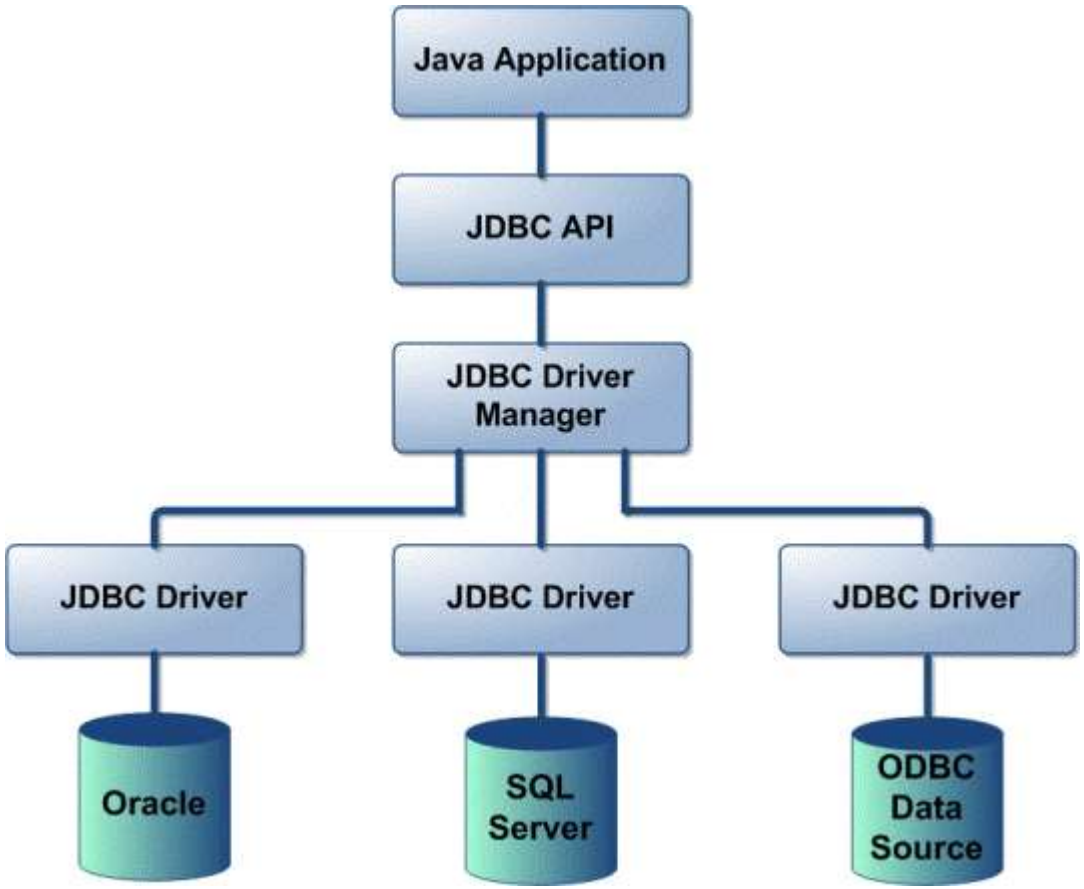
- Suporte fornecido por Sistemas Gerenciadores de Bancos de Dados (SGBDs)
  - Alguns SGBDs fornecem suas próprias bibliotecas/ APIs para uma ou mais linguagens de programação
  - Exemplos:
    - MySQL: APIs para C, PHP, Perl, Python, Ruby, TCL, ...
    - Oracle DB: bibliotecas para C, C++, PHP, Ruby, Perl, ...
    - PosgreSQL: APIs para C, C++, Perl, PHP, TCL, Python, ...

Bibliotecas e APIs

- Suporte através de ODBC
  - Diversos SGBDs possuem interfaces que seguem o padrão ODBC (*Open DataBase Connectivity*)
  - ODBC permite que o código de aplicações seja independente de SGBD; com isso, é possível trocar de SGBD sem modificar o código da aplicação
  - Linguagens possuem bibliotecas/APIs que se conectam à interface ODBC do SGBD
  - Exemplo: aplicação em PHP pode usar um módulo ODBC para acessar um BD do Microsoft Access

JDBC

- JDBC (*Java DataBase Connectivity*)
  - API do Java para acesso a BDs
  - É necessário utilizar um *driver*, que geralmente é fornecido pela empresa fabricante do SGBD



JDBC

- Tipos de *Drivers* JDBC
  - Tipo 1: JDBC-ODBC *Bridge*
    - É uma "ponte" entre JDBC e ODBC, que permite que a aplicação Java veja o *driver* ODBC como um *driver* JDBC
    - Fornecido pela classe *sun.jdbc.odbc.JdbcOdbcDriver*
  - Tipo 2: *Native API Driver*
    - Executa comandos invocando uma API/biblioteca proprietária fornecida pelo fabricante do SGBD
    - Sua implementação é dependente de plataforma e requer que o cliente possua a API/biblioteca do SGBD
    - Apresenta melhor desempenho que o *driver* tipo 1

JDBC

- *Drivers* JDBC (cont.)
  - Tipo 3: *Network Protocol Driver*
    - Escrito 100% em Java (multi-plataforma)
    - Emprega uma camada intermediária (*middleware*), que converte comandos JDBC para um protocolo de rede suportado pelo SGBD utilizado
    - O *middleware* pode suportar diversos SGBDs
  - Tipo 4: *Native Protocol Driver*
    - Escrito 100% em Java (multi-plataforma)
    - Utiliza protocolo proprietário do SGBD para acessá-lo
    - Apresenta melhor desempenho que os outros tipos

JDBC

- Utilização
  - A API JDBC pode ser invocada a partir de qualquer código Java, incluindo *scriptlets* JSP, *Servlets* e *beans* gerenciados JSF
  - Etapas de acesso a uma base de dados:
    - Estabelecimento de conexão com o SGBD
    - Execução de comando SQL
    - Obtenção do resultado da consulta

JDBC

- Estabelecimento de conexão:
  1. Carregar *driver* JDBC:
     - Instanciar um objeto da classe *java.sql.Driver*:
       ```
       Class.forName(path.to.driver);
       ```
     - Forma alternativa na execução da JVM:
       ```
       java -Djdbc.drivers=path.to.driver MyApp
       ```
  2. Instanciar objeto da classe *java.sql.Connection*:
     ```
     Connection conn = DriverManager.
             getConnection(URL, login, passwd);
     ```

JDBC

- Execução de Comandos SQL
  - O envio de comandos é feito através das classes:
    - `java.sql.Statement`: representa um comando SQL
    - `java.sql.PreparedStatement`: comando SQL pré-compilado, para execução eficiente múltiplas vezes
  - Principais métodos das classes *Statement*
    - `.executeQuery(String SQL)`: executa consulta
    - `.executeUpdate(String SQL)`: executa *update*, *insert* ou *delete*

JDBC

- Obtenção de Resultados de Consultas
  - Resultados de comandos são recebidos em um objeto `ResultSet` ao executar um `Statement`:
    ```
    java.sql.ResultSet rs = stmt.executeQuery(SQL);
    ```
  - O método `rs.next()` retorna um valor *booleano* indicando se há mais dados para serem lidos
  - Se houver um novo valor, este pode ser lido com um método `rs.getTipo(col)`, onde `Tipo` é o tipo correspondente ao dado retornado (ex.: *String*, *Int*, etc.) e `col` é o número ou nome da coluna

Bibliotecas de *Tags*

- A Biblioteca de *Tags* JSTL SQL define *tags* que são usadas para acessar banco de dados
  - *Tags* efetuam operações de consulta, inserção, atualização e remoção de dados em um BD
  - São úteis para criação de aplicações simples e para prototipagem rápida
  - Já em aplicações complexas, deve-se encapsular dados em objetos (*JavaBeans* ou DAOs) ou utilizar um *framework* de mapeamento objeto-relacional

JSTL SQL

- Biblioteca de *Tags* padrão do JSP para acesso a BDs
- Forma de utilização:
  - Importação:

```
<%@ taglib prefix="sql" uri="http://java.sun.com/jsp/jstl/sql" %>
```

  - Definição da fonte de dados:

```
<sql:setDataSource var="db" driver="org.apache.derby.jdbc.Driver"
  url="jdbc:derby://localhost:1527/DBClientes" user="joao"
  password="123"/>
```

  - Consulta:

```
<sql:query dataSource="${db}" var="clientes" />
   SELECT codigo, nome, sobrenome FROM Clientes
</sql:query>
```

JSTL SQL

- Forma de utilização (cont.)
  - Acesso a resultados de consultas:

```
<c:forEach var="c" items="${clientes.rows}">
  <p>Cliente ${c.codigo}: ${c.nome} ${c.sobrenome}</p>
</c:forEach>
```

  - Alteração de dados (INSERT, UPDATE e DELETE):

```
<sql:update dataSource="${db}" var="numLinhasMod" />
  INSERT INTO Clientes VALUES (111, 'João', 'Silva')
</sql:update>
```

  - Definição de parâmetros:

```
<sql:update dataSource="${db}" var="numLinhasMod">
 DELETE FROM Clientes WHERE codigo= ?
 <sql:param value="${param.codigo}" />
</sql:update>
```

Frameworks de Mapeamento

- *Frameworks* de Mapeamento de Dados
  - Mapeiam dados mantidos em um banco de dados de/para estruturas de dados de uma linguagem de programação
  - Efetuam automaticamente as conversões necessárias entre os tipos de dados utilizados no BD e os tipos existentes na linguagem
  - Permitem que os dados contidos em um BD sejam acessados de dentro de um programa sem executar instruções de acesso ao banco (ex.: consultas SQL)

Frameworks de Mapeamento

- *Frameworks* de Mapeamento Objeto-Relacional
  - Mapeiam as tabelas de BDs relacionais de/para objetos de uma linguagem OO
  - De modo geral, tabelas são mapeadas para classes
    - cada coluna da tabela se torna um atributo da classe
    - cada linha corresponde a um objeto (instância) da classe
    - a chave primária se torna o identificador do objeto
    - chaves estrangeiras e tabelas de relacionamento são geralmente representados por associações entre classes (atributo indica o objeto ou lista de objetos relacionados)

Frameworks de Mapeamento

- Exemplos de *Frameworks* de Mapeamento Objeto-Relacional:
  - Java: implementações das especificações JPA (*Java Persistence API*) e JDO (*Java Data Objects*), além de *frameworks* independentes, como Apache Cayenne.
  - .NET: ADO.NET *Entity Framework*, NHibernate, ...
  - PHP: CakePHP, *Zend Framework*, ...
  - Python: Django, Storm, Tryton, web2py, ...
  - Ruby: ActiveRecord (parte do *Ruby on Rails*), ...