Interface Gráfica e Banco de Dados em Java

Acesso a bancos de dados com JDBC

Licença para uso e distribuição

Este material está disponível para uso não-comercial e pode ser derivado e/ou distribuído, desde que utilizando uma licença equivalente.

Atribuição-Uso Não-Comercial-Compartilhamento pela mesma licença, versão 2.5

http://creativecommons.org/licenses/by-nc-sa/2.5/deed.pt

Você pode copiar, distribuir, exibir e executar a obra, além de criar obras derivadas, sob as seguintes condições: (a) você deve dar crédito ao autor original, da forma especificada pelo autor ou licenciante; (b) você não pode utilizar esta obra com finalidades comerciais; (c) Se você alterar, transformar, ou criar outra obra com base nesta, você somente poderá distribuir a obra resultante sob uma licença idêntica a esta.

Objetivos desta parte

- Introduzir conceitos básicos de persistência em bancos de dados relacionais (BDRs);
- Apresentar a linguagem SQL de forma a permitir a manipulação simples de dados em BDRs;
- Explicar como conectar em BDRs e enviar comandos SQL em Java com a API JDBC;
- Apresentar o padrão de projeto DAO como uma forma organizada de persistência de objetos.

Armazenamento persistente

- A grande maioria das aplicações precisa armazenar informações em mídia persistente;
- Persistente = não se apaga após terminado o processo que a criou;
- Tipos de armazenamento:
  - Arquivos binários (serialização);
  - Arquivos texto (ex.: XML);
  - Bancos de dados.

Banco de dados

- Coleção organizada de dados;
- SGBD – Sistema Gerenciador de Banco de Dados:
  - Em inglês, *Database Management System* (DBMS);
  - Software que gerencia o armazenamento e recuperação de dados em arquivos;
  - Provê uma linguagem para manipulação dos dados;
  - Usuário não precisa saber como os dados são armazenados em disco.

Paradigmas de SGBDs

- SGBDs Relacionais (SGBDRs):
  - Baseados em álgebra relacional (eficácia matematicamente provada);
  - Forte indústria promovendo seu uso.
- SGBDs Orientado a Objetos (SGBDOOs):
  - Paradigma compatível com o desenvolvimento;
  - Indústria em crescimento.
- SGBDs Objeto/Relacional (SGBDORs):
  - Relacional com extensões OO.

Foco na tecnologia relacional

- Este curso foca na tecnologia de bancos de dados relacionais;
- O acesso a SGBDORs é feito de forma semelhante, adicionando algumas facilidades;
- O acesso a bancos OO é muito mais simplificado, visto que o paradigma é o mesmo (persistência transparente);
- Persistência transparente pode ser alcançada também com *frameworks* de mapeamento objeto/relacional.

Abstração de armazenamento

- Em SGBDRs, os dados são armazenados em tabelas:
  - Cada tabela representa uma classe de entidades do sistema;
  - Cada coluna da tabela representa propriedades daquela classe de entidades;
  - Cada linha da tabela representa uma entidade (objeto) existente no sistema;
  - Uma ou mais colunas identificam univocamente cada linha da tabela (são as chaves primárias);
  - Relacionamentos entre tabelas são possíveis por transposição de chaves.

Exemplo de banco de dados

CD:

Id	Gravadora	Artista	Nome	Preço	Duplo?
١	١	١	Crash	R$ ٢٤,٩٠	Não
٢	٢	٢	Construção	R$ ٤٢,٩٠	Não
٣	١	١	Under the Table and Dreaming	R$ ٢٦,٩٠	Não
٤	٣	٣	Música para Acampamentos	R$ ٤٥,٠٠	Sim

Gravadora:

Id	Nome
١	Sony Music
٢	Universal Music
٣	EMI

Artista:

Id	Nome	Banda?
١	Dave Matthews Band	Sim
٢	Chico Buarque	Não
٣	Legião Urbana	Sim

Linguagem de manipulação de dados

- Um SGBD provê três linguagens para manipulação de dados:
  - DDL (*Data Definition Language*): linguagem para definição dos dados – define a estrutura do banco;
  - DML (*Data Manipulation Language*): linguagem de manipulação de dados – permite inserir, atualizar, recuperar e excluir dados do banco;
  - DCL (*Data Control Language*): linguagem de controle de dados – controla aspectos de autorização.
- A grande maioria dos SGBDRs atuais utiliza a linguagem SQL para tudo isso.

SQL – *Structured Query Language*

- Criada pela IBM, padronizada pela ANSI em 1986, ISO em 1987, revisões em 92, 99 e 2003;
- Permite extensões proprietárias;
- Provê comandos de DDL, DML e DCL para bancos de dados relacionais;
  - Apesar do nome ser “Linguagem Estruturada de Consulta”, permite todo o tipo de manipulação.
- Fortemente baseada em álgebra relacional;
- Fácil de escrever e entender (principalmente para quem fala inglês).

SGBDs que usam SQL

- Apache Derby;
- Caché;
- DB2;
- Ingres;
- InterBase;
- MySQL;
- Oracle;
- PostgreSQL;
- Microsoft SQL Server;
- SQLite;
- Sybase;
- Informix;
- Firebird;
- HSQLDB;
- PointBase.

Fonte: Wikipedia

SQL 101

- Veremos conceitos básicos de bancos de dados relacionais com SQL;
- Experimentaremos apenas o suficiente para armazenamento/recuperação simples de dados;
- Usaremos o banco de dados HSQLDB (www.hsqldb.org):
  - Banco de dados totalmente feito em Java;
  - Não é robusto, porém é pequeno (JAR de 624 KB), ideal para aplicações desktop.

Iniciando o HSQLDB

- O HSQLDB pode funcionar em diversos modos:
  - Servidor (protocolo de comunicação proprietário);
  - Servidor *Web* (HTTP);
  - Servlet (HTTP);
  - *Standalone / In-Process* (acesso direto);
  - Somente memória (acesso direto).
- Para iniciar o HSQL:
  - Servidor: `java -cp hsqldb.jar org.hsqldb.Server`
  - Web: `java -cp hsqldb.jar org.hsqldb.WebServer`
  - *Standalone* ou memória: basta usar o driver.

Arquivos de configuração

- A configuração do HSQLDB em modo servidor e *Web* pode ser passada como argumento ou em arquivos de configuração;
- `server.properties`:

```
server.database.0=file:javadiscos
server.dbname.0=javadiscos
```


- `webserver.properties`:

```
server.port=8090
server.database.0=file:javadiscos
server.dbname.0=javadiscos
```

Ferramenta de gerenciamento

- O HSQLDB possui uma ferramenta de gerenciamento bastante rudimentar;
- Para iniciá-la:

```
java -cp hsqldb.jar
org.hsqldb.util.DatabaseManager
```

Conectando ao HSQLDB

- Servidor normal:
  - Tipo: `HSQL Database Engine Server`;
  - URL: `jdbc:hsqldb:hsql://localhost/javadiscos`.
- Servidor *Web*:
  - Tipo: `HSQL Database Engine Web Server`;
  - URL: `jdbc:hsqldb:http://localhost:8090/javadiscos`.

Criação do banco de dados

- Para criar as tabelas, utilizamos comandos de linguagem de definição de dados da SQL;
  - `CREATE DATABASE;`
  - `CREATE TABLE;`
  - `ALTER TABLE;`
  - `DROP TABLE.`
- É mais fácil se utilizarmos uma ferramenta de projeto de banco de dados:
  - DBDesigner, ERWin, Glom, PHPMyAdmin, ferramentas proprietárias do próprio SGBD, etc.

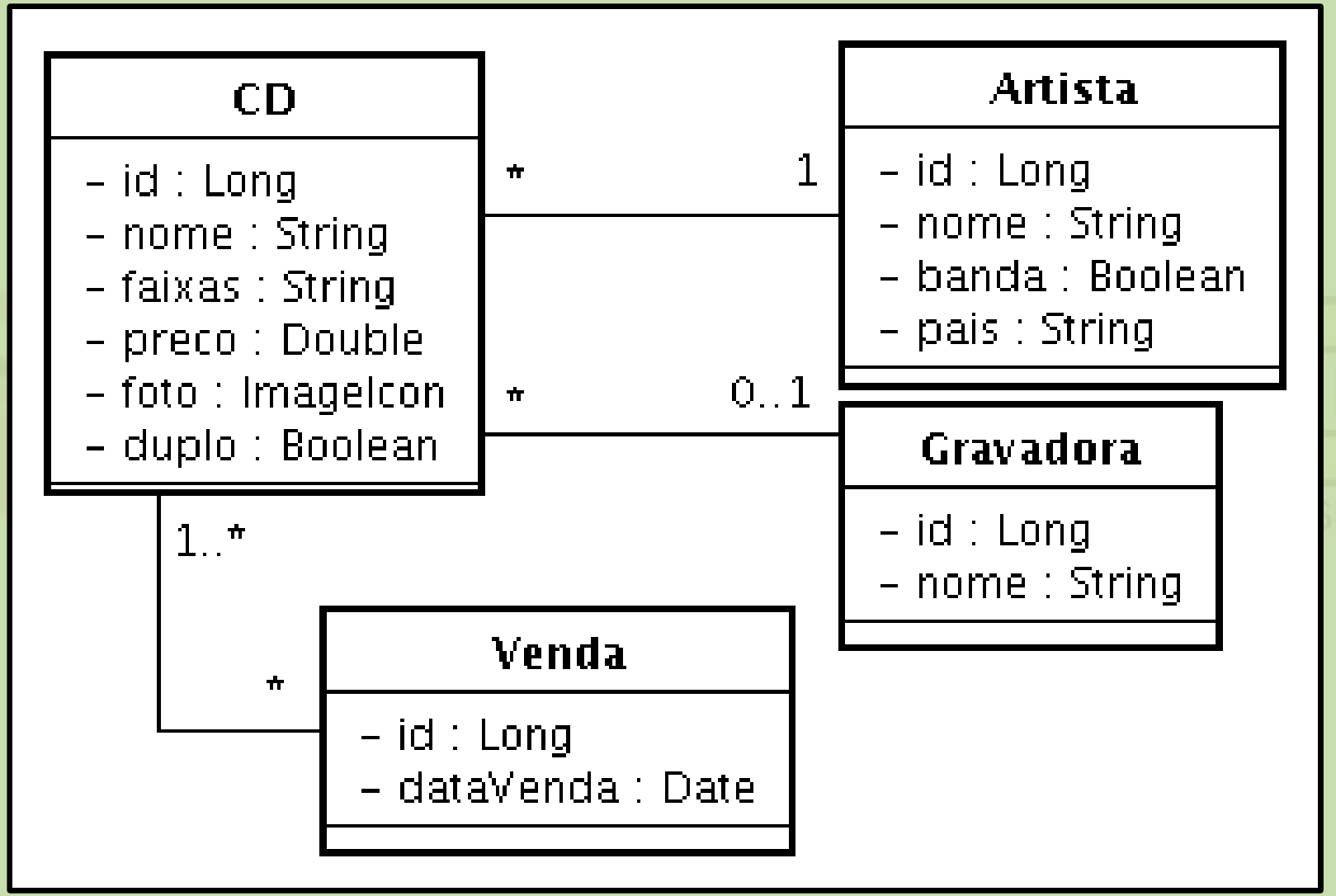
Há várias em Java: http://www.java-source.net/open-source/sql-clients

Modelagem das tabelas

- Tabelas representam entidades do domínio;
- Em sistemas OO, são as classes de negócio:
  - Cada atributo é uma coluna da tabela;
  - Define-se a chave-primária (PK);
  - Relacionamentos são feitos por transposição de PK.
- Definição da chave-primária:
  - Conjunto de atributos que identificam um objeto univocamente (CPF para pessoa, ISBN para livro, ...);
  - Criação de um atributo específico para a PK (recomendado).

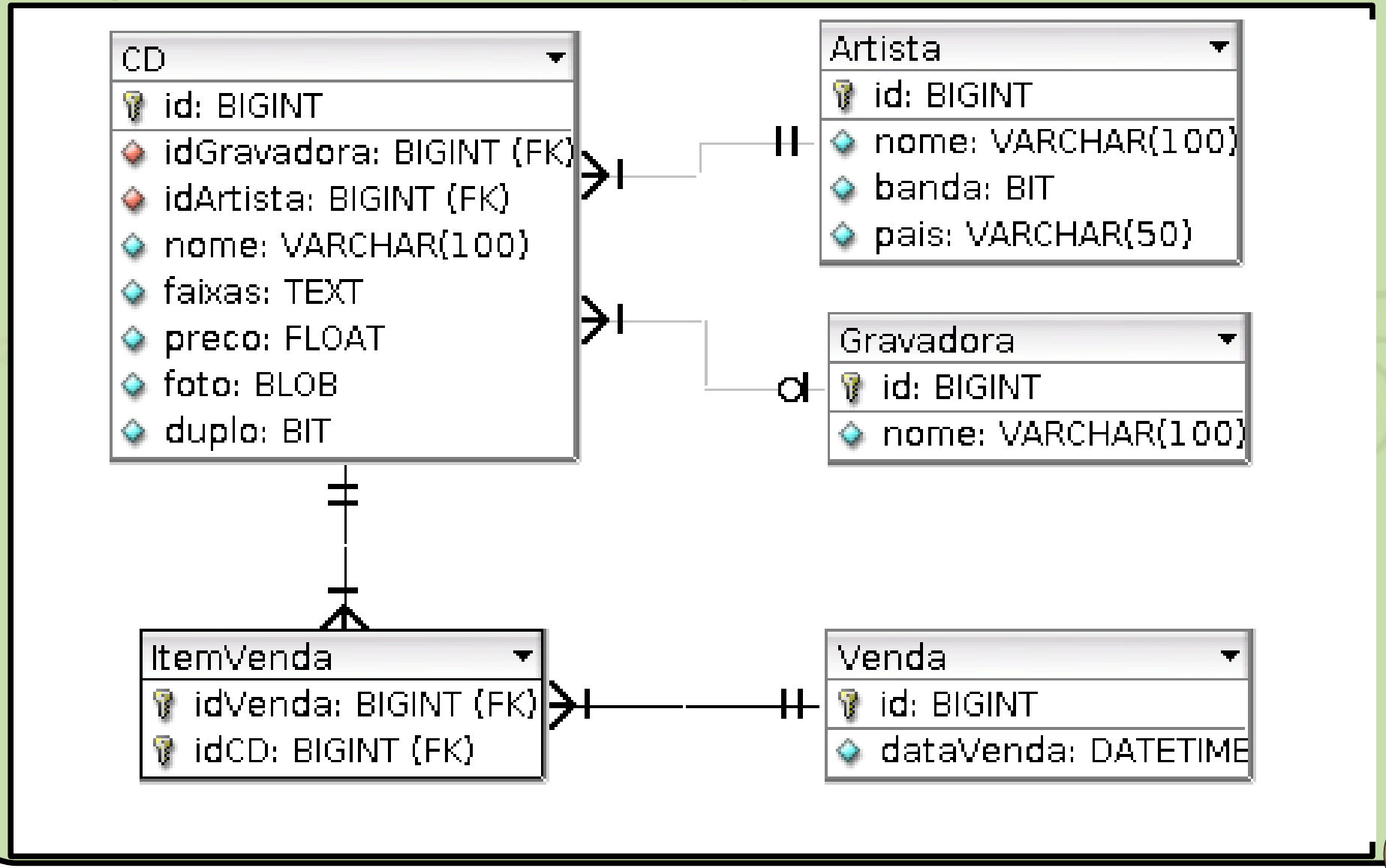
Sistema exemplo: Java Discos

- Usaremos uma loja de CDs como exemplo:



O banco de dados da Java Discos

- Mapeamento de classes para tabelas:



O banco de dados da Java Discos

```
CREATE TABLE Gravadora (
  id BIGINT NOT NULL IDENTITY,
  nome VARCHAR(100) NOT NULL,
  PRIMARY KEY(id)
);
CREATE TABLE Venda (
  id BIGINT NOT NULL IDENTITY,
  dataVenda DATETIME NULL,
  PRIMARY KEY(id)
);
CREATE TABLE Artista (
  id BIGINT NOT NULL IDENTITY,
  nome VARCHAR(100) NOT NULL,
  banda BIT NULL,
  pais VARCHAR(50) NOT NULL,
  PRIMARY KEY(id)
);
```

O banco de dados da Java Discos

```
CREATE TABLE CD (
  id BIGINT NOT NULL IDENTITY,
  idGravadora BIGINT,
  idArtista BIGINT NOT NULL,
  nome VARCHAR(100) NOT NULL,
  faixas LONGVARCHAR NULL,
  preco FLOAT NOT NULL,
  foto BINARY NULL,
  duplo BIT NULL,
  PRIMARY KEY(id),
  FOREIGN KEY(idArtista)
    REFERENCES Artista(id)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);
```

O banco de dados da Java Discos

```
CREATE TABLE ItemVenda (
  idVenda BIGINT NOT NULL,
  idCD BIGINT NOT NULL,
  PRIMARY KEY(idVenda, idCD),
  FOREIGN KEY(idVenda)
    REFERENCES Venda(id)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  FOREIGN KEY(idCD)
    REFERENCES CD(id)
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
);
```

Inserção de dados

- Comando SQL INSERT:

```
INSERT INTO tabela [(coluna [, ...])]
VALUES (expressão [, ...]);
```

- Onde:
  - tabela = nome da tabela na qual serão inseridos os dados;
  - coluna = nome de uma das colunas da tabela;
  - expressão = valor a ser inserido na coluna especificada.

Obs.: SQL não é case sensitive (não diferencia maiúsculas de minúsculas).

Expressões

- Formato depende do tipo:
  - Número inteiro: `0`, `100`, `-50`;
  - Número real: `0.5`, `-4.9`, `29.90`;
  - Booleano: `1`, `TRUE`, `0`, `FALSE`;
  - String: `'Abc'`, `''`, `'Com ''aspas'' simples'`;
  - Data: `'2006-10-07'`, `'1981-06-15'`;
  - Hora: `'18:49:00'`, `'07:15:16'`.

Experimente:

```
-- Insere as gravadoras Sony, Universal e EMI:
INSERT INTO Gravadora (nome) VALUES ('Sony Music');
INSERT INTO Gravadora (nome) VALUES ('Universal');
INSERT INTO Gravadora (nome) VALUES ('EMI');

-- Insere Dave Matthews, Chico Buarque e Legião:
INSERT INTO Artista (nome, banda, pais)
   VALUES ('Dave Matthews Band', TRUE, 'EUA');
INSERT INTO Artista (nome, banda, pais)
   VALUES ('Chico Buarque', FALSE, 'Brasil');
INSERT INTO Artista (nome, banda, pais)
   VALUES ('Legião Urbana', TRUE, 'Brasil');

-- Insere “Crash”, da DMB, por R$ 21.90, não duplo:
INSERT INTO CD (idGravadora, idArtista, nome, preco,
duplo) VALUES (0, 0, 'Crash', 21.90, FALSE);
```

Recuperação de dados

- Comando SQL SELECT simples:

```
SELECT colunas FROM tabela;
```

- Onde:
  - colunas = nomes das colunas cujos dados serão recuperados, separados por vírgulas;
  - tabela = nome da tabela da qual serão recuperados os dados.

Podemos utilizar * para representar todas as colunas.

Experimente:

```
-- Obtém todos os dados de todas as gravadoras.
SELECT * FROM Gravadora;

-- Obtém o nome e o país de todos os artistas/bandas.
SELECT nome, pais FROM Artista;

-- Obtém o nome e o preço de todos os CDs.
SELECT nome, preco FROM CD;
```

- Responda às perguntas:
  - Qual é o código da gravadora Sony Music?
  - Quais CDs foram gravados pela Sony Music?
  - Quais CDs do Chico Buarque são vendidos pela Java Discos?

Critérios de seleção

- Cláusula WHERE:

```
SELECT colunas FROM tabela
WHERE condição;
```

- Onde:
  - condição = expressão lógica (resulta em verdadeiro ou falso).

Operadores & constantes

- Lógicos: NOT, OR, AND;
- Relacionais: =, <, <=, >, >=, <>, !=;
- String:
  - LIKE 'padrão';
  - Uso de coringas: % ou _.
- Outros:
  - IS NULL, IS NOT NULL;
  - BETWEEN x AND y;
  - IN (x1, x2, ..., xn).

Experimente:

```
-- Gravadora com chave-primária = 0.
SELECT * FROM Gravadora WHERE id = 0;

-- Artistas cujo nome termina com a letra e.
SELECT * FROM Artista WHERE nome LIKE '%e';

-- CDs que custam entre R$ 20,00 e R$ 30,00.
SELECT * FROM CD WHERE preco BETWEEN 20 AND 30;
```

- Responda às perguntas:
  - Quais os CDs de Chico Buarque e Marisa Monte?
  - Quais CDs custam menos de R$ 25,00?
  - Quais CDs foram feitos de forma independente (sem gravadora)?

Ordenação de resultados

- Cláusula ORDER BY:

```
SELECT colunas FROM tabela
WHERE condição
ORDER BY colunas;
```

- Onde:
  - colunas = nomes das colunas usadas na ordenação e o critério (ASC = ascendente, DESC = descendente).

Experimente:

```
-- CDs em ordem crescente de nome.
SELECT * FROM CD ORDER BY nome ASC;

-- CDs em ordem decrescente de nome.
SELECT * FROM CD ORDER BY nome DESC;

-- Artistas em ordem crescente de país, depois nome.
SELECT * FROM Artista ORDER BY pais, nome;
```

- Tente ordenar:
  - Bandas em ordem crescente de nome, depois artistas em ordem crescente de nome;
  - CDs por preço (ascendente e descendente).

Junção de tabelas

- Dados de entidades diferentes ficam em tabelas diferentes para evitar redundância;
- Muitas vezes gostaríamos de recuperar dados de diversas tabelas ao mesmo tempo;
- Precisamos fazer junção de tabelas, utilizando as chaves-primárias transpostas;
- Quando houver conflito dos nomes das colunas, devemos usar o nome qualificado:

```
Tabela.coluna
```

Junção de tabelas

- Junção com WHERE:

```
SELECT colunas FROM T1, T2
WHERE T1.id = T2.idT1;
```

- Experimente:

```
-- Nome, preço e artista dos CD.
SELECT CD.nome, preco, Artista.nome
FROM CD, Artista
WHERE CD.idArtista = Artista.id;
```

- Tente juntar também a tabela Gravadora para mostrar a gravadora dos CDs.

Junção de tabelas

- Junção com INNER JOIN:

```
SELECT colunas FROM T1 INNER JOIN T2
ON T1.id = T2.idT1;
```

- Experimente:

```
-- Nome, preço e artista dos CD.
SELECT CD.nome, preco, Artista.nome
FROM CD INNER JOIN Artista
ON CD.idArtista = Artista.id;
```

- Tente juntar também a tabela Gravadora para mostrar a gravadora dos CDs.

Renomeando tabelas e colunas

- Para simplificar nossas consultas, podemos criar aliases para tabelas ou colunas:

```
-- Nome, preço e artista dos CD.
SELECT CD.nome, preco, A.nome AS artista
FROM CD INNER JOIN Artista AS A
ON CD.idArtista = A.id;

-- Podemos omitir o “AS”.
SELECT CD.nome, preco, A.nome artista
FROM CD INNER JOIN Artista A
ON CD.idArtista = A.id;
```

Funções úteis do HSQLDB

- `ABS(num)`: valor absoluto;
- `AVG(coluna)`: média dos valores;
- `SUM(coluna)`: soma dos valores;
- `CEILING(num)`, `FLOOR(num)`: teto e piso;
- `CONCAT(str1, str2)`: concatenação;
- `LENGTH(str)`: tamanho da string;
- `CURRENT_DATE`: data atual;
- `DAYOFMONTH(data)`: dia do mês;
- Dentre muitas outras...

Atualização de dados

- Comando SQL UPDATE:

```
UPDATE tabela SET
col1 = expr1, col2 = expr2, ...
WHERE condição;
```

- Experimente:

```
-- Altera o nome de 'CD Demo' para 'Demo CD'.
UPDATE CD SET nome = 'Demo CD'
WHERE id = 4;
```

- Tente dar 10% de desconto em todos os CDs da Dave Matthews Band!

Exclusão de dados

- Comando SQL DELETE:

```
DELETE FROM tabela
WHERE condição;
```

- Experimente:

```
-- Apaga os CDs com preços abaixo de R$ 20,00.
DELETE FROM CD
WHERE preco < 20.0;
```

- Tente apagar os CDs gravados pela Universal e EMI (dica: use o operador OR ou IN).

Recapitulando...

- Bancos de dados relacionais são o padrão do mercado;
- A linguagem de definição e manipulação de dados padrão é a SQL;
- Aprendemos os seguintes aspectos da SQL:
  - Criar as tabelas;
  - Inserir e atualizar dados;
  - Consultar dados (com critério, com ordenação);
  - Excluir dados.
- Prosseguindo: onde Java entra na história?

Java Database Conectivity

- JDBC é a API padrão da plataforma Java para acesso a bases de dados;
  - Geralmente usada para bancos de dados relacionais.
- O JDK e a JRE da Sun implementam o básico, voltado para aplicações *desktop*:
  - Pacote `java.sql`;
  - Pacote `javax.sql` e subpacotes.

Instalação

- Para programar em Java com acesso a bancos de dados é preciso:
  - Instalar o kit de desenvolvimento Java (opcionalmente, uma IDE);
  - Instalar o SGBD de sua preferência;
  - Obter o *driver* JDBC do seu SGBD.

Uso

- Para consultas (SELECT), são 5 passos:
  1º) Obtenha uma conexão a partir do *driver*;
  2º) Crie um *statement* a partir da conexão;
  3º) Envie uma consulta SQL por meio do *statement*;
  4º) Manipule o conjunto de resultados da consulta;
  5º) Feche o *statement* e a conexão.
- Para outros comandos, pule o passo 4.

O *driver* JDBC

- Cada BD possui um protocolo diferente de comunicação:
  - A API JDBC define somente as interfaces para conexão com o banco;
  - O *driver* do fornecedor implementa o acesso.
- Tipos de *driver*:
  - Tipo 1: ponte JDBC-ODBC (Microsoft);
  - Tipo 2: *drivers* para API nativa (JNI);
  - Tipo 3: clientes para servidores de dados Java;
  - Tipo 4: conexão direta ao banco, implementada em Java puro.

Passo 1: obtendo uma conexão

- Usaremos a classe `java.sql.DriverManager`;
- Obteremos uma `java.sql.Connection`;
- Pré-requisitos:
  - SGBD instalado e funcionando;
  - Banco de dados criado;
  - Ter um login/senha com permissão de acesso;
  - Estar com o *driver* JDBC no CLASSPATH;
  - Saber a URL de conexão com o banco (consulte a documentação do *driver*).

Passo 1: obtendo uma conexão

- Procedimento:
  - Carregar o driver: `Class.forName(nome);`
  - Conectar: `DriverManager.getConnection(url, login, senha).`

```
Connection conn = null;

// URL de conexão com o banco de dados.
String url="jdbc:hsqldb:hsql://localhost/javadiscos";

// Carrega o driver do HSQLDB e conecta.
Class.forName("org.hsqldb.jdbcDriver");
conn = DriverManager.getConnection(url, "sa", "");
```

Passo 2: criação do *statement*

- O *statement* é um objeto que permite a execução de SQL no banco de dados;
- Para obtê-lo, usaremos o método `createStatement()` do objeto `Connection`;
- Retorna `java.sql.Statement`.

```
// Cria o statement.
Statement stmt = conn.createStatement();
```

Passo 3: execução da SQL

- A classe `Statement` possui dois métodos principais para execução de comandos SQL:
  - `executeUpdate(sql)`: retorna o número de linhas afetadas pelo comando;
  - `executeQuery(sql)`: retorna os dados de uma consulta (SQL SELECT).

```
// Monta o comando SQL.
String sql = "INSERT INTO Gravadora (nome) VALUES
('Java Records');";

// Executa o comando no banco de dados.
stmt.executeUpdate(sql);
```

Passo 4: obtendo dados de consultas

- O método `executeQuery()` retorna um objeto que implementa `java.sql.ResultSet`;
- O *result set* (conjunto de resultados) funciona como um iterador pelas linhas retornadas:
  - Ele começa ANTES da primeira linha;
  - O método `next()` movimenta para a próxima linha ou retorna `false` se não há mais linhas;
  - Os métodos `getString()`, `getInt()`, `getDate()`, `getBoolean()`, etc. retornam os dados;
  - Tais métodos podem receber o nome da coluna ou seu índice (começando por 1).

Passo 4: obtendo dados de consultas

```
// Monta a consulta SQL.
String sql = "SELECT CD.nome, preco, A.nome AS " +
   "artista, G.nome AS gravadora FROM CD " +
   "INNER JOIN Artista A ON CD.idArtista = A.id " +
   "INNER JOIN Gravadora G ON CD.idGravadora = G.id;";

// Executa o comando no banco de dados.
ResultSet rset = stmt.executeQuery(sql);

// Navega pelos resultados.
while (rset.next()) {
   String nome = rset.getString("nome");
   double preco = rset.getDouble("preco");
   String artista = rset.getString("artista");
   String gravadora = rset.getString("gravadora");
   System.out.println(nome + " / " + preco + " / " +
   artista + " / " + gravadora);
}
```

Passo 4: obtendo dados de consultas

```
// Monta a consulta SQL.
String sql = "SELECT CD.nome, preco, A.nome AS " +
   "artista, G.nome AS gravadora FROM CD " +
   "INNER JOIN Artista A ON CD.idArtista = A.id " +
   "INNER JOIN Gravadora G ON CD.idGravadora = G.id;";

// Executa o comando no banco de dados.
ResultSet rset = stmt.executeQuery(sql);

// Navega pelos resultados.
while (rset.next()) {
   String nome = rset.getString(1);   // Equivalente!
   double preco = rset.getDouble(2);
   String artista = rset.getString(3);
   String gravadora = rset.getString(4);
   System.out.println(nome + " / " + preco + " / " +
   artista + " / " + gravadora);
}
```

Passo 5: fechando

- Sempre feche o *statement* e a conexão, para não ocupar recursos do sistema.

```
// Fecha o statement e a conexão.
stmt.close();
conn.close();
```

- Não é preciso fechar o *result set*. Ele é fechado automaticamente pelo *statement*;
- Só pode haver um *result set* por *statement*. Para efetuar outra consulta, crie outro *statement*.

Experimente!

- Crie um programa que imprima o nome de todos os CDs de bandas americanas;
- Crie um programa que calcule quanto custa comprar todos os CDs de cantores (não inclua bandas) brasileiros;
- Crie um programa que receba como parâmetros nome e preço do CD, nome do artista e nome da gravadora e cadastre o CD no banco de dados (o artista e a gravadora devem existir).

Uso avançado

- Alguns recursos avançados do JDBC:
  - *Prepared statements*;
  - Meta-dados do conjunto de resultados;
  - Tipos de cursores e concorrência do *result set*;
  - Conjunto de resultados navegável;
  - Uso de *stored procedures*;
  - A interface `RowSet`;
  - Transações.

Prepared statements

- Interface `PreparedStatement`, estende `Statement`;
- Pré-compila a SQL, agilizando sua execução;
- Útil para comandos executados repetidas vezes;
- Uso:
  - `stmt = conn.preparedStatement(sql);`
  - `setString(idx, str)`, `setInt(idx, num)`, etc.;
  - `stmt.executeUpdate()`.

Prepared statements

```
// Cria o statement.
PreparedStatement stmt = conn.prepareStatement("INSERT " +
"INTO Artista (nome, banda, pais) VALUES (?, ?, ?);");

// Monta o vetor de dados.
Object[][] dados = new Object[][] {
      {"Jamiroquai", Boolean.TRUE, "Inglaterra"},
      {"Norah Jones", Boolean.FALSE, "EUA"},
      {"Nirvana", Boolean.TRUE, "EUA"}
};

// Insere todos os dados no banco.
for (int i = 0; i < dados.length; i++) {
   stmt.setString(1, (String)dados[i][0]);
   boolean banda = ((Boolean)dados[i][1]).booleanValue();
   stmt.setBoolean(2, banda);
   stmt.setString(3, (String)dados[i][2]);
   int qtd = stmt.executeUpdate();
   System.out.println(qtd + " linha(s) afetada(s)");
}
```

Meta-dados do *result set*

- Ao realizar uma consulta, é possível obter meta-dados sobre a mesma:
  - Número de colunas retornadas;
  - Nomes das colunas retornadas;
  - Tipo de dado (classe Java) de cada coluna;
  - Etc.
- Uso:
  - Método `getMetaData()` de `ResultSet`;
  - Objeto da classe `ResultSetMetaData`.

Meta-dados do *result set*

```
// Obtém meta-dados.
ResultSetMetaData meta = rset.getMetaData();
int colCount = meta.getColumnCount();
int[] spacing = new int[colCount];

// Imprime um cabeçalho para os resultados.
for (int i = 0; i < colCount; i++) {
   System.out.print('|');
   String colName = meta.getColumnName(i + 1);
   spacing[i] = meta.getColumnDisplaySize(i + 1);
   System.out.print(colName);
   for (int j = colName.length() + 1; j < spacing[i];
      j++) System.out.print(' ');
}
System.out.println();
```

Tipos de cursores

- Um *result set* utiliza um cursor interno que navega pelos resultados;
- Existem tipos diferentes de cursores, representados por constantes de `ResultSet`:
  - `TYPE_FORWARD_ONLY`: só navega para frente;
  - `TYPE_SCROLL_INSENSITIVE`: navega em qualquer direção;
  - `TYPE_SCROLL_SENSITIVE`: navega em qualquer direção e atualiza os dados se houver alteração.

Modo de concorrência do *result set*

- Outras constantes de `ResultSet`:
  - `CONCUR_READ_ONLY`: não pode ser atualizado;
  - `CONCUR_UPDATABLE`: alterações no conjunto refletem no banco de dados.
- Ao criar o *statement*, podemos especificar o tipo do cursor e o modo de concorrência:

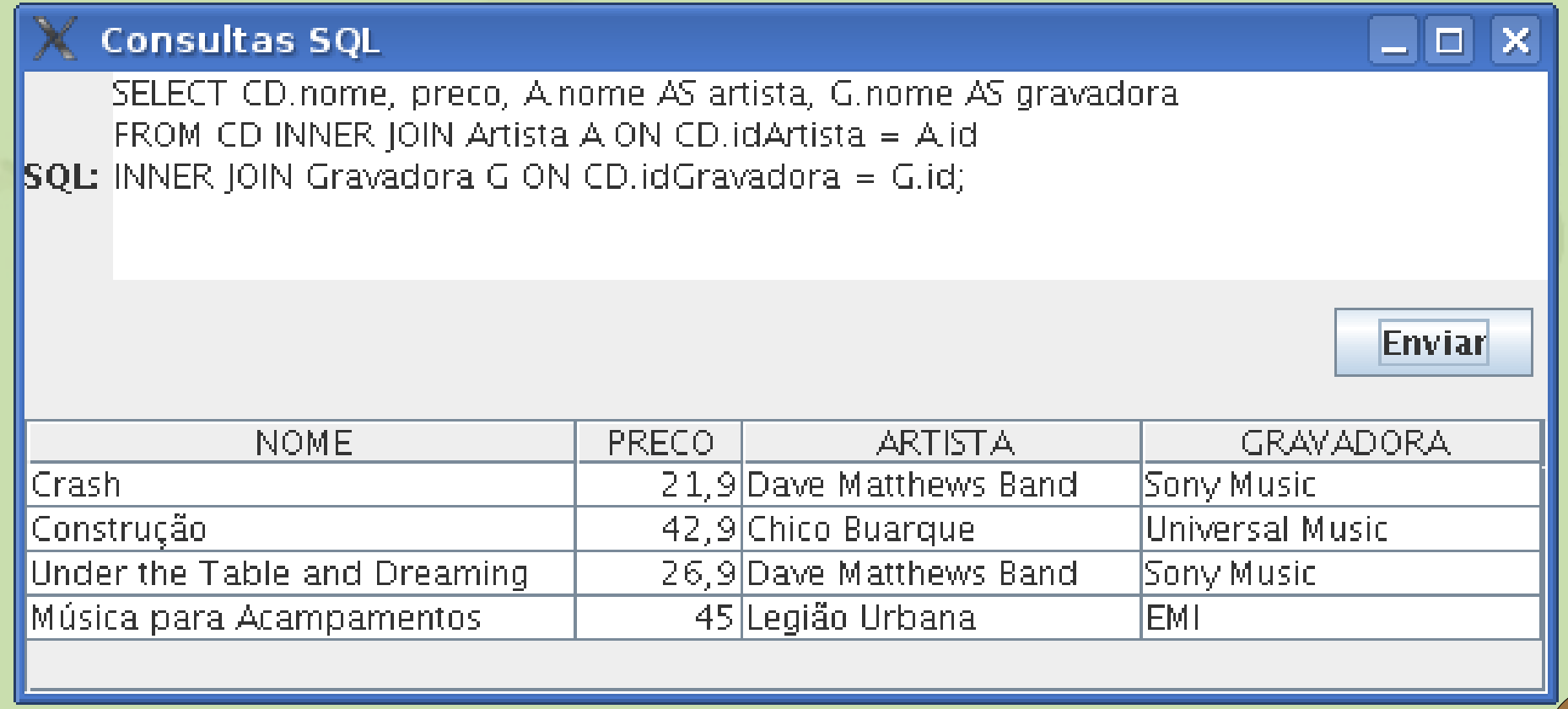
```
// Statement que obtém result sets que navegam em
// qualquer direção e podem ser atualizados.
stmt = conn.createStatement(
   ResultSet.TYPE_SCROLL_SENSITIVE,
   ResultSet.CONCUR_UPDATABLE
);
```

Conjunto de resultados navegável

- Configurando o *result set* como TYPE_SCROLL, podemos navegar em qualquer direção:
  - `next()`: próxima linha;
  - `previous()`: linha anterior;
  - `absolute(linha)`: linha específica;
  - `first()`: primeira linha;
  - `last()`: última linha;
  - `getRow()`: número da linha.

Desafio!

- Construa uma GUI para consultas SQL:
  - Implemente um TableModel que responda aos métodos usando um *result set* e seus meta-dados.



Uso de *stored procedures*

- São procedimentos armazenados no banco de dados e executados a partir de uma chamada;
- Use o método `prepareCall(nome)` da classe `Connection`;
- Retorno é objeto que implementa `java.sql.CallableStatement`;
- `CallableStatement` estende `PreparedStatement`.

A interface RowSet

- Criada no Java 1.4 e melhorada no Java 5.0, facilita a conexão e consultas em BDs;
- Dois tipos principais:
  - `JdbcRowSet`: *result set* automaticamente rolável e atualizável;
  - `CachedRowSet`: *result set* desconectado – lê os dados do banco, desconecta e armazena em cache.
- `RowSet` com cache:
  - Pode ser serializado (enviado pela rede!);
  - Tem limite de quantidade (limite da memória).

A interface RowSet

```
// Cria e executa o RowSet.
JdbcRowSet rowSet = new JdbcRowSetImpl();
rowSet.setUrl(Config.BD_URL);
rowSet.setUsername(Config.BD_LOGIN);
rowSet.setPassword(Config.BD_SENHA);
rowSet.setCommand(sql);
rowSet.execute();

// Navega pelos resultados.
while (rowSet.next()) {
      /* Igual ao ResultSet... */
}
```

Transações

- JDBC funciona por padrão com *commit* automático de transações;
- Para desabilitar, use `setAutoCommit(false)` no objeto `Connection`;
- Transações são iniciadas automaticamente. Para confirmar, use `commit()` em `Connection`;
- Para cancelar, use `rollback()`;
- O seu SGBD e seu *driver* devem ter suporte à transações para funcionar.

Recapitulando...

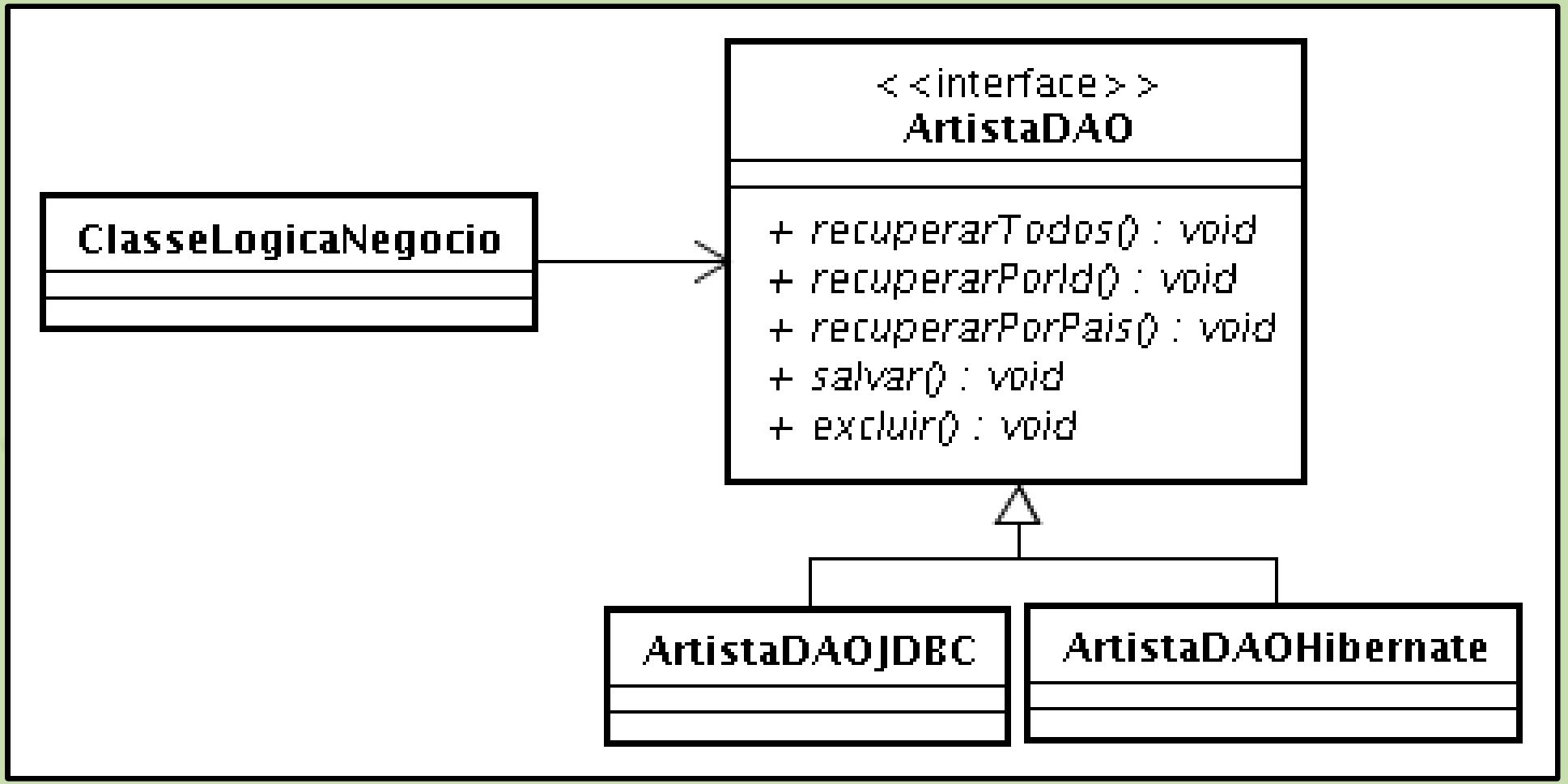
- Vimos muitos conceitos sobre JDBC:
  - O básico (*drivers*, conexão, *statement*, *result set*, ...);
  - Meta-dados;
  - Tipos de navegação e concorrência do *result set*;
  - *Prepared statement* e *stored procedures*;
  - Facilitando o acesso com RowSet.
- Prosseguindo...
  - Como utilizar JDBC de forma organizada?
  - Como realizar a persistência dos meus objetos de negócio?

O padrão de projeto DAO

- Delega-se a uma classe separada a tarefa de persistência de uma classe de negócio;
- Para cada classe de domínio há:
  - Uma interface que define as operações do DAO para aquela classe;
  - Uma implementação para cada tecnologia de persistência (JDBC, Hibernate, etc.).
- Pode ser combinado com o padrão de projeto Fábrica para desacoplar a tecnologia.

Fonte: http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

Exemplo


<<interface>>
ArtistaDAO
+ recuperarTodos() : void
+ recuperarPorId() : void
+ recuperarPorPais() : void
+ salvar() : void
+ excluir() : void
ClasseLogicaNegocio
ArtistaDAOJDBC
ArtistaDAOHibernate

Conclusões

- Vimos nesta parte do curso:
  - Conceitos básicos de SGBDRs e SQL;
  - Conexão a banco de dados em Java com JDBC;
  - O padrão de projeto DAO.
- Próximos passos:
  - Conhecer os *frameworks* de mapeamento objeto/relacional (ORM) – ainda neste curso;
  - Aprender mais sobre SQL e SGBDRs;
  - Investigar a tecnologia de bancos de dados orientada a objetos.