Universidade Federal do ABC

MC0037 – Programação para Web

Aula 2: Persistência / JDBC

2 Q / 2018

Persistência

➢Muitas aplicações precisam armazenar as informações de maneira permanente.

➢Em particular, para aplicações web que acessam imensas quantidades de dados para um grande número de usuários, o gerenciamento desses dados é de fundamental importância para o correto funcionamento do sistema.

➢O processo de armazenamento de dados de forma não volátil é chamado de Persistência.

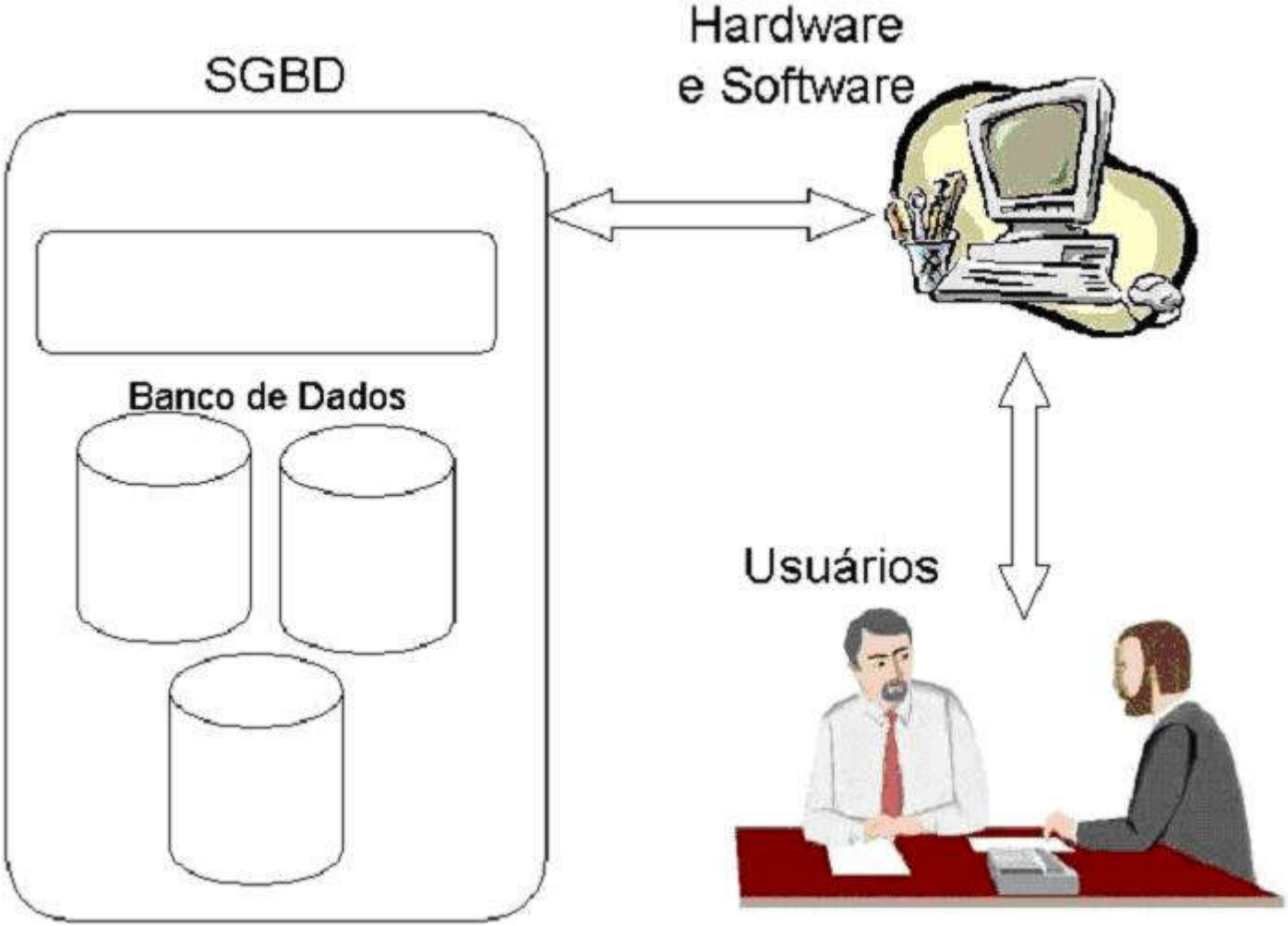
Banco de Dados

➢Uma banco de dados (BD) é uma coleção estruturada de dados relacionados.

➢O tipo mais comum é o banco de dados relacional, inventado por E. F. Codd (IBM) em 1970.

- Os dados são organizados em tabelas, cada linha corresponde a um registro (tupla ou item de dados) e cada coluna um atributo (ou campo) de cada registro.
- Tipicamente usa o SQL (Structured Query Language) para definir, gerenciar e buscar dados.

Banco de Dados



Abstração

➢SGBDs e bancos de dados precisam conhecer o modelo de dados o qual usualmente é representado por um modelo conceitual de dados.

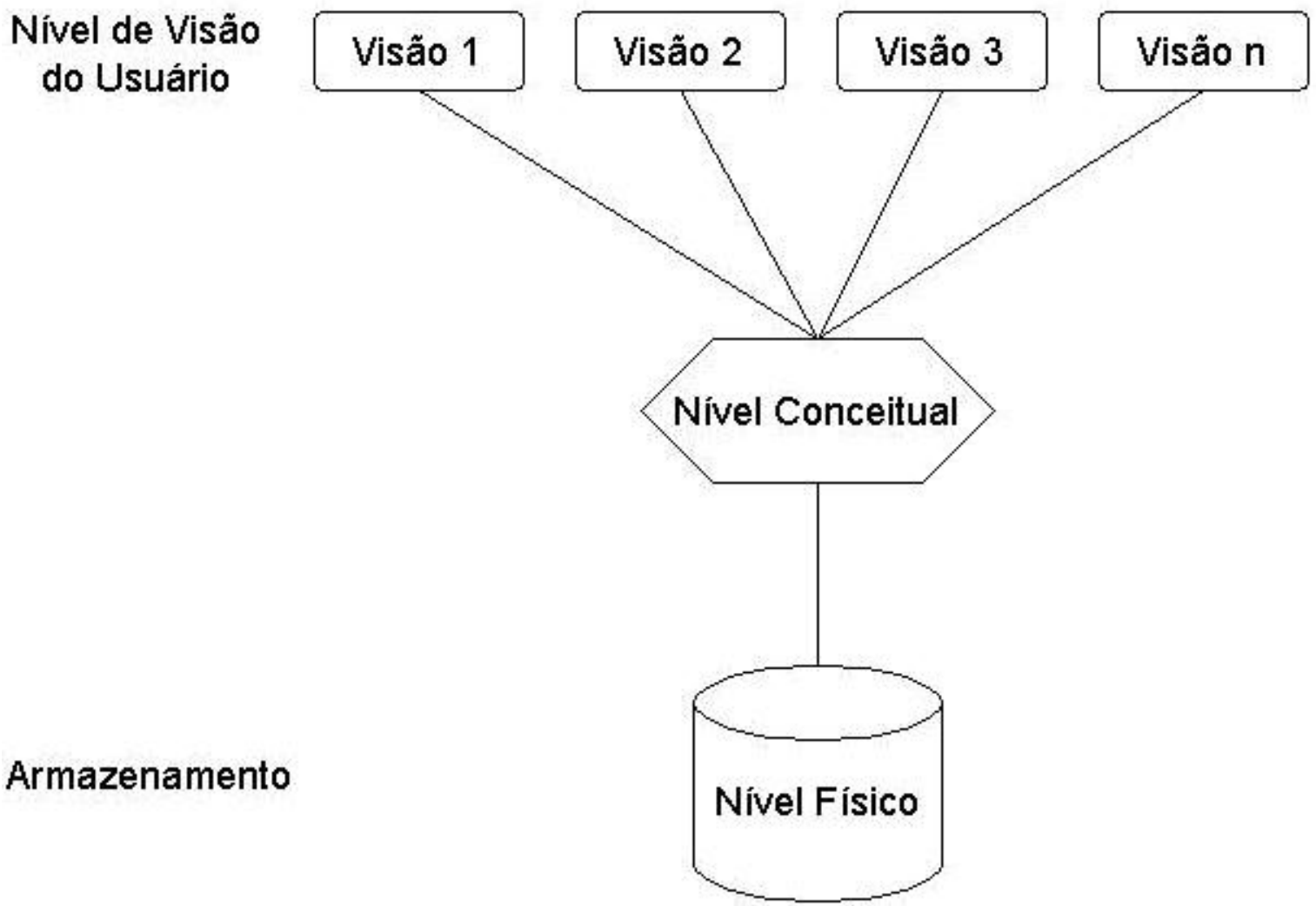
➢Define como os elementos/entidades de nosso sistema serão organizadas no banco de dados e, consequentemente, como os mesmos serão acessados.

Modelo de dados

- Um modelo de dados é uma descrição formal da estrutura de um BD.
- Modelo conceitual: descrição da estrutura do BD independente da implementação (SGBD). Apresenta quais dados podem estar no BD, mas não como estão armazenados de fato.
  - Diagrama Entidade-Relacionamento (ER): representação do modelo conceitual

- Modelo lógico: representa a estrutura de um BD conforme vista pelo usuário do SGBD.
  - Em SGBD relacional, os dados são organizados em forma de tabelas. O modelo lógico define quais as tabelas que o BD possui e para cada tabela, os nomes das colunas.

Modelo de dados



Modelo de dados

- Uma entidade é:
  - "uma coisa ou um objeto no mundo real que pode ser identificada de forma unívoca em relação a todos os outros objetos" (Silberchatz et al'1999)

- Uma entidade é representada por um conjunto de atributos – as propriedades específicas que a descrevem

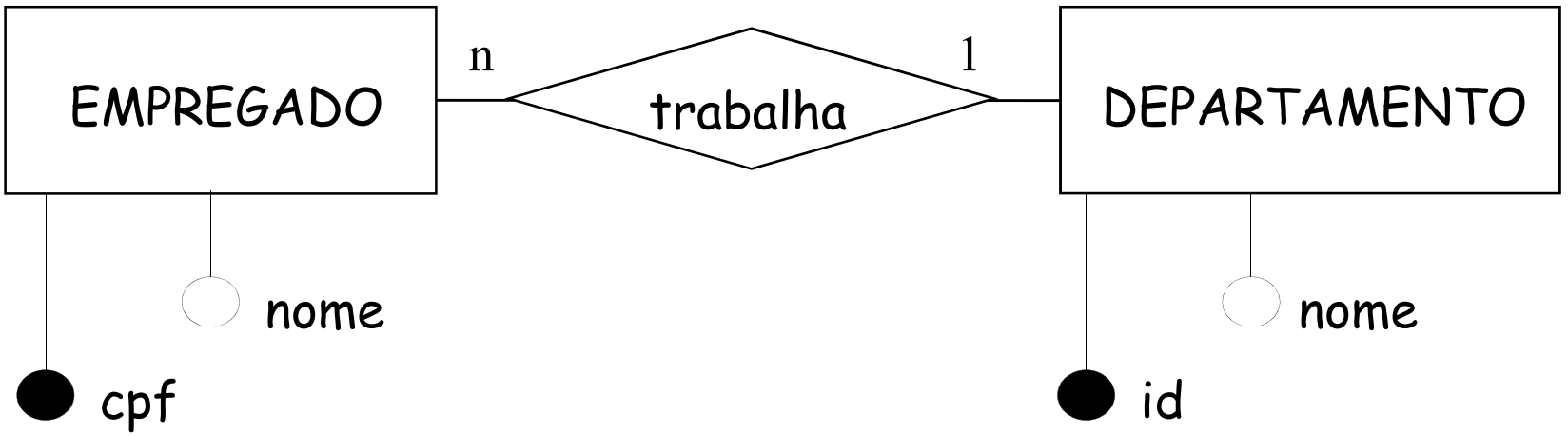
Diagrama Entidade-Relacionamento

- É um modelo de dados conceitual, freqüentemente utilizado para o projeto conceitual de bancos de dados
- Orienta a criação de um esquema conceitual – **Diagrama Entidade-Relacionamento (DER)** – para o banco de dados
- O modelo DER descreve dados como entidades, relacionamentos e atributos

Diagrama Entidade-Relacionamento

➢Exemplo de um **Diagrama Entidade-Relacionamento** DER parcial (Modelo conceitual, independente da implementação)

- ➢ Entidades + Atributos
- ➢ Relacionamentos
- ➢ Cardinalidade



Exercícios

- **Identificar entidades, relacionamentos e cardinalidades**

Deseja-se construir um banco de dados para um sistema de vendas. Em cada venda são vendidos vários produtos e um determinado produto pode aparecer em diferentes vendas. Cada venda é efetuada por um vendedor para um determinado cliente.

SQL

- SGBD: Sistema de Gerenciamento de Bancos de Dados é um conjunto de programas que incorpora as funções de definição, recuperação e alteração em banco de dados.
  - Exemplos de SGBDs populares: Oracle, MySQL, Microsoft SQL Server, PostGreSQL, DB2, Microsoft Access, SQLite, etc.

SQL

- SQL: Structured Query Language
- Linguagem padrão para acessar bancos de dados relacionais.
- Funcionalidades:
  - Criar esquemas/tabelas (comando CREATE)
  - Inserir dados (registros) em tabelas (comando INSERT)
  - Alterar dados na tabela (comando UPDATE)
  - Recuperar dados da tabela (comando SELECT)
  - Remover linhas da tabela (comando DELETE)
  - Remover tabelas, esquemas (comando DROP)
  - Transações: garante que os dados estão sempre em estado consistente
  - Controle: especifica permissões de acesso

SQL

- Criação de tabelas

```
CREATE TABLE nome_da_tabela (
  nome_coluna1 tipo_dado [restrições],
  nome_coluna2 tipo_dado [restrições],
  ...
  restrições_tabela );
```

SQL

- Restrições de domínio
  - **Tipo de dados**

    Ex: cliNome VARCHAR2(40)
  - **Restrição a valores nulos**

    Ex: cliLogradouro VARCHAR2(40) NOT NULL
  - **Valores padrão**

    Ex: cliSexo CHAR(1) DEFAULT ‘M’

SQL

- Restrição de Integridade de Chave
  - **Chave primária**

    Ex: PRIMARY KEY (cliCodigo)

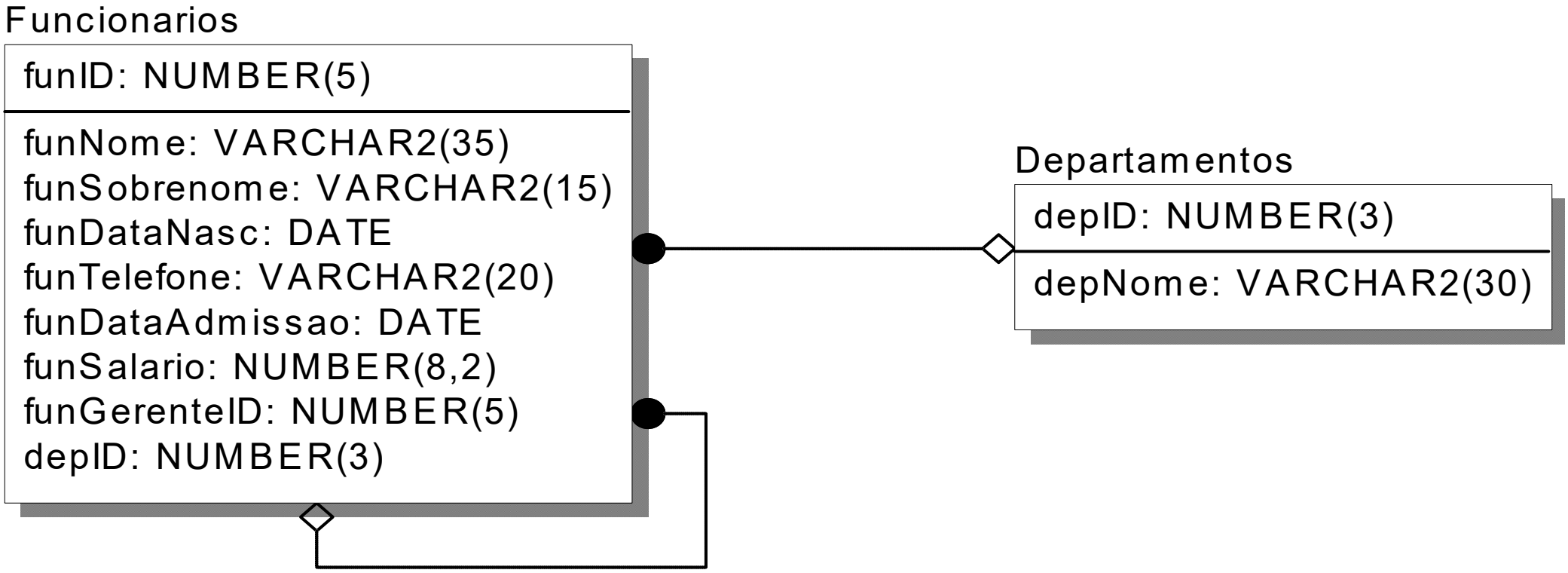
- Restrição de Integridade Referencial
  - **Chave estrangeira**

    Ex: FOREIGN KEY (prodCodigo) REFERENCES Produto (prodCodigo)

SQL

- Tipos de dados
  - **Char(*n*):** caracteres
  - **Date:** data / hora
  - **Float:** valores de ponto flutuante
  - **Integer:** número inteiros
  - **Number (precisão, escala):** a precisão é o número total de dígitos decimais e a escala é o número de dígitos à direita da vírgula decimal
  - **Varchar2(*n*):** armazena caracteres no tamanho máximo de *n*

Exemplo


Funcionarios
funID: NUMBER(5)
funNome: VARCHAR2(35)
funSobrenome: VARCHAR2(15)
funDataNasc: DATE
funTelefone: VARCHAR2(20)
funDataAdmissao: DATE
funSalario: NUMBER(8,2)
funGerenteID: NUMBER(5)
depID: NUMBER(3)
Departamentos
depID: NUMBER(3)
depNome: VARCHAR2(30)

Exercícios

- **SQL**

Com base no diagram apresentado elabore instruções SQL para os seguintes casos:

1. Inserir um departamento
2. Atualizar um funcionário
3. Selecionar funcionários com sobrenome Horita
4. Remover apenas funcionários com salários maiores que R$ 5.000
5. Selecionar apenas funcionários do departamento TI com salários menores que R$ 500

SQL

```
/* Table: Departamentos */

CREATE TABLE Departamentos
(
    depID NUMBER(3)
        CONSTRAINT dep_depID_NN NOT NULL
        CONSTRAINT dep_PK PRIMARY KEY,
    depNome VARCHAR(30)
        CONSTRAINT dep_depNome_NN NOT NULL
);
```

SQL

```
/* Table: Funcionarios */


CREATE TABLE Funcionarios
(
   funID NUMBER(5) NOT NULL,
   funNome VARCHAR(35) NOT NULL,
   funSobrenome VARCHAR(15) NOT NULL,
   funDataNasc DATE,
   funTelefone VARCHAR(20),
   funDataAdmissao DATE NOT NULL,
   funSalario NUMBER(8,2) NOT NULL
      CONSTRAINT fun_funSalario_ck
            CHECK (funSalario > 0),
   ...
```

SQL

- Exclusão de tabelas existentes

```
DROP TABLE nome_da_tabela;
```

SQL

- Inserção de dados em tabelas

```
INSERT INTO nome_da_tabela
(nome_coluna1, nome_coluna2, ...)
VALUES (valor1, valor2, ...);
```

SQL

```
INSERT INTO Departamentos (depID, depNome)
VALUES (10, 'Recursos Humanos');

INSERT INTO Departamentos
VALUES (15, 'CPD');
```

SQL

- Alteração de dados em tabelas

```
UPDATE nome_da_tabela
SET nome_coluna1 = valor1
    [, nome_coluna2 = valor2, ...]
WHERE condição;
```

SQL

```
UPDATE Departamentos
SET depNome = 'Informática'
WHERE depID = 15;

UPDATE Funcionarios
SET funTelefone = '(14) 3642-3030',
    funSalario = funSalario * 1.10
WHERE funID = 100;
```

SQL

- Exclusão de dados em tabelas

```
DELETE FROM nome_da_tabela
WHERE condição;
```

SQL

```
/* Exclusão de todos os dados da tabela */

DELETE FROM Funcionarios;


/* Exclusão de alguns dados da tabela */

DELETE FROM Funcionarios
WHERE depID = 10 OR depID = 30;
```

SQL

- Seleção de dados em tabelas

```
SELECT coluna1, coluna2, ..., colunaN
FROM nome_da_tabela
WHERE condição;
```

SQL

```
/* Seleção de todas as tuplas e colunas de uma tabela */

SELECT * FROM Funcionarios;


/* Seleção de algumas colunas de todas as tuplas de uma tabela */

SELECT funID, funNome
FROM Funcionarios;
```

SQL

```
/* Seleção de todos os funcionários que
trabalham no departamento 30 */

SELECT * FROM Funcionarios
WHERE depID = 30;


/* Seleção de todos os funcionários cujo
nome completo tem a palavra SILVA */

SELECT * FROM Funcionarios
WHERE UPPER(funNome) LIKE '%SILVA%'
   OR UPPER(funSobrenome) LIKE '%SILVA%'
```

SQL

- Seleção de dados em tabelas – ORDER BY

```
SELECT coluna1, coluna2, ..., colunaN
FROM nome_da_tabela
WHERE condição
ORDER BY nome_coluna1, ..., nome_colunaN
    [ASC | DESC];
```

SQL

```
/* Seleção de todos os dados dos funcionários,
ordenados pelo nome por ordem crescente */

SELECT * FROM Funcionarios
ORDER BY funNome [ASC]

/* Seleção de todos os dados dos funcionários,
ordenados alfabeticamente por ordem decrescente */

SELECT * FROM Funcionarios
ORDER BY funNome DESC
```

SQL

```
/* Seleção de todos os dados dos funcionários,
ordenados primeiramente pelo número do
departamento em que trabalham e, em seguida, por
ordem crescente de sobrenome */

SELECT * FROM Funcionarios

ORDER BY depID, funSobrenome
```

SQL

- Operadores de comparação

Igual a	=
Diferente de	<>
Maior que	>
Maior que ou Igual a	>=
Menor	<
Menor que ou Igual a	<=
BETWEEN ... AND ...	Entre dois valores (inclusivo)
IN (conjunto)	Corresponde a qualquer
LIKE	Corresponde a contém
IS NULL	É um valor nulo

SQL

- Operadores lógicos

AND	E
OR	Ou
NOT	Negação

- Uso do NOT com operadores de comparação

IS NOT NULL	Não é um valor nulo
NOT IN (conjunto)	Corresponde a nenhum
NOT BETWEEN	Fora do intervalo de valores
NOT LIKE	Corresponde a não contém

SQL

```
/* Seleção do código de identificação e o nome dos
funcionários que trabalham no departamento 15 e
que ganham R$ 800,00 ou mais */

SELECT funID, funNome
FROM funcionarios
WHERE depID = 15
   AND funSalario >= 800;
```

SQL

```
/* Seleção do ID e do nome completo dos funcionários
cujo salário esteja entre R$ 1500,00 e R$ 3000,00 */


SELECT funID,
       funNome || ' ' || funSobrenome
FROM Funcionarios
WHERE funSalario BETWEEN 1500 AND 3000;
```

SQL

- Seleção de dados em mais de uma tabela

```
SELECT coluna1, coluna2, ..., colunaN
FROM nome_da_tabela
WHERE condição_join;
```

SQL

```
/* USING - Especificando colunas para o join */
/* Seleção de todos os funcionários, inclusive com o
nome do departamento em que trabalham */

SELECT funID, funNome, funSobrenome,
    depID, depNome
FROM Funcionarios JOIN Departamentos
USING (depID)
```

SQL

```
/* JOINS com a cláusula WHERE */
/* Seleção de todos os funcionários, inclusive com o
nome do departamento em que trabalham */

SELECT funID, funNome, funSobrenome,
    f.depID, depNome
FROM Funcionarios f, Departamentos d
WHERE f.depID = d.depID
```

Criando um banco de dados

➢Vamos criar um banco de dados para armazenar os registros de contacorrentes (o nome do banco de dados é progweb):

➢CREATE DATABASE 'progweb';

➢Vamos criar uma tabela de nome contacorrentes para armazenar os dados dos contacorrentes (nome, email, endereço).

➢Para criar a tabela contacorrentes:

➢CREATE TABLE 'contacorrentes' (
'id' INT NOT NULL AUTO_INCREMENT,
'nome' VARCHAR(255),
'email' VARCHAR(255),
'endereco' VARCHAR(255),
PRIMARY KEY ('id')
);

Chave primária
é gerada automaticamente pelo banco de dados

Obs.: Consulte o guia de instalação do MySQL / H2 no Repositório, para dicas de configuração e exemplos de consultas SQL.

Criando um banco de dados

➢No H2, o nome do BD é configurado na tela inicial.

➢Insira o nome do banco de dados (progweb) e o usuário e senha para criar o banco e acessar.

➢Vamos criar uma tabela de nome contacorrentes para armazenar os dados dos contacorrentes (nome, email, endereço).

➢Para criar a tabela contacorrentes:

create table contacorrentes (id integer PRIMARY KEY AUTO_INCREMENT NOT NULL, nome VARCHAR(255), email VARCHAR(255), endereco VARCHAR(255));

Chave primária
é gerada automaticamente
pelo banco de dados

Exemplos de consultas

- Lista todos os registros
- SELECT * from contacorrentes;
- Lista todos os registros ordenados pela coluna nome:
- SELECT * from contacorrentes order by nome;
- Lista todos os registros cujo nome='joao'
- SELECT * FROM contacorrentes WHERE nome='joao';
- Busca o registro cujo id=2

  SELECT nome, email FROM contacorrentes WHERE id=2;

Outros exemplos de busca (considerando as tabelas Empregado/Departamento no slide 7):

- Devolve todos os empregados cujo código do departamento = 1
- SELECT nome FROM Empregado WHERE depto = 1;
- Devolve todos os empregados do departamento de 'vendas'
- SELECT e.nome FROM Empregado e join Departamento d WHERE d.nome = 'Vendas';

Exemplos de consultas

- Insere um novo registro na tabela
- INSERT INTO contacorrentes (nome, email, endereco) VALUES ('platao', 'platao@gmail.com', 'Av. dos Estados');

- Altera o email do registro cujo id=1
- UPDATE contacorrentes SET email='pt@gmail.com' WHERE id=1;

- Remove o registro cujo id=2
- DELETE FROM contacorrentes WHERE id=2;

Obs.: Há algumas diferenças na sintaxe do SQL do MySQL para o H2. Para maiores detalhes da sintaxe dos comandos, consulte o site do

Exemplos de consultas

- Remove uma tabela:
- DROP TABLE contacorrentes;

- Adiciona colunas à tabela
- ALTER TABLE contacorrentes ADD (create_clause1, create_clause2, ...);

- Remove a coluna column_name da tabela contacorrentes
- ALTER TABLE contacorrentes DROP column_name;

Conexão com o banco de dados

➢Após a criação do BD, precisamos de uma forma para possibilitar a comunicação de um programa Java com o BD para que possam ser realizadas as operações desejadas.

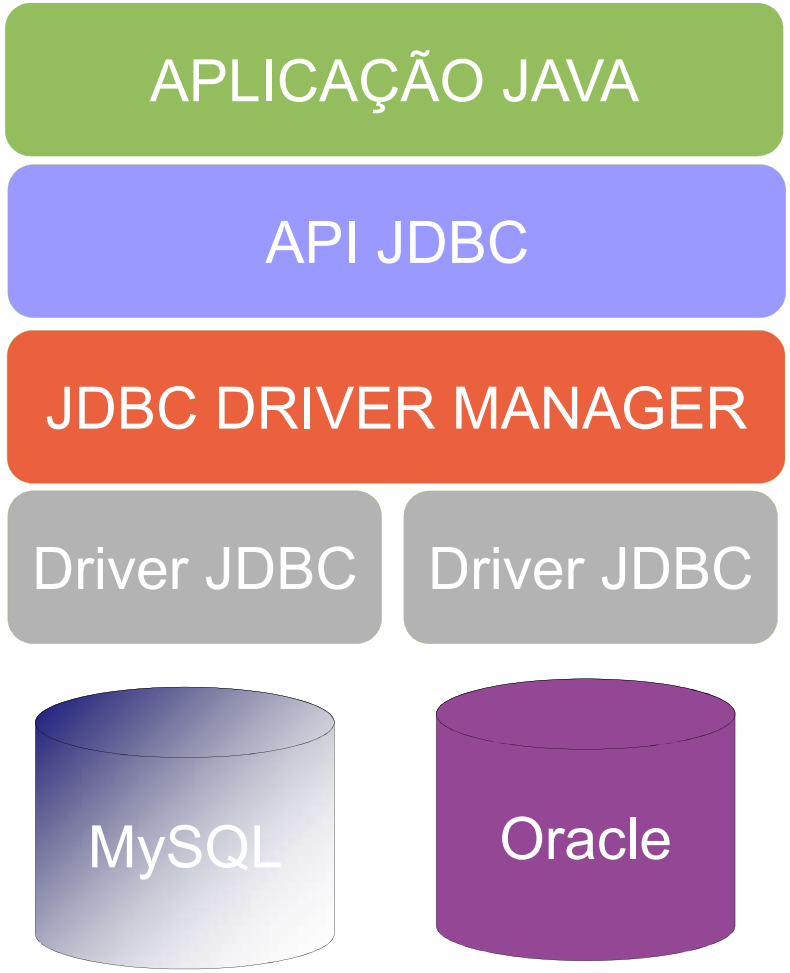
➢Há basicamente duas formas para se fazer isso:

- API básica JDBC (Java Database Connectivity)
- APIs de Mapeamento Objeto-Relacional
  - JPA (Java Persistence API) – especificação padrão do Java
  - Hibernate – framework que implementa JPA (será visto mais pra frente no curso)

JDBC

- JDBC: Java Database Connectivity
- API para possibilitar que uma aplicação Java se comunique com um banco de dados relacional.
- É uma especificação, contém interfaces que permitem, por exemplo, estabelecer uma conexão, executar comandos SQL, etc.
- Uma implementação da API é chamada de driver JDBC e cada banco de dados possui um driver JDBC específico.

JDBC


APLICAÇÃO JAVA
API JDBC
JDBC DRIVER MANAGER
Driver JDBC
Driver JDBC
MySQL
Oracle

JDBC

- Para a criar a conexão com um BD é preciso utilizar um driver JDBC.
- Para obter o driver JDBC para o Postgres acesse:
- https://jdbc.postgresql.org/
  - baixar o arquivo, salvar em algum diretório e descompactar.
- Para o H2, o driver encontra-se na pasta H2/bin, é um arquivo jar (o nome do arquivo da versão mais atual é: h2-1.4.187.jar).
- É necessário adicionar um desses arquivos .jar (dependendo do BD utilizado) na pasta de bibliotecas da aplicação (projeto Java).

Conexão com o banco de dados

➢Para abrir uma conexão com o BD, usar a classe DriverManager do pacote java.sql indicando qual banco de dados queremos conectar:

```
➢DriverManager.getConnection(stringConexao, usuario, senha);
```

➢Exemplo: conectando com o BD progweb:

```
➢// string de conexão
➢String url = "jdbc:postgresql://localhost/progweb";
•    // String url = "jdbc:h2:tcp://localhost/~progweb"; //H2

• // gerando o objeto do tipo Connection
• Connection conexao = DriverManager.getConnection(url,
  "root", "root");
```

Conexão com o banco de dados

- Teste de conexão com o BD

```
public class TestaConexao {
        public static void main(String[] args) {
                Connection conexao = null;
                try {
                        String url = "jdbc:postgresql://localhost/progweb";
                        conexao = DriverManager.getConnection(url, "root", "root");
                // para o H2
                // String url = "jdbc:h2:tcp://localhost/~/progweb";
                // conexao =  DriverManager.getConnection(url,"admin","admin");
                        System.out.println("Conectou!");
                } catch (SQLException e1) {
                        System.out.println("Erro ao abrir a conexao" +
                                                        e1.getMessage());
                } finally {
                        try {
                                conexao.close();
                        } catch (SQLException e2) {
                                System.out
                                .println("Erro ao fechar a conexão" +
                                        e2.getMessage());
                        }
                } } }
```

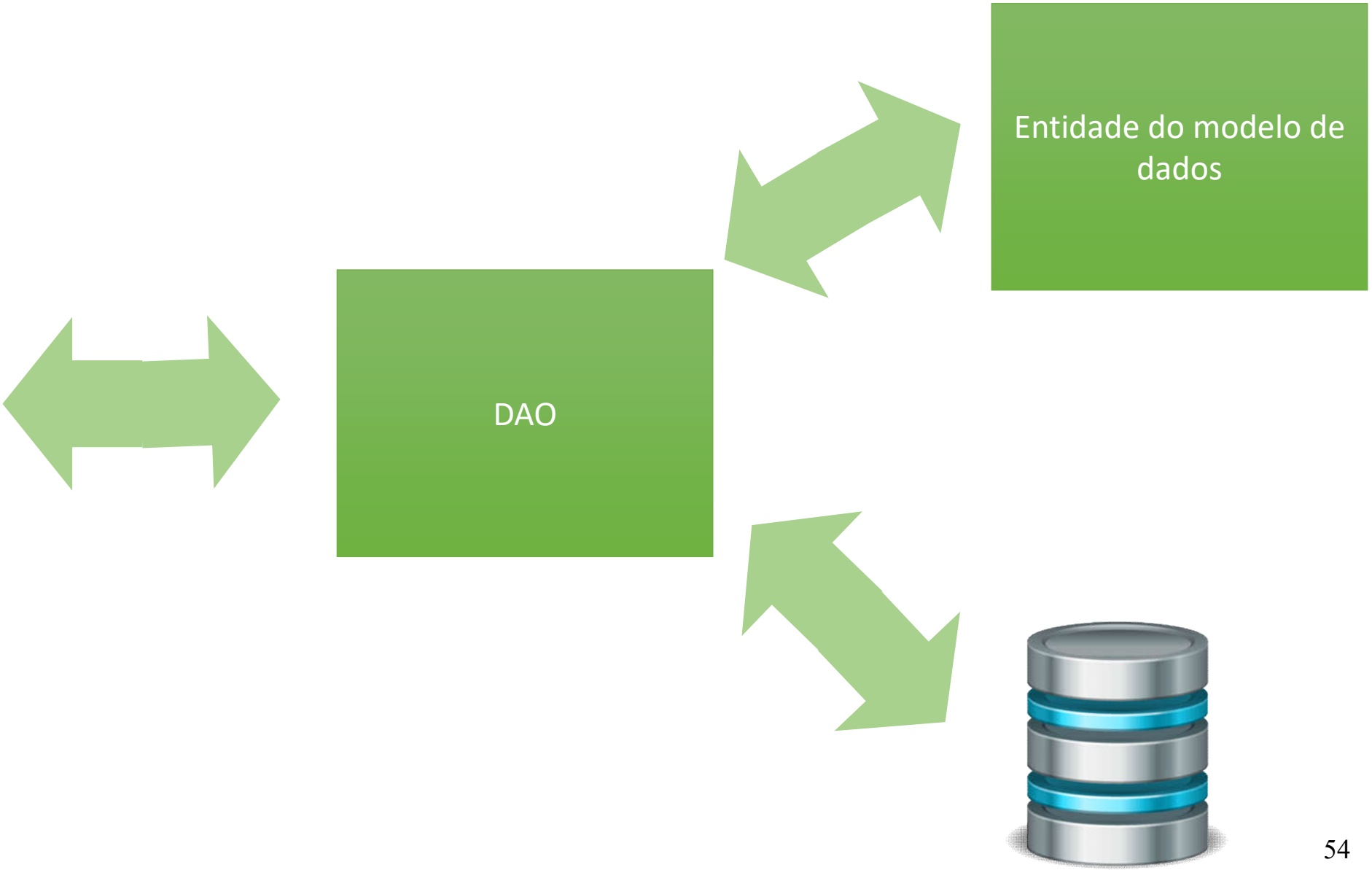
DAO

➢DAO (Data Access Object) é um padrão de projeto do Java EE patterns.

➢A idéia básica do DAO é encapsular as operações básicas de acesso ao BD, conhecidas como CRUD (Create, Read, Update, Delete).

➢Na prática, são classes que tratam exclusivamente das operações do banco de dados.

ORM (Object-relational mapping)



Exemplo: classe ContaCorrenteDAO

➢ Classe ContaCorrenteDAO.java (métodos da classe)

```
public class ContaCorrenteDAO {
	private Connection connection;

	public ContaCorrenteDAO() {
		// cria uma conexao com o BD
		this.connection = new ConnectionFactory().
			getConnection();
	}

	public void insere(ContaCorrente cc) {...}

	public void remove(ContaCorrente cc) {...}

	public void altera(ContaCorrente cc) {...}

	public void getLista(ContaCorrente cc) {...}

}
```

Exemplo: inserção

- Classe ContaCorrenteDAO.java - inserir um registro

```
public void insere(ContaCorrente cc) {
        String sql = "insert into contacorrente (numero)
                        values (?)";

        try { // prepared statement para inserção
                PreparedStatement stmt = connection.
                                        prepareStatement(sql);
                // seta valores
                stmt.setString(1, cc.getNumero());
                // executa
                stmt.execute();
                // fecha statement
                stmt.close();
        } catch (SQLException e) {
                throw new RuntimeException(e);
        }
}
```

Exemplo: remoção

- Classe ContaCorrenteDAO.java - remover um registro

```
public void remove(ContaCorrente cc) {
        try {
                PreparedStatement stmt =
                    connection.prepareStatement(
                    "delete from contacorrente where id=?");
        stmt.setLong(1, cc.getId());
                stmt.execute();
                stmt.close();
        } catch (SQLException e) {
                throw new RuntimeException(e);
        }
}
```

Exemplo: alteração

- Classe ContaCorrenteDAO.java - alterar um registro

```
public void altera(ContaCorrente cc) {
        String sql = "update contacorrente set numero=?
                        where id=?";
        try {
                PreparedStatement stmt =
                        connection.prepareStatement(sql);
                stmt.setString(1, cc.getNumero());
                stmt.setLong(2, cc.getId());
                stmt.execute();
                stmt.close();
        } catch (SQLException e) {
                throw new RuntimeException(e);
        }
}
```

Exemplo: listagem de registros

- Classe ContaCorrenteDAO.java - lista todos os registros

```
public List<ContCorrente> getLista() {
        List<ContaCorrente> ccs = new ArrayList<ContaCorrente>();
        PreparedStatement stmt;
        try {
                stmt = this.connection.prepareStatement("select *
                                from contacorrente order by numero");
                ResultSet rs = stmt.executeQuery();
                while (rs.next()) {
                        ContaCorrente cc = new ContaCorrente();
                        cc.setId(rs.getLong("id"));
                        cc.setNumero(rs.getString(“numero"));
                        ccs.add(cc);
                }
                rs.close();
                stmt.close();
        } catch (SQLException e) {
                throw new RuntimeException(e);
        }
        return ccs;
}
```

Acessando os métodos do DAO

- Testes: classe CriaContaCorrente (método main)

```
public class Application {
	public static void main(String[] args) {
		ContaCorrente cc = new ContaCorrente();
		cc.setNumero(“12323");

		try {
			ContaCorrenteDAO dao = new ContaCorrenteDAO();
			// gravando registro
			dao.insere(cc);
			System.out.println("Inserção OK!");
			// alterando um registro
			cc.setEmail(“098765");
			dao.altera(cc);
			System.out.println("Alteração OK!");

			// continua ...
```

Acessando os métodos do DAO

- Testes: classe CriaContaCorrente (método main) - continuação

```
                        // listando os registros
                        List<ContaCorrente> ccs = dao.getLista();
                        for (ContaCorrente cc1 : ccs) {
                                System.out.println("Número: " +
                                        cc1.getNumero ());
                } catch (Exception e) {
                        System.out.println("Ocorreu um erro! " + e);
                }
        }
}
```

Criando uma Fábrica de conexões

- Podemos encapsular um código repetitivo em uma classe.
- Uma fábrica de conexões é responsável por criar conexões com o BD.
- O método getConnection() devolve um objeto da classe Connection.
- Facilidade de manuntenção: quando precisar mudar a forma para obter conexões (ou mesmo a mudança de driver do banco de dados).
- Implementa o padrão *Factory** (encapsulamento da construção de objetos).

- * Padrões de Projeto, Erick Gama e outros autores.

Criando uma Fábrica de conexões

```
public class ConnectionFactory {

        public Connection getConnection() {
                System.out.println("Conectando ao banco de dados");
                try {
                        String url = "jdbc:postgreqsql://localhost/progweb";
                        return DriverManager.getConnection(url, "root",
                                                "root");
                // para o H2
                // String url = "jdbc:h2:tcp://localhost/~/progweb";
                // return  DriverManager.getConnection(url,"admin",
                                "admin");

                } catch (SQLException e) {
                        throw new RuntimeException(e);
                }
        }
}
```

Referências

- Deitel, H. M. e Deitel, P. J.; *JAVA – Como Programar*; 6ª edição, Editora Pearson Prentice-Hall, 2005.

- Heuser, C. A., Projeto de Banco de Dados, Ed. Bookman, 4ª edição, 2008.

- Sintaxe SQL no H2:
- http://www.h2database.com/html/grammar.html