Aula 022

Professores:

Geraldo Xexéo

Geraldo Zimbrão

Conteúdo:

Conversão OO para Relacional

Fundação CECIERJ
Consórcio cederj

Roteiro

- Introdução
- Conversão do Modelo OO para Relacional
  - Identidade
  - Classes
  - Atributos
  - Métodos
  - Herança
  - Relações - associação, agregação e composição
  - Restrições
- Considerações finais

Introdução

- O Modelo Relacional é basedo em tabelas, atributos e relacionamentos
  - O modelo OO (classes e objetos) é semanticamente mais rico que o relacional
  - É possível convertê-lo para um modelo relacional
  - Há vários padrões de conversão bem simples e diretos
  - Há várias ferramentas que fazem essa conversão de forma transparente
    - Java: Hibernate

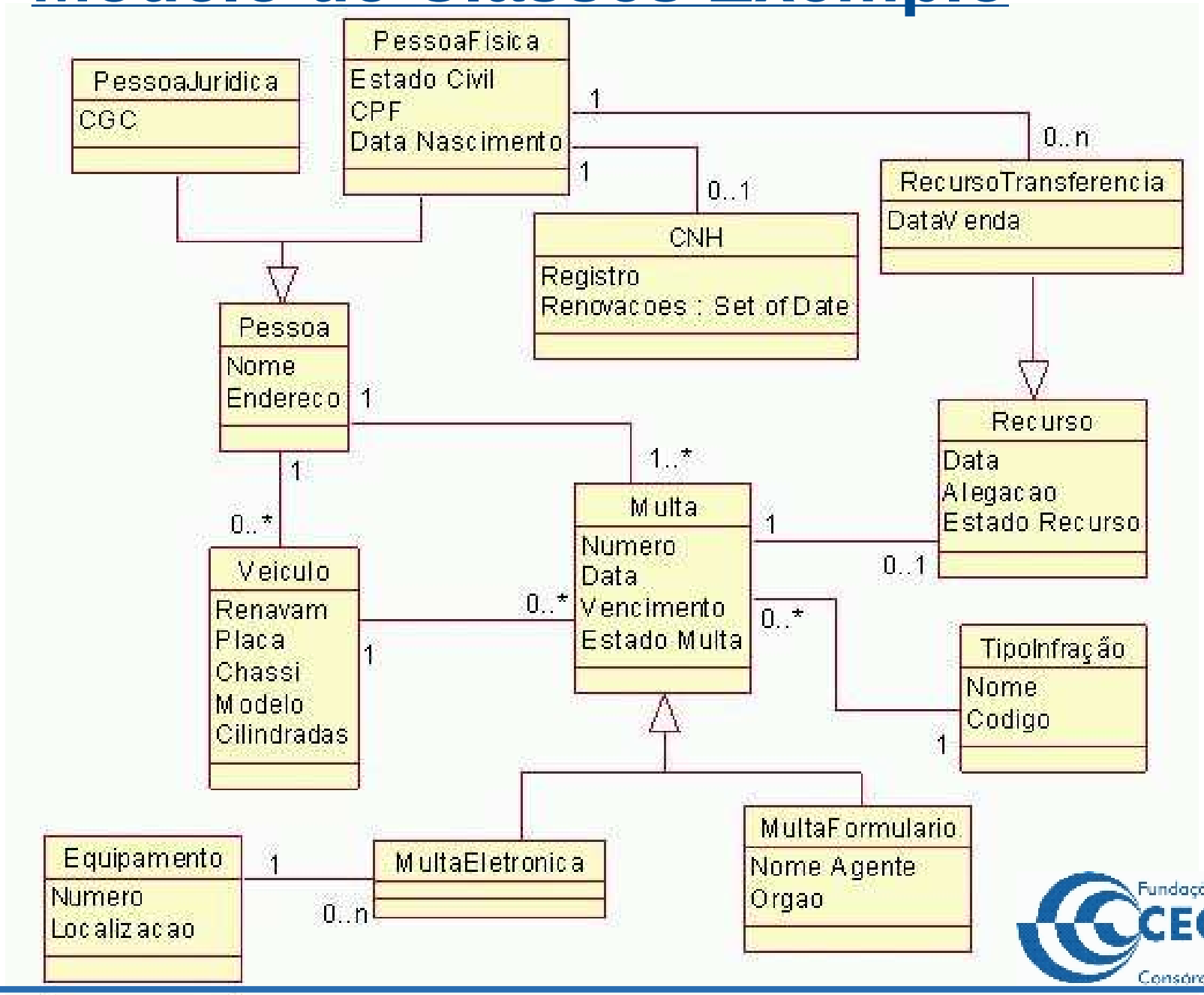
Conversão de um Modelo OO para Relacional

- As construções do Modelo OO serão mapeadas para tabelas, atributos, relacionamentos e restrições
  - Chaves primárias artificiais serão utilizadas para os identificadores
  - Os métodos serão mapeados para procedimentos

Pontos Principais

Construção	Mapeamento
Identidade	Chaves primárias artificiais
Classes	Tabelas
Atributos	Colunas de tabelas
Métodos	*Stored procedures* ou funções fora do banco de dados
Relações	Chaves estrangeiras e tabelas de relacionamento
Herança	Tabelas, visões e particionamentos horizontais ou verticais

Modelo de Classes Exemplo



Identificadores

- Usaremos chaves primárias artificiais formadas por um atributo inteiro sequencial (surrogate)
  - Usualmente chamado de ID
  - Apenas esse atributo deve ser utilizado como chave estrangeira nas restrições de integridade
  - Outras chaves alternativas são permitidas
  - Um gerador de números sequenciais deve ser providenciado
  - IDs nunca mudam de valor
    Não se reaproveita IDs de objetos removidos

Classes

- Cada classe simples será mapeada para uma tabela também simples
  - São aquelas cujos atributos podem ser diretamente mapeados para os tipos básicos do SGBD;
  - Não fazem parte de nenhuma hierarquia de classes;
  - Não possuem atributos listas ou conjuntos (multivalorados)

Criação da Tabela

```
Create table Veiculo
(
  ID integer not null primary key,
  Renavam char(9) not null unique,
  Placa char(7) not null unique,
  Chassi char(18) not null unique,
  Modelo varchar(40) not null,
  Cilindradas integer not null
);
```

- Estabelecemos os tipos refinando o modelo
  - Detalhar os requisitos
  - Várias chaves alternativas

Atributos

- Quatro características são muito importantes
  - Tipo
  - Cardinalidade
  - Persistência
  - Visibilidade

Tipos para Atributos

- Tipos Simples
  - Correspondência direta com os tipos do SGBD
- Faixas de valores
  - Utilizar restrições do tipo check:
    - *CHECK(CILINDRADAS BETWEEN 50 AND 6000)*
- Tipos enumerados
  - Tipos que podem ter um número finito de valores
    - Campos alfanuméricos combinados com faixas
    - Campos inteiros combinados com tabelas auxiliares

Tipos Enumerados - 1

```
Create table Pessoa
(
  ID integer not null primary key,
  Nome varchar(80) not null,
  EstadoCivil varchar(10) not null,
  ...
  Check( EstadoCivil in ( 'Solteiro',
                          'Casado',
                          'Separado',
                          'Divorciado',
                          'Viúvo' )
);
```

Prós

- Simples de implementar
- Bom desempenho
- Falicita a escrita de consultas

Contras

- Prejudica a manutenção
- Dificulta a escrita da aplicação
  - Como descobrir os possíveis valores?

Tipos Enumerados - 2

```
Create table Pessoa
(
  ID integer not null primary key,
  Nome varchar(80) not null,
  EstadoCivil integer
    references TipoEstadoCivil(id),
  ...
);

Create table TipoEstadoCivil
(
  ID integer not null primary key,
  Valor varchar(10) not null unique
);
```

- Vantagens
  - Manutenção simples
  - Facilita à aplicação descobrir os possíveis valores do tipo
- Desvantagens
  - Requer uma junção a mais
  - Proliferação de tabelas "Tipo"

Tipos Enumerados - 3

```
Create table Pessoa
(
  ID integer not null primary key,
  Nome varchar(80) not null,
  EstadoCivil integer
    references TiposEnumerados(id),
  ...
);

Create table TiposEnumerados
(
  ID integer not null primary key,
  Tipo varchar(32) not null,
  Valor varchar(32) not null
);
```

Vantagens

- Manutenção simples
- Facilita à aplicação descobrir os possíveis valores da faixa
- Apenas uma tabela de tipos

Desvantagens

- Requer uma junção a mais
- Verificação de consistência é mais complexa

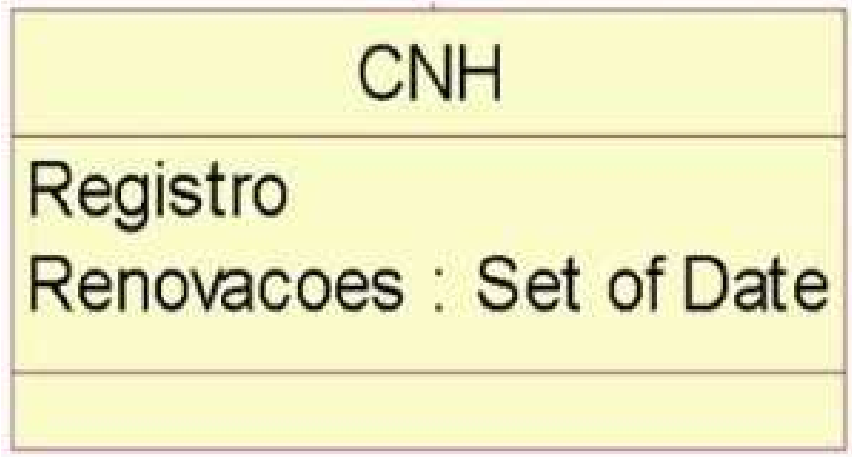
Cardinalidade de Atributos

- A cardinalidade de um atributo pode ser zero, um ou mais de um (coleção)
  - Restrições *not null* nos dois primeiros casos
  - Tabela auxiliar no terceiro, com chave estrangeira para a tabela da classe principal
  - Se a coleção for sem repetição, restrição de unicidade com a chave primária e estrangeira na tabela auxiliar

Cardinalidade

```
Create table CNH
(
  ID integer not null primary key,
  Registro char(20) unique not null
  ...
);

Create table RenovacoesCNH
(
  CNH_ID integer not null
    references CNH(ID)
    on delete cascade,
  Data date not null,
  unique( CNH_ID, Data )
);
```



- A tabela RenovacoesCNH existe apenas para representar o atributo Renovacoes
- *On delete cascade* faz com que se um objeto CNH for removido, sua coleção de renovações também o será

Persistência e Visibilidade

- Nem todos os atributos de uma classe devem ser armazenados
  - Alguns são calculados (derivados)
    - Utilizar funções
    - Utilizar visões
- Não há alternativas satisfatórias em um SGBD relacional para a implementação de restrições de visibilidade
  - Usar o mecanismo de controle de acesso do SGBD

Métodos

- Não há muito o que fazer em relação aos métodos
  - *Triggers* e *stored procedures*
  - Funções auxiliares em bibliotecas compartilhadas pelas aplicações

Relacionamentos

- Pouca diferença entre a técnica usada para converter o modelo ER pra o relacional
  - Todas as tabelas possuem uma chave primária inteira, o ID
  - Todos os relacionamentos irão utilizar uma chave estrangeira para esse ID
  - Não há diferença na implementação de associações, agregações e composições
  - Mesmas soluções do mapeamento ER para a cardinalidade dos relacionamentos

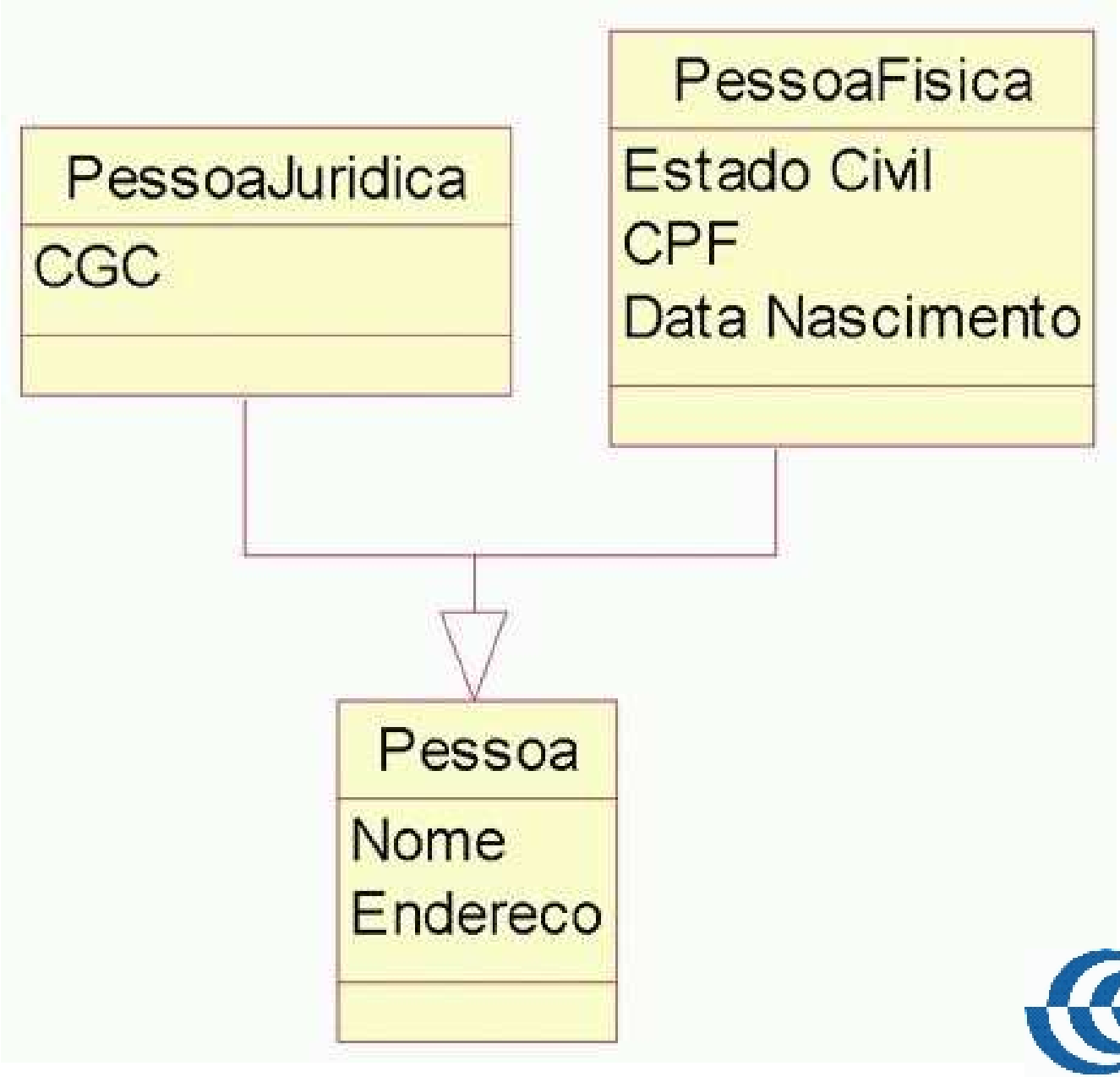
Herança

- Três opções básicas diferentes
  - Uma única tabela para toda a hierarquia
  - Várias tabelas com fragmentação vertical
  - Várias tabelas com fragmentação horizontal
- Cada uma apropriada a um caso específico
- A escolha "errada" pode afetar significativamente o desempenho
- Difícil evoluir o modelo alterando a escolha
  - Reescrever muitas consultas

Tabela Única

- Criamos uma única tabela para toda a hierarquia de classes
  - Um atributo **tipo** para indicar qual a classe de cada objeto
  - Vantagens
    - Abordagem mais simples
    - Facilita a escrita de consultas
    - Todos os atributos de todas as classes da hierarquia estarão nessa tabela
  - Desvantagens
    - Desperdício de espaço
    - A verificação da consistência da classe com os atributos é complexa
    - A verificação da consistência dos relacionamentos é mais complexa ainda

Hierarquia com Tabela única - 1



Hierarquia com Tabela única - 2

```
Create table Pessoa
(
  ID integer not null primary key,
  Tipo char(1) not null,
  Nome varchar(50) not null,
  Endereco varchar(60) not null,
  CPF char(8) unique,
  DataNascimento date,
  CGC char(12) unique,
  Check( Tipo in ('F','J') ),
  Check( (Tipo = 'F' and CPF is not null and DataNascimento is not null) or
           (Tipo <> 'F' and CPF is null and DataNascimento is null) ),
  Check( (Tipo = 'J' and CGC is not null) or
           (Tipo <> 'J' and CGC is null)
);
```

Hierarquia com Tabela única - 3

- Várias restrições para verificar a consistência dos atributos com a classe
- Não temos como garantir facilmente que uma CNH estará ligada a uma Pessoa Física
  - A tabela é Pessoa
  - Uso de triggers
  - Funções auxiliares na aplicação

Fragmentação Vertical

- Um objeto será "cortado" em duas ou mais partes
  - Uma tabela pra cada classe da hierarquia com os atributos correspondentes
  - É como se a tabela da primeira abordagem fosse "cortada" na **vertical**
  - Não há necessidade do atributo **Tipo**
  - A chave primária de cada subclasse será também chave estrangeira da tabela que representa a classe imediatamente superior na hierarquia

Hierarquia com Fragmentação Vertical - 1

```
Create table Pessoa (
  ID integer not null primary key,
  Nome varchar(50) not null,
  Endereco varchar(60) not null
);

Create table PessoaFisica (
  ID integer not null primary key references Pessoa on delete cascade,
  CPF char(8) not null unique,
  DataNascimento date not null
);

Create table PessoaJuridica (
  ID integer not null primary key references Pessoa on delete cascade,
  CGC char(12) not null unique
);
```

Hierarquia com Fragmentação Vertical - 2

- Vantagens
  - Fácil manutenção, alterações localizadas
  - Não há necessidade de verificar a consistência dos atributos com a classe
  - Posível representar os relacionamentos com as subclasses diretamente
    - CNH irá possuir uma chave estrangeira para a tabela PessoaFisica
- Desvantagens
  - Escrita de consultas envolve junções
  - Desempenho

Fragmentação Horizontal

- Corresponde a realizar um "corte" horizontal na tabela da primeira abordagem
  - Apenas as classes concretas serão representadas no modelo
    - Classes que possuem instâncias
- Os atributos que existirem nas superclasses de cada classe serão repetidos nas tabelas que representam as classes concretas

Hierarquia com Fragmentação Horizontal - 1

```
Create table PessoaFisica
(
  ID integer not null primary key,
  Nome varchar(50) not null,
  Endereco varchar(60) not null
  CPF char(8) not null unique,
  DataNascimento date not null
);

Create table PessoaJuridica
(
  ID integer not null primary key,
  Nome varchar(50) not null,
  Endereco varchar(60) not null
  CGC char(12) not null unique
);
```

Hierarquia com Fragmentação Horizontal - 2

- Vantagens
  - Cada objeto está contido em uma única tabela
  - Não é necessário junções para escrever as consultas
- Desvantagens
  - A manipulação da superclasse se torna complicada
    - União de várias tabelas
  - Relacionamentos para a superclasse não podem ser implementados com chave estrangeira
    - Multa se relaciona com Pessoa
  - Manutenção trabalhosa

Restrições

- As restrições do modelo de classes devem ser implementadas utilizando-se
  - Restrições do próprio SGBD
  - *Triggers* e *Procedures*
  - Funções de biblioteca na aplicação

Considerações Finais

- Os mecanismos presentes no SGBD devem ser utilizados sempre que possível
- O grau de fidelidade ao modelo de classes deve sempre ser levado em conta nas escolhas
  - Balanceado com a relação custo / benefício de cada alternativa
  - Nem todas as normalizações propostas pelo modelo relacional devem ser realizadas para não distanciar o modelo relacional final do modelo de classes