Engenharia de Software III

Aula 8
Persistência de Dados

l.bertholdo@ifsp.edu.br

Conteúdo

- Persistência de Dados
- Abordagens de Persistência de Dados
- Mapeamento Objeto-Relacional (ORM)
- Técnicas de Mapeamento Objeto-Relacional
- *Frameworks* ORM

l.bertholdo@ifsp.edu.br

Persistência de Dados

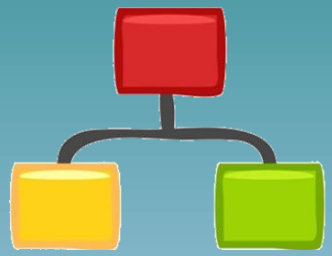
- No contexto de software, no qual temos objetos representando as mais diversas entidades (pessoas, produtos, clientes etc), **persistência** significa a possibilidade de esses objetos perdurarem em um meio de armazenamento **não volátil** e **externo** à aplicação que os criou.

- Os **bancos de dados relacionais**, concebidos na década de 1970, ainda continuam sendo o meio mais usado para essa função, embora outras abordagens de persistência tenham surgido nas últimas décadas.

*Apesar do sucesso dos **bancos de dados relacionais**, quando pensamos na questão da persistência em **aplicações orientadas a objetos**, existe uma grande diferença entre os conceitos envolvidos. Os bancos de dados foram construídos para armazenar dados, ao passo que, no paradigma orientado a objetos, além de dados (atributos), temos comportamentos (métodos) e outras estruturas e relacionamentos complexos.*

l.bertholdo@ifsp.edu.br

Persistência de Dados



- Ao longo dos anos, alguns bancos de dados relacionais foram adicionando recursos da orientação a objetos, como a **herança**.
- Ainda assim a estrutura relacional, representada pela relação entre tabelas, nas quais os dados residem em linhas e colunas, é incapaz de retratar com precisão a complexidade do mundo orientado a objetos.
- Desse modo, torna-se necessário utilizar alguma abordagem para compatibilizar esses dois modelos (de objetos e relacional).
- Isso geralmente é feito por meio de técnicas de **mapeamento objeto-relacional**, que têm como objetivo final viabilizar a correta **persistência** de objetos em bancos de dados relacionais.

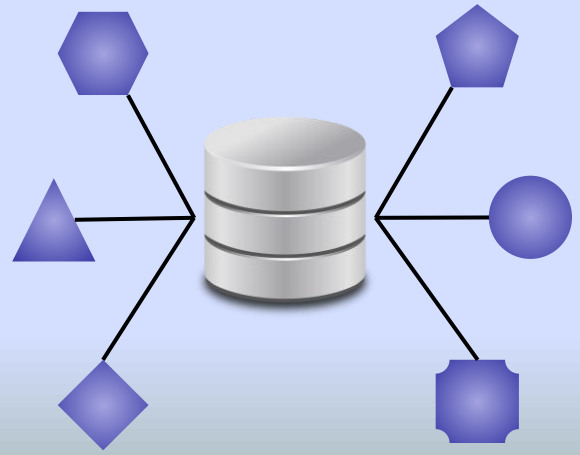
l.bertholdo@ifsp.edu.br

Abordagens de Persistência de Dados

- Além do modelo relacional, em que os dados são representados como uma coleção de tabelas bidimensionais, existem outras abordagens de persistência de dados.

- Um exemplo são os **bancos de dados orientados a objetos**, que surgiram após a consolidação do uso da orientação a objetos no desenvolvimento de aplicações.

*Essa categoria de banco de dados dispensa o mapeamento entre as **classes da aplicação** e suas respectivas **estruturas de armazenamento**. No entanto, sua adoção ainda é muito pequena, dado que os bancos de dados relacionais são utilizados há muito mais tempo, fornecendo ferramentas mais testadas, aceitas e maduras.*



l.bertholdo@ifsp.edu.br

Abordagens de Persistência de Dados

- Outro exemplo mais recente, que vem ganhando espaço nos últimos anos, é o **NoSQL (*Not Only SQL*)**, uma denominação genérica para abordagens de persistência não relacionais.

- Geralmente, refere-se a um tipo de banco de dados baseado em documentos, chave-valor, colunares ou orientados a grafos.

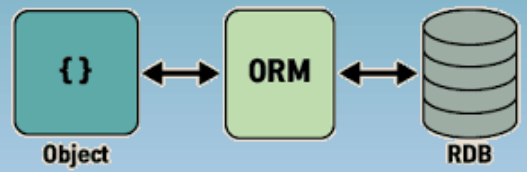
Bancos baseados em documentos *permitem que cada* ***documento (registro)*** *de uma* ***coleção (tabela)*** *tenha uma* ***estrutura específica****. Isso torna possível, por exemplo, cadastrar vários clientes, cada um deles com um conjunto de dados diferente dos outros.*

A tendência atual é a ***persistência poliglota****, ou seja, a ideia de empregar várias abordagens de persistência de dados, com base na maneira como os dados são usados pelas aplicações. Uma aplicação moderna pode usar um* ***banco relacional*** *para a maior parte de suas necessidades, mas usar um* ***banco orientado a documentos*** *para uma funcionalidade em que os relacionamentos entre as entidades não são importantes e a estrutura de armazenamento precisa ser flexível.*

l.bertholdo@ifsp.edu.br

Mapeamento Objeto-Relacional

- O **mapeamento objeto-relacional (*Object Relational Mapper* – ORM)** consiste na transposição do **modelo de classes** para o **modelo relacional**, ou vice-versa.



- Por exemplo, ao transpor um **objeto** para uma **tabela**, os atributos do objeto podem ser mapeados em colunas dessa tabela. Porém, as maiores diferenças entre objetos e tabelas estão nos **relacionamentos**.

*Os objetos trabalham com **ponteiros** para outros objetos, ao passo que os bancos de dados relacionam suas estruturas por meio de **chaves** (primárias e estrangeiras) presentes nas tabelas.*

- Para realizar este mapeamento, existem diversas **técnicas**, bem como alguns ***frameworks*** que automatizam esse processo.

*O mapeamento para bancos orientados a documentos é chamado **Object Document Mapper (ODM)**, que consiste na transposição do modelo de classes para uma estrutura de documentos, os quais podem ser baseados em formatos similares ao JSON (JavaScript Object Notation).*

l.bertholdo@ifsp.edu.br

Técnicas ORM – Identity Field

- Os bancos de dados relacionais conferem unicidade aos seus registros por meio das **chaves primárias** das tabelas, nas quais criamos uma coluna cujo valor não se repete, nem pode ser nulo.
- Os objetos em memória de um sistema não precisam de tal chave para se tornarem únicos. A unicidade é controlada pela própria linguagem de programação, por meio de um **OID (*Object Identifier*)**.
- Sendo assim, para compatibilizar esta unicidade entre os modelos, é preciso propagar as chaves das tabelas para seus respectivos objetos de negócio da aplicação.

A adoção da chave primária nos objetos é de suma importância, pois sem isso seria impossível ler os dados de um registro da tabela, instanciar um objeto com tais dados, modificar esse objeto e, na sequência, armazenar esse objeto novamente no banco de dados, substituindo seus dados originais.

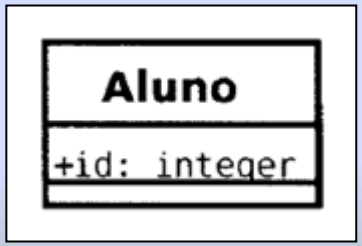
Técnicas ORM – Identity Field

- Visando esse mapeamento, é importante não usar **atributos ligados ao domínio de negócio** nas chaves primárias, por exemplo: o RG parece ser um dado estável, mas uma resolução do governo pode alterá-lo.

- Em vez disso, pode-se utilizar recursos do próprio banco de dados, como **campos genéricos** "*auto-increment*" como estratégia para gerar automaticamente esses identificadores únicos para os objetos.

- O uso de **chaves compostas** em tabelas também não é recomendado, pois torna o mapeamento mais complexo. Por isso, é essencial utilizar um único campo como chave primária, preferencialmente numérico, pois tende a ser mais rápido em buscas, além de facilitar a criação de sequências.

Classe **Aluno** com Identity Field

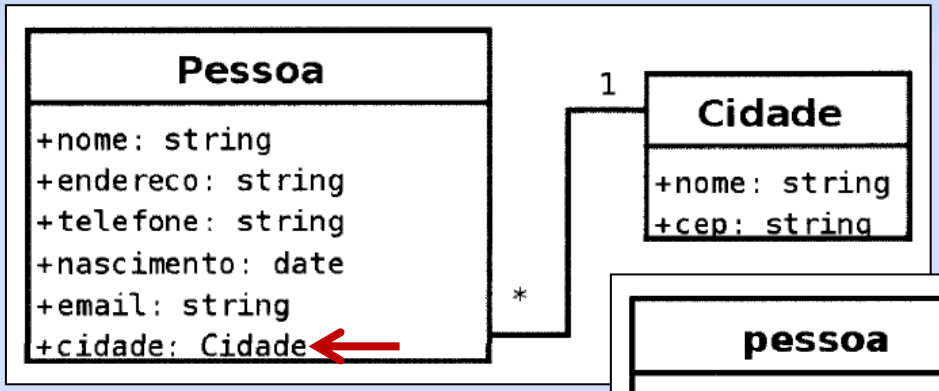


l.bertholdo@ifsp.edu.br

Técnicas ORM – Foreign Key Mapping

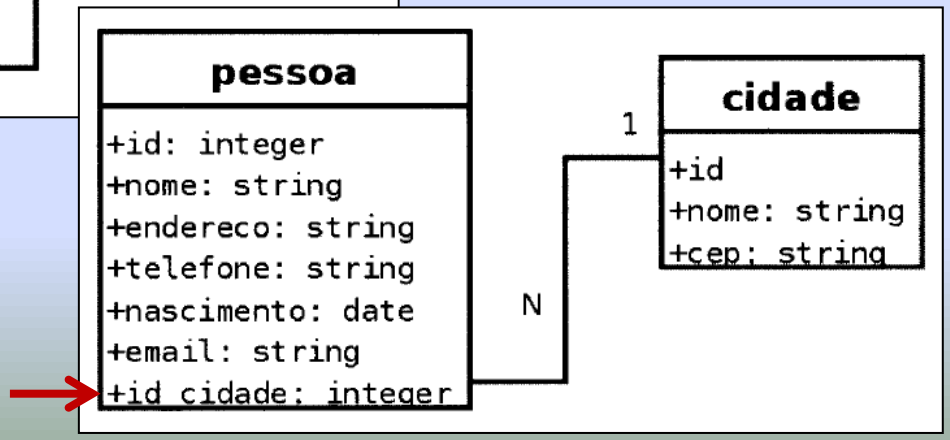
- Na orientação a objetos, **associação** é um tipo de relacionamento em que um objeto tem um atributo que referencia um outro objeto.
- **Exemplo:** A classe **Pessoa** tem uma associação com a classe **Cidade**, que por sua vez é representada pelo atributo do tipo **Cidade** na classe **Pessoa**.

Classes **Pessoa** e **Cidade**



*Para representar tal estrutura no banco de dados, é preciso criar uma chave estrangeira na tabela **pessoa**, apontando para a chave primária da tabela **cidade**.*

Tabelas **pessoa** e **cidade**



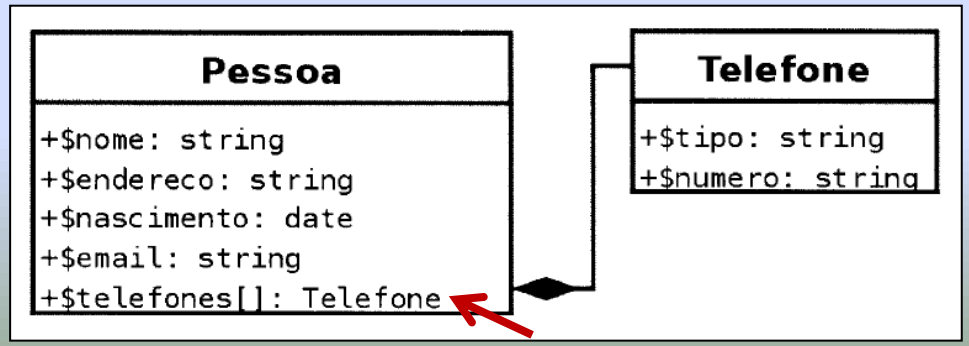
l.bertholdo@ifsp.edu.br

Técnicas ORM – Foreign Key Mapping

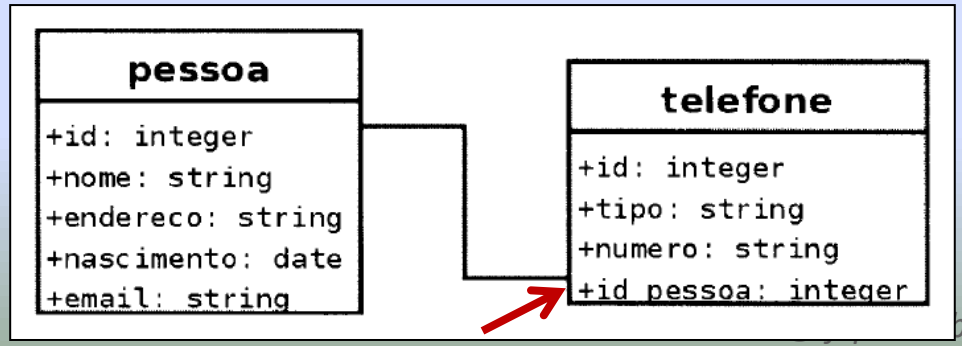
- Outro tipo de relacionamento é a **composição**, no qual um objeto pode conter uma ou várias instâncias de um outro objeto.
- **Exemplo:** A classe **Pessoa** é composta pela classe **Telefone**, que por sua vez é representada pelo atributo "*array*" do tipo **Telefone** na classe **Pessoa**, indicando que uma pessoa pode ter vários telefones.

*Os objetos **Telefone** (parte) e **Pessoa** (todo) estão fortemente ligados. Ao ser criado, o objeto **Pessoa** também cria seus objetos **Telefone** e, ao ser destruído, o mesmo ocorre com estes objetos **Telefone**. Além disso, um objeto **Telefone** fará parte apenas de um único objeto **Pessoa**. Para representar tal estrutura no banco de dados, é preciso adicionar uma chave estrangeira na tabela **telefone**, apontando para a chave primária da tabela **pessoa**.*

Classes **Pessoa** e **Telefone**



Tabelas **pessoa** e **telefone**

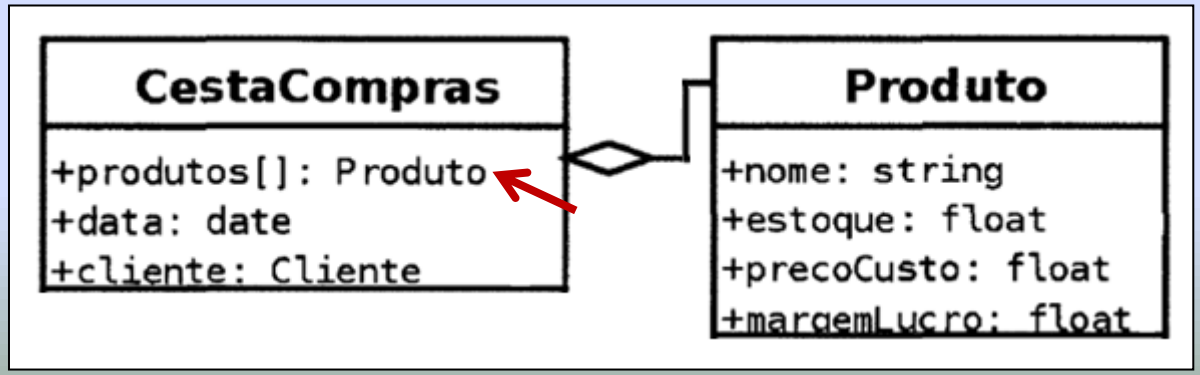


Técnicas ORM – Association Table Mapping

- Outro relacionamento no qual também há uma **relação todo/parte** é a **agregação**. **Exemplo:** A classe **CestaCompras** se agrega à classe **Produto**, que por sua vez é representada pelo atributo "*array*" do tipo **Produto** na classe **CestaCompras**, indicando que uma cesta pode ter vários produtos.

*Os objetos **Produto** (parte) e **CestaCompras** (todo) não estão fortemente ligados. Os objetos **Produto** são criados independentemente do objeto **CestaCompras**, diferentemente da composição, na qual os **objetos parte** são criados no construtor da classe do **objeto todo**. Além disso, um objeto **Produto** pode fazer parte de vários objetos **CestaCompras**.*

Classes **CestaCompras** e **Produto**



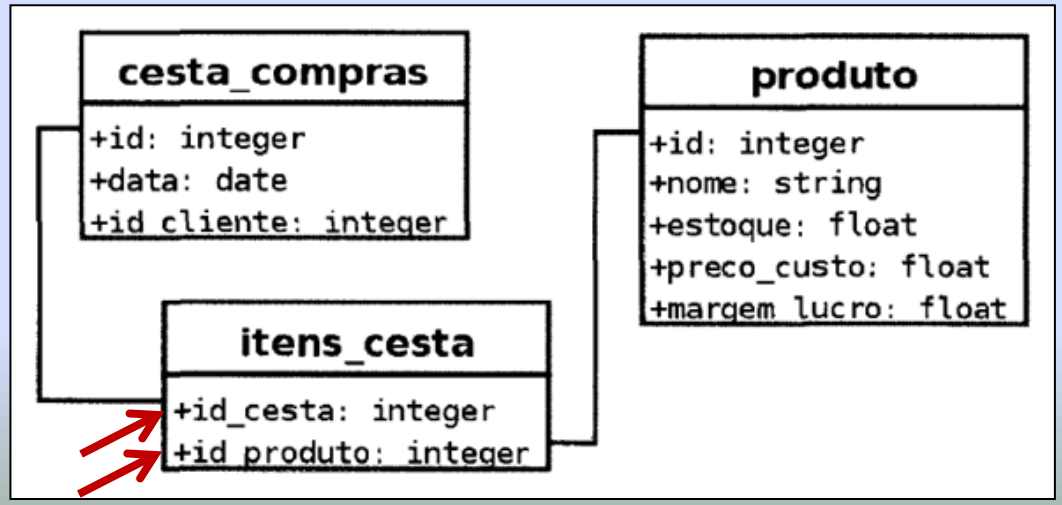
l.bertholdo@ifsp.edu.br

Técnicas ORM – Association Table Mapping

- Nesse caso, não podemos criar uma chave estrangeira na tabela **produto**, apontando para a chave primária da tabela **cesta_compras**, pois isso limitaria um produto a fazer parte apenas de uma cesta.

*Para resolver esse problema, é necessário criar uma tabela intermediária no banco de dados (**itens_cesta**) contendo as chaves primárias das outras duas tabelas, permitindo, então, que, nesta tabela, possamos representar diversos produtos em uma cesta e também que um mesmo produto possa estar em cestas de compras diferentes.*

Tabelas **cesta_compras**, **produto** e **itens_cesta**



*É importante dizer que a tabela **itens_cesta** será totalmente manipulada pela classe **CestaCompras**, ou seja, será totalmente transparente ao programador. Em nível de aplicação, teremos então só dois objetos (**CestaCompras** e **Produto**), que ao serem armazenados no banco de dados terão seus dados distribuídos em três tabelas.*

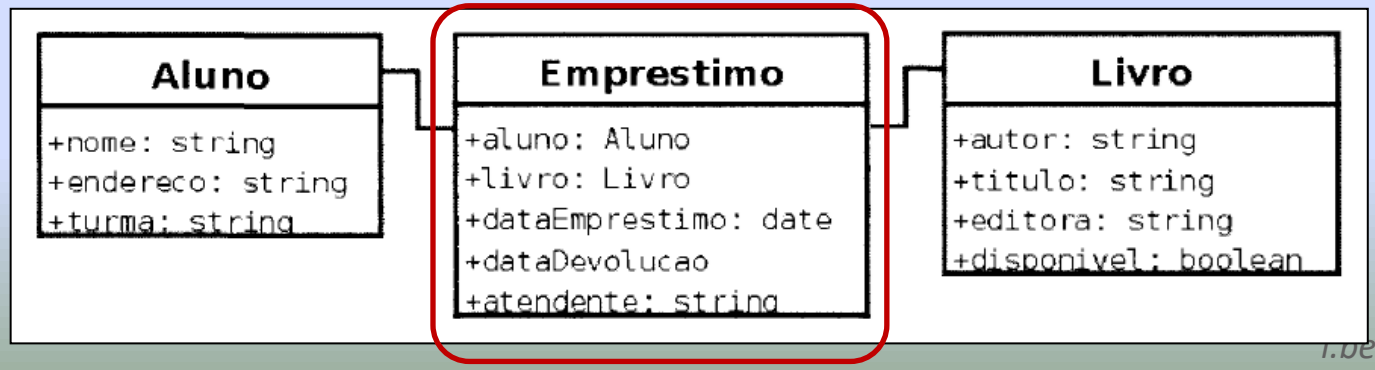
l.bertholdo@ifsp.edu.br

Técnicas ORM – Association Table Mapping

- Em alguns casos, o relacionamento entre duas tabelas extrapola o que pode ser representado com uma **agregação simples** entre duas classes.
- **Exemplo:** O relacionamento entre **aluno** e **livro** em um sistema de biblioteca, em que um aluno pode emprestar vários livros e um livro pode ser emprestado por vários alunos.

*Existem atributos de negócio importantes nesse relacionamento (data de empréstimo, data de devolução, atendente), os quais não conseguiríamos representar apenas com uma agregação entre as classes **Aluno** e **Livro**. Portanto, devido a esses atributos, é preciso promover o **relacionamento de empréstimo** a uma **classe** no nível de aplicação.*

Classes **Aluno**, **Emprestimo** e **Livro**



r.bertholdo@ifsp.edu.br

Técnicas ORM – Single Table Inheritance

- A **herança** é outro relacionamento que pode ocorrer entre objetos. Existem três formas de mapeá-lo em um banco de dados.
- **Exemplo:** Supondo uma biblioteca, o objeto **Material** pode ser do tipo **Livro** ou do tipo **DVD**. As classes **Livro** e **DVD** tem suas peculiaridades, mas também têm características em comum, como os atributos **titulo** e **genero**.
- A **primeira forma** de mapear esse relacionamento para uma banco de dados é criar uma tabela contendo todos os atributos das três classes.

Herança entre as classes **Material**, **Livro** e **DVD**

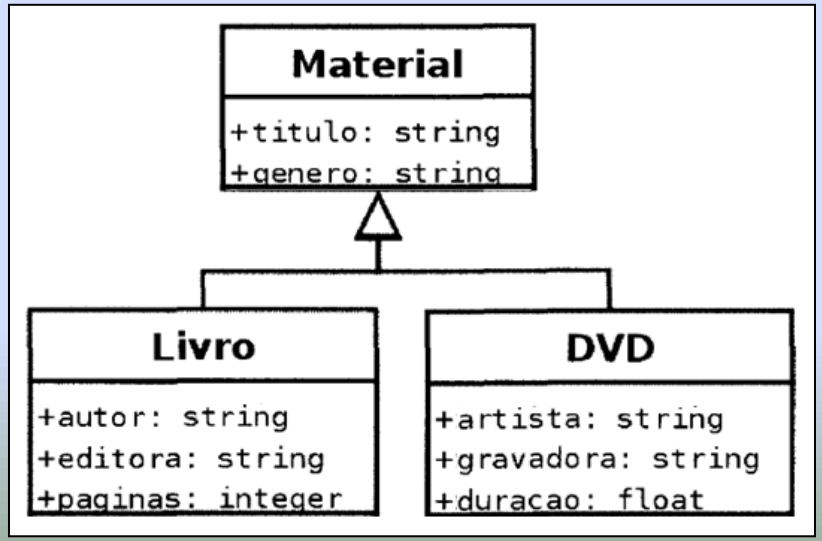
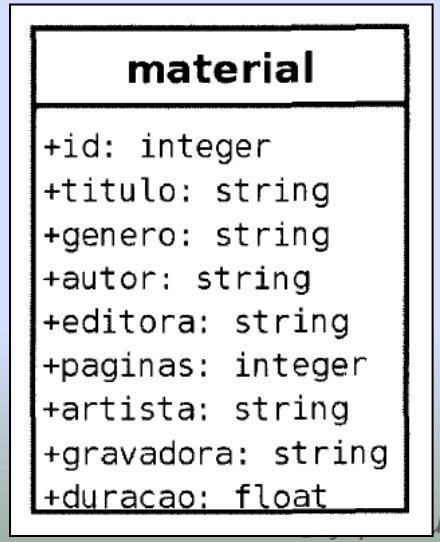


Tabela **material**



Técnicas ORM – Single Table Inheritance

- Nesse caso, ao instanciar um objeto **Livro** ou **DVD** a partir do banco de dados, é preciso saber a qual destes objetos o registro se refere.
- Para isso, é essencial criar na tabela **material** um campo para o **tipo do objeto**, que armazenaria "L" para Livro e "D" para DVD, por exemplo.
- Note que nunca teremos todos os campos da tabela preenchidos. Se o material for um Livro, artista, gravadora e duracao ficarão vazios. Se for um DVD, autor, editora e paginas ficarão vazios.

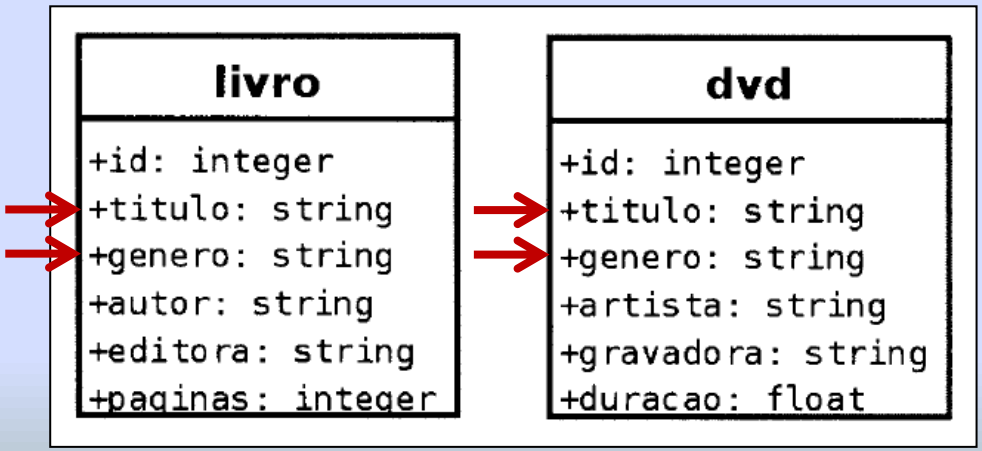
*Esta abordagem é a mais fácil de implementar. Além disso, gera uma estrutura simples de manipular, pois dispensa o uso de **junções** entre tabelas. Com um **SELECT** simples, conseguimos trazer quaisquer informações para a memória, bem como instanciar objetos **Livro** e **DVD**. Entretanto, o programador pode estranhar o fato de existirem alguns campos preenchidos em alguns casos e em outros não.*

Técnicas ORM – Concrete Table Inheritance

- A **segunda forma** de mapear um relacionamento de herança para um banco de dados é criar uma tabela para cada **classe filha** (ou **classe filha concreta**, caso a **classe mãe** seja **abstrata**).

- Dessa forma, no exemplo da biblioteca, haveria duas tabelas: uma para armazenar objetos **Livro** e outra para objetos **DVD**.

Tabelas **livro** e **dvd**



*Nesse caso, cada tabela deve ter, além dos atributos da **classe filha** mapeada, todos os atributos da **classe mãe**. Assim, as tabelas **Livro** e **DVD** iriam conter, além dos atributos de suas respectivas classes, os atributos **titulo** e **genero** da classe **Material**.*

l.bertholdo@ifsp.edu.br

Técnicas ORM – Concrete Table Inheritance

- A vantagem deste mapeamento é que cada tabela contém somente os campos relacionados ao seu contexto, diferentemente do primeiro mapeamento, no qual sempre teremos vários campos vazios.
- Aqui, uma questão importante é que um objeto da classe **Material** é único na memória, independentemente se ele é do tipo **Livro** ou **DVD**. Portanto, no banco de dados, as tabelas **livro** e **dvd** também devem compartilhar apenas uma unicidade.
- Por exemplo, se a tabela **livro** tiver um livro com o ID = 234, a tabela **dvd** não poderá ter um DVD com este mesmo ID.

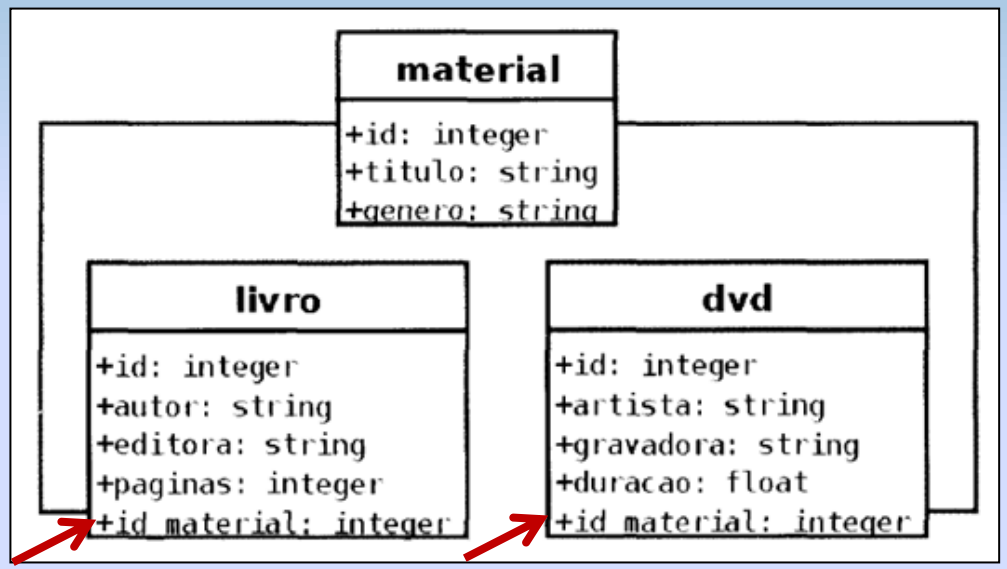
*Supondo uma tela em que o usuário pode consultar qualquer tipo de material (livro ou DVD), uma desvantagem dessa estratégia, é que seria preciso fazer um **SELECT** para cada tabela, ou fazer um **SELECT com junção**, agrupando os resultados em uma única consulta.*

l.bertholdo@ifsp.edu.br

Técnicas ORM – Class Table Inheritance

- Por fim, a **terceira forma** de mapear um relacionamento de herança para um banco de dados é criar uma tabela para cada classe envolvida e relacionar estas tabelas por meio de **chaves primárias** e **estrangeiras**.

Tabelas **material**, **livro** e **dvd**



*Esta abordagem proporciona uma maior **normalização** do banco de dados, quando comparada às duas formas de mapeamento anteriores. Cada **classe** do modelo de objetos possui uma **tabela** correspondente no modelo de banco de dados, o que leva a um melhor entendimento.*

*Porém, quanto mais normalizada for a estrutura do banco de dados, mais complexo e custoso será para carregar seus objetos na memória. **Por exemplo:** para listar todos os tipos de materiais e seus respectivos dados, teremos que fazer junções entre as três tabelas; já para listar apenas livros, teremos que fazer uma junção entre as tabelas **livro** e **material**.*

Técnicas ORM – Class Table Inheritance

- Outra questão a ser analisada nessa abordagem é que há situações em que um objeto pode **transitar** dentro da hierarquia de classes.

- **Exemplo:** Um livro dificilmente se transformará em um DVD, mas supondo as classes **Pessoa**, **Funcionario** e **Estagiario**, sendo **Funcionario** e **Estagiario** classes filhas de **Pessoa**, devemos considerar que um estagiário pode ser efetivado como um funcionário, ou seja, trocaria de tipo.

*Nesta abordagem, teríamos que **transferir um registro de uma tabela para outra**, ao passo que na primeira forma de mapeamento teríamos apenas que alterar o **campo que indica o tipo de objeto**.*

*Cada **técnica de mapeamento de herança** tem suas vantagens e desvantagens, cabe ao projetista descobrir qual delas é a mais indicada para cada caso.*

l.bertholdo@ifsp.edu.br

Técnicas ORM – Lazy Initialization

- Em um modelo com muitos objetos se relacionando entre si, carregar todos os objetos relacionados de um determinado objeto, ao recuperá-lo da base de dados, pode consumir uma quantidade enorme de memória, dependendo da quantidade de objetos relacionados.
- Para evitar essa situação, a técnica ***Lazy Initialization*** (inicialização tardia) faz com que os objetos relacionados sejam instanciados na aplicação **apenas quando são realmente necessários**.
- Uma forma de implementar essa técnica é usar **métodos interceptadores**, que são executados automaticamente sempre que uma propriedade não existente em um objeto é requisitada.

*Diferentemente das técnicas anteriores, o objetivo da **inicialização tardia** não é mapear dados ou relacionamentos entre os modelos de objetos e relacional, mas sim **otimizar** o maneira como os objetos recuperados do banco de dados são carregados na memória do programa.*

l.bertholdo@ifsp.edu.br

Técnicas ORM – Lazy Initialization

```php
class Pessoa {
    private $nome;
    private $cidadeID;

    function __construct($nome, $cidadeID) {
        $this->nome     = $nome;
        $this->cidadeID = $cidadeID;
    }

    function __get($propriedade) {
        if ($propriedade == 'cidade');
            return new Cidade($this->cidadeID);
    }
}
```

```php
class Cidade {
    private $id;
    private $nome;

    function __construct($id) {
        $data[1] = 'São Paulo';
        $data[2] = 'Rio de Janeiro';
        $data[3] = 'Belo Horizonte';

        $this->id = $id;
        $this->nome = $data[$id];
    }

    function setNome($nome) { $this->nome = $nome; }
    function getNome() { return $this->nome; }
}
```

*Quando instanciamos um objeto **Pessoa**, passamos como parâmetro o seu **nome** e o **id da cidade**. Já para um objeto **Cidade**, passamos apenas seu **id**, pois seu **nome** é definido com base em um array (**data**), que simula o banco de dados.*

*Ao instanciar o objeto **Pessoa**, o objeto **Cidade** não é instanciado em conjunto. Mesmo assim, é possível acessar a cidade de uma pessoa (**p1->cidade**). Isto é possível devido ao método **__get()** (específico da linguagem PHP), que verifica se está sendo requisitada alguma propriedade não existente em um objeto. Nesse exemplo, caso essa propriedade se chame "cidade", automaticamente é instanciado um objeto **Cidade**, o qual é retornado. Assim, o objeto relacionado (**Cidade**) é instanciado na memória apenas quando se torna de fato necessário, neste caso por meio da propriedade inexistente "cidade".*

```php
$p1 = new Pessoa('Maria Pereira', 2);
$p2 = new Pessoa('João da Silva', 3);

echo 'Cidade da pessoa 1: ' . $p1->cidade->getNome() . "<br>";
echo 'Cidade da pessoa 2: ' . $p2->cidade->getNome();
```

```
Cidade da pessoa 1: Rio de Janeiro
Cidade da pessoa 2: Belo Horizonte
```

Frameworks ORM

- Atualmente, existem diversos ***frameworks* ORM** desenvolvidos para facilitar o processo de mapeamento objeto-relacional.
- Essas ferramentas garantem maior produtividade no desenvolvimento de aplicações, pois evitam a necessidade de reescrever enormes quantidades de instruções para criar e manipular bancos de dados.
- Algumas linguagens de programação e seus principais *frameworks* ORM disponíveis no mercado:

Linguagens de Programação	Frameworks
Java	Hibernate, EclipseLink, ActiveJPA
Kotlin	Exposed
C#	Entity Framework, NHibernate, Dapper
PHP	Doctrine, Eloquent
Ruby	Ruby On Rails ActiveRecord, Datamapper
Python	DjangoORM, SqlAlchemy

l.bertholdo@ifsp.edu.br

Referências

- DALL'OGLIO, Pablo. **PHP:** Programando com orientação a objetos. 2 ed. São Paulo: Novatec Editora, 2007.

- MACHADO, Felipe Nery Rodrigues. **Projeto e implementação de banco de dados**. 3. ed. São Paulo: Érica, 2014.

- MACIASZEK, Leszek A. **Requirements Analysis and System Design**. 3. ed. Harlow, England: Pearson Education, 2007.