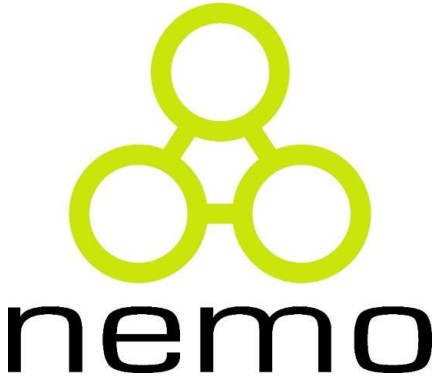


Projeto de Sistemas de Software

Jordana S. Salamon

jssalamon@inf.ufes.br

jordanasalamon@gmail.com

DEPARTAMENTO DE INFORMÁTICA

CENTRO TECNOLÓGICO

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

Introdução

- A maioria dos sistemas requer alguma forma de armazenamento de dados.
- Para tal, há várias alternativas, dentre elas a persistência em arquivos e bancos de dados.
- Em especial os sistemas de informação envolvem grandes quantidades de dados e fazem uso de sistemas gerenciadores de bancos de dados (SGBDs).
- Há diversos tipos de SGBDs, dentre eles os relacionais e os orientados a objetos, sendo os primeiros os (ainda) mais utilizados no desenvolvimento de sistemas de informação.

Introdução

- Quando SGBDs Relacionais são utilizados, é necessário um mapeamento entre as estruturas de dados dos modelos orientado a objetos e relacional, de modo que objetos possam ser armazenados em tabelas.

- Dentre as principais diferenças entre esses modelos, destacam-se as diferentes formas como objetos e tabelas tratam ligações e na ausência do mecanismo de herança no modelo relacional.

- Essas diferenças levam à necessidade de transformações das estruturas de dados entre objetos e tabelas, tratadas como **mapeamento objeto-relacional**.

Introdução

- Além das diferenças estruturais, outros aspectos têm de ser tratados durante o projeto da persistência, dentre eles o modo como a camada de lógica de negócio se comunica com o banco de dados, o problema comportamental que diz respeito a como obter vários objetos do banco e como salvá-los, e o tratamento de conexões com o banco de dados e transações.

- É importante enfatizar que muitos desses problemas são tratados por *frameworks* de persistência de objetos em bancos de dados relacionais (ou *frameworks* de mapeamento objeto-relacional), tal como o Hibernate.

- Os desenvolvedores desses frameworks têm despendido muitos esforços trabalhando nesses problemas e tais ferramentas são bem mais sofisticadas do que a maioria das soluções específicas que podem ser construídas pelos próprios desenvolvedores em um projeto específico.

Introdução

- Contudo, mesmo quando um framework de mapeamento objeto-relacional (O/R) é utilizado, é importante estar ciente dos padrões usados. Boas ferramentas de mapeamento O/R dão várias opções de mapeamento para um banco de dados e esses padrões ajudam a entender quando selecionar as diferentes opções.

- Por fim, deve-se enfatizar que um bom projeto do mecanismo de persistência deve levar em conta a ideia de separação entre interface com o usuário, lógica de negócio e persistência.

O Modelo Relacional

- Em um modelo de dados relacional, os conjuntos de dados são representados por tabelas de valores.

- Cada tabela, denominada de relação, é bidimensional, sendo organizada em linhas e colunas. Esse modelo está fortemente baseado na teoria matemática sobre relações, daí o nome relacional.

- Os principais conceitos do modelo relacional são os seguintes:

O Modelo Relacional

- • **Tabela ou Relação:** tabela de valores bidimensional organizada em linhas e colunas

Matrícula	Nome	CPF	Dt-Nasc
0111	Marcos	17345687691	11/04/66
0208	Rita	56935101129	21/02/64
0789	Mônica	81176628911	01/11/70
1589	Márcia	91125769120	20/10/80

O Modelo Relacional

- • **Linha ou Tupla**: representa uma entidade de um conjunto de entidades. Ex: A funcionária Mônica do conjunto de funcionários.

- • **Coluna**: representa um atributo de uma entidade. Ex.: Matrícula, Nome, CPF, Dt-Nasc.

- • **Célula**: Item de dado da linha i, coluna j. Ex.: Rita (linha 2, coluna 2).

- • **Chave Primária**: coluna ou combinação de colunas que possui a propriedade de identificar de forma única uma linha da tabela e que é utilizada para estabelecer associações entre entidades via transposição de chave.

O Modelo Relacional

- • **Chave Estrangeira ou Transposta**: é a forma utilizada para associar linhas de tabelas distintas.

- A chave primária de uma tabela é transposta como uma coluna na outra tabela, onde é considerada uma chave estrangeira. A Figura ilustra um relacionamento 1:N entre as tabelas Departamentos e Funcionários, indicando que um departamento pode lotar vários funcionários, enquanto um funcionário tem de estar lotado em um departamento.

Departamentos

Código	Nome
INF	Informática
LET	Letras
MAT	Matemática

Funcionários

Matrícula	Nome	Cod-Depto
0158	José	MAT
5295	Ricardo	INF
7712	Rosane	INF

Chave Estrangeira

O Modelo Relacional

- • **Tabelas Associativas**: usadas para representar relacionamentos n-para-n entre tabelas.

- No exemplo, uma pessoa pode ter interesse em vários assuntos, enquanto um assunto pode ser de interesse de várias pessoas. A tabela Interesses é uma tabela associativa, sendo suas duas colunas chaves transpostas de outras tabelas.

Pessoas

CPF	Nome
96100199	José
83467187	Maria
02765140	Luiza

Interesses

CPF-Pessoa	Código-Assunto
96100199	COMP
96100199	MUS
02765140	ENG

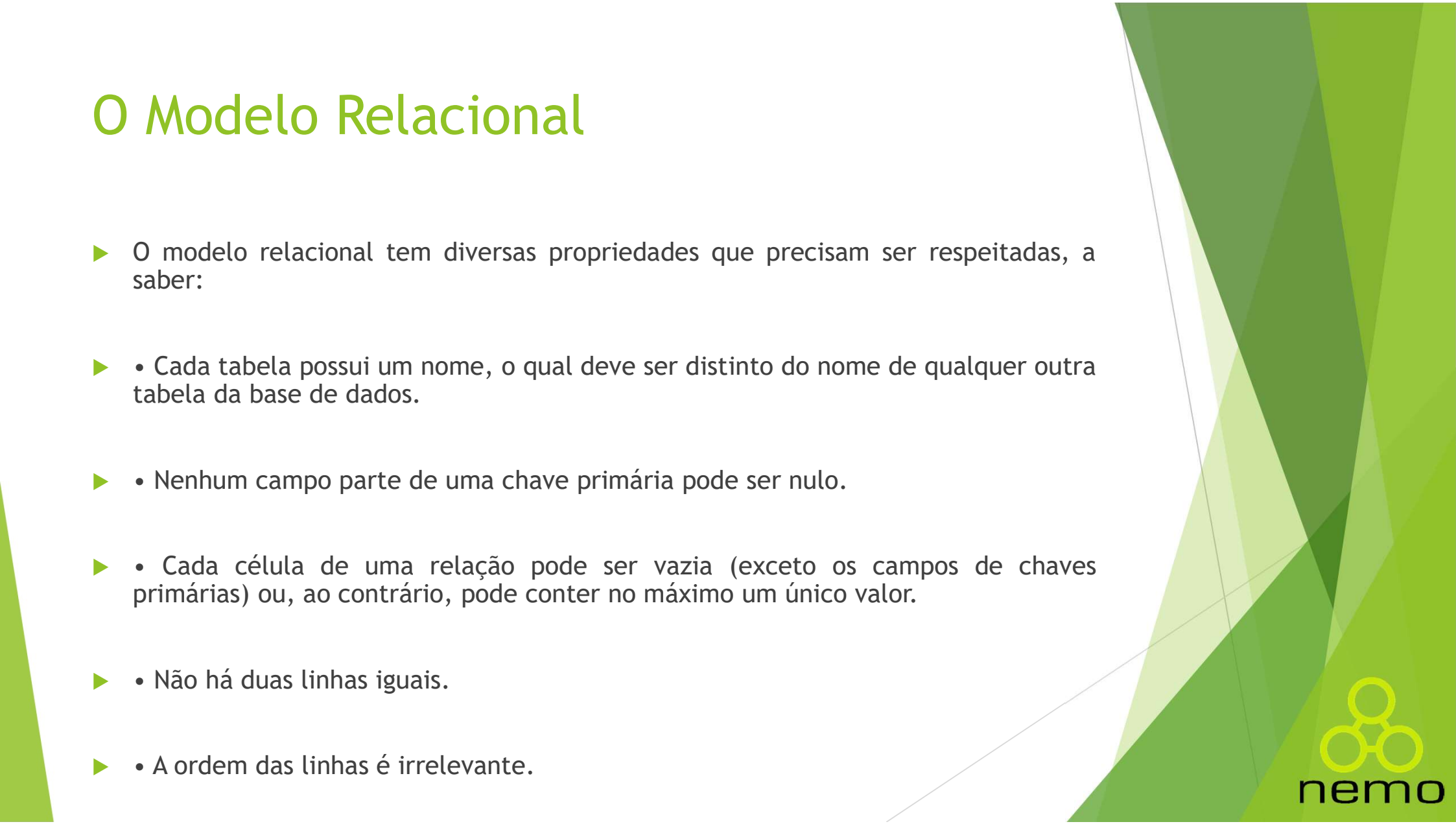
Assuntos

Código	Nome
ENG	Engenharia
COMP	Computação
MUS	Música

nemo

O Modelo Relacional

- O modelo relacional tem diversas propriedades que precisam ser respeitadas, a saber:
- • Cada tabela possui um nome, o qual deve ser distinto do nome de qualquer outra tabela da base de dados.
- • Nenhum campo parte de uma chave primária pode ser nulo.
- • Cada célula de uma relação pode ser vazia (exceto os campos de chaves primárias) ou, ao contrário, pode conter no máximo um único valor.
- • Não há duas linhas iguais.
- • A ordem das linhas é irrelevante.



O Modelo Relacional

- • Cada coluna tem um nome, o qual deve ser distinto dos demais nomes das colunas de uma mesma tabela.
- • Usando-se os nomes para se fazer referência às colunas, a ordem destas torna-se irrelevante.
- • Os valores de uma coluna são retirados todos de um mesmo conjunto, denominado domínio da coluna.
- • Duas ou mais colunas distintas podem ser definidas sobre um mesmo domínio.
- • Um campo que seja chave estrangeira (ou parte dela) só pode assumir valor nulo ou um valor para o qual exista um registro na tabela onde ele é chave primária.

O Modelo Relacional

- Muitas vezes, durante o projeto de bancos de dados relacionais, é útil representar graficamente as tabelas e as ligações entre elas.

- Para tal, um Diagrama Relacional pode ser desenvolvido, representando as ligações entre tabelas de um modelo relacional.

Diagrama Relacional

Tabelas do Modelo Relacional

Departamentos		
Código	Nome	Matrícula-Chefe
INF	Informática	00877
MAT	Matemática	06001
QUI	Química	13888

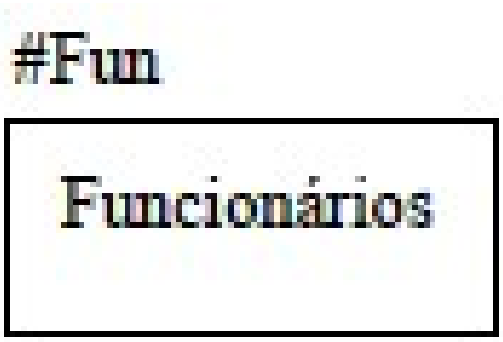
Funcionários	
Matrícula	Nome
13888	Jorge
00877	Dede
06001	Pedro

nemo

O Modelo Relacional

- Em um Diagrama Relacional são representados os seguintes elementos:

- • **Tabelas:** são representadas por retângulos, com uma referência à chave primária em cima da tabela.



O Modelo Relacional

- • **Relacionamentos**: são representadas por linhas contínuas, associadas aos símbolos abaixo:

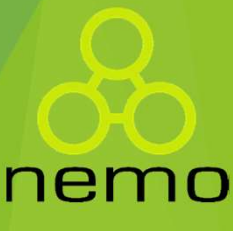
Cardinalidade	Relacionamento
(0,1)	
(1,1)	
(0,N)	
(1,N)	

- • **Chaves estrangeiras**: quando uma chave transposta não fizer parte da chave primária da relação destino, a mesma é representada em cima do retângulo da relação destino com um subscrito “t”.

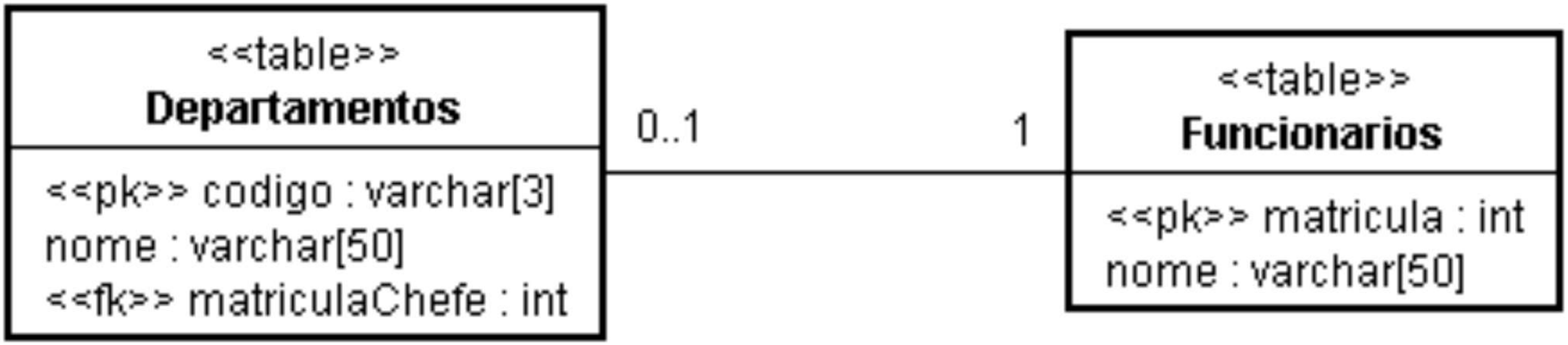
nemo

O Modelo Relacional

- Outra opção para representar diagramas relacionais é utilizar um perfil UML.
- Neste caso, tabelas são representadas como classes com o estereótipo <<table>>, tabelas associativas são representadas como classes com o estereótipo <<association table>> e colunas são representadas como atributos.
- Quando uma coluna é chave primária (ou parte da chave primária), ela é estereotipada com <<pk>>.
- Quando uma coluna é chave estrangeira (ou parte da chave estrangeira), ela é estereotipada com <<fk>>.



O Modelo Relacional



nemo

O Modelo Relacional

- Ainda que no exemplo anterior as chaves primárias tenham sido escolhidas dentre atributos das classes (Departamentos e Funcionários), essa não é necessariamente a melhor escolha. Muito pelo contrário.

- Para trabalhar bem a manutenibilidade dos sistemas resultantes, sobretudo quando se utiliza o paradigma orientado a objetos, é mais interessante utilizar como chave primária de uma tabela uma coluna criada exclusivamente para este fim, sem nenhum significado no domínio do problema e, portanto, uma coluna que não será manipulada pela camada de Lógica de Negócio, mas apenas pela camada de persistência.

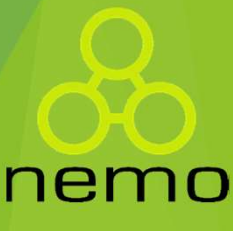
- Essa abordagem, além de facilitar a manutenção, facilita grandemente o desenvolvimento de infraestruturas genéricas de persistência de objetos, uma vez que todas as chaves primárias são do mesmo tipo de dados e podem ser tratadas uniformemente.

Mapeamento Objeto-Relacional

- Conforme se pode observar, há diferenças significativas entre o modelo de classes de um projeto orientado a objetos e o modelo relacional.

- Uma diferença significativa é a forma como relacionamentos são tratados nos modelos de objetos e relacional: objetos armazenam referências a outros objetos (p.ex., endereços de memória), enquanto bancos de dados relacionais ligam tabelas por meio de chaves transpostas.

- Ainda neste contexto, objetos usam coleções para tratar relacionamentos e atributos multivalorados, enquanto células de uma tabela só podem ter no máximo um valor.



Mapeamento Objeto-Relacional

- Outra diferença substancial é o mecanismo de herança, o qual não é suportado por bancos de dados relacionais.

- Por outro lado, tabelas têm de ter uma chave primária, enquanto objetos são únicos por essência, ficando transparente para o desenvolvedor a existência de identificadores.

- Assim, para que a persistência de objetos seja feita em um banco de dados relacional, é necessário **realizar um mapeamento entre esses dois tipos de modelos**.

Mapeamento Objeto-Relacional

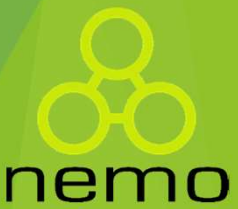
- O termo mapeamento objeto-relacional (O/R) é usado tipicamente para referenciar um mapeamento estrutural entre objetos que residem em memória e tabelas em bancos de dados.

- Ele é fundamental para o projeto da camada de persistência quando um SGBD relacional é utilizado e só deve ser visível na camada de persistência, isolando as demais camadas da arquitetura de software do impacto da tecnologia de bancos de dados.

- No mapeamento O/R, as seguintes questões devem ser abordadas:
- (i) mapeamento de classes e objetos;
- (ii) mapeamento de herança; e
- (iii) mapeamento de associações entre objetos.

Mapeamento Objeto-Relacional

- **Mapeando Classes e Objetos**
- Quando não há herança, cada classe tipicamente é mapeada em uma tabela e cada instância da classe (objeto) em uma linha dessa tabela.
- O modelo de classes deve ser normalizado previamente, eliminando-se atributos multivalorados.
- Nesse processo de normalização das classes, surge uma importante questão.



Mapeamento Objeto-Relacional

- **Mapeando Classes e Objetos**

- No modelo relacional, toda tabela tem de ter uma chave primária, isto é, uma ou mais colunas, cujos valores identificam univocamente uma linha da mesma.

- Objetos, por sua vez, têm identidade própria, independentemente dos valores de seus atributos.

- Assim, deve-se definir que identificador único deve ser usado para designar objetos no banco de dados relacional.

Mapeamento Objeto-Relacional

- **Mapeando Classes e Objetos**

- Uma solução possível consiste em observar se há um atributo na classe com a propriedade de identificação única e utilizá-lo, então, como chave primária.

- Caso não haja um atributo com tal característica, deverá ser criado um. Contudo, essa abordagem não é a mais indicada.

- De fato, ela só deve ser utilizada quando já houver uma base de dados legada sendo utilizada por outros sistemas.

Mapeamento Objeto-Relacional

- **Mapeando Classes e Objetos**

- Uma maneira mais eficaz, sobretudo para permitir a construção de componentes mais genéricos de persistência, consiste em dar a cada objeto um atributo chamado de identificador de objeto (id).

- Os ids são utilizados como chaves primárias nas tabelas do banco de dados relacional e não devem possuir nenhum significado de negócio.

- Essa abordagem é chamada de padrão Campo de Identidade (Identity Field), no qual o identificador salva a chave primária da correspondente linha da tabela no objeto, de modo a manter um rastro entre o objeto em memória e sua representação como linha de uma tabela do banco de dados.

Mapeamento Objeto-Relacional

- **Mapeando Herança**

- Uma vez que os bancos de dados relacionais não suportam o mecanismo de herança, é necessário estabelecer uma forma de mapeamento desse mecanismo.

- A grande questão no mapeamento da herança diz respeito a como organizar os atributos herdados no banco de dados.

- Existem três soluções principais para mapear herança em um banco de dados relacional, a saber:
- • Utilizar uma tabela para toda a hierarquia;
- • Utilizar uma tabela por classe concreta na hierarquia;
- • Utilizar uma tabela por classe na hierarquia.

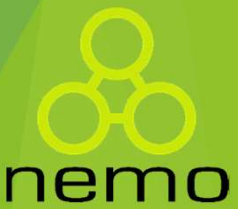
Mapeamento Objeto-Relacional

- **Mapeando Herança**

- No primeiro caso, a tabela derivada contém os atributos de todas as classes na hierarquia.

- Esta solução é descrita como o padrão Herança em Tabela Única (Single Table Inheritance).

- A vantagem dessa solução é a simplicidade. Além disso, ela suporta bem o polimorfismo e facilita a designação de ids, já que todos os objetos estão em uma única tabela.



Mapeamento Objeto-Relacional

- **Mapeando Herança**

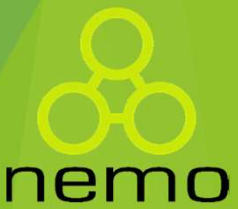
- O problema fundamental dessa solução é que, se as subclasses têm muitos atributos diferentes, haverá muitas colunas que não se aplicam aos objetos individualmente, provocando grande desperdício de espaço no banco de dados.

- Além disso, sempre que um atributo for adicionado a qualquer classe na hierarquia, um novo atributo deve ser adicionado à tabela.

- Isso aumenta o acoplamento na hierarquia, pois, se um erro for introduzido durante a adição desse atributo, todas as classes na hierarquia podem ser afetadas e não apenas a classe que recebeu o novo atributo.

Mapeamento Objeto-Relacional

- **Mapeando Herança**
- No segundo caso, utiliza-se uma tabela para cada classe concreta na hierarquia.
- Cada tabela derivada para as classes concretas inclui tanto os atributos da classe quanto os de suas superclasses.
- Essa solução é descrita como o padrão Herança em Tabela Concreta (Concrete Table Inheritance).
- A grande vantagem é a facilidade de processamento sobre as subclasses concretas, já que todos os dados de uma classe concreta estão armazenados em uma única tabela.



Mapeamento Objeto-Relacional

- **Mapeando Herança**

- Da mesma forma que o caso anterior, a designação de ids é facilitada, com a vantagem de se eliminar o desperdício de espaço.

- Contudo, há também desvantagens. Quando uma superclasse é alterada, é necessário alterar as tabelas.

- Além disso, quando há muito processamento envolvendo a superclasse, há uma tendência de queda do desempenho da aplicação, já que passa a ser necessário manipular várias tabelas ao invés de uma.

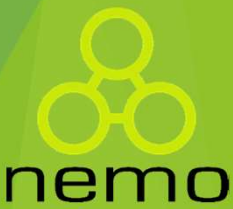
Mapeamento Objeto-Relacional

- **Mapeando Herança**

- A terceira solução é a mais genérica: utiliza-se uma tabela por classe, não importando se concreta ou abstrata.

- Deve haver uma tabela para cada classe e visões para cada uma das classes derivadas (subclasses).

- Essa solução é descrita como o padrão Herança em Tabela de Classe (Class Table Inheritance).



Mapeamento Objeto-Relacional

- **Mapeando Herança**

- Essa abordagem é a que provê o mapeamento mais simples entre classes e tabelas. É muito mais fácil modificar uma superclasse e acrescentar subclasses, já que é necessário apenas alterar ou acrescentar uma tabela.

- Uma desvantagem é o grande número de tabelas no banco de dados, uma para cada classe.

- Além disso, pode levar mais tempo para acessar dados de uma classe, uma vez que pode ser necessário acessar várias tabelas. Podem ser necessárias múltiplas uniões (joins) de tabelas para recuperar um único objeto, o que usualmente reduz o desempenho.

Mapeamento Objeto-Relacional

- **Mapeando Associações**

- Já percebemos que há diferenças substanciais na forma de lidar com relacionamentos nos modelos orientado a objetos e relacional.

- Assim, é fundamental mapear associações em um modelo de objetos para um modelo relacional. Associações 1:1 e 1:N são mapeadas por meio da transposição de chaves.

- Para tal, aplica-se o padrão Mapeamento de Chave Estrangeira (Foreign Key Mapping). Já associações N:N requerem uma tabela associativa, com preconiza o padrão Mapeamento de Tabela Associativa (Association Table Mapping).

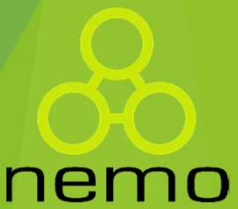
Mapeamento Objeto-Relacional

- **Mapeando Associações 1:1**

- O mapeamento de associações 1:1 é feito transpondo-se a chave primária de uma tabela para a outra.

- Quando a associação for obrigatória nas duas extremidades (multiplicidade mínima 1 em ambas as extremidades), pode-se escolher qualquer das chaves para transpor.

- Quando a associação for opcional em pelo menos uma das duas extremidades (multiplicidade mínima 0), é melhor transpor a chave que dará origem a uma coluna mais densa, isto é, que terá menos valores nulos.



Mapeamento Objeto-Relacional

- **Mapeando Associações 1:1**
- Outro fator a ser levado em consideração é a navegabilidade da associação.
- Sempre que possível, deve-se transpor a chave que facilite a navegabilidade escolhida.
- Por fim, é bom frisar que sempre que a associação for obrigatória (multiplicidade mínima 1), a coluna resultante da chave transposta não poderá ter valores nulos.

Mapeamento Objeto-Relacional

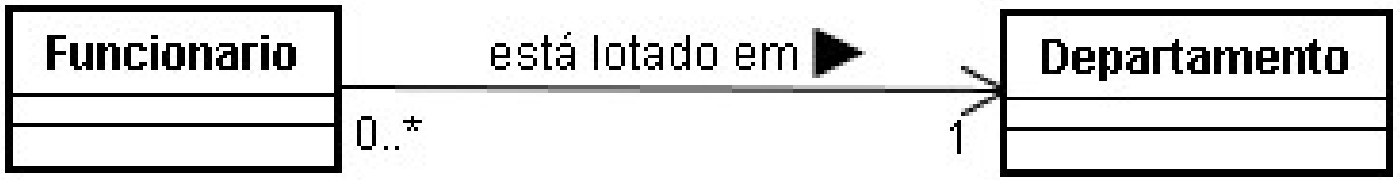
- **Mapeando Associações 1:N**

- O mapeamento de associações 1:N é feito transpondo-se a chave primária da tabela correspondente à classe cuja extremidade da associação tem multiplicidade máxima 1 para a tabela que corresponde à classe cuja extremidade da associação tem multiplicidade máxima n.

- É bom lembrar que sempre que a associação for obrigatória (multiplicidade mínima 1), a coluna resultante da chave transposta não poderá ter valores nulos

Mapeamento Objeto-Relacional

- **Mapeando Associações 1:N**



nemo

Mapeamento Objeto-Relacional

- **Mapeando Associações N:N**

- O mapeamento de associações N:N é feito utilizando-se uma tabela associativa, uma vez que bancos de dados relacionais não são capazes de manipular diretamente relacionamentos N:N.

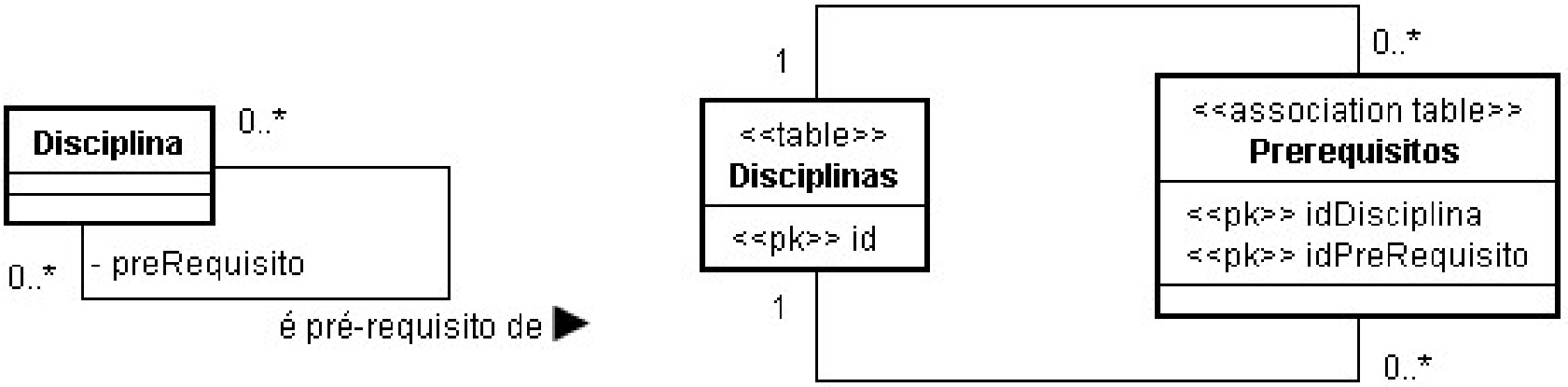
- Vale frisar que, muitas vezes, as chaves transpostas são parte da chave primária da tabela associativa.

- Quando este for o caso, elas são identificadas no modelo como sendo chaves primárias.

nemo

Mapeamento Objeto-Relacional

- **Mapeando Associações N:N**



nemo

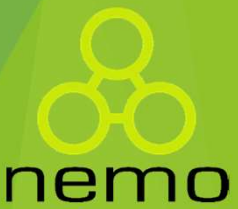
Padrões Arquitetônicos para a Camada de Gerência de Dados

- A Camada de Gerência de Dados - CGD (ou Camada de Persistência) provê a infraestrutura básica para o armazenamento e a recuperação de objetos no sistema.

- Sua finalidade é isolar os impactos da tecnologia de gerenciamento de dados sobre a arquitetura do software.

- A despeito da opção de persistência adotada (SGBD relacional, SGBD orientado a objetos, arquivos), há uma importante questão a ser considerada no projeto da CGD:

- Que classes devem suportar a persistência dos objetos?



Padrões Arquitetônicos para a Camada de Gerência de Dados

- Uma alternativa é tornar cada classe a ser persistida (tipicamente, classes do domínio do problema), ao longo de toda a arquitetura de software, responsável por suas próprias atividades de persistência.

- Essa abordagem é descrita no padrão Registro Ativo (Active Record), no qual uma classe mantém uma linha de uma tabela ou visão do banco de dados, encapsula o acesso à base de dados e adiciona à lógica de domínio o tratamento da persistência de seus dados.

Padrões Arquitetônicos para a Camada de Gerência de Dados

- Em outras palavras, esse padrão coloca a lógica de acesso a dados nas classes de domínio do problema e, portanto, não há efetivamente uma camada de persistência, já que não há separação entre lógica de negócio e gerência de dados.

- Obviamente, nessa abordagem, a arquitetura torna-se completamente dependente da tecnologia de persistência e, se, por exemplo, a organização migrar de um SGBD relacional para outro, essa migração provavelmente vai ter impactos em várias classes do sistema.

- Em geral, essa abordagem é desaconselhável, só devendo ser aplicada em aplicações muito simples.

Padrões Arquitetônicos para a Camada de Gerência de Dados

- Uma abordagem mais elegante consiste em isolar completamente a lógica de negócio e o banco de dados, criando uma camada responsável pelo mapeamento entre objetos do domínio e tabelas do banco de dados.

- Os padrões Mapeador de Dados (Data Mapper) e Objeto de Acesso a Dados (Data Access Object - DAO) adotam esta filosofia, de modo que apenas uma parte da arquitetura de software fica ciente da tecnologia de persistência adotada.

Padrões Arquitetônicos para a Camada de Gerência de Dados

- Essa parte, o Componente de Gerência de Dados (CGD), serve como uma camada intermediária separando objetos do domínio de objetos de gerência de dados.

- Via conexões de mensagem, o CGD lê e escreve dados, estabelecendo uma comunicação entre a base de dados e os objetos do sistema.

- Qualquer código SQL está confinado nessas classes, de modo que não há código desse tipo em outras classes da arquitetura do software.

Padrões Arquitetônicos para a Camada de Gerência de Dados

- **Padrão Data Mapper**

- O padrão Mapeador de Dados prescreve uma camada de objetos mapeadores que transferem dados entre objetos em memória e o banco de dados, mantendo-os independentes uns dos outros e dos mapeadores em si.

- Os objetos de domínio não têm qualquer conhecimento acerca do esquema do banco de dados e não precisam de nenhuma interface para código SQL.

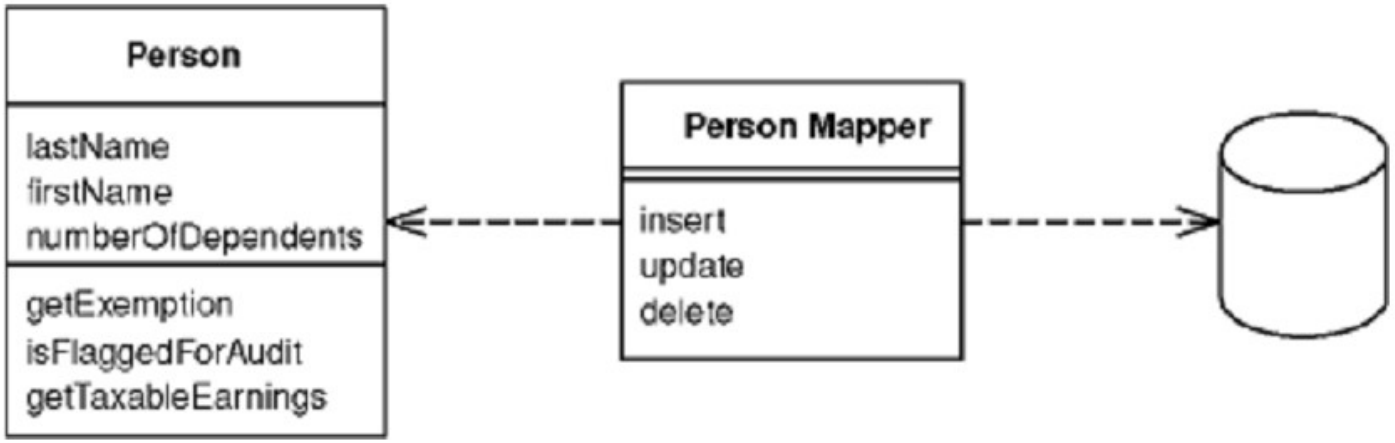
- De fato, eles não precisam saber sequer que há um banco de dados.

- O banco de dados, por sua vez, desconhece completamente os objetos que o utilizam.

Padrões Arquitetônicos para a Camada de Gerência de Dados

- **Padrão Data Mapper**

- Em sua versão mais simples, para cada classe a ser persistida em uma tabela, há uma correspondente classe mapeadora (ou classe sombra).



nemo

Padrões Arquitetônicos para a Camada de Gerência de Dados

- **Padrão DAO**

- O padrão DAO define uma interface de operações de persistência, incluindo métodos para criar, recuperar, alterar, excluir e fazer consultas de diversos tipos, relativa a uma particular entidade persistente, agrupando o código relacionado à persistência daquela entidade.

- Seguindo esse padrão, a camada de persistência é implementada por duas hierarquias paralelas: interfaces à esquerda e implementações à direita.

Padrões Arquitetônicos para a Camada de Gerência de Dados

- **Padrão DAO**

- As operações básicas de armazenamento e recuperação de objetos são agrupadas em uma interface genérica (*GenericDAO*) e uma superclasse genérica (*GenericDAOHibernate*).

- Esta última implementa as operações com uma particular solução de persistência (no caso, Hibernate).

- A interface genérica é estendida por interfaces para entidades específicas que requerem operações adicionais de acesso a dados.

Padrões Arquitetônicos para a Camada de Gerência de Dados

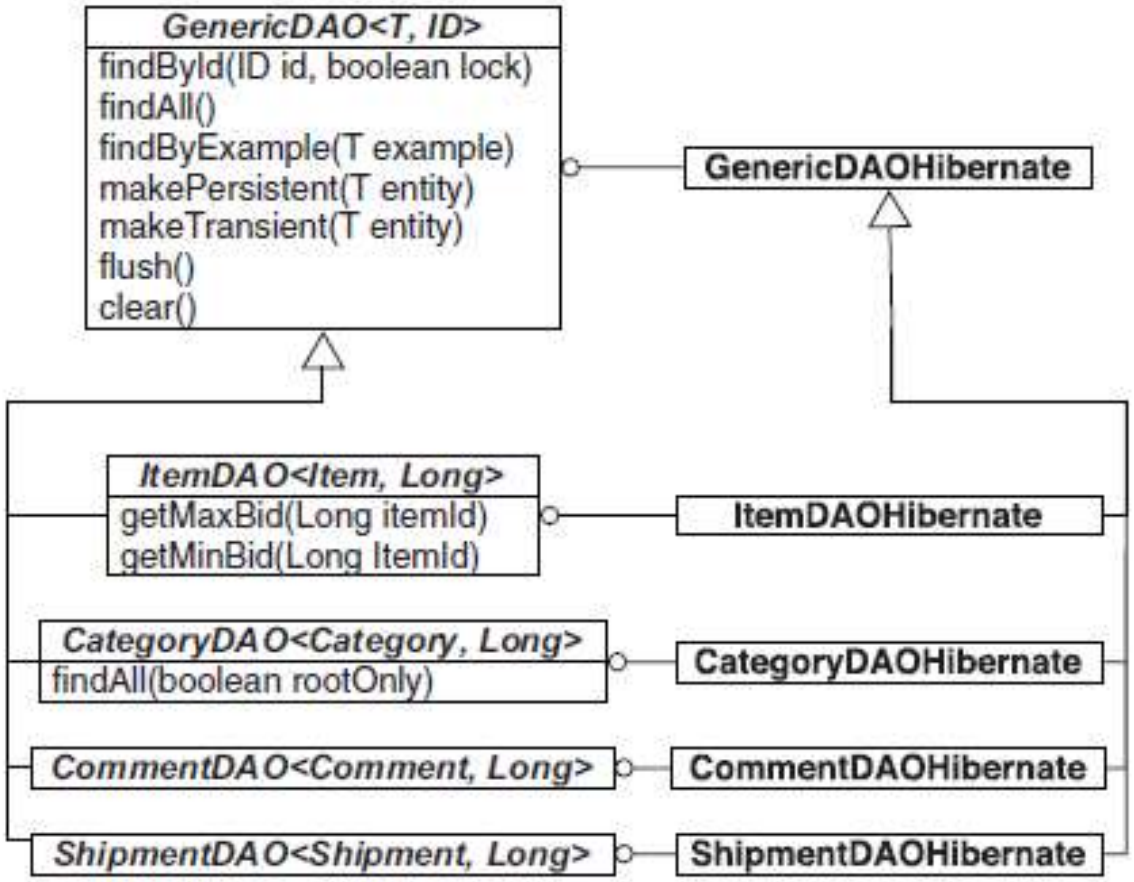
- **Padrão DAO**

- O mesmo ocorre com a hierarquia de classes de implementação.

- Uma característica marcante desta solução é que é possível ter várias implementações de uma mesma interface DAO.

- A estrutura básica do padrão é:

Padrões Arquitetônicos para a Camada de Gerência de Dados



nemo

Frameworks de Persistência

- Atualmente há muitos *frameworks* de persistência disponíveis, tais como Hibernate, Java Data Objects – JDO e Oracle Toplink, todos esses para a linguagem Java.

- Esses *framewoks* se encarregam do mapeamento objeto-relacional e tornam o projeto da camada de persistência mais simples.

- Usando um *framework* dessa natureza, ao invés de obter os dados dos objetos e mesclá-los a uma string de consulta SQL a ser enviada ao SGBD relacional, o desenvolvedor deve informar ao framework como transformar objetos/atributos em tabelas/colunas e chamar métodos simples, como salvar(), excluir() e recuperarPorId(), disponíveis no *framework*.

Frameworks de Persistência

- Em geral, esses *frameworks* disponibilizam também uma linguagem de consulta similar à SQL, porém orientada a objetos, para que consultas possam ser realizadas com facilidade.

- O Hibernate, por exemplo, disponibiliza a linguagem de consulta HQL.

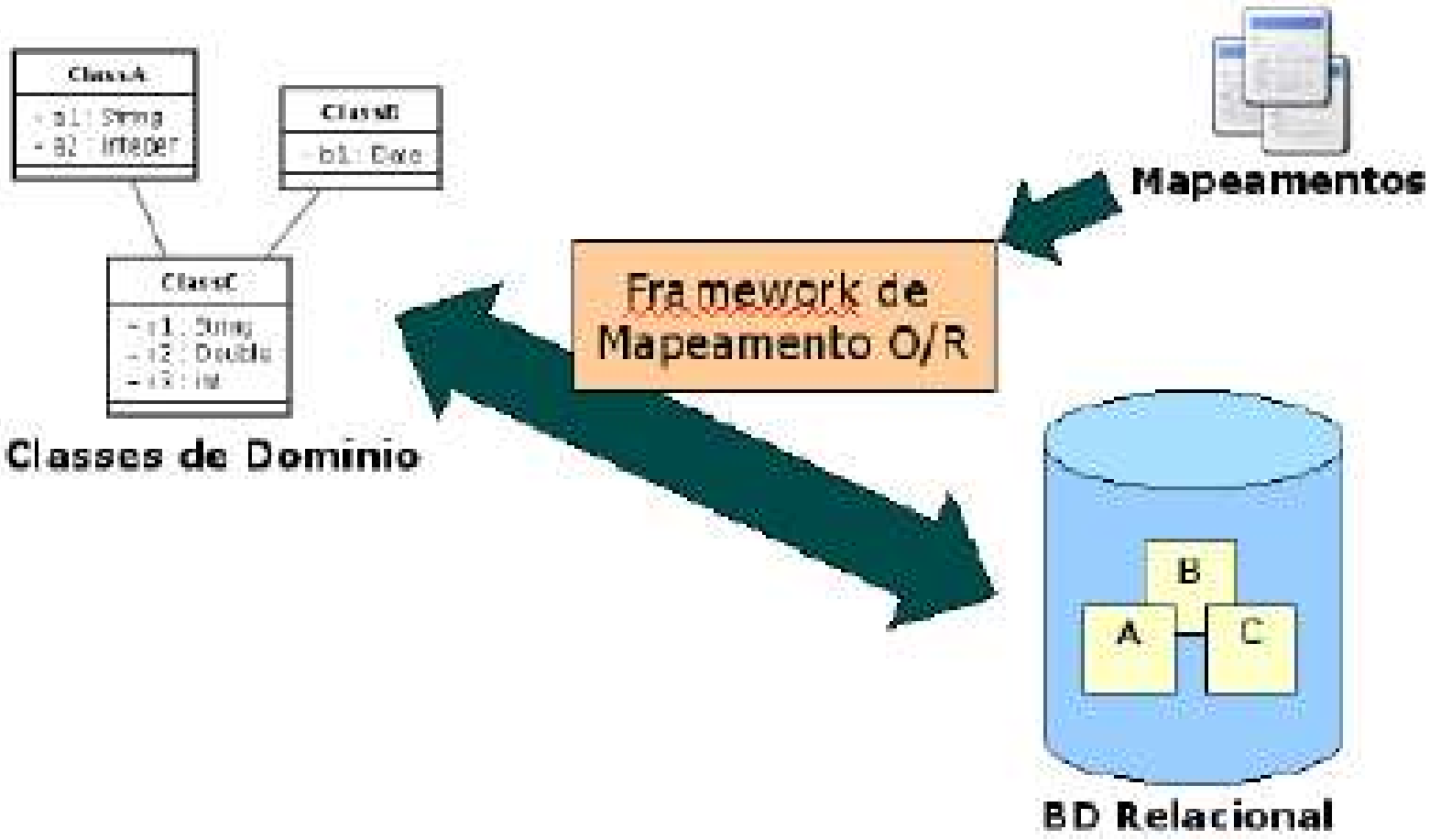
- Usando um framework de persistência, é praticamente desnecessário projetar tabelas, formatos de campos e outros aspectos típicos do projeto de bancos de dados relacionais.

Frameworks de Persistência

- Contudo, às vezes é necessário efetuar ajustes no mecanismo de persistência para acomodar objetos com características especiais ou satisfazer requisitos não funcionais específicos do sistema.

- Assim, é muito importante que o projetista saiba como o mapeamento é tipicamente feito nesses frameworks.

Frameworks de Persistência



nemo

Frameworks de Persistência

- Usando o padrão DAO em conjunto com um *framework* de persistência, o projeto da camada de persistência se restringe à definição das interfaces e classes relativas às entidades do domínio a serem persistidas.

- Em 2006, dada a existência de vários *frameworks* para persistência de objetos Java, foi lançada Java Persistence API (JPA).

- A JPA permite o armazenamento de dados em bancos de dados relacionais sem se ficar preso a um *framework* de persistência específico.

Frameworks de Persistência

- Por meio da JPA, operações de manipulação de tabelas são delegadas para um *framework* de persistência que implemente a JPA.

- Realizado o mapeamento O/R, é possível utilizar um *framework*, que suporte JPA (como Hibernate, Oracle TopLink, Kodo, OpenJPA, entre outros), para fazer as devidas inserções, buscas, exclusões e alterações nos dados da aplicação nas tabelas do banco de dados.

- Dessa forma, o código da aplicação fica independente de *frameworks* de persistência, pois todo o código utilizado para manipular o banco de dados passa a ser da JPA.

That’s all Folks!