Mapeamento Objeto Relacional

Apesar do paradigma orientado a objetos estar sendo cada vez mais difundido no processo de desenvolvimento de software, não existem hoje soluções comerciais robustas e amplamente aceitas neste paradigma para a persistência de dados. Mercado este dominado pelos bancos de dados relacionais. Neste contexto, o mapeamento do modelo orientado a objetos para o relacional é uma necessidade cada vez mais importante no processo de desenvolvimento.

Embora o uso de frameworks de persistência seja comum, é fundamental o entendimento das técnicas de mapeamento objeto relacional. Isso facilitará tanto o uso dos frameworks como também uma análise mais criteriosa dos mapeamentos realizados de forma automática caso seja necessário algum ajuste no código implementado.

Modelo Relacional

A abordagem relacional está baseada no princípio de que as informações em uma base de dados podem ser consideradas relações matemáticas e que estão representadas de maneira uniforme com o uso de tabelas bidimensionais.

Ao se fazer a análise para o desenvolvimento de uma aplicação, é muito importante saber se os tipos de dados a serem armazenados são tipos simples, tais como strings e números. Se este for o caso, é mais indicada a utilização de SGBDs relacionais.

Vantagens

Os SGBDs relacionais possuem uma característica muito importante: são extremamente confiáveis e mais eficientes, se comparados à maioria dos SGBDs orientados a objetos disponíveis no mercado. Além disso, o modelo de dados relacional foi criado para permitir a representação de uma grande variedade de problemas usando um pequeno conjunto de conceitos simples.

Através da linguagem de consulta SQL (Structured Query Language) é possível fazer a busca e recuperação dos dados necessários de forma bastante eficiente.

Dentre as vantagens do modelo relacional, talvez a mais importante esteja relacionada com a persistência dos dados. As regras e rotinas para tratamento da persistência dos dados podem ser criadas no próprio banco de dados relacional.

Desvantagens

Uma das grandes desvantagens do modelo relacional pode ser observada no momento da realização da análise de um sistema a ser implementado: a grande dificuldade em abstrair a realidade, ou seja, traduzir para um modelo de tabelas, com suas relações entre si, suas chaves primárias e estrangeiras, um problema do mundo real.

Modelo Orientado a Objetos

O modelo de dados orientado a objetos é uma extensão do paradigma orientado a objetos, possuindo basicamente os mesmos conceitos. O paradigma orientado a objetos se baseia na construção de aplicações a partir de objetos abstraídos da realidade, com dados e comportamento associados. Esses objetos possuem atributos (propriedades que contém valores que descrevem o objeto) e métodos (especificações de comportamentos dos objetos).

Além disso, o modelo Orientado a Objetos possui outras características importantes, dentre as quais se pode destacar as mostradas na Tabela 1. Já na Tabela 2 podem ser observadas as principais características de um objeto.

Conceito	Descrição

Encapsulamento	Os valores dos atributos e os detalhes da implementação dos métodos estão escondidos de outros objetos. Em banco de dados se diz que um objeto está encapsulado quando o estado é oculto ao usuário e o objeto pode ser consultado e modificado exclusivamente por meio das operações a ele associadas.
Mensagens	Meio de comunicação entre objetos. Troca de mensagens significa chamar um método do objeto.
Herança	**Relacionamento entre classes numa hierarquia**. É a capacidade de criação de uma nova classe a partir de outra existente. Quando uma classe herda características de mais de uma classe, diz-se que houve herança múltipla. As principais vantagens de herança são prover uma maior expressividade na modelagem dos dados, facilitar a reusabilidade de objetos e definir classes por refinamento, podendo fatorar especificações e implementações como na adaptação de métodos gerais para casos particulares.
Polimorfismo	Capacidade de existir diferentes implementações para métodos com a mesma assinatura em diferentes classes da mesma hierarquia de herança. Em sistemas polimórficos, uma mesma operação pode se comportar de diferentes formas em classes distintas.

Tabela 1. Principais características do Modelo Orientado a Objetos

Característica	Descrição
Estado	Indica como se encontram as informações encapsuladas pelo objeto.
Comportamento	Define o modo que um objeto age e reage em termos das suas mudanças de estado.
Identidade	É a propriedade que distingue um objeto de outro. Cada objeto possui uma identidade única.

Tabela 2. Principais características de um Objeto

Vantagens

No modelo Orientado a Objetos, a abstração da realidade é mais simplificada, pois um objeto nada mais é do que uma abstração direta da realidade. Por exemplo: um computador tem teclado, cpu e mouse. O teclado é um objeto, a cpu é outro objeto e o mouse também é um objeto.

Outra grande vantagem deste modelo é a questão da reutilização. Um objeto pode ser facilmente reutilizado no mesmo programa ou até mesmo em programas diferentes.

O modelo orientado a objetos foi projetado para a criação e representação de estruturas complexas de uma maneira coerente e uniforme. Isto representa uma grande vantagem em relação ao modelo de dados relacional, que foi criado sem se preocupar com o armazenamento de estruturas mais complexas. Outro ponto a favor do modelo orientado a objetos é a facilidade em expressar relações complexas entre objetos.

Desvantagens

O paradigma orientado a objetos apresenta um problema: os bancos de dados atuais não oferecem uma boa aderência aos princípios da orientação a objetos. Isto é, em termos de armazenamento, os bancos de dados orientados a objetos não conseguiram ainda substituir a tecnologia comprovadamente eficiente dos bancos de dados relacionais.

Em função disso, os administradores e desenvolvedores de bancos de dados acabam ficando em uma situação difícil, pois mesmo querendo adotar a orientação a objetos, eles têm que parar na hora de manipular seus dados. Até mesmo linguagens de programação orientadas a objetos, como Java, acessam os bancos de dados de maneira convencional e utilizando SQL. Com isso, um mesmo programa acaba tendo uma parte orientada a objetos e outra parte estruturada, fazendo com que as características da orientação a objetos não possam ser exploradas na sua plenitude.

Devido a isso, o que está ocorrendo nos dias de hoje é que o desenvolvimento é orientado a objetos, mas o banco de dados é relacional ou objeto-relacional. No modelo orientado a objetos, a implementação acaba se tornando mais complicada do que no modelo relacional, pois as estruturas de dados usadas no banco e as usadas na programação podem ser completamente diferentes. Isso reforça a necessidade da criação de uma camada de persistência, fazendo com que se gaste um bom tempo do desenvolvimento para mapear as estruturas da programação em estruturas do banco de dados.

Bancos de Dados Orientados a Objetos

Os bancos de dados orientados a objetos podem ser divididos em dois grupos:

- Bancos de Dados Puramente Orientados a Objetos (BDPOO): baseia-se somente no modelo de dados orientado a objetos. Está baseado no conceito de objetos persistentes e usa declarações de classes muito semelhantes às declarações das linguagens orientadas a objetos;

- Bancos de Dados Objeto-Relacionais (BDOR): correspondem a bancos relacionais que possibilitam o armazenamento de objetos.

Características dos BDPOO

As principais características dos Bancos de Dados Puramente Orientados a Objetos são:

- Permitem a representação de relacionamentos 1-n, n-n e de herança;

- A representação de um relacionamento é feita incluindo-se dentro de um objeto os “object identifiers” dos outros objetos com os quais ele se relaciona;

- “Object Identifier” é um identificador interno do banco de dados para cada objeto. Os “object identifiers” são atribuídos e utilizados somente pelo SGBD, nunca pelos usuários.

- Não possuem um padrão único de implementação como os bancos relacionais. Assim, cada produto tem a sua própria especificação para criação de classes e relacionamentos;

Os BDPOO tendem a seguir um padrão estabelecido pelo ODMG (Object Database Management Group). A última versão disponibilizada pelo ODMG é a versão 3.0, que pode ser obtida em no Operational Database Management Systems.

Características dos BDOR

Com relação aos Bancos de Dados Objeto-Relacionais, pode-se dizer que as principais características são:

- Um banco objeto-relacional é um banco que permite o armazenamento de objetos em suas tabelas;

- Uma classe representa um domínio, atuando como um data type para uma coluna. A classe, diferentemente dos bancos orientados a objetos, não representa mais um elemento envolvido em relacionamentos;

- Todas as regras de um banco relacional continuam válidas;

- Uma coluna cujo data type seja uma classe, só poderá ter uma instância desta classe. Imagine, por exemplo, uma coluna cujo data type seja uma classe TELEFONE. Nos BDOR, esta coluna não poderá ter várias instâncias da classe TELEFONE, mas apenas uma. Se o banco fosse OO, poderia ter mais do que uma.

Dentre as principais vantagens dos BDOR, pode-se citar que todo o suporte para ao paradigma orientado a objetos está presente. Além disso, já existe um padrão de implementação estabelecido, determinando uma certa facilidade de convivência com os sistemas já existentes.

Na Tabela 3 podem ser observados alguns dos principais BDOR disponíveis no mercado.

Fabricante	Descrição
IBM	Informix e DB2
Oracle Corporation	Oracle 9i
PostgreSQL Global Development Group	PostgreSQL
Micro Database System, Inc.	TITANIUM
Intersystems	Caché

Tabela 3. Principais BDOR no mercado

Mapeamento Objeto Relacional

Em termos conceituais, uma Camada de Persistência de Objetos é uma biblioteca que permite a realização do processo de persistência (isto é, o armazenamento e manutenção do estado dos objetos em algum meio não-volátil, como um banco de dados) de forma transparente.

Dentre as diversas vantagens obtidas com a utilização da Camada de Persistência, pode-se destacar o fato do analista/programador poder trabalhar como se estivesse em um sistema completamente orientado a objetos. Outra vantagem muito importante é que os acessos realizados diretamente ao banco de dados da aplicação são isolados, e os processos de construção de consultas e operações de manipulação de dados são centralizados em uma camada de objetos inacessível ao programador. Esse encapsulamento torna as aplicações mais confiáveis, permitindo até mesmo que o próprio SGBD ou a estrutura de suas tabelas possam ser alterados sem a necessidade de revisar e recompilar os programas.

Apesar disso, o modelo de classes de um projeto orientado a objetos não pode simplesmente ser traduzido em um script SQL de criação de banco de dados, é necessário realizar o mapeamento de objetos em um modelo de dados Entidade-Relacionamento.

Regras para Mapeamento

Para garantir que todas as características do modelo de objetos sejam reproduzidas fielmente pelo banco de dados relacional, alguns aspectos merecem ser destacados. Os objetos formam unidades que encapsulam atributos e operações. Os bancos de dados relacionais representam de forma bastante eficiente os atributos, mas são limitados na representação das operações. Pode-se tentar estabelecer uma analogia entre objetos e tabelas, e entre atributos e colunas.

Para mapear um objeto em uma tabela relacional, geralmente os atributos desse objeto são representados através de colunas na tabela. Entretanto, essa não é uma regra que pode ser seguida ao pé da letra, pois existem outras considerações a serem feitas (tipos dos dados, tamanho dos campos, entre outras). Tais considerações podem fazer com que um atributo seja mapeado em várias colunas, ou então que vários atributos sejam mapeados em apenas uma coluna.

Ao se realizar uma transposição de um modelo orientado a objetos para um modelo relacional, algumas regras devem ser seguidas (vale destacar aqui que essas regras não são rígidas):

- Todas as tabelas (ou relações) devem ter uma chave primária. Em um sistema orientado a objetos, cada objeto é único. Essa unicidade é garantida através da introdução de um “identificador de objetos” (OID – Object IDentifier). Esse OID é gerado pelo sistema, não podendo ser alterado pelo usuário. Seu valor não depende dos valores dos atributos do objeto.

Para fazer essa implementação no modelo relacional, deve ser criado um atributo OID para cada tabela. Na Figura 1 é mostrado um exemplo de como seria o mapeamento do objeto Pedido para a tabela Pedido.

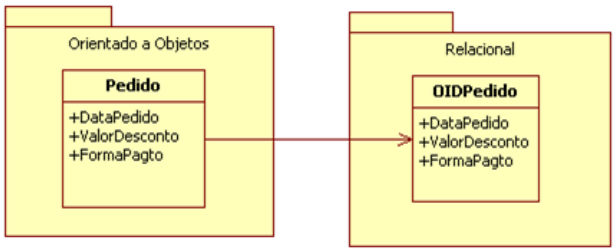

Figura 1. Exemplo 1

- Mapeamento de atributos. Existem três tipos de atributos para serem mapeados. Os atributos simples são mapeados para colunas. Os atributos compostos podem ser mapeados em várias colunas. Já os atributos multivalorados devem ser mapeados em tabelas onde a chave primária é composta pela chave primária da tabela que representa a classe que contém o atributo multivalorado e pela chave primária que representa o atributo multivalorado.

Na Figura 2 pode ser observado um exemplo de mapeamento de um atributo simples (Nome e DtNascimento) e de um atributo composto (Endereco). Já na **Figura 3** pode ser observado um exemplo de mapeamento de um atributo multivalorado (Telefone).

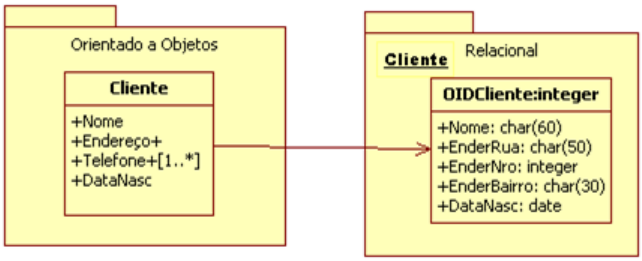

Figura 2. Atributos simples e compostos

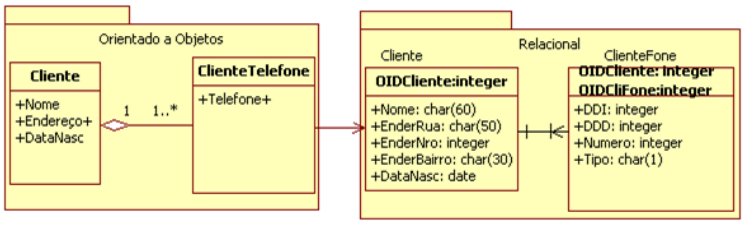

Figura 3. Atributos multivalorados

- Herança: pode-se mapear este conceito de três formas. A primeira é simplesmente criar uma tabela para cada classe. A segunda forma é criar uma única tabela para toda a hierarquia de classes. Por último, pode-se criar uma tabela para cada classe concreta.

Na técnica de criação de uma tabela para cada classe (Figura 4), os atributos da tabela são os atributos específicos da classe e mais uma coluna de chave estrangeira que referencia a chave primária da tabela pai. As desvantagens do uso dessa técnica é que são geradas muitas tabelas no banco de dados, fazendo com que haja uma demora maior para ler e gravar os dados. Por esse mesmo motivo, algumas consultas acabam sendo bastante dificultadas, obrigando a criação de views para agilizar o processo.

Se for utilizada a segunda forma (Figura 5), a classe raiz é tomada por base, pois é nela que todos os atributos são armazenados. Essa técnica facilita as consultas, pois os dados de um objeto estão em uma única tabela O principal problema é que, potencialmente, há um desperdício de espaço no banco de dados. Além disso, a performance pode ser prejudicada.

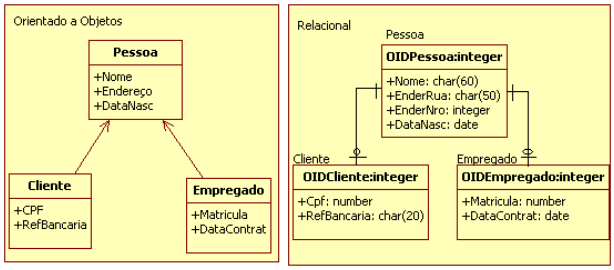

Figura 4. Mapeamento de uma tabela por classe

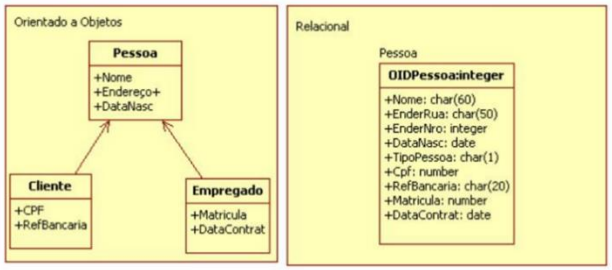

Figura 5. Uma única tabela para toda a hierarquia de classes

Caso se opte por criar uma tabela para cada classe concreta (Figura 6), deve-se incluir em cada tabela tanto os atributos específicos, quanto os atributos herdados da classe que ela representa. Como os dados de uma classe ficam todos em uma única tabela, as consultas são facilitadas.

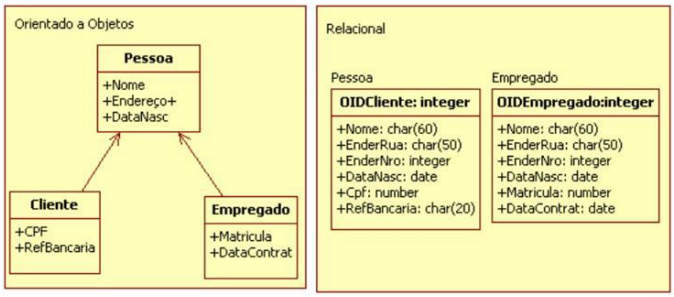

Figura 6. Uma tabela para cada classe concreta

- Associações Muitos-para-Muitos: neste caso devemos criar uma tabela associativa em que a chave primária é composta pelas chaves primárias das tabelas associadas à famosa e já conhecida entidade ou tabela associativa. Na Figura 7 pode ser visto um exemplo da transposição de uma associação muitos-para-muitos. No modelo orientado a objetos havia dois objetos (Aluno e Disciplina), que no modelo relacional passaram a ser representados por três tabelas (Aluno, Disciplina e Frequenta).

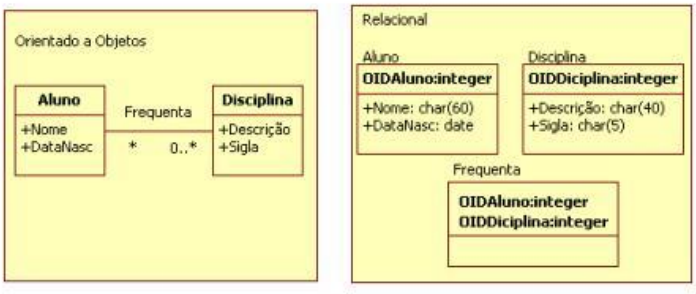

Figura 7. Associação muitos-para-muitos

- Associações Muitos-para-Muitos com Classe de Associação: neste caso, aplica-se a regra da associação muitos-para-muitos e os atributos da classe associativa permanecerão na tabela que é gerada para mapear a associação. Na Figura 8 pode-se observar um exemplo de como é feita a transposição de um associação de muitos para muitos

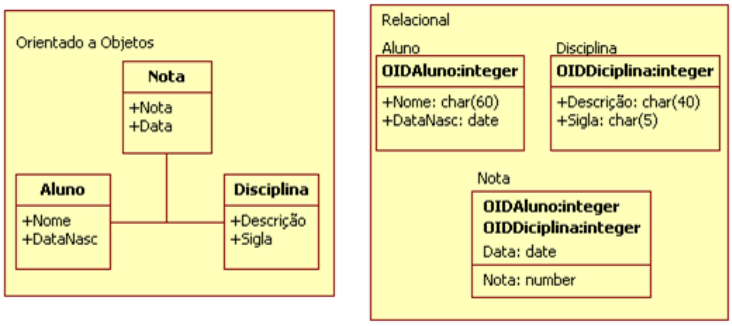

Figura 8. Associação muitos-para-muitos com agregação

- Associações Um-para-Muitos: neste caso, a tabela cujos registros podem ser endereçados diversas (lado Muitos do relacionamento) vezes é a que herda a referência da tabela cuja correspondência é unitária. Na Figura 9 pode-se observar um exemplo onde a tabela Pedido possui uma chave estrangeira (FK) que referencia a tabela Cliente

Figura 9. Associação um-para-muitos

- Associações Um-para-Muitos com Classe de Associação: neste caso, aplica-se a regra da associação um-para-muitos e os atributos da classe associativa são herdados como atributos normais pela tabela que herda a chave estrangeira (ver Figura 10)

Figura 10. Associação um-para-muitos com agregação

- Associações Um-para-Um: nesse tipo de associação, o analista deve optar por gerar uma única tabela no modelo relacional, ou então gerar duas tabelas (uma para cada classe).

Se for escolhida a primeira opção, os atributos da classe agregada devem ser colocados na mesma tabela da classe agregadora. Nessa técnica, a performance é melhor, pois é preciso acessar uma única

tabela. Além disso, o objeto agregado é automaticamente excluído quando o objeto agregador é eliminado, não havendo a necessidade da implementação de triggers ou rotinas especiais na aplicação para garantir a consistência do banco de dados. A grande desvantagem dessa técnica é que a incorporação dos atributos aumenta a quantidade de páginas que são recuperadas em cada acesso à tabela.

No caso da opção ser pela geração de duas tabelas, uma delas deve herdar como um atributo normal (chave estrangeira) a chave primária da outra tabela. Essa técnica facilita a manutenção das tabelas e torna a estrutura do banco de dados mais flexível. Porém, as consultas necessitam de uma operação de junção (join) ou pelo menos dois acessos ao banco de dados. Se o acesso aos dados de ambas as tabelas não for muito frequente, esta alternativa é aceitável, caso contrário, essa alternativa pode não ser a mais adequada. Outra desvantagem de gerar uma tabela para cada classe é que para manter a consistência do banco de dados entre essas tabelas, por ocasião de operações de exclusão de objetos, é necessário implementar triggers ou rotinas especiais na aplicação.