Modelagem de Dados Tutorial

O que é modelagem de dados?

Modelagem de dados é o ato de explorar estruturas orientadas a dados. Como outros artefatos de modelagem, modelos de dados podem ser usados para uma variedade de propósitos, desde modelos conceituais de alto nível até modelos físicos de dados. Do ponto de vista de um desenvolvedor atuando no paradigma orientado a objetos, modelagem de dados é conceitualmente similar à modelagem de classes. Com a modelagem de dados identificamos tipos de entidades da mesma forma que na modelagem de classes identificamos classes. Atributos de dados são associados a tipos de entidades exatamente como associados atributos e operações às classes. Existem associações entre entidades, similar às associações entre classes – relacionamento, herança, composição e agregação são todos conceitos aplicáveis em modelagem de dados.

Modelagem de dados tradicional é diferente da modelagem de classes porque o seu foco é totalmente nos dados – modelos de classes permitem explorar os aspectos comportamentais e de dados em um domínio de aplicação, já com o modelo de dados podemos apenas explorar o aspecto dado. Por causa deste foco, projetistas de dados tendem a serem melhores em identificar os dados "corretos" em uma aplicação do que modeladores de objetos. No entanto, algumas pessoas modelam métodos de banco de dados (stored procedures, stored functions e triggers) quando estão realizando a modelagem física dos dados.

Apesar de o foco deste artigo ser modelagem de dados, existem normalmente alternativas para artefatos orientados a dados. Por exemplo, quando estamos na modelagem conceitual, os diagramas ORM (Object Role Model) não são a única opção. Além do Modelo Lógico de Dados, é comum a criação de diagramas de classes da UML.

Como modelos de dados são usados na prática?

Embora as questões de metodologias sejam abordadas depois, precisamos discutir como modelos de dados podem ser usados na prática para melhor entendê-los. Provavelmente, iremos nos deparar a três estilos básicos de modelos de dados:

- **Modelos de dados conceituais**: esses modelos, algumas vezes chamados modelos de domínio, são tipicamente usados para explorar conceitos do domínio com os envolvidos no projeto. Em equipes ágeis, modelos conceituais de alto nível são normalmente criados como parte do esforço inicial do entendimento dos requisitos do sistema, pois eles são usados para explorar as estruturas e conceitos de negócio estáticos de alto nível. Em equipes tradicionais (não ágeis), modelos de dados conceituais são normalmente criados como precursores aos modelos lógicos de dados (MLD) ou suas alternativas.
- **Modelos Lógico de Dados (MLDs)**: MLDs são usados para explorar os conceitos do domínio e seus relacionados. Isso pode ser feito para o escopo de um simples projeto ou para uma empresa inteira. MLDs descrevem os tipos de entidades lógicas, tipicamente referenciadas simplesmente como tipos de entidades, os atributos de dados que descrevem essas entidades e os relacionamentos entre as entidades. MLDs são raramente usados em projetos ágeis apesar de normalmente estarem presentes em projetos tradicionais (onde eles raramente adicionam muito valor na prática).
- **Modelos Físicos de Dados (MFDs)**: MFDs são usados para projetar o esquema interno de um banco de dados, descrevendo as tabelas de dados, as colunas de dados das tabelas e o relacionamento entre as tabelas. MFDs normalmente são bastante úteis em projetos ágeis e tradicionais, por isso este será o foco deste artigo: modelagem física dos dados.

Embora MLDs e MFDs parecerem similares, e eles de fato são, o nível de detalhes que eles modelam pode ser significativamente diferente. Isso porque o objetivo de cada diagrama é diferente – podemos usar um MLD

para explorar conceitos do domínio com os envolvidos no projeto e MFD para definir o projeto do banco de dados. A **Figura 1** apresenta um simples MLD e a **Figura 2** um simples MFD, ambos modelando o conceito de clientes e endereços, assim como o relacionamento entre eles. Ambos os diagramas seguem a notação de Barker, que será descrita a seguir. Note como o MFD mostra mais detalhes, incluindo uma tabela associativa necessária para implementar a associação, assim como as chaves necessárias para manter os relacionamentos. Mais detalhes sobre esses conceitos serão descritos a seguir.

MFDs devem também refletir os padrões de nomenclatura de banco de dados da organização. Neste caso, uma abreviação do nome da entidade é colocado para cada nome de coluna e uma abreviação para "número" foi consistentemente introduzida. Um MFD deve também indicar os tipos de dados das colunas, tais como integer e char(5). Apesar de a **Figura 2** não mostrá-las, tabelas de referência como para o endereço é usado, assim como para estados e países estão implícitos pelos atributos END_USADO_CODIGO,END_ESTADO_CODIGO, END_PAIS_CODIGO.

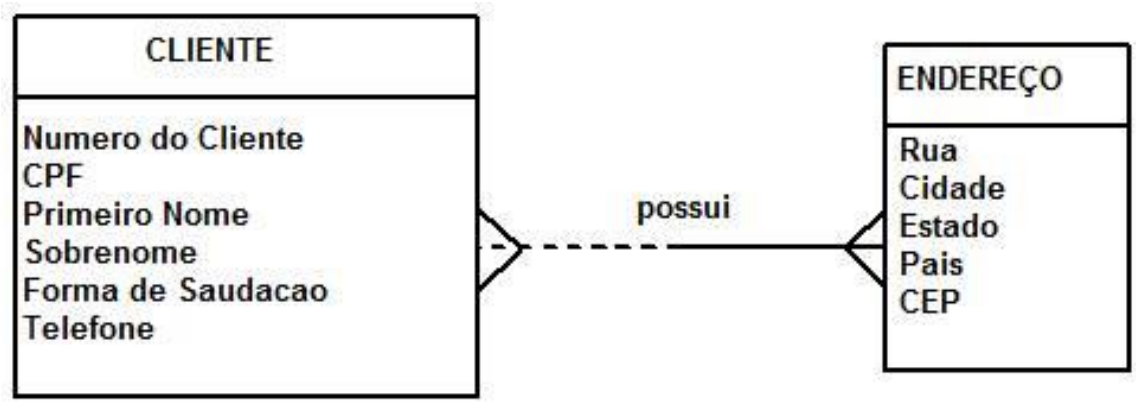

Figura 1. Um simples modelo lógico de dados

Figura 2. Um simples modelo físico de dados

Modelos de dados podem ser usados efetivamente tanto no nível da empresa como de projetos. Os arquitetos da empresa normalmente criarão um ou mais MLDs de alto nível que descreve as estruturas de dados que apoiam toda a empresa, normalmente chamados de modelos de dados da empresa ou modelos de informação da empresa. Um modelo de dados da empresa é uma das várias visões que os arquitetos da empresa podem escolher para manter e apoiar – outras visões podem explorar a infraestrutura de rede/hardware, a estrutura da organização, infraestrutura de softwares o processo de negócios, dentre outros. Esses modelos provêm informações que uma equipe de projeto pode usar como conjunto de restrições e também como descrição da estrutura do sistema.

Equipes de projeto tipicamente criarão MLDs como um dos principais artefatos de análise quando seu ambiente de implementação é predominantemente procedural por natureza, por exemplo quando estão usando COBOL estruturado como linguagem de implementação. MLDs são também boas escolhas quando um projeto é orientado a dados, como um data warehouse ou sistema de relatório. No entanto, MLDs são normalmente escolhas ruins quando uma equipe de projeto está usando tecnologias orientadas a objeto ou baseadas em componentes porque os desenvolvedores trabalhariam melhor com diagramas UML ou quando o projeto não é orientado a dados. Como uma dica de modelagem, aplique os artefatos corretos para aquele trabalho a ser desenvolvido.

Quando um banco de dados relacional é usado para armazenar dados, equipes de projeto são aconselhadas a criar um MFD para modelar um esquema interno. MFD normalmente é apenas um dos artefatos de projeto críticos para projetos de desenvolvimento de aplicações de negócio.

O que dizer sobre modelos conceituais?

Muitos profissionais de dados preferem criar um ORM (Object-Role Model), como o apresentado no exemplo da **Figura 3**, em vez de um MLD para um modelo conceitual. A vantagem é que a notação é muito simples, algo que os envolvidos no projeto podem rapidamente interpretar, apesar da desvantagem que seria o fato de os modelos se tornarem grandes rapidamente. ORMs nos permite primeiramente explorar os exemplos de dados reais em vez de simplesmente saltar para uma abstração potencialmente incorreta – por exemplo, a **Figura 3** examina o relacionamento entre clientes e um endereço em detalhe.

Figura 3. Um simples ORM (Object-Role Model)

Normalmente ORMs são usados para explorar o domínio da aplicação com os envolvidos no projeto, mas depois ele é substituído por um artefato mais tradicional, como um MLD, um **diagrama de classes** ou até um MFD.

Notações comuns de modelagem de dados

A **Figura 4** apresenta um resumo da sintaxe das quatro notações mais comuns para modelagem de dados: Engenharia da Informação (EI), Notação de Barker, IDEF1X e **UML** (Unified Modeling Language). Este diagrama não tem a pretensão de ser altamente compreensivo, mas sim prover uma visão geral básica sobre as notações. Além disso, para não se estender muito no texto, não foi possível descrever a abordagem altamente detalhada para nomenclatura de relacionamento como sugerido por Barker. Apesar disso, foi provida uma breve discussão de cada notação na **Tabela 1**.

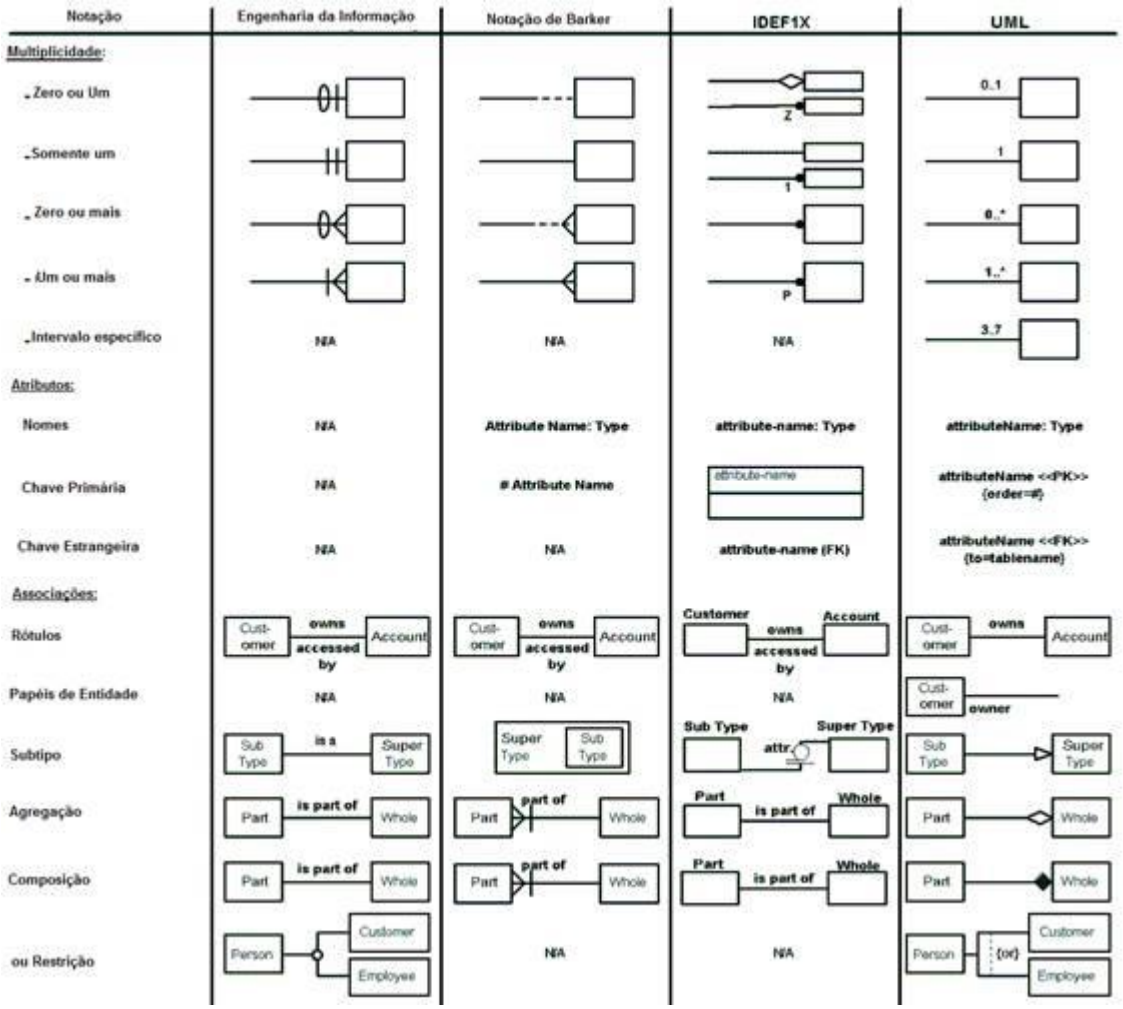

Figura 4. Comparando a sintaxe das notações comuns para modelagem de dados

Notação	Comentários
EI	A notação EI é simples e fácil de ser lida, e é bem abrangente para modelagem de dados de negócio e modelagem lógica de alto nível. O único ponto negativo desta notação é que ela não suporta a identificação de atributos de uma entidade. Assume-se que os atributos serão modelados com outro diagrama ou simplesmente descrito em uma documentação de apoio.
Barker	A notação de Barker é uma das mais populares, sendo apoiadas por várias ferramentas (ex: Oracle toolset), é bem abrangente para todos os tipos de modelos de dados. Esta abordagem pode se tornar complicada com hierarquias que possuem vários níveis de profundidade.
IDEF1X	Esta notação é a mais complexa, e foi originalmente intencionada para modelagem física, mas foi mal aplicada para modelagem lógica. Esta notação tem sido abandonada por todos, então evite-a se possível.
UML	Esta não chega a ser uma notação de modelagem de dados oficial. Apesar de várias sugestões para um perfil UML de modelagem de dados existirem, nenhum é completo e não são oficializados pela UML.

Tabela 1. Discutindo notações comuns de modelagem de dados

Como modelar dados

É crucial para um desenvolvedor de aplicação ter uma noção dos fundamentos de modelagem de dados não apenas para ler os modelos de dados, mas também para trabalhar efetivamente com os DBAs responsáveis pelos aspectos relacionados aos dados do projeto. O objetivo ao ler esta seção não é aprender como se tornar um modelador de dados, mas sim obter uma apreciação a respeito do que é envolvido nesta tarefa.

As seguintes tarefas são realizadas de forma iterativa:

- Identificar os tipos de entidade;
- Identificar atributos;
- Aplicar convenção de nomes;
- Identificar relacionamentos;
- Associar chaves;
- Normalizar para reduzir a redundância dos dados;
- Diversificar para melhorar o desempenho.

Identificar os tipos de entidade

Um tipo de entidade, ou simplesmente entidade, é conceitualmente similar ao conceito de orientação a objeto de uma classe – um tipo de entidade representa uma coleção de objetos similares. Um tipo de entidade pode representar uma coleção de pessoas, lugares, coisas, eventos ou conceitos. Exemplos de entidades em um sistema de vendas incluiria: Cliente, Endereço, Venda, Item e Taxa. Se estivéssemos modelando classes, esperaríamos descobrir classes exatamente com esses nomes. No entanto, a diferença entre uma classe e um tipo de entidade é que classes possuem dados e comportamentos, enquanto que tipos de entidade possuem apenas dados.

Idealmente, uma entidade deveria ser normal, descrevendo de forma coesa uma informação do mundo real. Uma entidade normalmente descreve um conceito, tal como uma classe coesa modela um conceito. Por exemplo, cliente e venda são claramente dois conceitos diferentes, portanto, faz sentido modelá-los como entidades diferentes.

Identificar atributos

Cada tipo de entidade terá um ou mais atributos de dados. Por exemplo, na **Figura 1** podemos ver que a entidade Cliente possui atributos como Primeiro Nome e Sobrenome e na **Figura 2** que a tabela TCLIENTE possui colunas de dados correspondentes CLI_PRIMEIRO_NOME e CLI_SOBRENOME (uma coluna é a implementação de um atributo de dados em um banco de dados relacional).

Atributos devem ser coesos do ponto de vista do domínio da aplicação. Na **Figura 1** decidimos que queríamos modelar o fato de pessoas possuírem primeiro nome e sobrenome em vez de apenas um nome (ex: “Cláudio” e “Dias” VS. “Cláudio Dias”). Usar o nível de detalhe correto pode ter um impacto significativo no esforço de desenvolvimento e manutenção. Refatorar uma simples coluna de dados em várias colunas pode ser difícil, o que pode resultar em construir o sistema com elementos desnecessários e, portanto, provoca um maior custo de desenvolvimento e de manutenção do que realmente necessário.

Aplicar convenções de nome

Sua organização deve dispor de normas e diretrizes aplicáveis à modelagem de dados, algo que você deve ser capaz de obter dos administradores da empresa (se não existir você deve fazer algum lobby para incluí-lo).Essas diretrizes devem incluir as convenções de nomenclatura para a modelagem lógica e física, as convenções de nomenclatura lógica devem ser focadas na capacidade de leitura de humanos, enquanto as convenções de nomenclatura física refletirão considerações técnicas. Você pode ver claramente que diferentes convenções de nomenclatura foram aplicadas nas **Figuras 1** e **2**.

A ideia básica é que desenvolvedores sigam um conjunto comum de padrões de modelagem em um projeto de software. Tal como é importante seguir convenções comuns de codificação, um código limpo que segue as diretrizes escolhidas é mais fácil de ser compreendido. Isso funciona da mesma forma para as convenções de modelagem de dados.

Identificar relacionamentos

No mundo real, entidades possuem relacionamentos entre elas. Por exemplo, clientes FAZEM compras, clientes MORAM EM endereços e itens de venda SÃO PARTE DAS vendas. Todos esses termos em maiúsculo definem relacionamentos entre entidades. Os relacionamentos entre entidades são conceitualmente idênticos aos relacionamentos (associações) entre objetos.

A **Figura 5** descreve um MLD parcial para um sistema de compra online. A primeira coisa a se notar são os vários estilos aplicados aos nomes dos relacionamentos e papéis – diferentes relacionamentos requerem diferentes abordagens. Por exemplo, o relacionamento entre Cliente e Venda possui dois nomes, compra e é comprado por, mesmo o relacionamento entre essas entidades sendo apenas um. Neste exemplo, tendo um segundo nome no relacionamento, a ideia seria especificar como ler o relacionamento em cada direção. O ideal seria colocar apenas um nome por relacionamento.

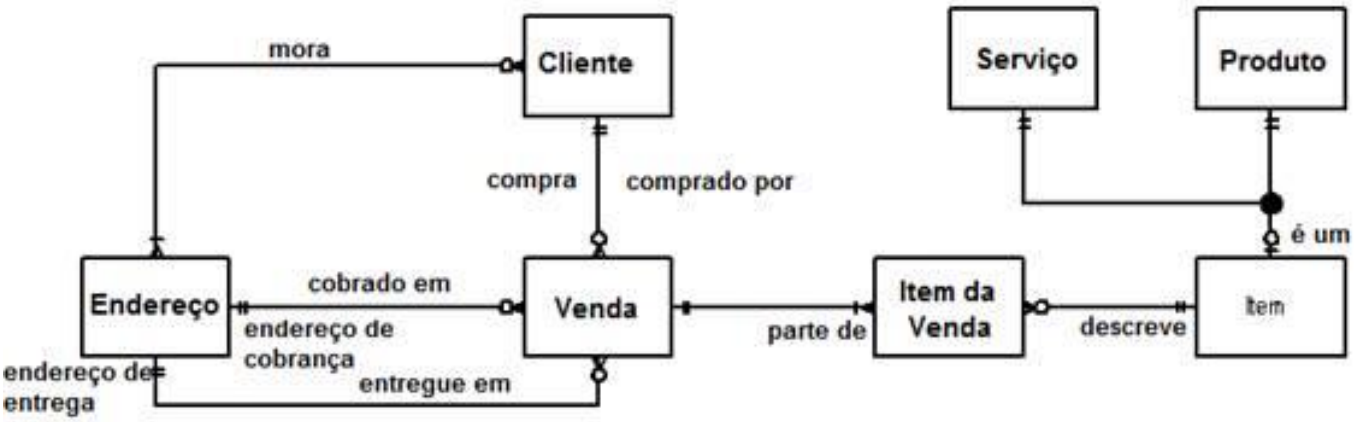

Figura 5. Um modelo lógico de dados (notações Engenharia da Informação)

Precisamos também identificar a cardinalidade e opcionalmente de um relacionamento (a UML combina os conceitos de cardinalidade e opcionalmente de um relacionamento em um conceito único de multiplicidade). Cardinalidade representa o conceito de "quantos" enquanto opcionalmente representa o conceito de "se é obrigatória a existência da entidade". Por exemplo, não é suficiente saber que clientes fazem vendas. Quantas vendas um cliente pode realizar? Nenhuma, uma ou várias? Além disso, os relacionamentos existem nos dois sentidos: não apenas clientes fazem vendas, mas vendas são realizadas por clientes. Isso nos leva a questões como: quantos clientes podem ser envolvidos em uma dada venda e é possível ter uma venda com nenhum cliente envolvido? A **Figura 5** mostra que clientes fazem nenhuma ou mais vendas e que qualquer venda é realizada por um e somente um cliente. Ela também mostra que um cliente possui um ou mais endereços e que qualquer endereço possui zero ou mais clientes associados a ele.

Apesar de a UML distinguir entre diferentes tipos de relacionamentos – associações, hierarquia, agregação, composição e dependência – modeladores de dados normalmente não estão por dentro dessa questão. Subtipo,

uma aplicação de hierarquia, é normalmente encontrada em modelos de dados. Agregação e composição são muito menos comum, assim como dependências, que são tipicamente uma construção de software e portanto não aparecem no modelo de dados, a menos que tenhamos um modelo físico de dados bastante detalhado que mostre como `views`, `triggers` ou `stored procedures` dependem de outros aspectos do esquema do banco de dados.

Associar chaves

Existem duas estratégias fundamentais para associar chaves às tabelas. Primeiro, podemos associar uma chave natural que é um ou mais atributos de dados existentes que são únicos para o conceito do negócio. Imaginemos uma tabela Cliente, por exemplo. Ela possui de imediato duas chaves candidatas, as colunas `NumeroCliente` e `CPF`. A segunda forma é introduzindo uma nova coluna, chamada chave substituta, que é uma chave que não possui qualquer significado para o negócio. Um exemplo disso seria uma coluna `idEndereco` de uma tabela `Endereco`. Endereços não possuem uma chave natural “trivial” porque seria necessário usar todas as colunas da tabela `Endereco` para formar uma chave. Assim, introduzir uma chave substituta é uma opção muito melhor neste caso.

O debate entre "natural vs. substituta" é um das grandes questões religiosas na comunidade de banco de dados. O fato é que não existe estratégia perfeita, e com o tempo percebemos que na prática algumas vezes fazem sentido usar chaves naturais e em outras situações é mais adequado o uso de chaves substitutas.

Normalizar para reduzir redundância de dados

Normalização de dados é um processo no qual atributos de dados em um modelos de dados são organizados para aumentar a coesão dos tipos de entidade. Em outras palavras, o objetivo da normalização de dados é reduzir e até eliminar redundância de dados, uma questão importante para desenvolvedores, pois é incrivelmente difícil armazenar objetos em um banco de dados relacional que mantém a mesma informação em vários lugares. A **Tabela 2** resume as três principais regras de normalização descrevendo como aumentar os níveis de normalização em tipos de entidade.

Com respeito à terminologia, um esquema de dados é considerado estar em um nível de normalização do seu tipo de entidade menos normalizado. Por exemplo, se todos os tipos de entidade estão na segunda forma normal (2NF) ou maior, então dizemos que o esquema de dados está na 2NF.

Nível	**Regra**
Primeira Forma Normal (1NF)	Uma entidade está na 1NF quando ela não contém grupos de dados repetidos.
Segunda Forma Normal (2NF)	Uma entidade está na 2NF quando ela está na 1NF e quando todos seus atributos que não são chaves primárias são completamente dependentes de sua chave primária.
Terceira Forma Normal (3NF)	Uma entidade está na 3NF quando ele está na 2NF e quando todos seus atributos são diretamente dependentes da chave primária.

Tabela 2. Regras de normalização de dados

A **Figura 6** descreve um esquema de banco de dados na ONF enquanto que a **Figura 7** descreve um esquema normalizado na 3NF.

Por que normalização de dados? A vantagem de ter do esquema de dados altamente normalizado é que a informação é armazenada em um lugar apenas, reduzindo a possibilidade de dados inconsistentes. Além disso, esquemas de dados altamente normalizados em geral são conceitualmente mais próximos dos esquemas orientados a objeto, pois os objetivos da orientação a objetos de promover alta coesão e pouco acoplamento entre as classes resulta em soluções similares (ao menos do ponto de visa de dados). Isso geralmente torna mais simples mapear os objetos para o esquema de dados. Infelizmente, a normalização normalmente traz um custo para o desempenho. Com o esquema de dados da **Figura 6** todos os dados para uma venda estão armazenados em uma linha (assumindo que vendas poderão ter até dois itens), simplificando o acesso. Com o esquema de dados da **Figura 6** podemos rapidamente determinar a quantidade total de uma venda lendo uma única linha da tabela. Para fazer o mesmo com o esquema de dados da **Figura 7**, precisamos ler dados a partir de uma linha na tabela Venda, dados a partir de linhas na tabela `ItemVenda` para aquela venda e dados a partir das linhas correspondentes na tabela Item. Para esta consulta, o esquema de dados da **Figura 6** provavelmente obtém melhor resultado.

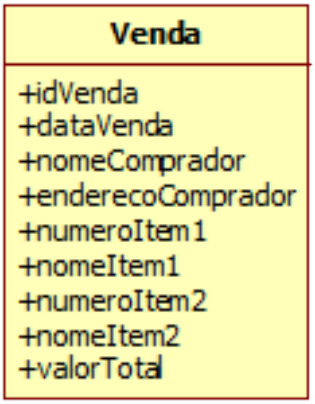

Figura 6. Um esquema de dados inicial para Venda (notação UML)

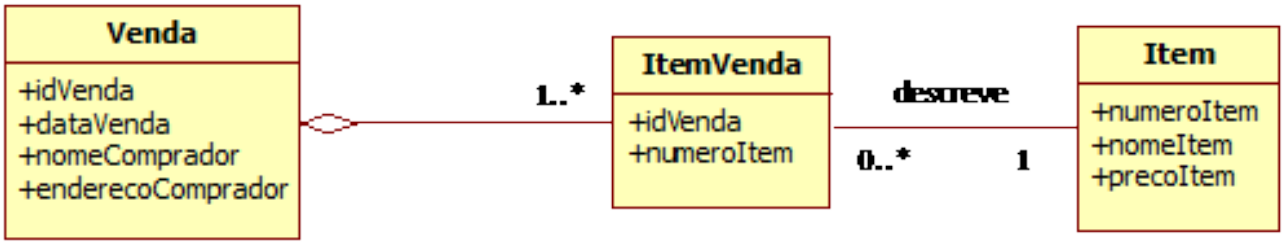

Figura 7. Um esquema normalizado em 3NF (notação UML)

Diversificar para melhorar desempenho

Esquemas de dados normalizados, quando colocados em produção, normalmente sofrem problemas de desempenho. Isso faz sentido – as regras de normalização focam em reduzir redundância de dados, não em melhorar desempenho do acesso aos dados. Uma parte importante da modelagem de dados é Diversificar porções do esquema de dados para melhorar tempo de acesso aos dados.

Observe que se o projeto inicial e normalizado dos dados atinge o desempenho necessário para a aplicação, nada precisa ser feito. A Diversificação deve ser aplicada apenas quando os testes de desempenho mostram que temos um problema com os objetos, revelando que precisamos melhorar o tempo de acesso aos dados.

Referência: https://www.devmedia.com.br/modelagem-de-dados-tutorial/20398

Autor: Por Arilo Em 2011