Falso Olho Provisório
(para ser substituído)

PROJETO E MODELAGEM DE BANCO DE DADOS

Folha de Rosto Provisória
(para ser substituída)

PROJETO E MODELAGEM DE BANCO DE DADOS

Toby Teorey
Sam Lighstone
Tom Nadeau



Favor conferir

Do original
Database modeling and design — 4th edition
Tradução autorizada do idioma inglês da edição publicada por Morgan Kaufmann Publishers — Elsevier Inc.

Copidesque

Editoração Eletrônica
DTPhoenix Editorial

Revisão Gráfica
Marco Antonio Correa

Projeto Gráfico
Editora Campus/Elsevier
A Qualidade da Informação
Rua Sete de Setembro, 111 — 16º andar
20050-006 — Rio de Janeiro — RJ — Brasil
Telefone: (21) 3970-9300 Fax (21) 2507-1991
E-mail: info@elsevier.com.br
Escritório São Paulo
Rua Quintana, 753 — 8º andar
04569-011 — Brooklin — São Paulo — SP
Telefone: (11) 5105-8555

Faltou o dígito

ISBN 13: 978-85-352-2114-?
ISBN 10: 85-352-2114-X
Edição original: ISBN 13: 978-0-07-299054-6 / ISBN 10: 0-07-299054-6

Nota: Muito zelo e técnica foram empregados na edição desta obra. No entanto, podem ocorrer erros de digitação, impressão ou dúvida conceitual. Em qualquer das hipóteses, solicitamos a comunicação à nossa Central de Atendimento, para que possamos esclarecer ou encaminhar a questão.

Nem a editora nem o autor assumem qualquer responsabilidade por eventuais danos ou perdas a pessoas ou bens, originados do uso desta publicação.

Central de atendimento
tel.: 0800-265340
Rua Sete de Setembro, 111, 16º andar — Centro — Rio de Janeiro
e-mail: *info@elsevier.com.br*
site: *www.campus.com.br*

CIP-Brasil. Catalogação-na-fonte.
Sindicato Nacional dos Editores de Livros, RJ

T29p Teorey, Toby J.
Projeto e modelagem de banco de dados / Toby Teorey, Sam Lightstone, Tom Nadeau; tradução de Daniel Vieira. — Rio de Janeiro: Elsevier, 2007

Tradução de: Database modeling and design (4th ed)
Anexo
Inclui bibliografia
ISBN 85-352-2114-X

1. Banco de dados relacionais. 2. Banco de dados — Projetos. I. Lightstone, Sam. II. Nadeau, Tom. III. Título.

CDD 005.756
CDU 004.652.4

06-2702

Para Matt, Carol e Marilyn
(Toby Teorey)

Para a minha esposa e filhos, Elisheva, Hodaya e Avishai
(Sam Lightstone)

Para Carol, Paula, Mike e Lagi
(Tom Nadeau)

Sumário

Será gerado quando eu finalizar a diagramação.

Prefácio

Os sistemas de banco de dados e a tecnologia de projeto de banco de dados passaram por uma evolução significativa nos últimos anos, à medida que as aplicações comerciais têm sido dominadas pelo modelo de dados relacional e por sistemas de bancos de dados relacionais. O modelo relacional tem permitido que o projetista de banco de dados focalize separadamente o projeto lógico (definindo os relacionamentos de dados e tabelas) e o projeto físico (armazenando e recuperando dados do repositório físico de forma eficiente). Outras novas tecnologias, como data warehousing, OLAP e data mining, além de bancos de dados orientados a objeto, espaciais, temporais e de multimídia, também tiveram um impacto importante no projeto de banco de dados.

Nesta quarta edição, continuamos a nos concentrar nas técnicas para projeto de banco de dados em sistemas de bancos de dados relacionais. Entretanto, devido às grandes e explosivas mudanças nas novas técnicas de projeto físico de banco de dados ocorridas nos últimos anos, reorganizamos os tópicos em dois livros separados:

- *Modelagem e projeto de banco de dados: Projeto lógico* (4a. Edição)
- *Physical Database Design*

O projeto lógico de banco de dados é, em grande parte, domínio dos projetistas de aplicações, que criam a estrutura lógica do banco de dados de acordo com os requisitos de manipulação de dados e consultas estruturadas da aplicação. Neste livro, consideramos a definição das tabelas de banco de

dados para um determinado fornecedor como sendo de domínio do projeto lógico, embora muitos profissionais de banco de dados se refiram a essa etapa como sendo parte do projeto físico.

O projeto físico de banco de dados, no contexto destes dois livros, é realizado por implementadores dos servidores de banco de dados, normalmente administradores de banco de dados (DBAs – Database Administrators), que precisam decidir como estruturar o banco de dados para uma determinada máquina (servidor) e otimizar essa estrutura para o desempenho e a administração do sistema. Em empresas menores, essas comunidades podem, na verdade, compreender as mesmas pessoas, mas em grandes empresas, existe uma maior distinção.

Iniciamos a discussão do projeto lógico de banco de dados com a abordagem entidade-relacionamento (ER) para a especificação de requisitos de dados e modelagem conceitual; então, oferecemos uma visão detalhada de uma outra abordagem dominante de modelagem de dados, a *Unified Modeling Language* (UML). As duas abordagens são usadas em todo o texto para todos os exemplos de modelagem de dados, de modo que o leitor pode selecionar qualquer uma (ou ambas) para ajudar a acompanhar a metodologia do projeto lógico. A discussão sobre os princípios básicos é complementada por exemplos comuns, baseados nas experiências reais e que foram bem testados em sala de aula.

Organização

O ciclo de vida do banco de dados é descrito no Capítulo 1. No Capítulo 2, apresentamos os conceitos mais fundamentais da modelagem de dados e fornecemos um conjunto simples de construtores notacionais (a notação Chen para o modelo ER) para representá-los. O modelo ER tem sido, tradicionalmente, um método popular para conceituar os requisitos de dados dos usuários. O Capítulo 3 apresenta a notação UML para a modelagem de dados. A UML (na realidade, UML-2) tornou-se um método padrão de modelagem de sistemas em grande escala para linguagens orientadas a objeto, tais como C++ e Java, e o componente de modelagem de dados da UML está rapidamente se tornando tão popular quanto o modelo ER. Acreditamos ser importante para o leitor entender as duas notações e o quanto elas têm em comum.

Os Capítulos 4 e 5 mostram como usar os conceitos de modelagem de dados no processo de projeto de banco de dados. O Capítulo 4 é dedicado à aplicação direta da modelagem de dados conceitual no projeto lógico de ban-

co de dados. O Capítulo 5 explica a transformação do modelo conceitual para o modelo relacional, especificamente a sintaxe da *Structured Query Language* (SQL).

O Capítulo 6 é dedicado aos fundamentos da normalização de banco de dados até a quinta forma normal, mostrando a equivalência funcional entre o modelo conceitual (tanto em ER quanto em UML) e o modelo relacional nas formas normais mais elevadas.

O estudo de caso no Capítulo 7 resume as técnicas apresentadas nos Capítulos de 1 a 6, com um novo escopo de problema.

O Capítulo 8 descreve as principais questões do projeto lógico de banco de dados na inteligência de negócios - *data warehousing*, processamento analítico on-line (Online Analytical Processing - OLAP) para sistemas de apoio à decisão e data mining.

O Capítulo 9 discute três das ferramentas de software mais populares atualmente para o projeto lógico: Rational Data Architect, da IBM, AllFusion ERwin Data Modeller, da Computer Associates, e PowerDesigner, da Sybase. São fornecidos alguns exemplos com o objetivo de demonstrar como cada uma dessas ferramentas pode ser usada para tratar dos problemas complexos de modelagem de dados.

O Apêndice contém uma revisão dos componentes básicos de definição e manipulação de dados da linguagem de consulta de banco de dados relacional SQL (SQL-99) para os leitores que não têm familiaridade com linguagens de consulta de banco de dados. Um banco de dados simples, com três tabelas, é utilizado como exemplo para ilustrar a capacidade de consulta da SQL.

O profissional de banco de dados pode usar este livro como um guia para a modelagem de banco de dados e de sua aplicação no projeto de bancos de dados empresariais e de negócio, e também para bancos de dados científicos e de engenharia bem-estruturados. Seja você um usuário iniciante de banco de dados ou um profissional experiente, este livro oferece novas idéias de modelagem de banco de dados e facilita a transição do modelo ER ou UML para o modelo relacional, incluindo a criação de definições de dados da SQL padrão. Assim, independente de você estar utilizando DB2 da IBM, Oracle, SQL Server da Microsoft ou MySQL, as regras de projeto destacadas aqui serão igualmente aplicáveis. Os estudos de casos usados ao longo deste livro são exemplos de bancos de dados reais, que foram projetados usando os princípios enunciados aqui. Este livro também pode ser usado por alunos de graduação avançados ou de pós-graduação iniciantes como um livro-texto complementar sobre introdução ao gerenciamento de banco

de dados, ou para um curso independente sobre modelagem de dados ou projeto de banco de dados.

Convenções tipográficas

Para facilitar a referência, nomes de entidade e classe (Empregado, Departamento e assim por diante) são iniciados com maiúsculas a partir do Capítulo 2. No decorrer do livro, nomes de tabela (**produto, produto_conta**) são impressos em negrito por questão de legibilidade.

Agradecimentos

Gostaríamos de agradecer aos nossos colegas e alunos que contribuíram para a continuidade técnica deste livro: James Bean, Mike Blaha, Deb Bolton, Joe Celko, Jarir Chaar, Nauman Chaudhry, David Chesney, Pat Corey, John DeSue, Yang Dongqing, Ron Fagin, Carol Fan, Jim Fry, Jim Gray, Bill Grosky, Wei Guangping, Wendy Hall, Paul Helman, Nayantara Kalro, John Koenig, Ji-Bih Lee, Marilyn Mantei Tremaine, Bongki Moon, Robert Muller, Wee-Teck Ng, Dan O'Leary, Kunle Olukotun, Dorian Pyle, Dave Roberts, Behrooz Seyed-Abbassi, Dan Skrbina, Rick Snodgrass, Il-Yeol Song, Dick Spencer, Amjad Umar e Susanne Yul. Também queremos agradecer ao Departamento de Engenharia Elétrica e Ciência da Computação (EECS) da Universidade de Michigan por fornecer recursos computacionais para a escrita e revisão. Finalmente, sou grato a Julie por sugerir a cidade de Ludington e por seu apoio inabalável (TJT), e agradeço pela generosidade de minha esposa e filhos, que me deram tempo para trabalhar neste texto (SL).

1 Introdução

A tecnologia de banco de dados tem evoluído rapidamente nas últimas três décadas, desde a ascensão e o conseqüente domínio dos sistemas de bancos de dados relacionais. Embora muitos sistemas de bancos de dados especializados (orientados a objeto, espaciais, multimídia etc.) tenham encontrado substanciais comunidades de usuários nos campos da ciência e engenharia, os sistemas relacionais continuam sendo a tecnologia de banco de dados dominante nas empresas comerciais.

O projeto de banco de dados relacional deixou de ser uma arte para se tornar uma ciência parcialmente implementável por um conjunto de recursos de projeto de software. Muitos desses recursos de projeto surgiram como componentes de banco de dados das ferramentas de engenharia de software auxiliada por computador (*Computer-Aided Software Engineering* — CASE), e muitos deles oferecem capacidade de modelagem interativa usando uma abordagem simplificada de modelagem de dados. O projeto lógico — ou seja, a estrutura dos relacionamentos básicos dos dados e a sua definição em um determinado sistema de banco de dados — é, em grande parte, de domínio dos projetistas de aplicações. Esses projetistas podem trabalhar eficientemente com ferramentas tais como ERwin Data Modeler ou Rational Rose com UML, assim como com abordagens puramente manuais. O projeto físico — ou seja, a criação de mecanismos eficientes de armazenamento e recuperação de dados em uma determinada plataforma de computação — é, normalmente, de domínio do administrador de banco de dados (*Database Administrator* — DBA). Os DBAs possuem, atualmente, diversas ferramentas fornecidas por terceiros para ajudá-los no projeto de bancos de dados mais eficientes. Este

livro é dedicado às metodologias e ferramentas de projeto lógico de bancos de dados relacionais mais populares. As metodologias e ferramentas do projeto físico são abordadas em um outro livro.

Neste capítulo, revisamos os conceitos básicos do gerenciamento de banco de dados e apresentamos o papel da modelagem de dados e do projeto de banco de dados no ciclo de vida do banco de dados.

1.1 Dados e gerenciamento de banco de dados

O componente básico de um arquivo em um sistema de arquivos é o *item de dados*, que é a menor unidade de dados identificável que tem significado no mundo real — por exemplo, sobrenome, nome, nome da rua, número de ID ou partido político. Um grupo de itens de dados relacionados, tratados como uma entidade isolada por uma aplicação, é chamado de *registro*. Alguns exemplos de tipos de registros são: pedido, vendedor, cliente, produto e departamento. Um *arquivo* é uma coleção de registros de um mesmo tipo. Os sistemas de banco de dados baseiam-se em arquivos e expandem suas definições: em um banco de dados relacional, um item de dados é chamado de *coluna* ou *atributo*; um registro é chamado de *linha* ou *tupla*; e um arquivo é chamado de *tabela*.

Um *banco de dados* é um objeto mais complexo; é uma coleção de dados armazenados e inter-relacionados, que atende às necessidades de vários usuários dentro de uma ou mais organizações, ou seja, coleções inter-relacionadas de muitos tipos diferentes de tabelas. As motivações para usar bancos de dados em vez de arquivos incluem a maior disponibilidade a um conjunto diversificado de usuários, a integração de dados para facilitar o acesso e a atualização de transações complexas, e a menor redundância de dados.

Um *sistema gerenciador de banco de dados* (SGBD) é um sistema de software genérico para manipular bancos de dados. Um SGBD admite uma visão lógica (esquema, subesquema); visão física (métodos de acesso, clustering de dados); linguagem de definição de dados; linguagem de manipulação de dados; e utilitários importantes, como gerenciamento de transação e controle de concorrência, integridade de dados, recuperação de falhas e segurança. Os SGBD relacionais, o tipo dominante de sistema nos bancos de dados que apóiam negócios bem definidos, também fornecem maior grau de independência de dados que os SGBDs hierárquicos e de rede (CODASYL), mais antigos. A *independência de dados* é a capacidade de fazer mudanças na estrutura lógica ou física do banco de dados sem exigir reprogramação dos programas de

aplicação. Ela também facilita bastante a conversão e a reorganização do banco de dados. Os SGBDs relacionais fornecem um grau de independência de dados muito mais elevado que os sistemas anteriores; e são o foco da nossa discussão sobre modelagem de dados.

1.2 O ciclo de vida do banco de dados

O ciclo de vida do banco de dados incorpora os passos básicos envolvidos no projeto de um esquema global do banco de dados lógico, a alocação dos dados por uma rede de computadores e a definição de esquemas locais específicos do SGBD. Quando o projeto termina, o ciclo de vida continua com a implementação e a manutenção do banco de dados. Este capítulo apresenta uma visão geral do ciclo de vida do banco de dados, como mostra a Figura 1.1. Nos capítulos seguintes, o foco estará no processo de projetar bancos de dados, desde a modelagem de requisitos até o projeto lógico (etapas I e II, a seguir). O resultado de cada etapa do ciclo de vida é ilustrado por uma série de diagramas na Figura 1.2. Cada diagrama mostra um formato possível da saída de cada etapa, de modo que o leitor possa ver a progressão do processo de projeto a partir de uma idéia até a implementação real do banco de dados. Esses formatos são discutidos com muito mais detalhes nos Capítulos de 2 a 6.

I. ***Análise de requisitos.*** Os requisitos do banco de dados são determinados entrevistando-se os produtores e os usuários dos dados; as informações são usadas para produzir uma especificação formal de requisitos. Essa especificação inclui os dados exigidos para o processamento, os relacionamentos de dados naturais e a plataforma de software para a implementação do banco de dados. Como exemplo, a Figura 1.2 (etapa I) mostra os conceitos de produtos, clientes, vendedores e pedidos sendo formulados na mente do usuário final durante o processo de entrevista.
II. ***Projeto lógico.*** O *esquema global*, um diagrama de modelo de dados conceitual que mostra todos os dados e seus relacionamentos, é desenvolvido usando técnicas como ER ou UML. As construções do modelo de dados por fim precisam ser transformadas em relações normalizadas (globais), ou tabelas. A metodologia de desenvolvimento do esquema global é a mesma para um banco de dados distribuído ou centralizado.
    a. *Modelagem de dados conceitual.* Os requisitos de dados são analisados e modelados por meio de um diagrama ER ou UML que inclui, por exemplo, a semântica dos relacionamentos opcionais, relacionamentos ter-

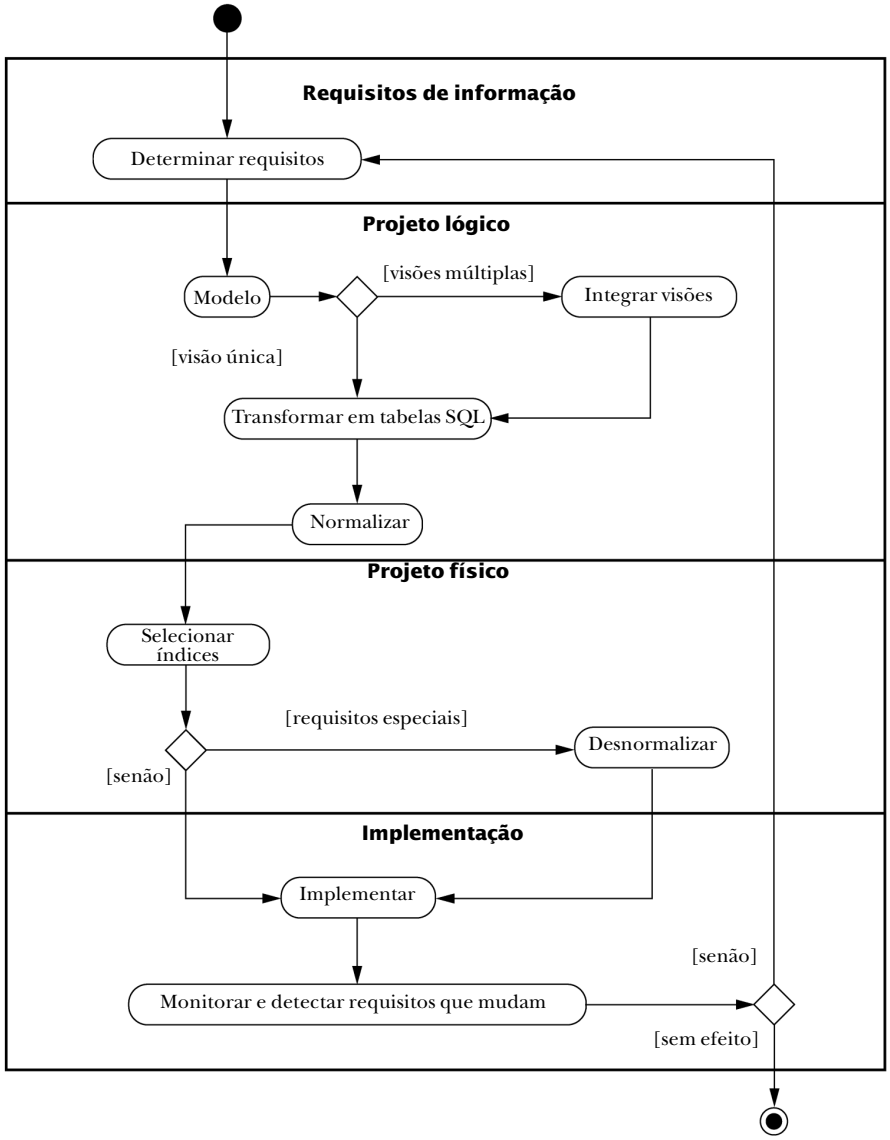


Figura 1.1 O ciclo de vida do banco de dados.

nários, supertipos e subtipos (categorias). Os requisitos de processamento são normalmente especificados usando-se expressões da linguagem natural ou comandos SQL, junto com a freqüência de ocorrência. A Figura 1.2 [etapa II(a)] mostra uma possível representação do modelo ER do banco de dados de produto/cliente na mente do usuário final.

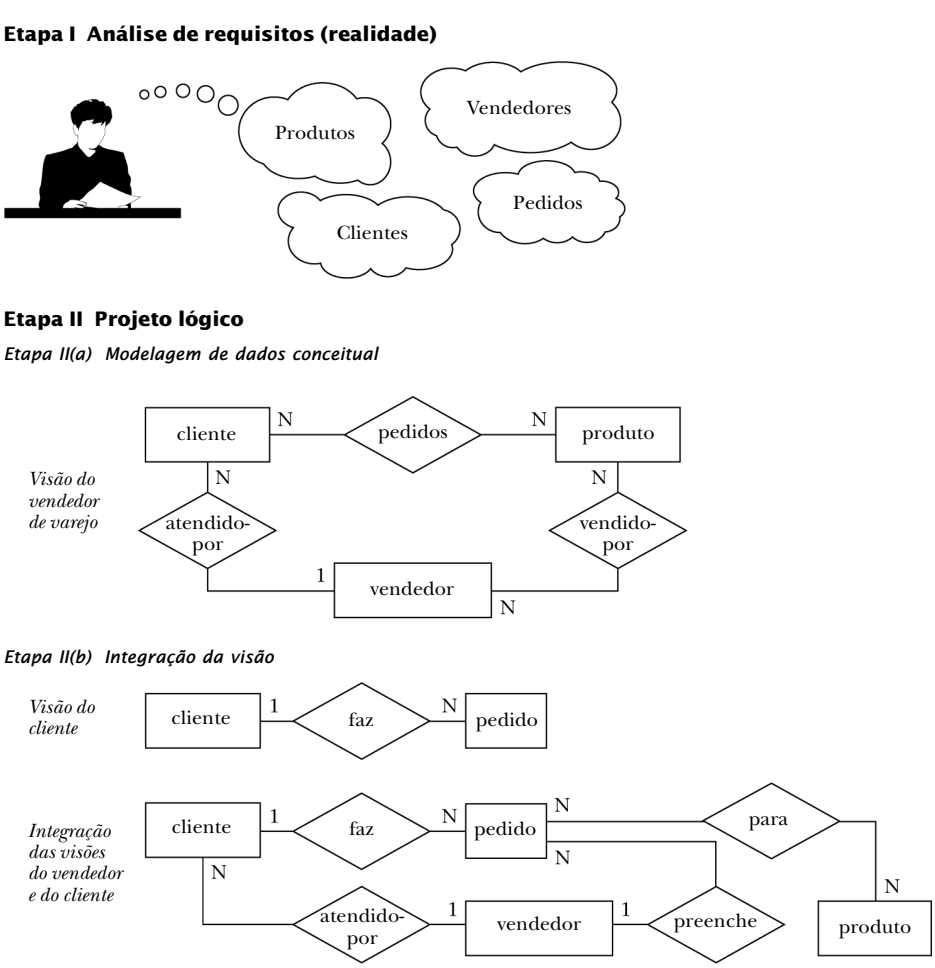


Figura 1.2 Resultados do ciclo de vida, etapa por etapa.

b. *Integração da visão.* Normalmente, quando o projeto é grande e há mais de uma pessoa envolvida na análise de requisitos, ocorrem várias visões dos dados e relacionamentos. Para eliminar redundâncias e inconsistências do modelo, essas visões por fim precisam ser "racionalizadas" (resolvendo inconsistências ocasionadas pela variação na taxonomia, contexto ou percepção) e depois consolidadas em uma única visão global. A integração da visão exige o uso de ferramentas semânticas da ER, como a identificação de sinônimos, agregação e generalização. Na Figura 1.2 [etapa II(b)], duas visões possíveis do banco de

dados de produto/cliente são mescladas em uma única visão global, com base em dados comuns de cliente e pedido. A integração da visão também é importante para a integração da aplicação.

c. *Transformação do modelo de dados conceitual em tabelas SQL.* Com base em uma categorização das construções de modelagem de dados e um conjunto de regras de mapeamento, cada relacionamento e suas entidades associadas são transformados em um conjunto de tabelas relacionais candidatas específicas do SGBD. Mostraremos essas transformações na SQL padrão no Capítulo 5. Tabelas redundantes são eliminadas como parte desse processo. Em nosso exemplo, as tabelas na etapa II(c) da Figura 1.2 são o resultado da transformação do modelo ER integrado na etapa II(b).

d. *Normalização de tabelas.* Dependências funcionais (DFs) são derivadas do diagrama do modelo de dados conceitual e da semântica dos relacionamentos de dados na análise de requisitos. Elas representam as dependências entre os elementos de dados que são identificadores exclusivos (chaves) das entidades. DFs adicionais, que representam as dependências entre atributos chaves e não-chaves dentro das entidades, podem ser derivadas a partir da especificação de requisitos. Tabelas relacionais candidatas, associadas a todas as DFs derivadas, são normalizadas (ou seja, modificadas pela decomposição ou divisão de tabelas em tabelas menores) por meio de técnicas padronizadas. Finalmente, as redundâncias de dados nas tabelas candidatas normalizadas são também analisadas em busca de possíveis eliminações, com a restrição de que a integridade dos dados precisa ser preservada. Um exemplo de normalização da tabela **Vendedor,** nas novas tabelas **Vendedor** e **Vendedor-Férias,** aparece na Figura 1.2, da etapa II(c) para a etapa II(d).

   Observamos aqui que os fornecedores de ferramentas de bancos de dados costumam usar o termo *modelo lógico* para se referir ao modelo de dados conceitual e utilizam o termo *modelo físico* para se referir ao modelo de implementação específico do SGBD (por exemplo, tabelas SQL). Observe também que muitos modelos de dados conceituais são obtidos não do zero, mas do processo de *engenharia reversa*, a partir de um esquema específico do SGBD [Silberschatz, Korth e Sudarshan, 2002].

III. ***Projeto físico.*** A etapa de projeto físico envolve a seleção de índices (métodos de acesso), particionamento e clustering de dados. A metodologia de projeto lógico na etapa II simplifica a técnica de projeto de grandes ban-

Etapa II(c) Transformação do modelo de dados conceitual em tabelas SQL

Cliente

num-cli	nome-cli	. . .

salto
o.i. pág. 7

Produto

num-produto	nome-prod	qtd-estoque

Vendedor

nome-vendedor	endereço	dept	nível-cargo	dias-férias

Pedido

num-pedido	nome-vendedor	num-cli

Pedido-produto

num-pedido	num-produto

Etapa II(d) Normalização de tabelas SQL

Decomposição de tabelas e remoção de anomalias de atualização

Vendedor

nome-vendedor	endereço	dept	nível-cargo

Vendedor-férias

nível-cargo	dias-férias

Etapa III Projeto físico

Indexação
Clustering
Particionamento
Views materializadas
Desnormalização

Figura 1.2 *(continuação)*

cos de dados relacionais, reduzindo o número de dependências de dados que precisam ser analisadas. Isso é alcançado pela inserção da modelagem de dados conceitual e pela integração das etapas [etapas II(a) e II(b) da Figura 1.2] à técnica tradicional de projeto relacional. O objetivo dessas etapas é uma representação precisa da realidade. A integridade de dados é preservada por meio da normalização das tabelas candidatas criadas quando o modelo de dados conceitual é transformado em um modelo relacional. A finalidade do projeto físico é otimizar o desempenho da melhor forma possível.

Como parte do projeto físico, o esquema global às vezes pode ser refinado, de maneira bastante limitada, para refletir os requisitos de processamento (consulta e transação) se forem obtidos ganhos óbvios e significativos na eficiência. Isso é chamado de *desnormalização*. Ela consiste em selecionar processos dominantes com base na alta freqüência, alto volume ou prioridade explícita; definir extensões simples às tabelas, o que melhorará o desempenho da consulta; avaliar o custo total para a consulta, atualização e armazenamento; e considerar os efeitos colaterais, como possível perda de integridade. Isso é particularmente importante para aplicações OLAP (*Online Analytical Processing*).

IV. ***Implementação, monitoração e modificação de bancos de dados.*** Quando o projeto é finalizado, o banco de dados pode ser criado por meio da implementação do esquema formal, usando a linguagem de definição de dados (LDD) de um SGBD. Depois, a linguagem de manipulação de dados (LMD) pode ser usada para consultar e atualizar o banco de dados, além de configurar índices e estabelecer restrições, como a integridade referencial. A linguagem SQL contém construtores da LDD e LMD; por exemplo, o comando *create table* representa um construtor da LDD, e o comando *select* representa um construtor da LMD.

À medida que o banco de dados inicia a operação, a monitoração indica se os requisitos de desempenho estão sendo atendidos. Se não estiverem sendo satisfeitos, devem ser feitas modificações para melhorar o desempenho. Outras modificações podem ser necessárias quando os requisitos mudam ou quando as expectativas dos usuários finais aumentam devido ao bom desempenho. Assim, o ciclo de vida continua com a monitoração, reprojeto e modificações. Nos dois capítulos seguintes, examinamos primeiro os conceitos básicos de modelagem de dados e depois — começando no Capítulo 4 - aplicamos esses conceitos ao processo de projeto de banco de dados.

1.3 Modelagem de dados conceitual

A modelagem de dados conceitual é o principal componente do projeto lógico do banco de dados. Vejamos como esse componente ocorre e por que é importante. Os diagramas de esquema foram formalizados na década de 1960 por Charles Bachman. Ele usou retângulos para indicar os tipos de registros e setas direcionadas de um tipo de registro para outro a fim de indicar um relacionamento um-para-muitos entre as instâncias de registros desses dois tipos. A técnica *entidade-relacionamento* (ER) para a modelagem de dados conceitual, uma das duas técnicas enfatizadas neste livro e descrita com detalhes no Capítulo 2, foi apresentada inicialmente em 1976 por Peter Chen. O formato Chen do modelo ER emprega retângulos para especificar entidades, que de certa forma são semelhantes a registros. Ele também usa objetos em forma de losango para representar os vários tipos de relacionamentos, que são diferenciados por números ou letras colocadas nas linhas que conectam os losangos aos retângulos.

A Unified Modeling Language (UML) foi introduzida em 1997 por Grady Booch e James Rumbaugh e tornou-se uma linguagem gráfica padrão para especificar e documentar sistemas de software em grande escala. O componente de modelagem de dados da UML (agora UML-2) tem muita semelhança com o modelo ER e é apresentado com detalhes no Capítulo 3. Usamos o modelo ER e a UML para ilustrar a modelagem de dados e exemplos de projeto lógico de banco de dados no decorrer deste livro.

Na modelagem de dados conceitual, a ênfase dominante está na simplicidade e na legibilidade. O objetivo do projeto do esquema conceitual, em que as abordagens ER e UML são mais úteis, é capturar os requisitos de dados do mundo real de uma maneira simples e significativa, que seja inteligível pelo projetista de banco de dados e pelo usuário final. O usuário final é a pessoa responsável por acessar o banco de dados e executar consultas e atualizações por meio do software do SGBD e, portanto, possui um interesse real no processo de projeto de banco de dados.

O modelo ER possui dois níveis de definição — um que é muito simples e outro que é consideravelmente mais complexo. O nível simples é aquele usado pela maioria das ferramentas de projeto atuais. Ele é muito útil ao projetista de banco de dados que precisa se comunicar com os usuários finais a respeito de seus requisitos de dados. Nesse nível, você simplesmente descreve, em forma de diagrama, as entidades, os atributos e os relacionamentos que ocorrem no sistema a ser conceituado, usando uma semântica definida em

um dicionário de dados. Construções especializadas, como entidades "fracas" ou notação de existência obrigatória/opcional, são também normalmente incluídas na forma simples. Contudo, muito pouca coisa a mais é incluída, para evitar encher o diagrama ER enquanto o conhecimento do modelo pelo projetista e pelo usuário final está sendo conciliado.

Um exemplo de uma forma simples do modelo ER usando a notação de Chen aparece na Figura 1.3. Nesse exemplo, queremos registrar as fitas de vídeo e os clientes em uma videolocadora. Os vídeos e os clientes são representados como entidades Vídeo e Cliente, e o relacionamento "aluga" mostra uma associação muitos-para-muitos entre eles. As entidades Vídeo e Cliente possuem alguns atributos que descrevem suas características, e o relacionamento "aluga" possui um atributo data-venc, que representa a data em que uma determinada fita de vídeo, alugada por um cliente específico, precisa ser devolvida.

Do ponto de vista do profissional de banco de dados, a forma simples do modelo ER (ou UML) é a forma preferida para a modelagem de dados e verificação final do usuário. Ela é fácil de entender e aplicável a uma grande variedade de problemas de projeto que podem ser encontrados na indústria e em pequenas empresas. Conforme demonstraremos, a forma simples pode ser facilmente traduzida para definições de dados SQL e assim possui uso imediato como um auxílio para a implementação do banco de dados.

O nível complexo da definição do modelo ER inclui conceitos que vão bem além do modelo simples. Ele inclui conceitos dos modelos semânticos da inteligência artificial e dos modelos de dados conceituais concorrentes. A modelagem de dados nesse nível ajuda o projetista de banco de dados a capturar mais semântica sem ter de lançar mão de explicações narrativas. Ela também é útil ao programador de aplicação de banco de dados, pois certas

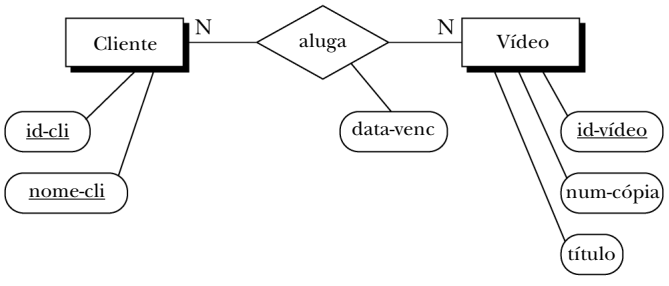


Figura 1.3 Uma forma simples do modelo ER usando a notação Chen.

restrições de integridade definidas no modelo ER se relacionam diretamente ao código — código que verifica os limites de intervalo sobre valores de dados e valores nulos, por exemplo. No entanto, o nível simples é recomendado como a ferramenta de comunicação básica para a verificação do projeto do banco de dados.

1.4 Resumo

O conhecimento das técnicas de modelagem de dados e projeto de banco de dados é importante para os profissionais de banco de dados e desenvolvedores de aplicações. O ciclo de vida do banco de dados mostra as etapas necessárias em uma abordagem metódica para o projeto de um banco de dados, desde o projeto lógico, que é independente do ambiente do sistema, até o projeto físico, que é baseado nos detalhes do sistema de gerenciamento de banco de dados escolhidos para implementar o banco de dados. Entre as várias abordagens de modelagem de dados, os modelos de dados ER e UML são comprovadamente os mais populares e em uso hoje em dia, devido à sua simplicidade e legibilidade. Uma forma simples desses modelos é utilizada na maioria das ferramentas de projeto; ela é fácil de aprender e se aplica a uma série de aplicações industriais e comerciais. Ela também é uma ferramenta muito útil para a comunicação com o usuário final sobre o modelo conceitual e para a verificação das suposições feitas no processo de modelagem. Uma forma mais complexa, um superconjunto da forma simples, é útil para o projetista mais experiente, que deseja capturar mais detalhes semânticos na forma de diagrama, enquanto evita descrições narrativas longas e tediosas para explicar certos requisitos e restrições.

1.5 Resumo da literatura

Grande parte do trabalho inicial de modelagem de dados foi feito por Bachman [1969, 1972], Chen [1976], Senko *et al.* [1973], além de outros. Os livros-texto de projeto de banco de dados que aderem a uma parte significativa do ciclo de vida do banco de dados relacional, descrito neste capítulo, são Teorey e Fry [1982], Muller [1999], Stephens e Plew [2000], Simsion e Witt [2001] e Hernandez e Getz [2003]. Bancos de dados temporais (variáveis no tempo) são definidos e discutidos em Jensen e Snodgrass [1996] e Snodgrass [2000]. Outras técnicas bastante usadas para a modelagem de dados conceitual incluem IDEFIX [Bruce, 1992; IDEFIX, 2005] e o componente de modela-

gem de dados do Zachmann Framework [Zachmann, 1987; Zachmann Institute for Framework Advancement, 2005]. A evolução do esquema durante o desenvolvimento, um problema que ocorre com freqüência, é explicado em Harriman, Hodgetts e Leo [2004].

2 O modelo entidade-relacionamento

Este capítulo define os principais conceitos de entidade-relacionamento (ER) aplicáveis à fase de modelagem de dados conceitual do ciclo de vida do banco de dados. Na Seção 2.1, examinamos o nível simples da modelagem ER, descrito no trabalho original de Chen e estendido por outros. A forma simples do modelo ER é usada como base para a comunicação eficaz com o usuário final sobre o banco de dados conceitual. A Seção 2.2, mais adiante neste capítulo, apresenta os conceitos mais avançados; embora geralmente seja de menor aceitação, eles são úteis para descrever determinadas semânticas que não podem ser construídas com o modelo simples.

2.1 Construtores fundamentais da ER

2.1.1 Objetos básicos: entidades, relacionamentos, atributos

O modelo ER básico consiste em três classes de objetos: entidades, relacionamentos e atributos.

*Entidades**

Entidades são os principais objetos de dados sobre os quais informações devem ser coletadas; elas normalmente representam uma pessoa, lugar, coisa

* *Nota dos Revisores Técnicos*: Alguns termos utilizados pelos autores neste texto divergem da nomenclatura utilizada por alguns outros autores na área de banco de dados, tais como Elmasri & Navathe no livro: *Fundamental of database systems,* 4th. Edition, Addison-Wesley, 2003. (Tradução: *Sistemas de Banco*

ou evento de interesse informativo. Uma ocorrência específica de uma entidade é chamada de *instância da entidade*, ou, às vezes, *ocorrência da entidade*. Em nosso exemplo, funcionário, departamento, divisão, projeto, habilidade e local são exemplos de entidades. Para facilitar a referência, os nomes de entidade, daqui por diante, serão iniciados com maiúsculas no decorrer deste texto (por exemplo, Funcionário, Departamento etc). O construtor da entidade é um retângulo, conforme representado na Figura 2.1. O nome da entidade é escrito dentro do retângulo.

Relacionamentos

Relacionamentos representam associações do mundo real entre uma ou mais entidades e, dessa forma, não possuem existência física ou conceitual além de

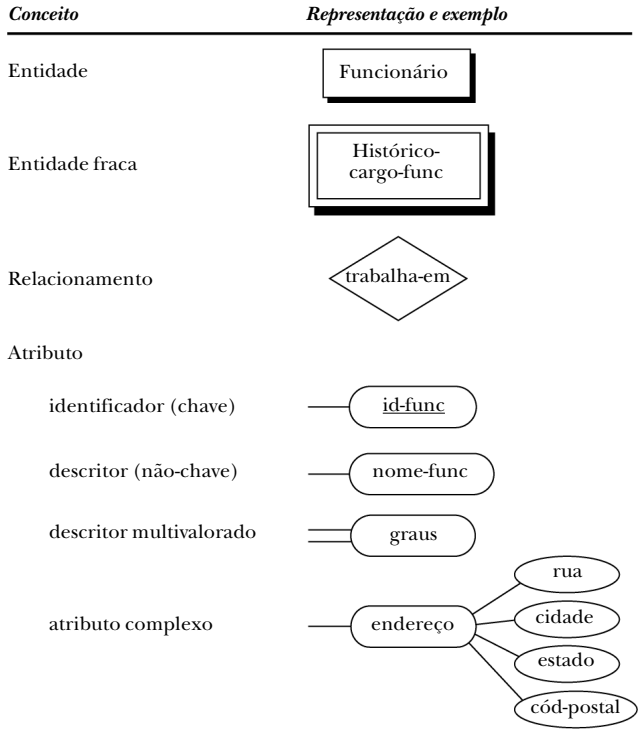


Figura 2.1 O modelo ER básico.

de Dados, Addison –Wesley, 2005). Exemplos dos termos equivalentes utilizados respectivamente pelos autores deste texto e outros são: entidade = tipo de entidade; relacionamento = tipo de relacionamento; ocorrência ou instância de ocorrência de entidade = entidade; ocorrência ou instância de ocorrência de relacionamento = relacionamento.

sua dependência das entidades associadas. Os relacionamentos são descritos em termos de grau, conectividade e existência. Esses termos são definidos nas próximas seções. O significado mais comum associado ao termo *relacionamento* é indicado pela conectividade entre ocorrências de entidade: um-para-um, um-para-muitos e muitos-para-muitos. O construtor de relacionamento é um losango que conecta as entidades associadas, como mostra a Figura 2.1. O nome do relacionamento pode ser escrito dentro ou próximo do losango.

Um *papel* é o nome de uma extremidade de um relacionamento quando cada extremidade precisa de um nome distinto para esclarecer o relacionamento. Na maioria dos exemplos dados na Figura 2.2, os nomes de papel não são obrigatórios, pois os nomes de entidade combinados com o nome do relacionamento definem com clareza os papéis individuais de cada entidade no relacionamento. Contudo, em alguns casos, os nomes de papel devem ser usados para esclarecer as ambigüidades. Por exemplo, no primeiro caso da Figura 2.2, o relacionamento binário recursivo "gerencia" usa dois papéis, "gerente" e "subordinado", para associar as conectividades apropriadas aos dois papéis diferentes da única entidade. Os nomes de papel normalmente são substantivos. Nesse diagrama, o papel de um funcionário é de "gerente" de até *n* outros funcionários. O outro papel de um funcionário é de "subordinado" gerenciado por exatamente um funcionário com papel de "gerente".

Atributos

Atributos são características de entidades que oferecem detalhes descritivos sobre elas. Uma ocorrência em particular de um atributo dentro de uma entidade ou relacionamento é chamada de valor de atributo. Os atributos de uma entidade como o da entidade Funcionário podem incluir id-func, nome-func, endereço-func, num-tel, num-fax, cargo e assim por diante. O construtor de atributo é uma elipse com o nome do atributo em seu interior (ou uma forma de retângulo arredondado, como mostra a Figura 2.1). O atributo é conectado à entidade que ele caracteriza.

Existem dois tipos de atributos: identificadores e descritores. Um identificador (ou chave) é usado para determinar exclusivamente uma instância de uma entidade; um descritor (ou atributo não-chave) é usado para especificar uma característica não-exclusiva de determinada instância da entidade. Identificadores e descritores podem ser simples ou compostos. Por exemplo, um identificador ou chave de Funcionário é id-func, e um descritor de Funcionário é nome-func ou cargo. Os atributos chaves são sublinhados no diagrama ER, como mostra a Figura 2.1. Observamos, rapidamente, que uma enti-

dade pode ter mais de um identificador (chave), ou pode ter um conjunto de atributos que compõem uma chave (ver Seção 6.1.2).

Alguns atributos, como área-especialidade, podem ser multivalorados. A notação para atributos multivalorados é uma linha de conexão dupla, como mostra a Figura 2.1. Outros atributos podem ser complexos, como um endereço que se subdivide em rua, cidade, estado e código postal. Atributos complexos são construtores que possuem atributos próprios; porém, às vezes, no início da modelagem, as partes individuais de um atributo complexo podem ser especificadas como atributos individuais. Qualquer forma é razoável na notação ER.

Entidades possuem identificadores internos que determinam com exclusividade a existência de instâncias de entidade, mas entidades fracas derivam sua identidade dos atributos de identificação de uma ou mais entidades "pai". Entidades fracas normalmente são representadas com um retângulo de borda dupla (ver Figura 2.1), que indica que todas as ocorrências dessa entidade dependem de uma entidade (forte) associada para a sua existência no banco de dados. Por exemplo, na Figura 2.1, a entidade fraca Histórico-cargo-func está relacionada à entidade Funcionário e depende de Funcionário para a sua própria existência.

2.1.2 Grau de um relacionamento

O grau de um relacionamento é o número de entidades associadas ao relacionamento. Relacionamentos binários e ternários são casos especiais em que os graus são 2 e 3, respectivamente. Um relacionamento *n*-ário é a forma geral de qualquer grau *n*. A notação para grau é ilustrada na Figura 2.2. O relacionamento binário, uma associação entre duas entidades, é o tipo mais comum no mundo natural. De fato, muitos sistemas de modelagem usam apenas esse tipo. Na Figura 2.2, vemos vários exemplos diferentes de associação entre duas entidades: Departamento e Divisão, Departamento e Funcionário, Funcionário e Projeto, e assim por diante. Um relacionamento binário recursivo (por exemplo, "gerencia", da Figura 2.2) relaciona um particular Funcionário a outro Funcionário através de gerencia. Ele é chamado recursivo porque uma instância de entidade se relaciona apenas com outras instâncias de seu próprio tipo. O construtor do relacionamento binário recursivo é um losango com as duas conexões com a mesma entidade.

Um relacionamento ternário é uma associação que envolve três entidades. Esse tipo de relacionamento é necessário quando os relacionamentos

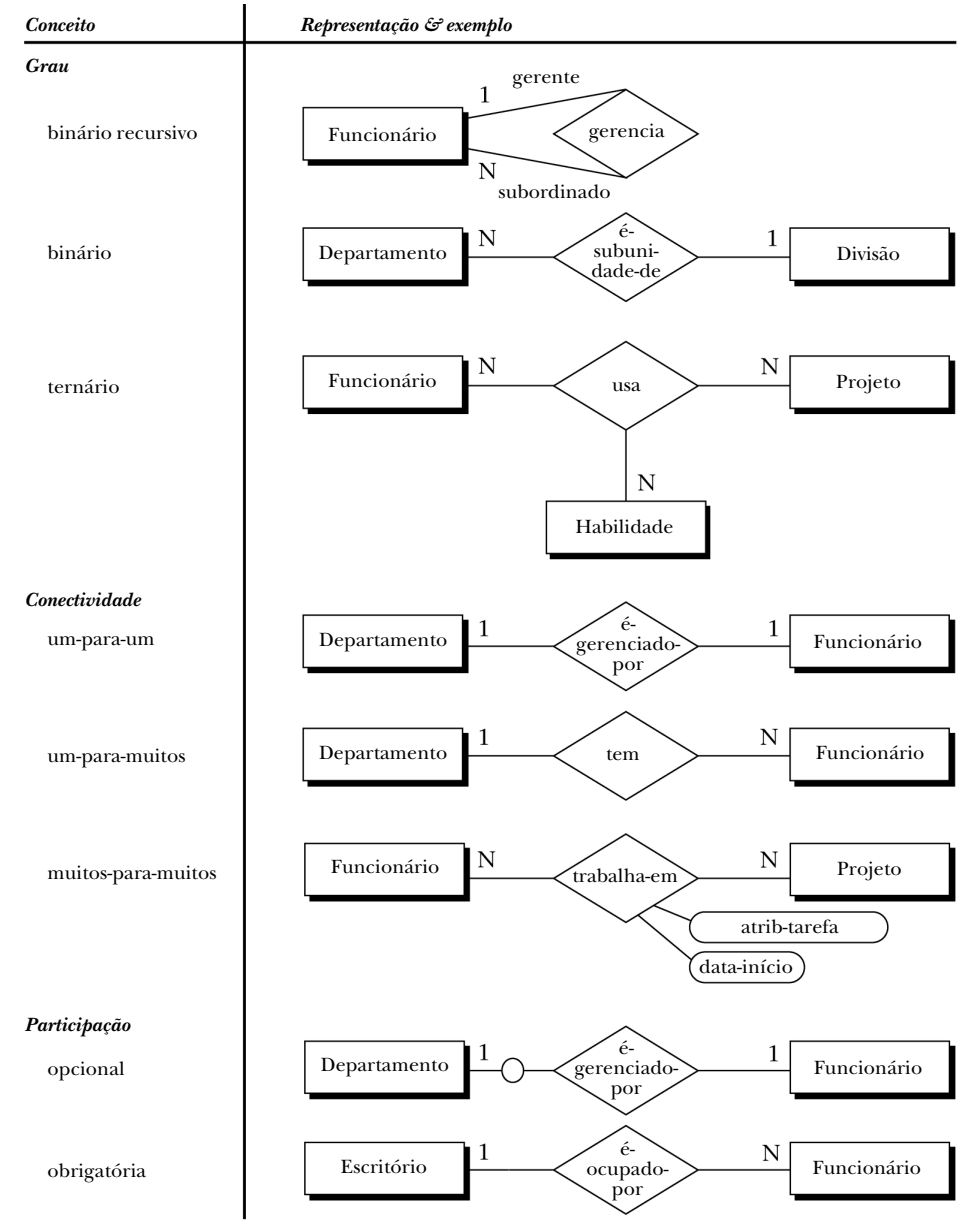


Figura 2.2 Graus, conectividade e atributos de um relacionamento.

binários não são suficientes para descrever com precisão a semântica da associação. O construtor do relacionamento ternário é um único losango que conecta três entidades, como mostra a Figura 2.2. Às vezes, um relacionamen-

to é por engano modelado como ternário, quando poderia ser decomposto em dois ou três relacionamentos binários equivalentes. Quando isso ocorre, o relacionamento ternário deve ser eliminado para alcançar a simplicidade e a pureza semântica. Os relacionamentos ternários são discutidos com mais detalhes na Seção 2.2.3 e no Capítulo 6.

Uma entidade pode estar envolvida em qualquer quantidade de relacionamentos, e cada relacionamento pode ser de qualquer grau. Além do mais, duas entidades podem ter qualquer quantidade de relacionamentos binários entre elas, e isso vale para qualquer *n* entidades (ver relacionamentos *n*-ários definidos na Seção 2.2.4).

2.1.3 Conectividade de um relacionamento

A *conectividade* de um relacionamento descreve uma restrição sobre a conexão das ocorrências de entidade associadas no relacionamento. Os valores para conectividade são "um" ou "muitos". No relacionamento entre as entidades Departamento e Funcionário, uma conectividade de um para Departamento e muitos para Funcionário significa que existe no máximo uma ocorrência da entidade Departamento associada a ocorrências de Funcionário. A contagem real de elementos associados à conectividade é chamada de *cardinalidade* da conectividade do relacionamento; ela é usada com muito menos freqüência do que a restrição de conectividade, pois os valores reais normalmente variam entre as instâncias dos relacionamentos. Observe que não existem termos padrão para o conceito de conectividade, de modo que advertimos o leitor a considerar a definição desses termos com cuidado quando usar determinada metodologia de projeto de banco de dados.

A Figura 2.2 mostra os construtores básicos de conectividade nos relacionamentos binários: um-para-um, um-para-muitos e muitos-para-muitos. No lado "um", o número um aparece na conexão entre o relacionamento e uma das entidades, e no lado "muitos", a letra *N* é usada na conexão entre o relacionamento e a entidade para designar o conceito de muitos.

No caso um-para-um, a entidade Departamento é gerenciada por exatamente um Funcionário, e cada Funcionário gerencia exatamente um Departamento. Portanto, as conectividades mínima e máxima no relacionamento "é-gerenciado-por" são exatamente um, tanto para Departamento quanto para Funcionário.

No caso um-para-muitos, a entidade Departamento está associada a ("tem") muitos Funcionários. A conectividade máxima é dada no lado Funcionário

(muitos) como o valor desconhecido *N*, mas a conectividade mínima é conhecida como um. No lado Departamento, as conectividades mínima e máxima são ambas um, ou seja, cada Funcionário trabalha dentro de exatamente um Departamento.

No caso muitos-para-muitos, um Funcionário em particular pode trabalhar em muitos Projetos, e cada Projeto pode ter muitos Funcionários. Vemos que a conectividade máxima para Funcionário e Projeto é *N* nas duas direções, e as conectividades mínimas são definidas (implicitamente) como um.

Algumas situações, embora raras, são tais que a conectividade máxima real é conhecida. Por exemplo, um time de basquete profissional pode estar limitado pelas regras da confederação em 12 jogadores. Nesse caso, o número 12 poderia ser colocado próximo de uma entidade chamada "membros do time" no lado muitos de um relacionamento com uma entidade "time". Entretanto, a maioria das situações possui conectividade variável no lado muitos, como aparece em todos os exemplos da Figura 2.2.

2.1.4 Atributos de um relacionamento

Atributos podem ser associados a certos tipos de relacionamentos, além das entidades. Um relacionamento muitos-para-muitos, como o relacionamento "trabalha-em" entre as entidades Funcionário e Projeto (Figura 2.2), poderia ter os atributos "designação-tarefa" e "data-início". Nesse caso, uma determinada designação de tarefa ou data de início só possui significado quando eles forem comuns tanto a uma instância associada a um determinado Funcionário quanto a uma instância de um determinado Projeto por meio do relacionamento "trabalha-em".

Os atributos dos relacionamentos normalmente são atribuídos apenas a relacionamentos binários muitos-para-muitos e a relacionamentos ternários. Eles normalmente não são atribuídos a relacionamentos um-para-um ou um-para-muitos, devido a possíveis ambigüidades. Por exemplo, no relacionamento binário um-para-um "é-gerenciado-por" entre Departamento e Funcionário, um atributo "data-início" poderia ser aplicado a Departamento para designar a data de início para esse departamento. Como alternativa, ele poderia ser aplicado a Funcionário como um atributo para cada instância de Funcionário, a fim de designar a data de início do funcionário como gerente desse departamento. Se, em vez disso, o relacionamento fosse muitos-para-muitos, de modo que um funcionário pudesse gerenciar muitos departamentos com o tempo, então o atributo "data-início" precisaria passar para o relacionamen-

to; de modo que cada instância do relacionamento que combina um funcionário a um departamento possa ter uma data de início exclusiva para esse funcionário como gerente desse departamento.

2.1.5 Participação de uma entidade em um relacionamento

A *participação* de uma ocorrência de entidade em um relacionamento é definida como obrigatória ou opcional. Se uma ocorrência de entidade no lado "um" ou "muitos" sempre tiver de participar do relacionamento para que ela exista, então a participação é obrigatória. Quando uma ocorrência dessa entidade nem sempre precisar participar do relacionamento, ela é considerada opcional. Por exemplo, na Figura 2.2, a entidade Funcionário pode ou não ser o gerente de algum Departamento, tornando opcional a participação de Funcionário no relacionamento "é-gerenciada-por" entre Funcionário e Departamento.

A participação opcional, definida por um zero na linha de conexão entre uma entidade e um relacionamento, define uma conectividade mínima de zero. A *participação obrigatória* define uma conectividade mínima de um. Quando a participação é desconhecida, consideramos que a conectividade mínima é um (ou seja, obrigatória).

Conectividades máximas são definidas explicitamente no diagrama ER como uma constante (se um número for mostrado no diagrama ER próximo de uma entidade) ou uma variável (como é o padrão, quando nenhum número aparece no diagrama ER próximo de uma entidade). Por exemplo, na Figura 2.2, o relacionamento "é-ocupado-por" entre a entidade Escritório e Funcionário implica que um Escritório pode conter de zero a algum número máximo variável (*N*) de Funcionários, mas um Funcionário precisa estar alocado a exatamente um Escritório, ou seja, obrigatório.

A participação normalmente é implícita no mundo real. Por exemplo, a participação de uma entidade Funcionário associada a uma entidade (fraca), Dependente, normalmente é opcional, mas a participação da entidade fraca é obrigatória. Usando o conceito de participação opcional, uma instância da entidade pode ser capaz de participar de vários relacionamentos, embora ela não participe de qualquer relacionamento em particular.

O termo participação também está associado à identificação de um objeto de dados. Muitos SGBDs fornecem identificadores exclusivos para linhas (ROWIDS do Oracle, por exemplo). A identificação de um objeto como uma linha pode ser usada para definir a participação desse objeto em um relacio-

namento. A participação também pode ser definida com base nos valores de atributos, identificando o objeto (linha) com os valores de um ou mais atributos ou colunas da tabela.

2.1.6 Notações alternativas de modelagem de dados conceitual

Neste ponto, faremos um rápido desvio para analisar outras notações de modelagem de dados conceitual que normalmente são usadas hoje, comparando-as com a abordagem de Chen. Uma forma popular alternativa para relacionamentos um-para-muitos e muitos-para-muitos utiliza a notação "pé de galinha" para o lado "muitos" (ver Figura 2.3a). Essa forma é usada por algumas ferramentas CASE, como a *Information Engineering Workbench* (IEW) da *Knowledgeware*. Os relacionamentos não possuem construtores explícitos, mas são indicados pela linha de conexão entre as entidades e um nome de relacionamento na linha de conexão. A conectividade mínima é especificada por um 0 (para zero) ou por uma linha perpendicular (para um) nas linhas de conexão entre as unidades. O termo *entidade de interseção* é usado para designar uma entidade fraca e, em especial, uma entidade que é equivalente a um relacionamento muitos-para-muitos. Outra forma popular usada hoje é a notação IDEFIX [IDEFIX, 2005], concebida por Robert G. Brown [Bruce, 1992]. As semelhanças com a notação de Chen ficam claras na Figura 2.3b. Felizmente, qualquer uma dessas formas é razoavelmente fácil de aprender e ler, e sua equivalência com os conceitos básicos de ER é óbvia a partir dos diagramas. No entanto, sem um padrão claro para o modelo ER, muitas outras construções estão sendo usadas hoje além dos três tipos mostrados aqui.

2.2 Construtores avançados da ER

2.2.1 Generalização: supertipos e subtipos

Já, há algum tempo, o modelo ER original vem sendo usado com eficiência na comunicação e definição dos principais dados e relacionamentos com o usuário final. Contudo, o seu uso no desenvolvimento e na integração de modelos conceituais com diferentes visões do usuário era bastante limitado até a sua extensão com a inclusão de conceitos de abstração de banco de dados, como a *generalização*. O relacionamento de generalização especifica que vários tipos de entidades com certos atributos comuns podem ser generalizados para um tipo de entidade de nível superior — uma entidade genérica

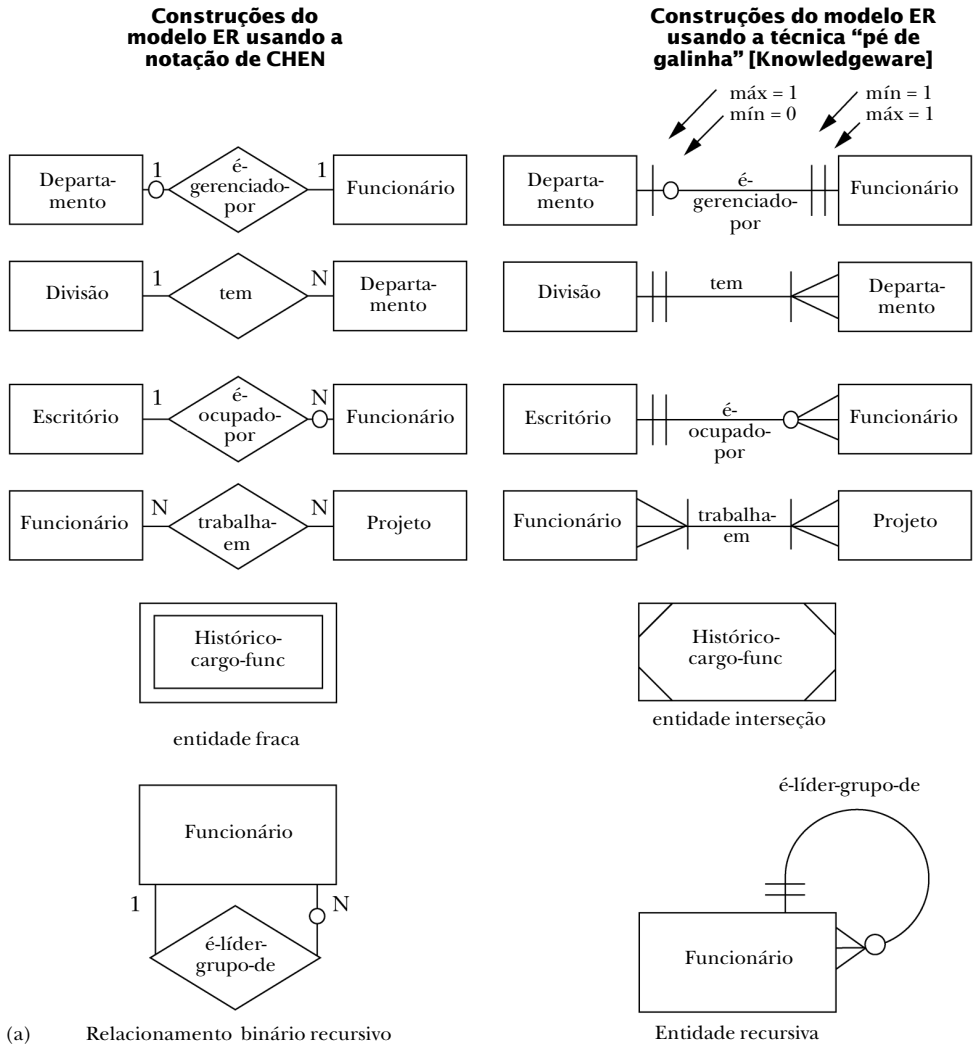


Figura 2.3 Notações da modelagem de dados conceitual.

ou superclasse, normalmente conhecida como entidade *supertipo*. As entidades do nível inferior — os *subtipos* em uma hierarquia de generalização - podem ser subconjuntos disjuntos ou sobrepostos da entidade supertipo. Como exemplo, a Figura 2.4 ilustra que a entidade Funcionário é uma abstração de nível superior de Gerente, Engenheiro, Técnico e Secretário, todos tipos disjuntos de Funcionário. O construtor do modelo ER para a abstração de generalização é a conexão de uma entidade supertipo com seus subtipos, usan-

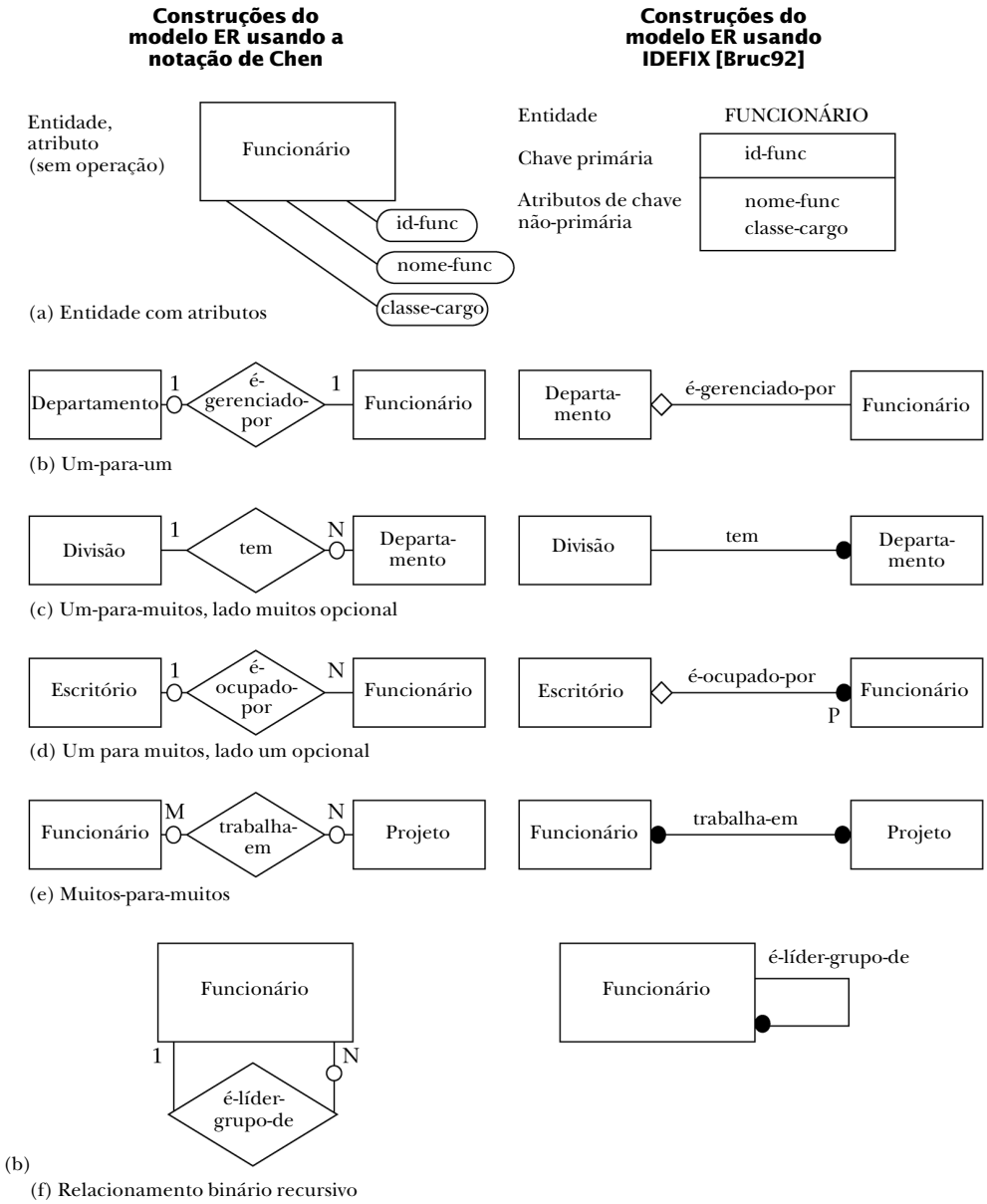


Figura 2.3 *(continuação)*

do um círculo e o símbolo de subconjunto nas linhas de conexão do círculo para as entidades subtipos. O círculo contém uma letra que especifica uma restrição de disjunção (ver a discussão seguinte). A *especialização*, o inverso da generalização, é uma inversão do mesmo conceito; ela indica que os subtipos especializam o supertipo.

Uma entidade supertipo em um relacionamento pode ser uma entidade subtipo em um outro relacionamento. Quando uma estrutura é formada pela combinação de relacionamentos supertipo/subtipo, essa estrutura é chamada de *hierarquia supertipo/subtipo*, ou *hierarquia de generalização*. A generalização também pode ser descrita em termos de herança, a qual especifica que todos os atributos de um supertipo são propagados para baixo na hierarquia até as entidades de um tipo inferior. A generalização pode ocorrer quando uma entidade genérica, que chamamos de entidade supertipo, é particionada por diferentes valores de um atributo comum. Por exemplo, na Figura 2.4, a entidade Funcionário é uma generalização de Gerente, Engenheiro, Técnico e Secretário sobre o atributo "cargo" em Funcionário.

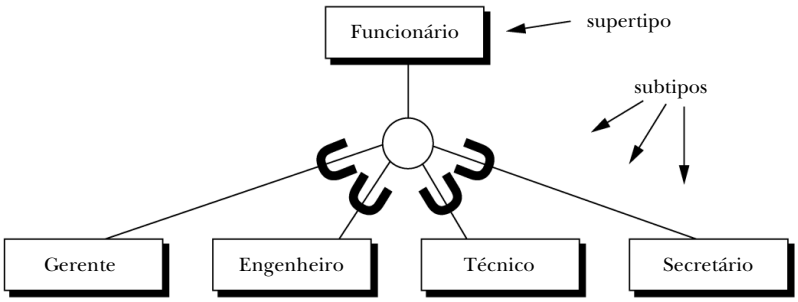
A generalização pode ainda ser classificada por duas restrições importantes nas entidades subtipo: *disjunção* e *integralidade*. A restrição de disjunção exige que as entidades subtipo sejam mutuamente exclusivas. Indicamos esse tipo de restrição com a letra "d" escrita dentro do círculo de generalização (Figura 2.4a). Os subtipos que não são disjuntos (ou seja, que se sobrepõem) são designados com o uso da letra "o" (*Overlap*) dentro do círculo. Como exemplo, a entidade supertipo Indivíduo possui duas entidades subtipo, Funcionário e Cliente; esses subtipos poderiam ser descritos como sobrepostos, ou não mutuamente exclusivos (Figura 2.4b). Independente de os subtipos serem disjuntos ou sobrepostos, eles podem ter outros atributos especiais além dos atributos genéricos (herdados) do supertipo.

A restrição de integralidade exige que os subtipos incluam totalmente o supertipo. Assim, os subtipos podem ser definidos como sendo de cobertura total ou parcial do supertipo. Por exemplo, em uma hierarquia de generalização com o supertipo Indivíduo e os subtipos Funcionário e Cliente, os subtipos podem ser descritos como totalmente inclusivo ou total. Indicamos esse tipo de restrição com uma linha dupla entre a entidade supertipo e o círculo. Isso é indicado na Figura 2.4b e implica que os únicos tipos de indivíduos a serem considerados no banco de dados são funcionários e clientes.

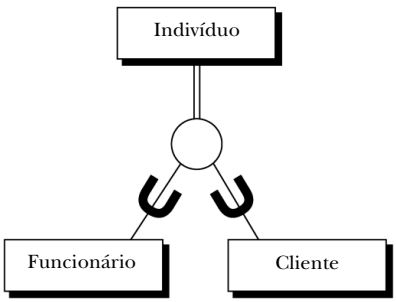
2.2.2 Agregação*

Agregação é uma forma de abstração entre um supertipo e a entidade subtipo e é significativamente diferente da abstração de generalização. A ge-

* *Nota dos Revisores Técnicos*: O termo agregação é utilizado como sinônimo do termo composição lógica por outros autores, na área de banco de dados. Esses termos são detalhados no Capítulo 3.



(a) Generalização com subtipos disjuntos



(b) Generalização com subtipos sobrepondo-se e restrições de integralidade

Figura 2.4 Supertipos e subtipos.

neralização é descrita em termos do relacionamento "é um" entre o subtipo e o supertipo — por exemplo, um Funcionário é um Indivíduo. A agregação, por outro lado, é o relacionamento entre o todo e suas partes e é descrita como um relacionamento "parte de" — por exemplo, um relatório e um protótipo do pacote de software são ambos parte dos produtos de um contrato. Assim, na Figura 2.5, a entidade Produto-software é vista como uma composição das partes componentes Programa e Guia do Usuário. O construtor de agregação é semelhante ao da generalização, no sentido de que a entidade supertipo é conectada às entidades subtipo com um círculo; nesse caso, a letra "A" aparece no círculo. Todavia, não existem símbolos de subconjunto, pois o relacionamento "parte de" não é um subconjunto. Além do mais, não existem atributos herdados na agregação; cada entidade tem seu próprio conjunto único de atributos.

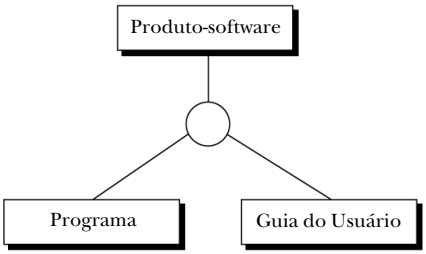


Figura 2.5 Agregação

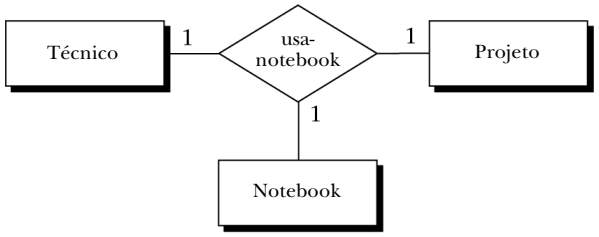
2.2.3 Relacionamentos ternários

Relacionamentos ternários são exigidos quando os relacionamentos binários não são suficientes para descrever com precisão a semântica de uma associação entre três entidades. Entretanto, os relacionamentos ternários são um pouco mais complexos do que os relacionamentos binários. A notação ER para um relacionamento ternário aparece na Figura 2.6 com três entidades conectadas a um único losango de relacionamento, e a conectividade de cada entidade é designada como "um" ou "muitos". Uma entidade em um relacionamento ternário possui conectividade "um" se uma instância dela somente puder estar associada a uma instância de cada uma das outras duas entidades associadas. Ela é "muitos" se mais de uma instância dela puder estar associada a uma instância de cada uma das outras duas entidades associadas. De qualquer forma, considera-se uma instância em relação a cada uma das outras entidades.

Como exemplo, o relacionamento "gerencia" da Figura 2.6c associa as entidades Gerente, Engenheiro e Projeto. As entidades Engenheiro e Projeto são consideradas "muitos"; a entidade Gerente é considerada "um". Isso é representado pelas afirmações a seguir.

- **Afirmação 1:** Um Engenheiro, trabalhando sob as ordens de um gerente, poderia estar trabalhando em muitos projetos.
- **Afirmação 2:** Um projeto, sob a direção de um gerente, poderia ter muitos engenheiros.
- **Afirmação 3:** Um Engenheiro, trabalhando em um projeto, só pode ter um único gerente.

A Afirmação 3 também poderia ser escrita de outra forma, usando uma seta (—>) em uma espécie de abreviação, chamada *dependência funcional*. Por exemplo:

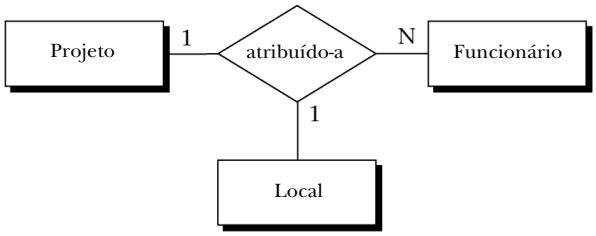


Um técnico usa exatamente um notebook para cada projeto. Cada notebook pertence a um técnico para cada projeto. Observe que um técnico ainda pode trabalhar em muitos projetos e manter diferentes notebooks para diferentes projetos.

Dependências funcionais

id-func, nome-projeto à num-notebook
id-func, num-notebook à nome-projeto
nome-projeto, num-notebook à id-func

(a) Relacionamento ternário um-para-um-para-um



Cada funcionário atribuído a um projeto trabalha em apenas um local para esse projeto, mas pode estar em diferentes locais para diferentes projetos. Em determinado local, um funcionário trabalha em apenas um projeto. Em um local específico, pode haver muitos funcionários atribuídos a determinado projeto.

Dependências funcionais

id-func, nome-loc à nome-projeto
id-func, nome-projeto à nome-loc

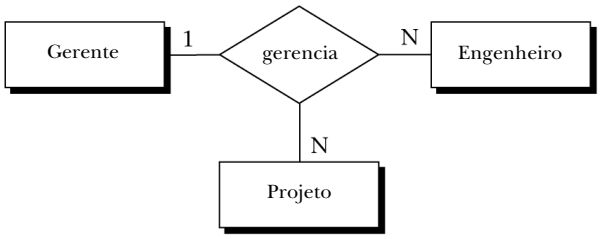
(b) Relacionamento ternário um-para-um-para-muitos

Figura 2.6 Relacionamentos ternários.

```
id-func, nome-projeto —> id-gerente
```

onde id-func é a chave (identificador exclusivo) associada à entidade Engenheiro, nome-projeto é a chave associada à entidade Projeto, e id-gerente é a chave da entidade Gerente. Em geral, para um relacionamento *n*-ário, cada entidade considerada como sendo um "um" tem a sua chave aparecendo no lado direito em exatamente uma dependência funcional (DF). Nenhuma entidade considerada "muitos" pode ter a sua chave aparecendo no lado direito de uma DF.

Todas as quatro formas de relacionamentos ternários são ilustradas na Figura 2.6. Em cada caso, a quantidade de entidades "um" resulta no número

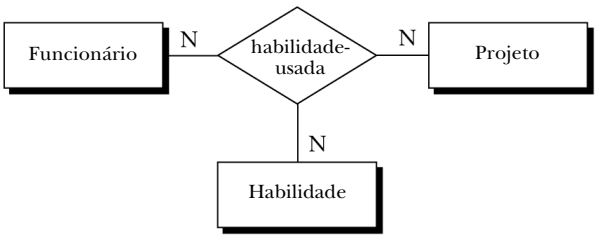


Cada engenheiro trabalhando em um determinado projeto tem exatamente um gerente, mas cada gerente de um projeto pode gerenciar muitos engenheiros, e cada gerente de um engenheiro pode gerenciar esse engenheiro em muitos projetos.

Dependências funcionais

nome-projeto, id-func à id-gerente

(c) Relacionamento ternário um-para-muitos-para-muitos



Os funcionários podem usar muitas habilidades em qualquer um dentre muitos projetos, e cada projeto possui muitos funcionários com várias habilidades.

Dependências funcionais

nenhuma

(d) Relacionamento ternário muitos-para-muitos-para-muitos

Figura 2.6 *(continuação)*

de DFs usadas para definir a semântica do relacionamento, e a chave de cada entidade "um" aparece no lado direito em exatamente uma DF desse relacionamento.

Os relacionamentos ternários podem ter atributos da mesma maneira como os relacionamentos binários muitos-para-muitos. Os valores desses atributos são determinados exclusivamente por alguma combinação das chaves das entidades associadas ao relacionamento. Por exemplo, na Figura 2.6d, o relacionamento "habilidade-usada" poderia ter o atributo "ferramenta" associado a um funcionário específico usando determinada habilidade em um certo projeto, indicando que um valor para ferramenta é determinado exclusivamente pela combinação de funcionário, habilidade e projeto.

2.2.4 Generalização para Relacionamentos *n*-ários

Generalizando a forma ternária para relacionamentos de grau maior, um relacionamento *n*-ário que descreve alguma associação entre *n* entidades é representado por um único losango de relacionamento com *n* conectores, um para cada entidade (ver Figura 2.7). O significado dessa forma pode ser mais bem descrito em termos das dependências funcionais entre as chaves das *n* entidades associadas. Pode haver algo entre zero e *n* DFs, dependendo do número de entidades "um". A coleção de DFs que descrevem um relacionamento *n*-ário precisa ter *n* componentes: *n*-1 no lado esquerdo (determinante) e 1 no lado direito. Um relacionamento ternário (n=3), por exemplo, tem dois componentes à esquerda e um à direita, como vimos nos exemplos da Figura 2.6. Em um banco de dados mais complexo, outros tipos de DFs também podem existir dentro de um relacionamento *n*-ário. Quando isso acontece, o modelo ER não oferece semântica suficiente sozinho e precisa ser suplementado com uma descrição narrativa dessas dependências.

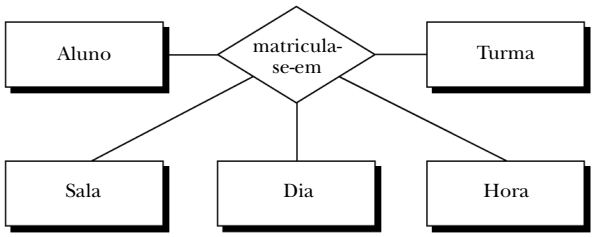


Figura 2.7 Relacionamento *n*-ário.

2.2.5 Restrição de exclusão

O tratamento normal ou *default* de vários relacionamentos é o *OR inclusivo*, que permite que algumas ou todas as entidades participem. Em algumas situações, porém, vários relacionamentos podem ser afetados pela restrição *OR exclusiva* (exclusão), que permite que no máximo uma instância de entidade, entre os vários tipos de entidade, participe do relacionamento com uma única entidade raiz. Por exemplo, na Figura 2.8, suponha que a entidade raiz Tarefa-trabalho tenha duas entidades associadas, Projeto-externo e Projeto-interno. No máximo uma das instâncias de entidade associadas poderia se aplicar a uma instância de Tarefa-trabalho.

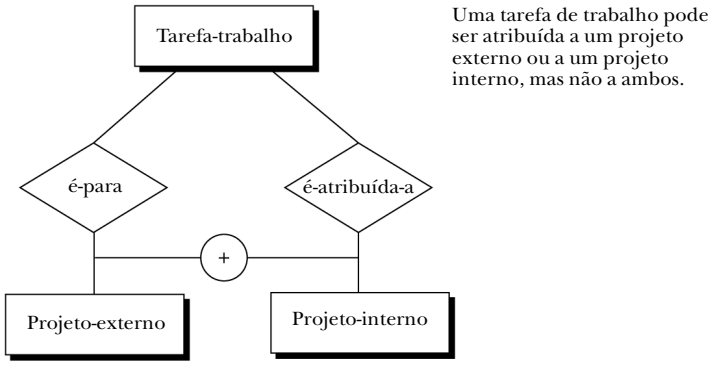


Figura 2.8 Restrição de exclusão

2.2.6 Integridade referencial

Observamos que uma chave estrangeira é um atributo de uma tabela (não necessariamente uma chave de algum tipo) que se relaciona a uma chave de uma outra tabela. A *integridade referencial* exige que, para cada instância de chave estrangeira que existe em uma tabela, a linha (e, portanto, a instância de chave) da tabela pai associada a essa instância de chave estrangeira também tenha de existir. A restrição de integridade referencial tornou-se integrante do projeto de banco de dados relacional e normalmente é sugerida como um dos requisitos necessários para a implementação de bancos de dados relacionais. (O Capítulo 5 discute a implementação SQL das restrições de integridade referencial.)

2.3 Resumo

Neste capítulo, foram descritos os conceitos básicos do modelo ER e seus construtores . Uma entidade é uma pessoa, local, coisa ou evento de interesse informativo. Os atributos são objetos que oferecem informações descritivas sobre entidades. Os atributos podem ser identificadores exclusivos ou descritores não-exclusivos. Os relacionamentos descrevem a conectividade entre instâncias de entidade: um-para-um, um-para-muitos ou muitos-para-muitos. O grau de um relacionamento é o número de entidades associadas: duas (binária), três (ternária) ou qualquer n (n-ária). O papel (nome), ou nome de relacionamento, define a função de uma entidade em um relacionamento.

O conceito de participação em um relacionamento determina se uma instância de entidade precisa participar do relacionamento (obrigatória) ou não

(opcional) para existir. Assim, por exemplo, a conectividade mínima de um relacionamento binário — ou seja, o número de instâncias de entidade em um lado, que estão associadas a uma instância no outro lado — pode ser zero, se for opcional, ou um, se for obrigatória. O conceito de generalização leva em conta a implementação das abstrações de supertipo e subtipo.

Os construtores mais avançados são usados esporadicamente nos diagramas ER e ainda não têm uma forma geralmente aceita. Eles incluem relacionamentos ternários, que definimos em termos do conceito de DFs dos bancos de dados relacionais; restrições de exclusão; e as restrições implícitas do modelo relacional, como a integridade referencial.

2.4 Resumo da literatura

A maioria das notações deste capítulo vem da definição de ER original de Chen [Chen, 1976]. O conceito de abstração de dados foi proposto inicialmente por Smith e Smith [1977] e aplicado ao modelo ER por Scheuermann, Scheffner e Weber [1980], Elmasri e Navathe [2003], Bruce [1992], IDEFIX [2005], entre outros. A aplicação do modelo de rede semântica ao projeto do esquema conceitual foi mostrada por Bachman [1977], McKleod e King [1979], Hull e King [1987] e Peckham e Maryanski [1988].

3 A UML (*Unified Modeling Language*)

A UML (*Unified Modeling Language*) é uma linguagem gráfica para comunicar especificações de projeto de software. A comunidade de desenvolvimento de software orientado a objeto criou a UML para atender às necessidades especiais de descrever o projeto de software orientado a objeto. A UML cresceu e se tornou um padrão para projetos de sistemas digitais em geral.

Existem diversos tipos diferentes de diagramas UML que atendem a diversas finalidades [Rumb05]. Os tipos de diagramas *classe* e *atividade* são particularmente úteis na discussão de questões sobre projetos de bancos de dados. Os diagramas de classe da UML capturam os aspectos estruturais encontrados nos esquemas de banco de dados. Os diagramas de atividade da UML facilitam a discussão sobre os processos dinâmicos envolvidos no projeto de banco de dados. Este capítulo apresenta uma visão geral da sintaxe e semântica dos construtores dos diagramas de classe e atividade usados neste livro. Esses mesmos conceitos são úteis para planejamento, documentação, discussão e implementação de bancos de dados. Estamos usando a UML 2.0, embora, para os propósitos dos diagramas de classe e diagramas de atividade mostrados neste livro, se você estiver acostumado com a UML 1.4 ou 1.5, provavelmente não perceberá quaisquer diferenças.

Os diagramas de classe da UML e os modelos entidade-relacionamento (ER) [Chen, 1976; Chen, 1987] são semelhantes em formato e semântica. Os criadores originais da UML apontam a influência dos modelos ER nas origens dos diagramas de classe [Rumbaugh, Jacobson e Booch, 2005]. A influência da UML, por sua vez, afetou a comunidade de banco de dados. Os diagramas

de classe agora aparecem com freqüência na literatura de banco de dados para descrever esquemas de banco de dados.

Os diagramas de atividade da UML são semelhantes em finalidade aos fluxogramas. Os processos são particionados em atividades constituintes junto com as especificações de fluxo de controle.

Este capítulo é organizado em três seções. A Seção 3.1 apresenta a notação do diagrama de classe, com alguns exemplos. A Seção 3.2 abrange a notação do diagrama de atividade, junto com exemplos ilustrativos. A Seção 3.3 conclui o capítulo com algumas regras práticas para o uso da UML.

3.1 Diagramas de classe

Uma *classe* é um descritor para um conjunto de objetos que compartilham alguns atributos e/ou operações. Conceituamos as classes de objetos de nossa vida diária. Por exemplo, um carro possui atributos, como número de identificação de veículo (chassis) e quilometragem. Um carro também tem operações, como acelerar e frear. Todos os carros possuem esses atributos e operações. Carros individuais diferem nos detalhes. Determinado carro tem seus próprios valores de chassis e quilometragem. Por exemplo, um carro poderia ter o chassis INXBR32ES3Z126369 e uma quilometragem de 22.137 quilômetros. Carros individuais são objetos que são instâncias da classe "carro".

Classes e objetos são as formas naturais de conceituar o mundo ao nosso redor. Os conceitos de classes e objetos também são os paradigmas que formam o alicerce da programação orientada a objeto. O desenvolvimento da programação orientada a objeto levou à necessidade de uma linguagem para descrever o projeto orientado a objeto, fazendo surgir a UML.

Há uma correspondência muito próxima entre os diagramas de classe da UML e os diagramas ER. As classes são semelhantes às entidades. Os esquemas de banco de dados podem ser diagramados usando a UML. É possível conceituar uma tabela de banco de dados como uma classe. As colunas na tabela são os atributos, e as linhas são objetos dessa classe. Por exemplo, poderíamos ter uma tabela chamada "Carro" com as colunas chamadas "chassis" e "quilometragem". Cada linha na tabela teria valores para essas colunas, representando um carro individual. Determinado carro poderia ser representado por uma linha com o valor "INXBR32ES3Z126369" na coluna chassis e 22.137 na coluna quilometragem.

A principal diferença entre classes e entidades é a falta de operações nas entidades. Observe que o termo *operação* é usado aqui no sentido definido

pela UML. Os procedimentos armazenados (*stored procedures*), funções, *triggers* e restrições são formas de comportamento nomeado que podem ser definidas nos bancos de dados; porém, estas não estão associadas ao comportamento das linhas individuais. O termo *operações* da UML refere-se aos métodos intrínsecos das classes de objetos. Esses comportamentos não são armazenados na definição de linhas dentro do banco de dados. Não existem operações chamadas "acelerar" ou "frear" associadas às linhas em nossa tabela "Carro". As classes podem ser exibidas com seus atributos e sem as operações da UML, que é o uso típico para os esquemas de banco de dados.

3.1.1 Notação básica do diagrama de classe

O topo da Figura 3.1 ilustra a sintaxe da UML para uma classe, mostrando *atributos* e *operações*. Também é possível incluir compartimentos nomeados definidos pelo usuário, como responsabilidades. Focalizamos os compartimentos nome de classe, atributos e operações. O ícone da UML para uma classe é um retângulo. Quando a classe aparece com atributos e operações, o retângulo é subdividido em três compartimentos horizontais. O compartimento superior contém o nome da classe, centralizado em negrito, começando com uma letra maiúscula. Normalmente, nomes de classe são substantivos. O compartimento intermediário contém nomes de atributos, alinhados à esquerda em texto normal, começando com uma letra minúscula. O compartimento inferior contém nomes de operações, alinhados à esquerda em texto normal, começando com uma letra minúscula e terminando com uma expressão entre parênteses. Os parênteses podem conter argumentos da operação.

A notação de classe tem algumas variações de acordo com a ênfase. As classes podem ser escritas sem o compartimento de atributo e/ou o compartimento de operações. As operações são importantes no software. Se o projetista do software quiser focalizar as operações, a classe poderá ser mostrada apenas com os compartimentos de nome de classe e operações. Mostrar operações e ocultar atributos é uma sintaxe muito comum usada pelos projetistas de software. Os projetistas de banco de dados, por outro lado, geralmente não lidam com as operações de classe; mas os atributos têm importância fundamental. As necessidades do projetista de banco de dados podem ser atendidas escrevendo-se a classe apenas com a apresentação dos compartimentos de nome de classe e de atributos. Ocultar operações e mostrar atributos é uma sintaxe incomum para um projetista de software, mas é comum para o projeto do banco de dados. Por fim, nos diagramas de alto nível, normalmente se

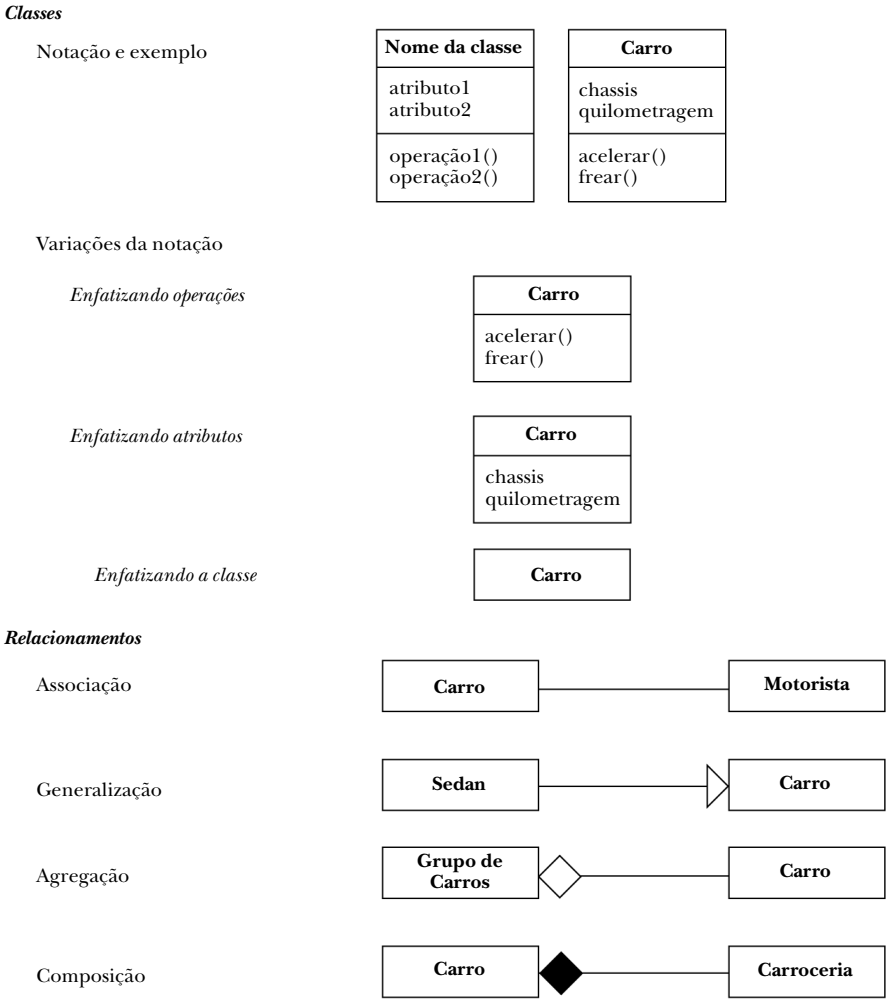


Figura 3.1 Construtores básicos do diagrama de classes da UML.

deseja ilustrar os relacionamentos das classes sem enchê-los com detalhes dos atributos e operações. As classes podem ser escritas apenas com o compartimento de nome de classe quando se deseja simplicidade.

Vários tipos de relacionamentos podem existir entre as classes. As *associações* são um tipo de relacionamento. A forma mais genérica de associação é desenhada com uma linha conectando duas classes. Por exemplo, na Figura 3.1, existe uma associação entre a classe "Carro" e a classe chamada "Motorista".

Alguns tipos de associações, como *agregação* e *composição*, são muito comuns. A UML designou símbolos para essas associações. A agregação indica

associações "parte de", em que as partes têm uma existência independente. Por exemplo, um Carro pode fazer parte de um Grupo de Carros. O Carro também existe por conta própria, independente de qualquer Grupo de Carros. Outro recurso que distingue a agregação é que a parte pode ser compartilhada entre vários objetos. Por exemplo, um Carro pode pertencer a mais de um Grupo de Carros. A associação de agregação é indicada com um losango vazio conectado à classe que mantém as partes. A Figura 3.1 indica que um Grupo de Carros agrega Carros.

A composição é uma outra associação "parte de" em que as partes são estritamente possuídas, e não compartilhadas. Por exemplo, uma Carroceria é parte de um único Carro. A notação para composição é uma associação adornada com um losango preto sólido conectado à classe que possui as partes. A Figura 3.1 indica que uma Carroceria faz parte da composição de um carro.

A generalização é outro relacionamento comum. Por exemplo, Sedan é um tipo de carro. A classe "Carro" é mais genérica do que a classe "Sedan". A generalização é indicada por uma linha sólida adornada com uma ponta de seta vazia apontando para a classe mais geral. A Figura 3.1 mostra a generalização da classe Sedan para a classe Carro.

3.1.2 Diagramas de classe para projeto de banco de dados

O leitor pode estar interessado nas semelhanças e diferenças entre os diagramas de classe da UML e os modelos ER. As Figuras de 3.2 a 3.5 colocam lado a lado algumas das figuras no Capítulo 2, permitindo a fácil comparação. Depois, voltamos nossa atenção para capturar a informação de chave primária na Figura 3.6. Concluímos esta seção com um exemplo de esquema de banco de dados da indústria musical, ilustrado pelas Figuras 3.7 a 3.10.

A Figura 3.2 ilustra os construtores da UML para relacionamentos com vários graus de associação e multiplicidades. Esses exemplos são paralelos aos modelos ER mostrados na Figura 2.2. Você pode se referir à Figura 2.2 se quiser comparar os construtores da UML com os da ER.

As associações entre classes podem ser reflexivas, binárias ou *n*-árias. A *associação reflexiva* (ou recursiva) é um termo emprestado da modelagem ER. Não é um termo definido na UML, embora mereça ser discutido. A associação reflexiva relaciona uma classe com ela mesma. A associação reflexiva da Figura 3.2 indica que um Funcionário no papel de gerente está associado a muitos Funcionários gerenciados. Os papéis das classes em um relacionamen-

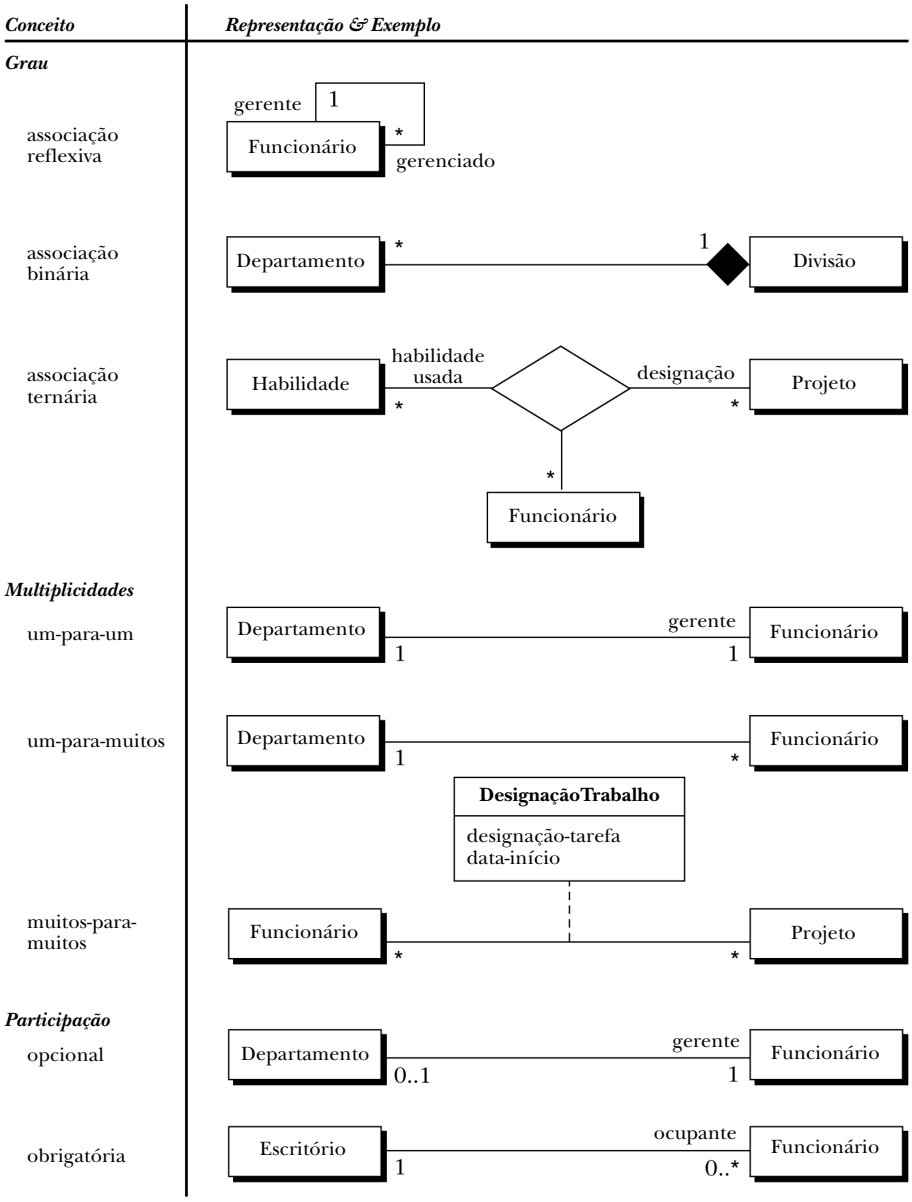


Figura 3.2 Tipos de relacionamento UML selecionados (paralelo à Figura 2.2).

to podem ser indicados nas extremidades do relacionamento. O número de objetos envolvidos no relacionamento, conhecido como *multiplicidade*, também pode ser especificado nas extremidades do relacionamento. Um asterisco indica que muitos objetos fazem parte da associação na extremidade do

relacionamento. As multiplicidades do exemplo de associação reflexiva da Figura 3.2 indicam que um Funcionário está associado a um gerente e que um gerente está associado a muitos Funcionários gerenciados.

Uma associação binária é um relacionamento entre duas classes. Por exemplo, uma Divisão possui muitos Departamentos. Observe o losango preto sólido na extremidade Divisão do relacionamento. O losango sólido é um adorno para as associações que indica composição. A Divisão é composta por Departamentos.

O relacionamento ternário da Figura 3.2 é um exemplo de uma associação *n*-ária — uma associação que relaciona três ou mais classes. Todas as classes participantes da associação estão conectadas a um losango vazio. Papéis e multiplicidades são opcionalmente indicados nas extremidades da associação *n*-ária. Cada extremidade do exemplo de associação ternária da Figura 3.2, marcada com um asterisco, o que significa muitos. O significado de cada multiplicidade é isolado das outras multiplicidades. Dada uma classe, se você pegar exatamente um objeto de cada uma das outras classes na associação, a multiplicidade será o número de objetos que podem estar associados à classe dada. Um Funcionário que esteja trabalhando em um Projeto usa muitas Habilidades. Um Funcionário usa uma Habilidade em muitas atribuições de Projeto. Uma Habilidade usada em um Projeto é satisfeita por muitos Funcionários.

Os três diagramas de classe seguintes da Figura 3.2 mostram diversas combinações de multiplicidades. A associação um-para-um ilustrada especifica que cada Departamento está associado a exatamente um Funcionário atuando no papel de gerente, e cada gerente está associado a exatamente um Departamento. O diagrama com a associação um-para-muitos significa que cada Departamento tem muitos Funcionários e que cada Funcionário pertence a exatamente um Departamento.

O exemplo de muitos-para-muitos da Figura 3.2 indica que cada Funcionário é associado a muitos Projetos e que cada Projeto está associado a muitos Funcionários. Este exemplo também ilustra o uso de uma classe de associação. Se uma associação tiver atributos, estes são escritos em uma classe que está conectada à associação com uma linha tracejada. A classe de associação chamada "DesignaçãoTrabalho" da Figura 3.2 contém dois atributos de associação chamados "atribuição-tarefa" e "data-início". A associação e a classe, juntas, formam uma classe de associação.

A multiplicidade pode ser um intervalo de inteiros, escritos com os valores mínimo e máximo separados por dois pontos. O asterisco por si só carrega o mesmo significado do intervalo [0..*]. Além disso, se o mínimo e máximo

forem iguais, então a multiplicidade poderá ser escrita com um único número. Por exemplo, [1..1] significa o mesmo que [1]. A participação opcional pode ser especificada usando um zero. O [0..1] no exemplo de participação opcional da Figura 3.2 indica que um Funcionário pode ou não participar da associação com o papel de gerente do Departamento .

A participação obrigatória é especificada sempre que uma multiplicidade começa com um inteiro positivo. O exemplo de participação obrigatória da Figura 3.2 indica que um Funcionário é ocupante de exatamente um Escritório. Uma extremidade de uma associação pode indicar a participação obrigatória, enquanto a outra extremidade pode usar a participação opcional. Isso acontece no exemplo, em que um Escritório pode ter qualquer quantidade de ocupantes, incluindo zero.

A generalização é um outro tipo de relacionamento; uma superclasse é uma generalização de uma subclasse. A especialização é o oposto da generalização; uma subclasse é uma especialização da superclasse. O relacionamento de generalização na UML é escrito com uma ponta de seta vazia apontando da subclasse para a superclasse generalizada. O exemplo no topo da Figura 3.3 mostra quatro subclasses: Gerente, Engenheiro, Técnico e Secretário. Essas quatro subclasses são todas especializações da superclasse mais geral Funcionário; ou seja, Gerentes, Engenheiros, Técnicos e Secretários são tipos de Funcionários.

Observe que os quatro relacionamentos compartilham uma ponta de seta comum. Semanticamente, ainda são quatro relacionamentos separados. O compartilhamento da ponta de seta é permissível em UML, para melhorar a clareza dos diagramas.

O exemplo inferior da Figura 3.3 ilustra que uma classe pode atuar como uma subclasse em um relacionamento e como uma superclasse em outro relacionamento. A classe chamada Indivíduo é uma generalização das classes Funcionário e Cliente. As classes Funcionário e Cliente, por sua vez, são superclasses da classe FuncCli. Uma classe pode ser uma subclasse em mais de um relacionamento de generalização. O significado no exemplo é que um objeto FuncCli é tanto um Funcionário quanto um Cliente.

Você pode ocasionalmente descobrir que a UML não fornece um símbolo padrão para o que está tentando comunicar. A UML incorpora alguma facilidade de extensão para acomodar as necessidades do usuário, como uma *nota*. Uma nota em UML é escrita como um retângulo com uma dobra no canto superior direito. A nota pode se conectar ao(s) elemento(s) pertinente(s) com uma linha tracejada. Escreva resumidamente na nota o que você deseja

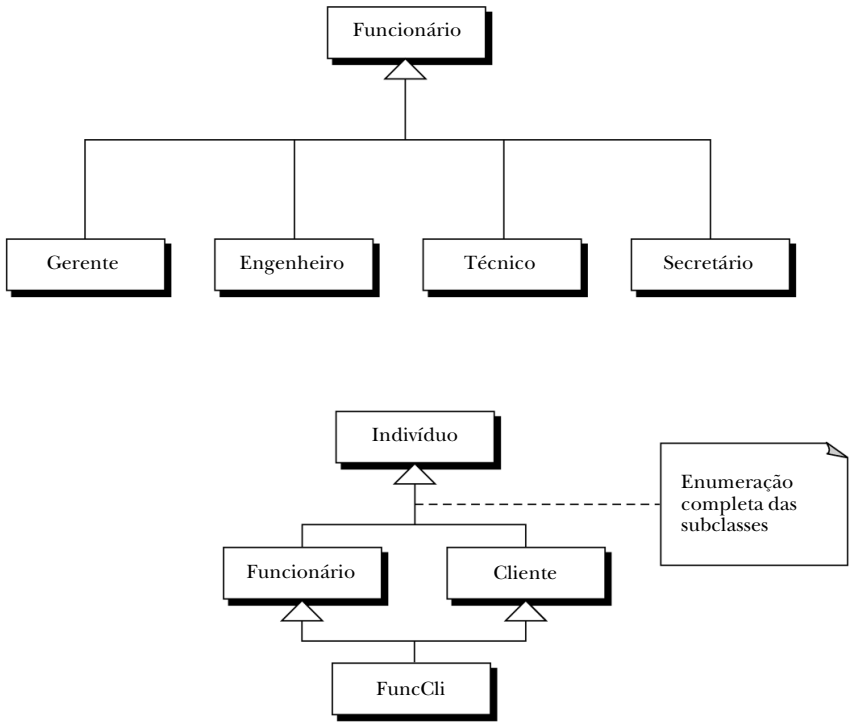


Figura 3.3 Construtores de generalização da UML (paralelas à Figura 2.4).

transmitir. O diagrama inferior da Figura 3.3 ilustra uma nota, que descreve as classes Funcionário e Cliente como sendo a "Enumeração completa das subclasses".

A distinção entre composição e agregação às vezes é indistinta para os que são novos à UML. A Figura 3.4 mostra um exemplo de cada, para ajudar a esclarecer. O diagrama do topo indica que um Programa e Documentação Eletrônica contribuem para a composição de um Produto de Software. A composição indica que as partes não existem sem que pertençam ao Produto de Software (não existe pirataria de software em nosso mundo ideal). O diagrama inferior especifica que um Professor e um Livro-texto são agregados por um curso. A agregação indica que o Professor e o Livro-texto fazem parte do Curso, mas também existem separadamente. Se um Curso for cancelado, o Professor e o Livro-texto continuarão a existir.

A Figura 3.5 ilustra um outro exemplo de relacionamento *n*-ário. O relacionamento *n*-ário pode ser esclarecido especificando-se os papéis próximos às

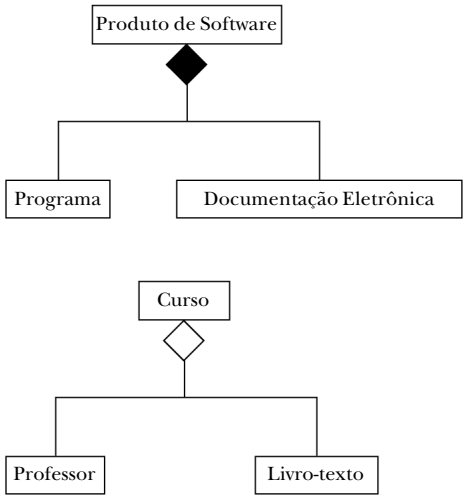


Figura 3.4 Construtores de agregação da UML (paralelo à Figura 2.5).

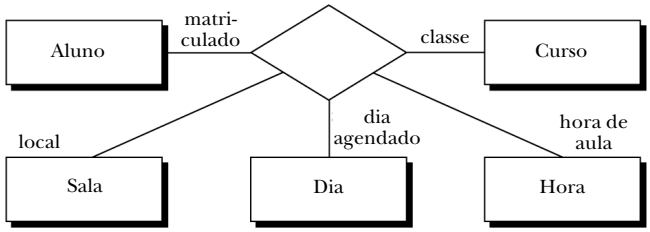


Figura 3.5 Relacionamento *n*-ário da UML (paralelo à Figura 2.7).

classes participantes. Um Aluno é um matriculado em uma classe, associada a um determinado local de Sala, Dia agendado e Hora de aula.

O conceito de uma chave primária surge no contexto de projeto de banco de dados. Normalmente, cada linha de uma tabela é identificada exclusivamente pelos valores contidos em uma ou mais colunas designadas como chave primária. Os objetos no software normalmente não são identificados dessa forma. Como resultado, a UML não possui um ícone que representa uma chave primária. Contudo, a UML é extensível. O significado de um elemento na UML pode ser estendido com um *estereótipo*. Os estereótipos são representados com uma palavra ou frase curta na linguagem natural, delimitada por

divisas: « e ». Tiramos proveito dessa capacidade de extensão, usando o estereótipo «pk», para designar atributos que foram a chave primária. A Figura 3.6 ilustra o mecanismo de estereótipo. O atributo chassis identifica um Carro específico. Uma regra digna de observação para chaves primárias: quando existe um relacionamento de composição, a chave primária da parte inclui a chave primária do objeto que a possui. O segundo diagrama da Figura 3.6 ilustra esse ponto.

3.1.3 Exemplo da indústria musical

Grandes esquemas de banco de dados podem ser apresentados em diagramas de alto nível. Os detalhes podem ser desmembrados em diagramas adicionais. O objetivo geral é apresentar idéias de uma maneira clara e organizada. A UML fornece variações notacionais e mecanismos organizacionais. É possível perceber que existem várias maneiras de representar um mesmo conceito em UML. As decisões tomadas em relação à sua representação dependem em parte da finalidade de um determinado diagrama. As Figuras 3.7 a 3.10 ilustram algumas das possibilidades, com um exemplo retirado da indústria musical.

Os *pacotes* podem ser usados para organizar as classes dentro de grupos. Os pacotes também podem, por si só, ser agrupados em pacotes. O objetivo de usar pacotes é tornar o projeto geral de um sistema mais compreensível.

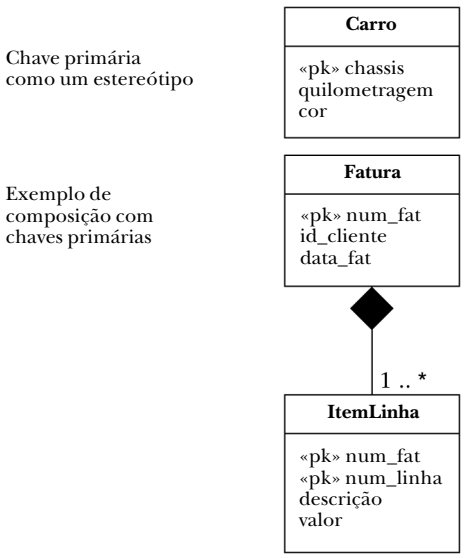


Figura 3.6 Construtores da UML ilustrando chaves primárias.

Os pacotes podem ser usados para representar esquemas. Você pode então mostrar vários esquemas de forma concisa. Outro uso para os pacotes é agrupar classes relacionadas dentro de um esquema e apresentar o esquema com clareza. Dado um determinado conjunto de classes, diferentes pessoas podem conceituar diferentes agrupamentos. A divisão é uma decisão de projeto, sem uma resposta certa ou errada. Qualquer que seja a decisão tomada, o resultado deverá melhorar a legibilidade. A notação para um pacote é um ícone de pasta, e o conteúdo de um pacote pode ser mostrado opcionalmente no corpo da pasta. Se o conteúdo aparecer, então o nome do pacote é colocado na guia, no alto da pasta. Se o conteúdo estiver oculto, então o nome do pacote é colocado no corpo do ícone.

Se a finalidade for ilustrar os relacionamentos dos pacotes, e as classes não forem importantes no momento, então é melhor ilustrar com o conteúdo oculto. A Figura 3.7 ilustra a notação com o exemplo da indústria musical em um nível bastante elevado. A Música é criada e colocada na Mídia. Depois, a Mídia é Distribuída. Há uma associação entre Música e Mídia, e entre Mídia e Distribuição.

Vejamos a organização das classes. A indústria da música é ilustrada na Figura 3.8 com as classes listadas. O pacote Música contém classes que são responsáveis por criar a música. Exemplos de Grupos são Beatles e os Bangles. Sarah McLachlan e Sting são Artistas. Grupos e Artistas estão envolvidos na criação da música. Veremos rapidamente as outras classes e como elas estão relacionadas. O pacote Mídia contém classes que mantêm fisicamente a gravação da música. O pacote Distribuição contém classes que levam a mídia para você.

O conteúdo de um pacote pode ser expandido para visualizar mais detalhes. Os relacionamentos entre as classes pertencentes ao pacote Música são ilustrados na Figura 3.9. Um Grupo é uma agregação de dois ou mais Artistas. Conforme indicado pela multiplicidade entre Artista e Grupo, [0..*], um Artista pode ou não estar em um Grupo, e pode estar em mais de um Grupo. Compositores, Letristas e Músicos são diferentes tipos de artistas. Uma Canção está associada a um ou mais Compositores. Uma Canção pode não ter qualquer Letrista, ou ter qualquer quantidade de Letristas. Uma Canção pode

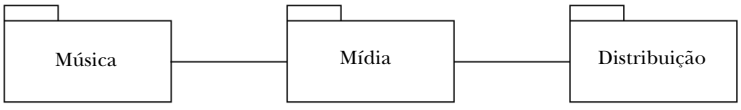


Figura 3.7 Exemplo de pacotes relacionados.

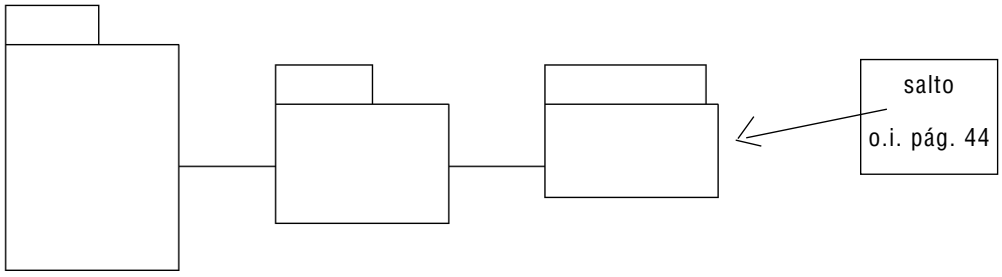


Figura 3.8 SALTO O.I. pág. 44.

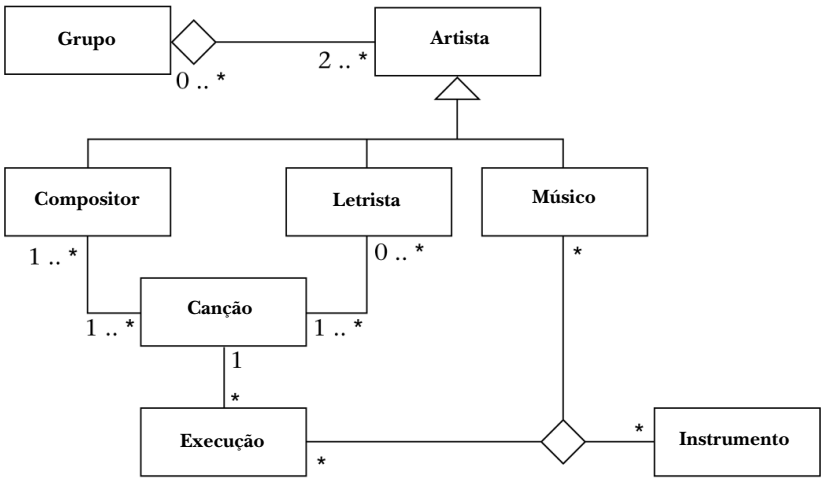


Figura 3.9 Relacionamentos entre classes no pacote música.

ter qualquer quantidade de Execuções. Uma Execução está associada a exatamente uma Canção. Uma Execução está associada aos Músicos e Instrumentos. Uma dada combinação Músico-Instrumento está associada a qualquer número de Execuções. Uma combinação específica de Execução-Músico pode estar associada a qualquer quantidade de Instrumentos. Uma determinada combinação Execução-Instrumento está associada a qualquer quantidade de Músicos.

Um sistema pode ser entendido mais facilmente focalizando individualmente cada pacote. Voltamos nossa atenção agora para as classes e os relacio-

namentos no pacote Mídia, mostrado na Figura 3.10. As classes pertencentes aos pacotes Música e Distribuição também aparecem, detalhando como o pacote Mídia está relacionado aos outros pacotes. A Mídia Musical está associada às classes Grupo e Artista, que estão contidas no pacote Música mostrado na Figura 3.8. A Mídia Musical também está associada às classes Gravadora, Estúdio e Produtor, que estão contidas no pacote Distribuição, mostrado na Figura 3.8. Álbuns e CDs são tipos de Mídia Musical. Álbuns e CDs são compostos de Faixas. Faixas estão associadas a Execuções.

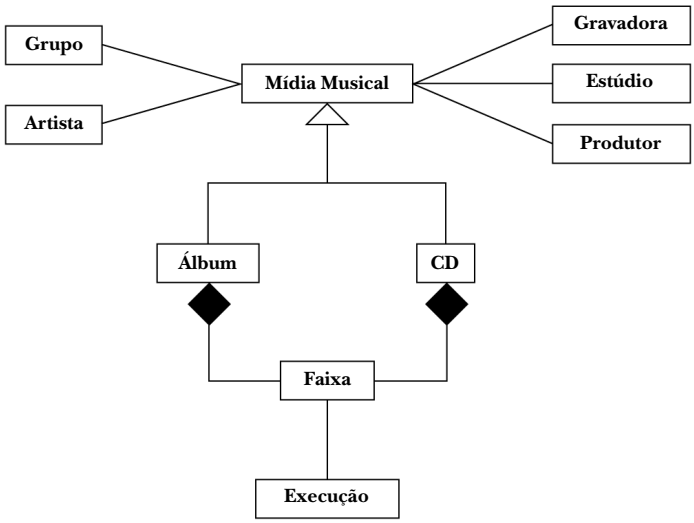


Figura 3.10 Classes de pacote de mídia e classes relacionadas.

3.2 Diagramas de atividade

A UML possui um conjunto completo de tipos de diagramas, cada um atende a uma necessidade de descrição de uma visão do projeto. Os *diagramas de atividade* da UML são usados para especificar as atividades e o fluxo de controle em um processo. O processo pode ser um fluxo de trabalho seguido por pessoas, organizações ou outros elementos físicos. Como alternativa, o processo pode ser um algoritmo implementado em software. A sintaxe e a semântica dos construtores da UML são iguais, não importando o processo descrito. Nossos exemplos foram retirados de fluxos de trabalho seguidos por pessoas e organizações, visto que são mais úteis para o projeto lógico de bancos de dados.

3.2.1 Descrição da notação do diagrama de atividade

Os diagramas de atividade possuem notação para os nós, fluxo de controle e organização. Os ícones que estamos descrevendo aqui são esboçados na Figura 3.11. A notação será esclarecida mais adiante, através de exemplos, na Seção 3.2.2.

Entre os nós estão os nós *inicial*, *final* e *de atividade*. O controle de todo o processo começa com o nó inicial, representado como um círculo preto sólido. O processo termina quando o controle atinge um nó final, representado como um círculo preto sólido cercado por um círculo concêntrico (ou seja, um olho de boi). Os nós de atividade são estados em que o trabalho especificado é processado. Por exemplo, uma atividade poderia se chamar "Gerar orçamento". O nome de uma atividade normalmente é um verbo no infinitivo ou uma frase verbal curta, escrito dentro de um losango com cantos arredondados. O controle permanece na atividade até que esta seja completada. Depois, o controle segue o fluxo de saída.

Entre os ícones de fluxo de controle estão os *fluxos*, *decisões, bifurcações* e *junções*. Um fluxo é desenhado com uma seta. O controle flui na direção das setas. Os nós de decisão são desenhados como um losango vazio com múltiplos fluxos de saída. Cada fluxo de saída de um nó de decisão precisa ter uma *condição de guarda*. Uma condição de guarda é escrita entre colchetes ao lado do fluxo. O controle flui exatamente em uma direção a partir de um nó de decisão e só segue um fluxo se a condição de guarda for verdadeira. Para evitar o comportamento não-determinístico, a condição de guarda associada a um nó de decisão precisa ser mutuamente exclusiva. Não pode haver ambigüidade quanto a que direção o controle irá fluir. Os guardas precisam abranger todas as condições de teste, de modo que o controle não fique bloqueado no nó de decisão. Um caminho pode ser guardado por [else]. Se um caminho for guardado por [else], então o controle fluirá nessa direção somente se todos os outros guardas falharem. Bifurcações e junções são formas de sincronismo escritas com uma barra sólida. A bifurcação tem um fluxo de entrada e vários fluxos de saída. Quando o controle flui para uma bifurcação, ele segue todos os fluxos de saída de forma concorrente. São conhecidos como *threads* (linhas de execução) concorrentes. As junções são o oposto das bifurcações; o construtor de junção tem vários fluxos de entrada e um fluxo de saída. O controle flui de uma junção apenas quando todos os fluxos de entrada tiverem alcançado a junção.

Os diagramas de atividade podem ser mais bem organizados usando partições, também conhecidas como raias. As partições dividem as atividades em

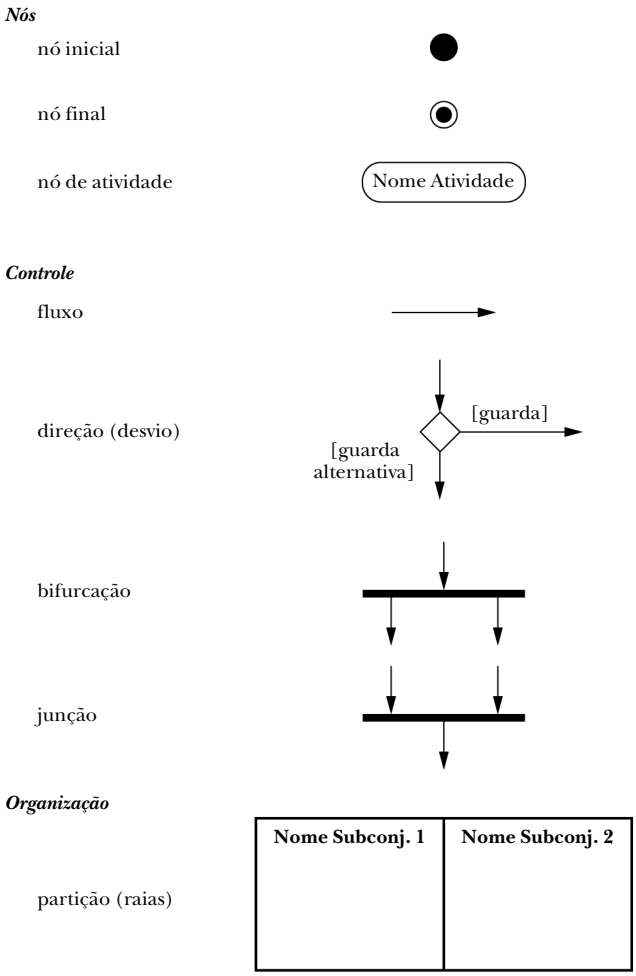


Figura 3.11 Construtores do diagrama de atividade da UML.

subconjuntos, organizados de acordo com o participante responsável. Cada subconjunto é nomeado e delimitado com linhas.

3.2.2 Diagramas de atividade para fluxo de trabalho

A Figura 3.12 ilustra os construtores do diagrama de atividade da UML usados neste livro. Esse diagrama é dividido em dois subconjuntos de atividades, organizados de acordo com o participante responsável: o subconjunto da esquerda contém atividades do Cliente, e o subconjunto da direita contém atividades do Fabricante. As partições de atividades às vezes são chamadas de raias,

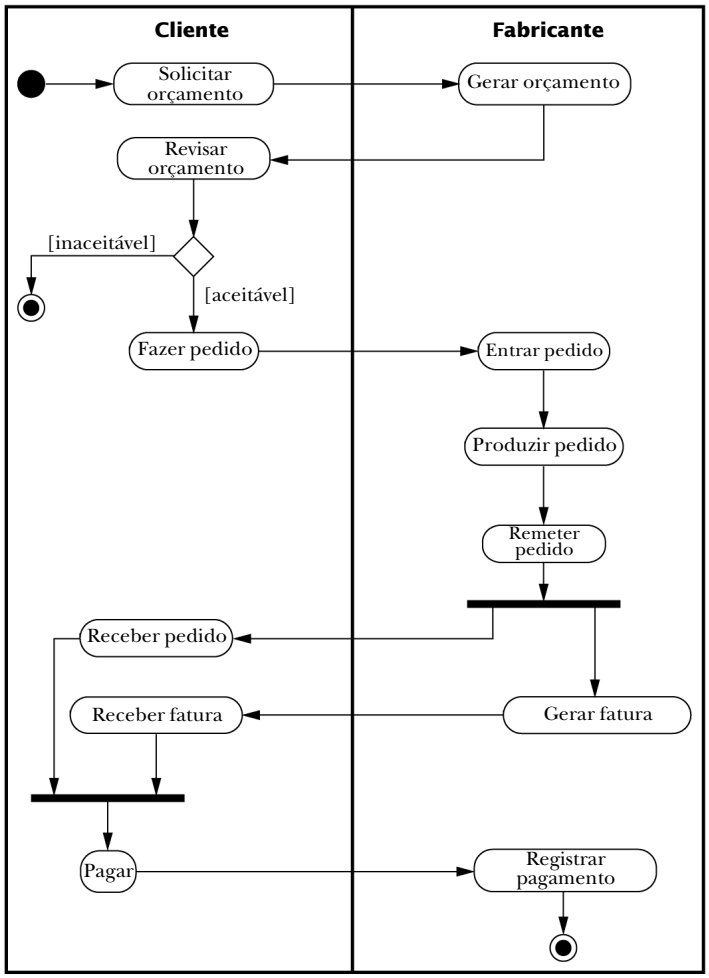


Figura 3.12 Diagrama de atividade da UML, com exemplo de manufatura.

devido à sua típica aparência. As partições de atividades podem ser dispostas verticalmente, horizontalmente ou em uma grade. Divisores curvos podem ser usados, embora isso não seja muito comum. Os diagramas de atividades também podem ser escritos sem uma partição. O construtor tem apenas a finalidade organizacional e não possui nenhuma semântica específica. O significado é sugerido por você quando escolhe um nome para o subconjunto.

O controle começa no estado inicial, representado pelo ponto sólido no canto superior esquerdo da Figura 3.12. O controle flui para a primeira atividade, na qual o cliente solicita um orçamento (Solicitar orçamento). O controle permanece em uma atividade até que ela termine; depois, o controle

segue a seta de saída. Quando a solicitação por um orçamento termina, o Fabricante gera um orçamento (Gerar orçamento). Depois, o cliente revisa o orçamento (Revisar orçamento).

O próximo construtor é um desvio, representado por um losango. Cada seta de saída de um desvio possui um guarda. O guarda representa uma condição que precisa ser verdadeira para que o controle flua por esse caminho. Os guardas são escritos como descrições de condição delimitadas por colchetes. Depois que o Cliente termina de revisar o orçamento na Figura 3.12, se ele for inaceitável, o processo atingirá o estado final e terminará. Um estado final é representado com um alvo (o olho de boi). Se o orçamento for aceitável, então o Cliente fará um pedido (Fazer pedido). O Fabricante dá a entrada (Entrar pedido), produz (Produz pedido) e envia o pedido (Remeter pedido).

Em uma bifurcação, o controle se divide em vários threads concorrentes. A notação é uma barra sólida com uma seta de entrada e várias setas de saída. Depois que o pedido é remetido na Figura 3.12, o controle atinge uma bifurcação e se divide em dois threads. O Cliente recebe o pedido (Receber pedido). Em paralelo à recepção do pedido pelo Cliente, o Fabricante gera uma fatura (Gerar fatura) e depois o Cliente recebe a fatura (Receber fatura). A ordem das atividades executadas nos threads não está definida. Assim, o Cliente pode receber o pedido antes ou depois que o fabricante gera a fatura, ou até mesmo depois que o Cliente receber a fatura.

Em uma junção, vários threads se juntam em um único. A notação é uma barra sólida com várias setas de entrada e uma seta de saída. Na Figura 3.12, depois que o Cliente recebe o pedido e a fatura, o Cliente efetua o pagamento (Pagar). Todos os threads de entrada precisam ser concluídos antes que o controle continue pela seta de saída.

Finalmente, na Figura 3.12, o Cliente paga, o Fabricante registra o pagamento (Registrar pagamento) e depois um estado final é atingido. Observe que um diagrama de atividade pode ter vários estados finais. No entanto, só pode haver um estado inicial.

Existem pelo menos dois usos para os diagramas de atividade no contexto do projeto de banco de dados. Os diagramas de atividade podem especificar as interações das classes em um esquema de banco de dados. Os diagramas de classe capturam a estrutura, os diagramas de atividade capturam o comportamento. Os dois tipos de diagramas podem apresentar aspectos complementares do mesmo sistema. Por exemplo, pode-se facilmente imaginar que a Figura 3.12 ilustra o uso de classes chamadas Orçamento, Pedido, Fatura e Pagamento. Outro uso dos diagramas de atividades no contexto do projeto de

banco de dados é na ilustração dos processos em torno do banco de dados. Por exemplo, o ciclo de vida do banco de dados pode ser ilustrado usando diagramas de atividade.

3.3 Regras práticas de uso da UML

1. Decida primeiro o que você deseja comunicar e depois concentre-se na sua descrição. Ilustre os detalhes que esclarecem sua finalidade e oculte o restante. A UML é como qualquer outra linguagem em que você pode se perder nos detalhes e se desviar de sua finalidade principal. Seja conciso.
2. Mantenha cada diagrama UML em uma página. Os diagramas são mais fáceis de entender se puderem ser vistos de uma só vez. Isso não significa que você deve se restringir; em vez disso, você deve dividir e organizar seu conteúdo em partes razoáveis, inteligíveis. Use pacotes para organizar a sua apresentação. Se você tiver muitas idéias brilhantes para transmitir (naturalmente que tem!), comece com um diagrama de alto nível, que mostra um panorama geral. Depois, continue com um diagrama dedicado a cada uma das suas idéias.
3. Use a UML quando ela for útil. Não se sinta obrigado a escrever um documento UML só por achar que precisa de um documento UML. A UML não é um fim por si só, mas é uma ferramenta de projeto excelente para problemas apropriados.
4. Acompanhe seus diagramas com descrições textuais, esclarecendo assim a sua intenção. Além disso, lembre-se de que algumas pessoas são orientadas verbalmente, outras visualmente. Um método eficiente é combinar a linguagem natural com a UML.
5. Tenha o cuidado de organizar cada diagrama de forma clara. Evite cruzar associações. Agrupe elementos se houver uma conexão em sua mente. Dois diagramas UML podem conter exatamente os mesmos elementos e associações, e um pode ser uma bagunça, enquanto o outro pode ser elegante e claro. Ambos transmitem o mesmo significado na UML, mas certamente a versão elegante terá muito mais sucesso na comunicação dos aspectos do projeto.

3.4 Resumo

A *Unified Modeling Language* (UML) é uma linguagem gráfica muito popular atualmente para comunicar especificações de projeto de software e, em

particular, para projetos lógicos de banco de dados, por meio de diagramas de classe. A semelhança entre a UML e o modelo ER é evidenciada por meio de alguns exemplos comuns, incluindo relacionamentos ternários e generalizações. Os diagramas de atividade da UML são usados para especificar as atividades e o fluxo de controle dos processos. O uso da UML no projeto lógico de banco de dados é resumido em cinco regras básicas.

3.5 Resumo da literatura

O manual de referência definitivo da UML é de Rumbaugh, Jacobson e Booch [2005]. Use Mullins [1999] para conhecer mais detalhes sobre a modelagem de banco de dados com UML. Outros textos úteis sobre UML são Naiburg e Maksimchuk [2001], Quatrani [2002] e Rumbaugh, Jacobson e Booch [2004].

4 Análise de requisitos e modelagem de dados conceitual

Este capítulo mostra como as abordagens da ER e UML podem ser aplicadas ao ciclo de vida do banco de dados, especialmente nas etapas de I a II(b) (conforme definido na Seção 1.2), que incluem os estágios de análise de requisitos e modelagem de dados conceitual do projeto lógico do banco de dados. O exemplo apresentado no Capítulo 2 é usado novamente para ilustrar os princípios de modelagem ER desenvolvidos neste capítulo.

4.1 Introdução

O projeto lógico de banco de dados é realizado com diversas abordagens, incluindo as metodologias *top-down*, *bottom-up* e combinadas. A abordagem tradicional, particularmente para bancos de dados relacionais, tem sido uma atividade de baixo nível, *bottom-up*, sintetizando os elementos de dados individuais a partir de tabelas normalizadas para depois realizar uma análise cuidadosa das interdependências dos elementos de dados definidos durante a análise de requisitos. Embora o processo tradicional tenha tido algum sucesso para bancos de dados de tamanho pequeno a médio, quando usado para grandes bancos de dados, sua complexidade pode ser muito grande, a ponto de os projetistas abandonarem o seu uso regular. Na prática, é usada uma combinação de técnicas *top-down* e *bottom-up*; na maior parte dos casos, as tabelas podem ser definidas diretamente a partir da análise de requisitos.

O modelo de dados conceitual tem sido mais bem-sucedido como uma ferramenta de comunicação entre o projetista e o usuário final durante as fases de análise de requisitos e projeto lógico. Seu sucesso deve-se ao fato

de o modelo, usando ER ou UML, ser fácil de entender e conveniente de representar. Outro motivo para sua eficácia é que essa é uma técnica *top-down*, que usa o conceito de abstração. O número de entidades em um banco de dados normalmente é muito menor do que o número de elementos de dados individuais, pois os elementos de dados normalmente representam os atributos. Portanto, usar entidades como uma abstração para os elementos de dados e focalizar os relacionamentos entre entidades reduz bastante o número de objetos em consideração e simplifica a análise. Embora ainda seja necessário representar os elementos de dados por atributos de entidades em nível conceitual, suas dependências normalmente são confinadas aos outros atributos dentro da entidade ou, em alguns casos, a atributos associados a outros elementos com um relacionamento direto com sua entidade.

As principais dependências entre atributos que ocorrem nos modelos de dados são as dependências entre as *chaves de entidade*, que são os identificadores exclusivos das diferentes entidades capturadas no processo conceitual de modelagem de dados. Casos especiais, como as dependências entre os elementos de dados de entidades não relacionadas, podem ser tratados quando forem identificados na análise de dados subseqüente.

A abordagem de projeto lógico de banco de dados definida aqui utiliza o modelo de dados conceitual e o modelo relacional em estágios sucessivos. Ele se beneficia da simplicidade e facilidade de uso do modelo de dados conceitual e da estrutura e formalismo associados ao modelo relacional. Para facilitar essa abordagem, é necessário construir uma estrutura para transformar os vários construtores do modelo de dados conceitual em tabelas que já estão normalizadas ou que podem ser normalizadas com um mínimo de transformação. A beleza desse tipo de transformação é que ele resulta em tabelas SQL normalizadas ou quase normalizadas desde o início; em geral, normalizações posteriores não são necessárias.

Contudo, antes disso, precisamos primeiro definir as principais etapas da metodologia do projeto lógico relacional no contexto do ciclo de vida do banco de dados.

4.2 Análise de requisitos

A etapa I, a análise de requisitos, é extremamente importante no ciclo de vida do banco de dados, e normalmente é a que exige mais trabalho. O projetista de banco de dados precisa entrevistar a população de usuários finais e

determinar exatamente a finalidade de uso do banco de dados e o que ele precisa conter. Os objetivos básicos da análise de requisitos são:

- Delinear os requisitos de dados da empresa em termos dos elementos de dados básicos.
- Descrever a informação sobre os elementos de dados e os relacionamentos entre eles necessários para modelar esses requisitos de dados.
- Determinar os tipos de transações que devem ser executadas no banco de dados e a interação entre as transações e os elementos de dados.
- Definir quaisquer restrições de desempenho, integridade, segurança ou administrativas que tenham de ser impostas sobre o banco de dados resultante.
- Especificar quaisquer restrições de projeto e de implementação, tais como tecnologias, hardware e software, linguagens de programação, políticas, padrões ou interfaces externas específicos.
- Documentar por completo todos os itens anteriores em uma especificação de requisitos detalhada. Os elementos de dados também podem ser definidos em um sistema de dicionário de dados, normalmente fornecido como parte integral do sistema de gerenciamento de banco de dados.

O modelo de dados conceitual ajuda os projetistas a capturarem com precisão os requisitos de dados reais, pois exige que focalizem nos detalhes semânticos dos relacionamentos de dados, que é maior do que os detalhes que seriam fornecidos apenas pelas DFs. A semântica do modelo ER, por exemplo, leva em conta as transformações diretas de entidades e relacionamentos para tabelas na primeira forma normal (1FN), pelo menos. Ela também fornece orientações claras para as restrições de integridade. Além disso, técnicas de abstração (como generalização) fornecem ferramentas úteis para integrar as visões do usuário final, a fim de definir um esquema conceitual global.

4.3 Modelagem de dados conceitual

Agora, vejamos mais de perto os elementos de dados e relacionamentos básicos que devem ser definidos durante a análise de requisitos e o projeto conceitual. Essas duas etapas do ciclo de vida normalmente são feitas ao mesmo tempo.

Considere as subetapas da etapa II(a), a modelagem de dados conceitual, usando o modelo ER:

- Classificar entidades e atributos (em UML, classificar classes e atributos).
- Identificar as hierarquias de generalização (para o modelo ER e a UML).
- Definir relacionamentos (em UML, definir associações e classes de associação).

O restante desta seção discute as tarefas envolvidas em cada subetapa.

4.3.1 Classificar entidades e atributos

Embora seja fácil definir os construtores de entidade, atributo e de relacionamento, não é tão fácil distinguir seus papéis na modelagem do banco de dados. O que torna um elemento de dados uma entidade, um atributo ou mesmo um relacionamento? Por exemplo, as centrais de projeto estão localizadas em cidades. "Cidade" deve ser uma entidade ou um atributo? Um currículo é mantido para cada funcionário. "Currículo" é uma entidade ou um relacionamento?

As seguintes orientações para classificar entidades e atributos ajudarão a convergir os pensamentos do projetista para um projeto de banco de dados relacional normalizado:

- Entidades devem conter informações descritivas.
- Atributos multivalorados devem ser classificados como entidades.
- Atributos devem estar conectados às entidades que descrevem mais diretamente.

Agora, vamos examinar cada orientação individualmente.

Conteúdo da entidade

Entidades devem conter informações descritivas. Se houver informação descritiva sobre um elemento de dados, este deverá ser classificado como uma entidade. Se um elemento de dados exigir apenas um identificador e não tiver relacionamentos, ele deverá ser classificado como um atributo. Com "cidade", por exemplo, se houver alguma informação descritiva como "país" e "população" para cidades, então "cidade" deverá ser classificado como uma entidade. Se apenas o nome da cidade for necessário para identificar uma cidade, então "cidade" deverá ser classificado como um atributo associado a alguma entidade, como Projeto. A exceção a essa regra é que, se a identidade do valor precisar ser restrita por inclusão em um conjunto, ele deverá ser criado como uma entidade. Por exemplo, "Estado é muito parecido com cidade,

mas você provavelmente desejará ter uma entidade Estado que contém todas as instâncias de Estado válidas." Alguns exemplos de outros elementos de dados no mundo real, normalmente classificados como entidades, são: Funcionário, Tarefa, Projeto, Departamento, Empresa, Cliente e assim por diante.

Atributos multivalorados

Atributos multivalorados devem ser classificados como entidades. Se mais de um valor de um atributo descritor corresponder a um valor de um identificador, o descritor deverá ser classificado como uma entidade, em vez de um atributo, embora ele mesmo não tenha descritores. Uma grande empresa, por exemplo, poderia ter muitas divisões, algumas delas possivelmente em cidades diferentes. Nesse caso, "divisão" poderia ser classificada como um atributo multivalorado de "empresa", mas seria mais bem classificada como uma entidade, com "endereço-divisão" como seu identificador. Se os atributos forem restritos a terem um único valor, as decisões posteriores de projeto e implementação serão simplificadas.

Conexão de atributo

Atributos devem estar conectados às entidades que descrevem mais diretamente. Por exemplo, "nome-prédio-escritórios" normalmente deve ser um atributo da entidade Departamento, em vez da entidade Funcionário. O procedimento de identificar entidades e conectar atributos a entidades é iterativo. Classifique alguns elementos de dados como entidades e conecte identificadores e descritores a eles. Se você encontrar alguma violação das orientações anteriores, mude alguns elementos de dados de entidade para atributo (ou de atributo para entidade), conecte atributos às novas entidades, e assim por diante.

4.3.2 Identificar as hierarquias de generalização

Se houver uma hierarquia de generalização entre as entidades, então coloque o identificador e os descritores genéricos na entidade supertipo e coloque o mesmo identificador e descritores específicos nas entidades subtipo.

Por exemplo, suponha que cinco entidades fossem identificadas no modelo ER mostrado na Figura 2.4a:

- Funcionário, com o identificador num-func e descritores nome-func, endereço e data-nascimento.

- Gerente, com identificador num-func e descritores nome-func e cargo.
- Engenheiro, com identificador num-func e descritores nome-func, formação e cargo.
- Técnico, com identificador num-func e descritores nome-func e especialidade.
- Secretário, com identificador num-func e descritores nome-func e melhor-habilidade.

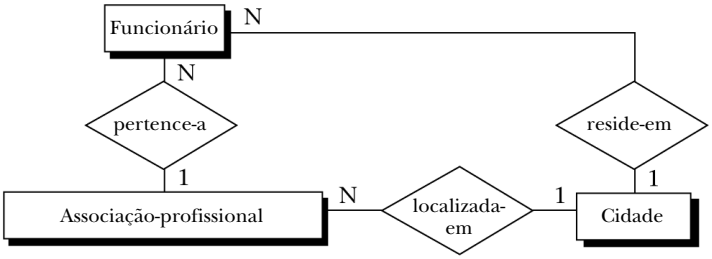
Digamos que tenhamos determinado, através de nossa análise, que a entidade Funcionário poderia ser criada como uma generalização de Gerente, Engenheiro, Técnico e Secretário. Então, colocamos o identificador num-func e os descritores genéricos nome-func, endereço e data-nascimento na entidade supertipo Funcionário; o identificador num-func e o descritor específico cargo da entidade subtipo Gerente; o identificador num-func e o descritor específico formação e cargo na entidade de subtipo Engenheiro etc. Mais tarde, se decidirmos eliminar **Funcionário** como uma tabela, os identificadores originais e os atributos genéricos poderão ser redistribuídos para todas as tabelas subtipo. (Observe que colocamos nomes de tabela em negrito no decorrer de todo o livro, para facilitar a leitura).

4.3.3 Definir relacionamentos

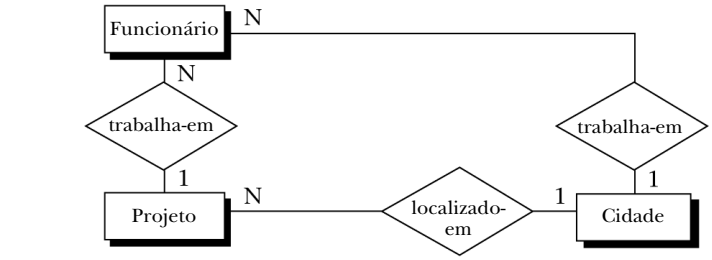
Agora, tratamos dos elementos de dados que representam associações entre entidades, que chamamos de relacionamentos. Alguns exemplos de relacionamentos típicos são "trabalha-em", "trabalha-para", "compra", "dirige" ou qualquer verbo que conecte entidades. Para cada relacionamento, os seguintes conceitos devem ser especificados: grau (binário, ternário etc.); conectividade (um-para-muitos etc.); participação opcional ou obrigatória; e quaisquer atributos associados ao relacionamento, e não às entidades. A seguir, vejamos algumas orientações para definir os tipos de relacionamentos mais difíceis.

Relacionamentos redundantes

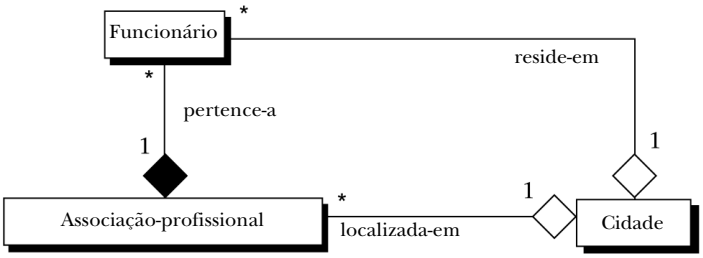
Analise os relacionamentos redundantes com cuidado. Dois ou mais relacionamentos usados para representar o mesmo conceito são considerados redundantes. Os relacionamentos redundantes provavelmente resultarão em tabelas não normalizadas quando o modelo ER for transformado em esquemas relacionais. Observe que dois ou mais relacionamentos são permitidos entre as mesmas duas entidades, desde que esses relacionamentos tenham



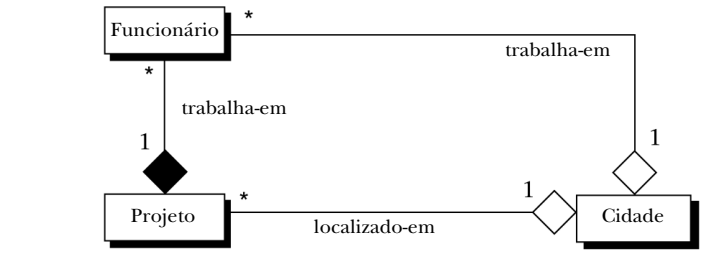
(a) Relacionamentos não-redundantes



(b) Relacionamentos redundantes usando transitividade



(c) Associações não-redundantes



(d) Associações redundantes usando transitividade

Figura 4.1 Relacionamentos redundantes.

significados diferentes. Nesse caso, eles não são considerados redundantes. Um caso importante de não redundância aparece na Figura 4.1a para o modelo ER e na Figura 4.1c para UML. Se "pertence-a" for um relacionamento um-para-muitos entre Funcionário e Associação-profissional; se "localizada-em" for um relacionamento um-para-muitos entre Associação-profissional e Cidade; e se "reside-em" for um relacionamento um-para-muitos entre Funcionário e Cidade; então "reside-em" não será redundante, pois os relacionamentos não estão relacionados. Entretanto, considere a situação mostrada na Figura 4.1b para o modelo ER e a Figura 4.1d para a UML. O funcionário trabalha em um projeto localizado em uma cidade, de modo que o relacionamento "trabalha-em" entre Funcionário e Cidade é redundante e pode ser eliminado.

Relacionamentos ternários

Defina os relacionamentos ternários com cuidado. Definimos um relacionamento ternário entre três entidades apenas quando o conceito não puder ser representado por vários relacionamentos binários entre essas entidades. Por exemplo, vamos supor que exista alguma associação entre as entidades Técnico, Projeto e Notebook. Se cada técnico pode trabalhar em qualquer um dos vários projetos e usar os mesmos notebooks em cada projeto, então podem ser definidos três relacionamentos binários muitos-para-muitos (ver Figura 4.2a para o modelo ER e a Figura 4.2c para a UML). No entanto, se cada técnico for limitado a usar exatamente um notebook para cada projeto e esse notebook pertencer a somente um técnico, então um relacionamento ternário um-para-um-para-um deverá ser definido (ver Figura 4.2b para o modelo ER e a Figura 4.2d para a UML). A técnica a ser usada na modelagem ER é primeiro tentar expressar as associações em termos de relacionamentos binários; se isso for impossível devido às restrições das associações, tente expressá-las em termos de um relacionamento ternário.

O significado da conectividade para os relacionamentos ternários é importante. A Figura 4.2b mostra que, para determinado par de instâncias de Técnico e Projeto, há somente uma instância correspondente de Notebook; para determinado par de instâncias de Técnico e Notebook, há somente uma instância correspondente de Projeto; e para determinado par de instâncias de Projeto e Notebook, há somente uma instância de Técnico. Em geral, sabemos, pela nossa definição de relacionamentos ternários, que se um relacionamento entre três entidades só puder ser expresso por uma dependência funcional (DF) envolvendo as chaves de todas as três entidades, então ele não poderá ser expresso usando apenas relacionamentos binários. Os relaciona-

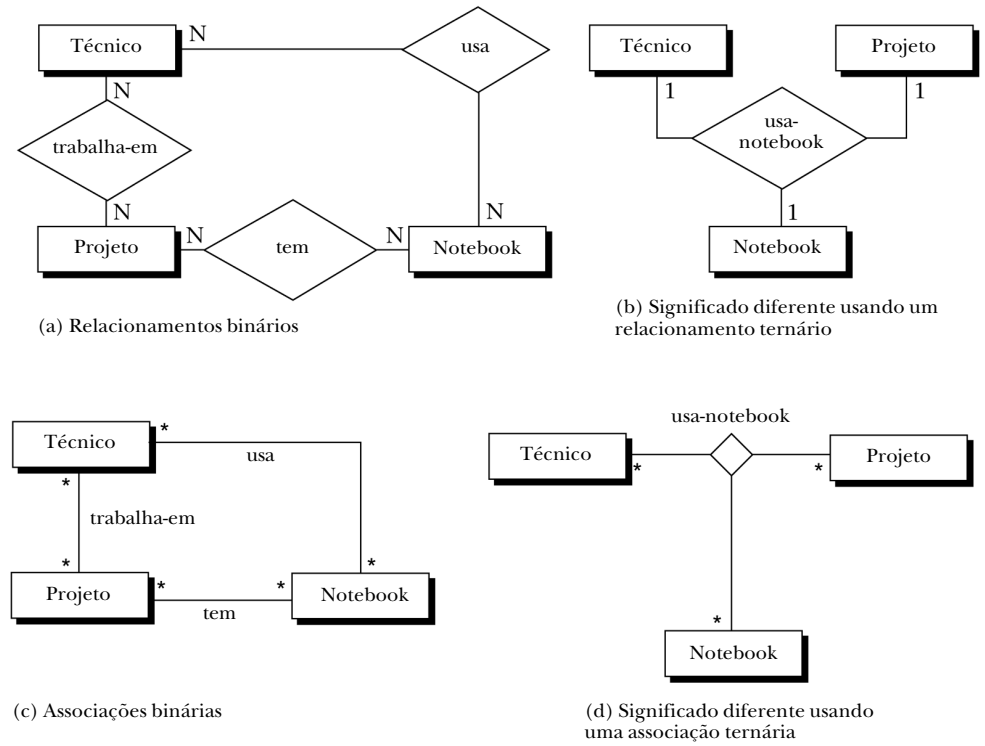


Figura 4.2 Relacionamentos ternários.

mentos binários só se aplicam às associações entre duas entidades. O projeto orientado a objeto oferece uma maneira comprovadamente melhor de modelar essa situação [Muller, 1999].

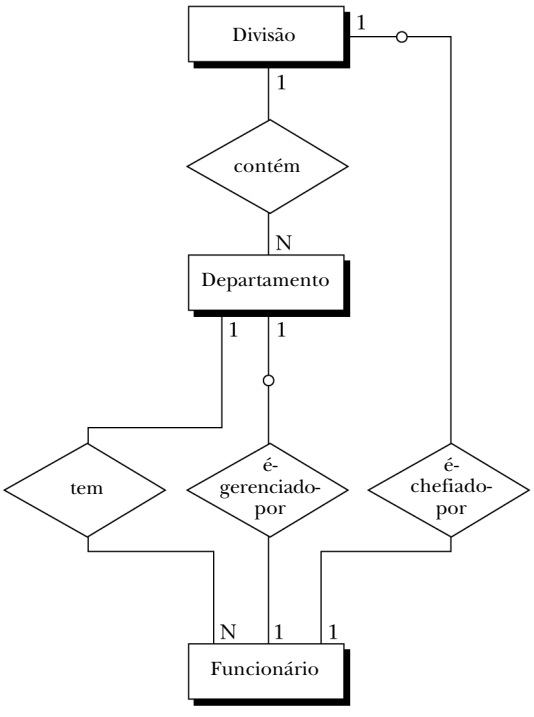
4.3.4 Exemplo de modelagem de dados: Banco de dados de pessoas e projetos da empresa

Modelagem ER de visões individuais com base nos requisitos

Vamos supor que se deseja montar um banco de dados completo de uma grande empresa de engenharia, que registra todo o pessoal de tempo integral, suas habilidades e projetos designados, os departamentos (e divisões) em que trabalham, as associações de profissionais de engenharia a que pertencem e os computadores desktop dos engenheiros alocados. Durante o processo de coleta de requisitos — ou seja, durante a entrevista dos usuários finais — obtivemos três visões do banco de dados.

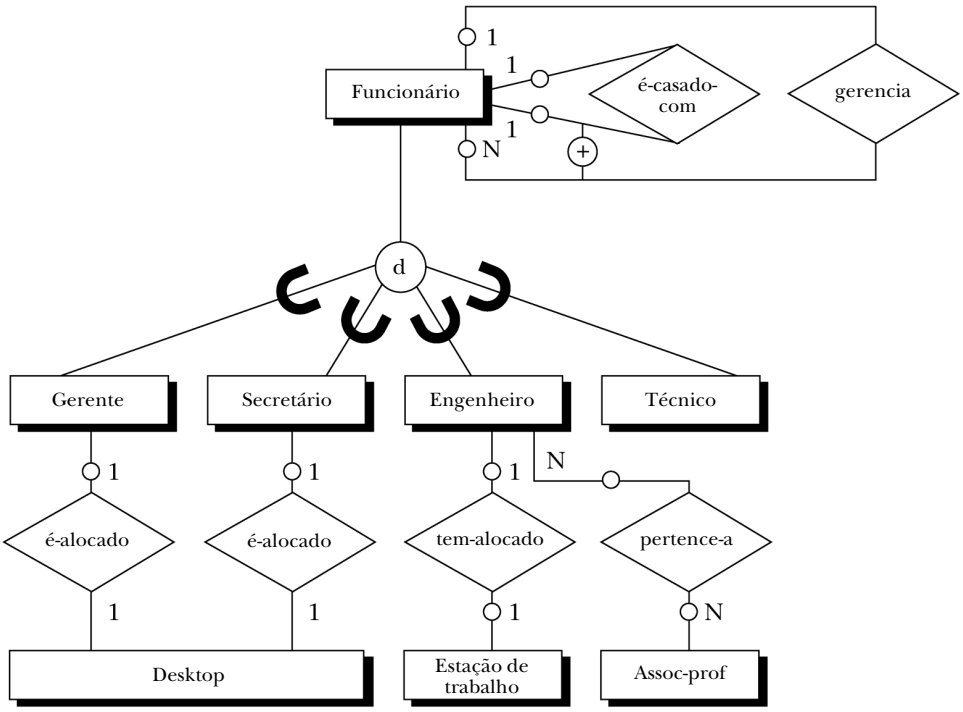
A primeira visão, uma visão gerencial, define cada funcionário como trabalhando em um único departamento e define uma divisão como a unidade básica na empresa, que consiste em vários departamentos. Cada divisão e departamento possui um gerente, e queremos manter a trilha de cada gerente. O modelo ER para essa visão aparece na Figura 4.3a.

A segunda visão define cada funcionário como tendo um cargo: engenheiro, técnico, secretário, gerente e assim por diante. Os engenheiros normalmente pertencem a associações profissionais e podem receber uma estação de trabalho (ou computador) de engenharia. Secretárias e gerentes recebem um computador desktop. Reservas de desktops e estações de trabalho são mantidas para atender novos funcionários e para empréstimos enquanto o computador de um funcionário estiver sendo consertado. Um funcionário pode estar casado com outro, e queremos registrar esses relacionamentos para evitar que um funcionário seja chefiado por seu cônjuge. Essa visão é ilustrada na Figura 4.3b.

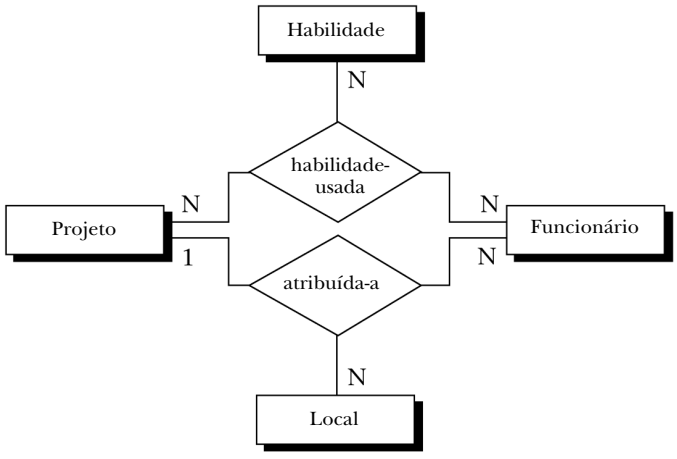


(a) Visão gerencial

Figura 4.3 Exemplo de modelagem de dados.



(b) Visão do funcionário



(c) Visão de atribuição de funcionário

Figura 4.3 *(continuação)*

A terceira visão, mostrada na Figura 4.3c, envolve a atribuição de funcionários, principalmente engenheiros e técnicos, a projetos. Os funcionários podem trabalhar em vários projetos ao mesmo tempo, e cada projeto pode ser sediado em diversos locais (cidades). Todavia, cada funcionário em determinado local trabalha em apenas um projeto nesse local. As habilidades do funcionário podem ser selecionadas individualmente para determinado projeto, mas nenhum indivíduo tem monopólio sobre habilidades, projetos ou locais.

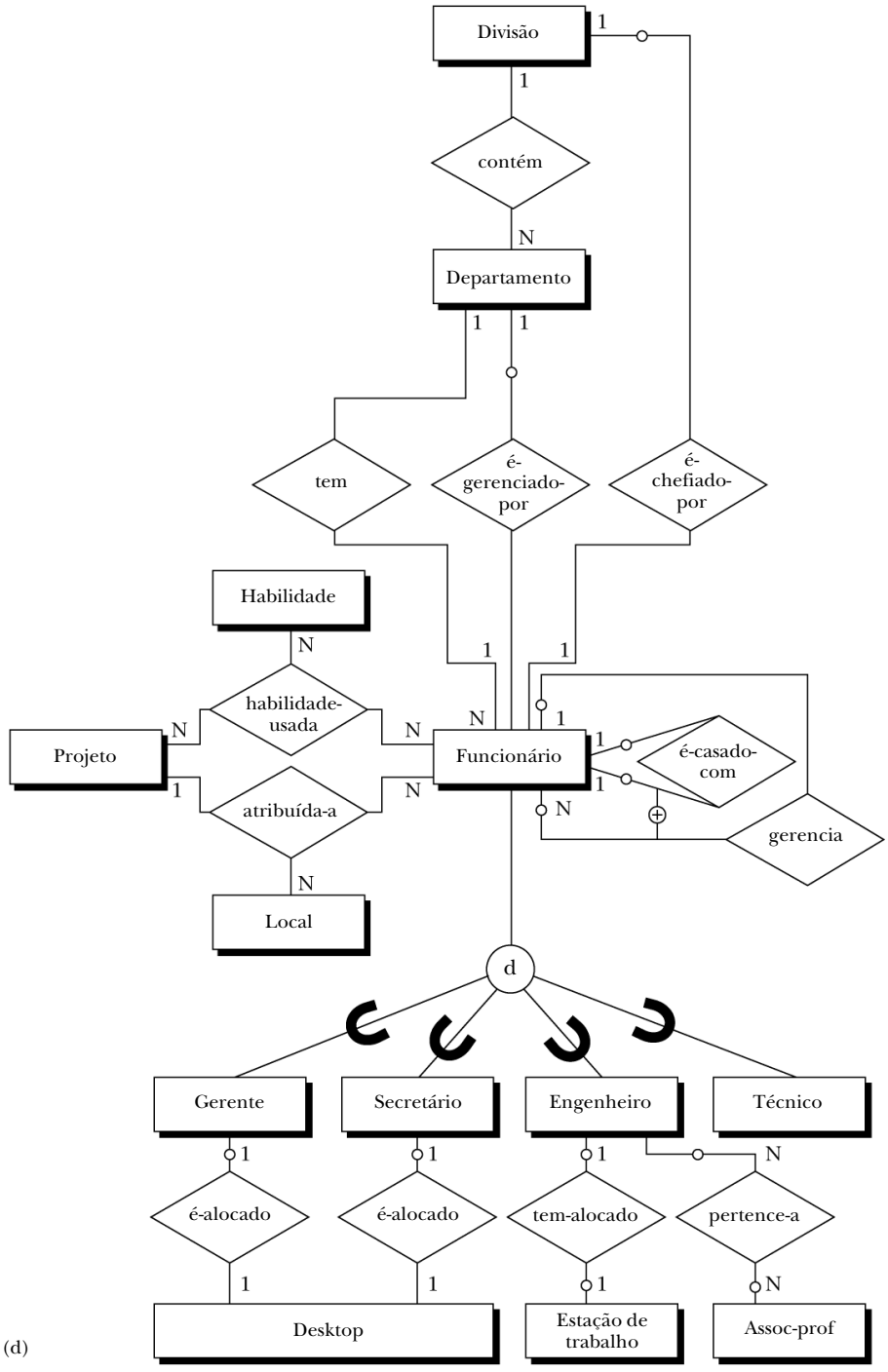
Esquema ER global

Uma simples integração das três visões sobre a definição da entidade Funcionário resulta no esquema (diagrama) ER global da Figura 4.3d, que se torna a base para o desenvolvimento das tabelas normalizadas. Cada relacionamento no esquema global é baseado em uma afirmação verificável sobre os dados reais da empresa, e a análise dessas afirmações leva à transformação desses construtores da ER em tabelas SQL candidatas, como mostra o Capítulo 5.

Observe que as visões equivalentes e a integração poderiam ter sido feitas com um modelo conceitual UML sobre a classe Funcionário. Entretanto, usaremos o modelo ER nos exemplos no restante deste capítulo.

O diagrama mostra exemplos de relacionamentos binários, ternários e binários recursivos; participação opcional e obrigatória nos relacionamentos; e generalização com a restrição de disjunção. Os relacionamentos ternários "habilidade-usada" e "atribuída-a" são necessários, pois os relacionamentos binários não podem ser usados como tendo noções equivalentes. Por exemplo, um funcionário e um local determinam exatamente um projeto (uma dependência funcional). No caso de "habilidade-usada", o uso seletivo de habilidades aos projetos não pode ser representado com relacionamentos binários (ver Seção 6.5).

O uso da participação opcional, por exemplo, entre Funcionário e Divisão, ou entre Funcionário e Departamento, é derivado do nosso conhecimento geral de que a maioria dos funcionários não será gerentes de qualquer divisão ou departamento. Em outro exemplo de participação opcional, mostramos que a alocação de uma estação de trabalho a um gerente pode nem sempre ocorrer, e nem todos os desktops ou estações de trabalho necessariamente serão alocados a alguém o tempo todo. Em geral, todos os relacionamentos, restrições de participação opcional e construtores de generalização precisam ser validados junto ao usuário final antes que o modelo ER seja transformado em tabelas SQL.



(d)

Figura 4.3 *(continuação)*

Resumindo, a aplicação do modelo ER ao projeto de banco de dados relacional oferece os seguintes benefícios:

- O uso da abordagem ER permite que o foco das discussões junto aos usuários sejam os relacionamentos importantes entre as entidades. Algumas aplicações são caracterizadas por casos especiais que afetam uma pequena quantidade de instâncias, e uma consideração extensa dessas instâncias pode distrair a atenção para os relacionamentos básicos.
- Uma sintaxe diagramática transmite muitas informações em um formato compacto, prontamente inteligível.
- As extensões ao modelo ER original, como as classes associativas opcionais e obrigatórias, são importantes em muitos relacionamentos. A generalização permite que as entidades sejam agrupadas segundo um papel funcional ou que sejam vistas como subtipos separados quando outras restrições forem impostas.
- Um conjunto completo de regras transforma os construtores ER em tabelas SQL geralmente normalizadas, as quais atendem facilmente aos requisitos do mundo real.

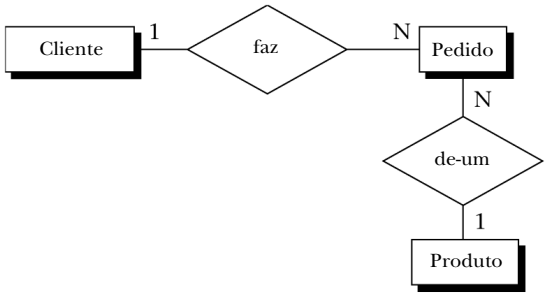
4.4 Integração de visão

Uma parte crítica do processo de projeto de banco de dados é a etapa II(b), a integração de diferentes visões do usuário em um esquema global unificado e não-redundante. As visões individuais dos usuários finais são representadas por modelos de dados conceituais; o esquema conceitual integrado resulta da análise adequada dessas visões dos usuários finais para resolver todas as diferenças de perspectivas e terminologias. A experiência tem mostrado que quase todas as situações podem ser resolvidas adequadamente por meio das técnicas de integração.

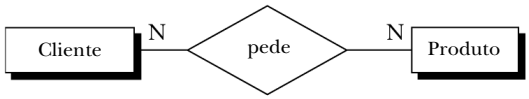
A diversidade de esquemas ocorre quando diferentes usuários ou grupos de usuários desenvolvem suas próprias perspectivas do mundo ou, pelo menos, da empresa a ser representada no banco de dados. Por exemplo, a divisão de marketing costuma tratar o produto completo como uma unidade básica de vendas, mas a divisão de engenharia pode se concentrar nas partes individuais do produto completo. Em um outro caso, um usuário poderá ver um projeto em termos de seus objetivos e de seu progresso para atingir seus objetivos conforme o tempo passa, mas outro usuário pode ver esse mesmo projeto em termos dos recursos de que precisa e do pessoal envolvido. Essas

diferenças fazem com que os modelos conceituais pareçam ter relacionamentos e terminologia incompatíveis. Essas diferenças tornam-se visíveis nos modelos de dados conceituais como diferenças nos níveis de abstração, conectividade de relacionamentos (um-para-muitos, muitos-para-muitos e assim por diante), ou como um mesmo conceito sendo modelado como uma entidade, atributo ou relacionamento, dependendo do ponto de vista do usuário.

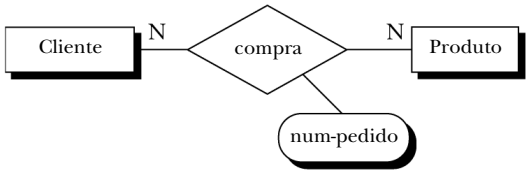
Como exemplo desse último caso, a Figura 4.4 mostra três perspectivas diferentes de uma mesma situação real — a preparação de um pedido de determinado produto. O resultado é uma série de esquemas. O primeiro esquema (Figura 4.4a) representa Cliente, Pedido e Produto como entidades, e "faz" e "de-um" como relacionamentos. O segundo esquema (Figura 4.4b), porém, define "pede" como um relacionamento entre Cliente e Produto e



(a) O conceito de pedido como uma entidade



(b) O conceito de pedido como um relacionamento



(c) O conceito de pedido como um atributo

Figura 4.4 Esquemas: preparação de um pedido.

omite completamente o Pedido como uma entidade. Finalmente, no terceiro caso (Figura 4.4c), o relacionamento "pede" foi substituído por outro relacionamento, "compra"; "num-pedido", o identificador (chave) de um pedido, é designado como um atributo do relacionamento "compra". Em outras palavras, o conceito do pedido foi representado de diversas formas: como uma entidade, um relacionamento e um atributo, dependendo do ponto de vista.

Existem quatro etapas básicas necessárias para a integração do esquema conceitual:

1. Análise de pré-integração
2. Comparação de esquemas
3. Adequação de esquemas
4. Unificação e reestruturação de esquemas

4.4.1 Análise de pré-integração

A primeira etapa, a análise de pré-integração, envolve a escolha de uma estratégia de integração. Normalmente, a escolha é entre uma técnica binária com dois esquemas unificados por vez e uma técnica *n*-ária com *n* esquemas unificados por vez, onde *n* está entre 2 e o número total de esquemas desenvolvidos no projeto conceitual. A técnica binária é atraente porque cada unificação envolve uma pequena quantidade de construtores do modelo de dados e é mais fácil de conceituar. A técnica *n*-ária pode exigir apenas uma grande unificação, mas a quantidade de construtores pode ser tão grande que é humanamente impossível organizar as transformações apropriadamente.

4.4.2 Comparação de esquemas

Na segunda etapa, a comparação de esquemas, o projetista examina como as entidades se correspondem e detecta conflitos que surgem da diversidade de esquemas - ou seja, de grupos de usuários que adotam diferentes pontos de vista em seus respectivos esquemas. Conflitos de nomes ocorrem com o uso de sinônimos e homônimos. Sinônimos ocorrem quando diferentes nomes são dados para o mesmo conceito e podem ser detectados analisando-se o dicionário de dados, se houver algum estabelecido para o banco de dados. Homônimos ocorrem quando o mesmo nome é usado para diferentes conceitos. Estes só podem ser detectados analisando-se os diferentes esquemas e procurando-se nomes comuns.

Conflitos estruturais ocorrem na própria estrutura do esquema. Conflitos de tipo envolvem o uso de diferentes construtores para modelar o mesmo conceito. Na Figura 4.4, por exemplo, uma entidade, um relacionamento ou um atributo podem ser usados para modelar o conceito de pedido em um banco de dados comercial. Conflitos de dependência acontecem quando os usuários especificam diferentes níveis de conectividade (um-para-muitos etc.) para conceitos semelhantes ou mesmo iguais. Um modo de resolver esses conflitos poderia ser usar apenas a conectividade mais genérica - por exemplo, muitos-para-muitos. Se isso não for semanticamente correto, mude os nomes das entidades de modo que cada tipo de conectividade tenha um nome de entidade diferente. Conflitos de chave ocorrem quando diferentes chaves são atribuídas à mesma entidade em diferentes visões. Por exemplo, um conflito de chave ocorre se o nome completo de um funcionário, o número de identidade do funcionário e o número do CPF forem atribuídos como chaves.

4.4.3 Adequação de esquemas

A solução de conflitos normalmente exige a interação entre usuário e projetista. O objetivo básico da terceira etapa é alinhar ou adequar os esquemas para torná-los compatíveis para integração. As entidades, bem como os atributos chaves podem precisar ser rebatizados. A conversão pode ser necessária para que apenas um dos conceitos modelados como entidades, atributos ou relacionamentos seja considerado. Os relacionamentos de mesmo grau, papéis e restrições de conectividade são fáceis de unificar. Aqueles com características diferentes são mais difíceis e, em alguns casos, impossíveis de unificar. Além disso, os relacionamentos que não são consistentes - por exemplo, um relacionamento que usa generalização em um local e o OR exclusivo em outro — precisam ser resolvidos. Finalmente, algumas certezas podem precisar ser modificadas para que as restrições de integridade permaneçam consistentes.

As técnicas usadas para a integração de visão incluem abstrações, como a generalização e a agregação para criar novos supertipos e subtipos, ou mesmo a introdução de novos relacionamentos. Como exemplo, a generalização de Indivíduo por diferentes valores do atributo descritor “cargo” poderia representar a consolidação das duas visões do banco de dados — uma baseada em um indivíduo como unidade básica de pessoa na organização e outra, baseada na classificação dos indivíduos por cargos e características especiais dentro dessas classificações.

4.4.4 Unificação e reestruturação de esquemas

A quarta etapa consiste na unificação e reestruturação de esquemas. Essa etapa é controlada pelos objetivos de completude, minimalidade e inteligibilidade. A completude exige que todos os conceitos correspondentes apareçam semanticamente intactos no esquema global. A minimalidade exige que o projetista remova todos os conceitos redundantes do esquema global. Exemplos de conceitos redundantes são entidades superpostas e relacionamentos verdadeiramente redundantes do ponto de vista semântico; por exemplo, Veículo-Terrestre e Automóvel poderiam ser duas entidades superpostas. Um relacionamento redundante poderia ocorrer entre Instrutor e Aluno. Os relacionamentos "pesquisa-direta" e "conselho" podem ou não representar a mesma atividade ou relacionamento, de modo que é preciso investigar melhor para determinar se são redundantes ou não. Inteligibilidade exige que o esquema global faça sentido para o usuário.

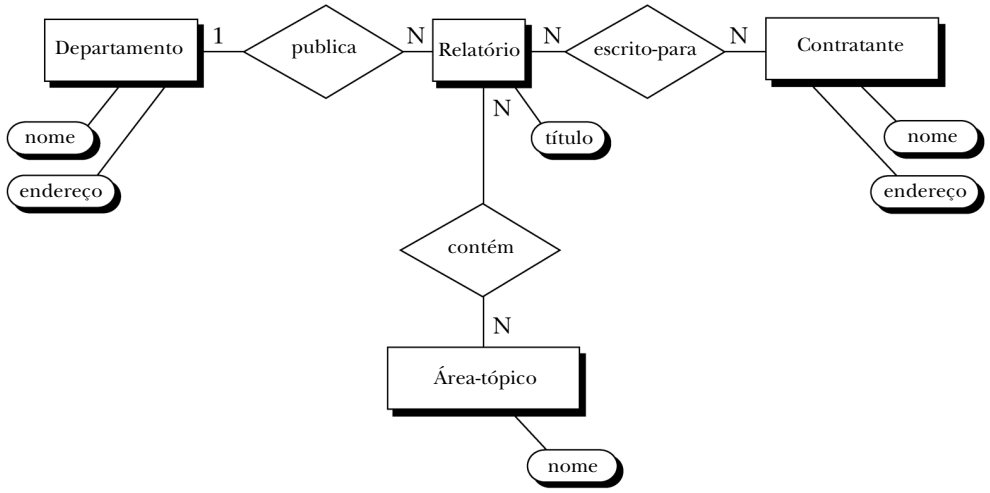
Os esquemas componentes são inicialmente unificados pela superposição dos mesmos conceitos e depois pela reestruturação do esquema integrado resultante buscando a inteligibilidade. Por exemplo, se uma combinação de supertipo/subtipo for definida como resultado da operação de unificação, as propriedades do subtipo poderão ser descartadas do esquema, pois são automaticamente fornecidas pela entidade do supertipo.

4.4.5 Exemplo de integração de visão

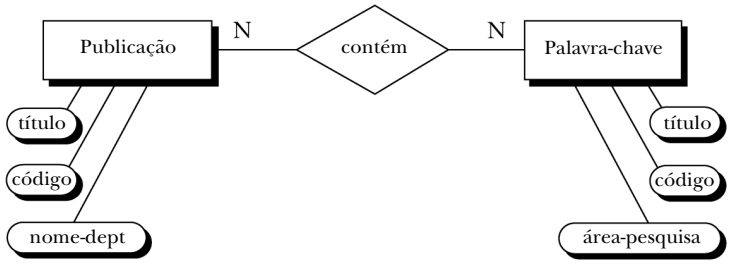
Vamos examinar duas visões diferentes dos dados superpostos. As visões são baseadas em duas entrevistas separadas de usuários finais. Adaptamos o exemplo interessante citado por Batini, Lenzerini e Navathe [1986] a uma situação hipotética relacionada ao nosso exemplo.

Na Figura 4.5a, temos uma visão que focaliza relatórios e inclui dados sobre departamentos que publicam os relatórios, áreas tópicos de relatórios e contratantes para os quais os relatórios são escritos. A Figura 4.5b mostra outra visão, com publicações como foco central e palavras-chave sobre publicação como dados secundários. Nosso objetivo é encontrar maneiras significativas de integrar as duas visões e manter a completude, minimalidade e inteligibilidade.

Primeiro, procuramos sinônimos e homônimos, principalmente entre as entidades. Observe que um sinônimo existe entre as entidades Área-tópico no esquema 1 e a Palavra-chave no esquema 2, embora os atributos não correspondam. Contudo, descobrimos que os atributos são compatíveis e



(a) Esquema original 1, focalizado em relatórios



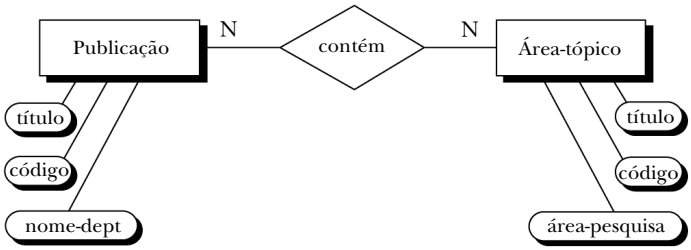
(b) Esquema original 2, focalizado em publicações

Figura 4.5 Integração de visão: encontrar maneiras significativas de integrar.

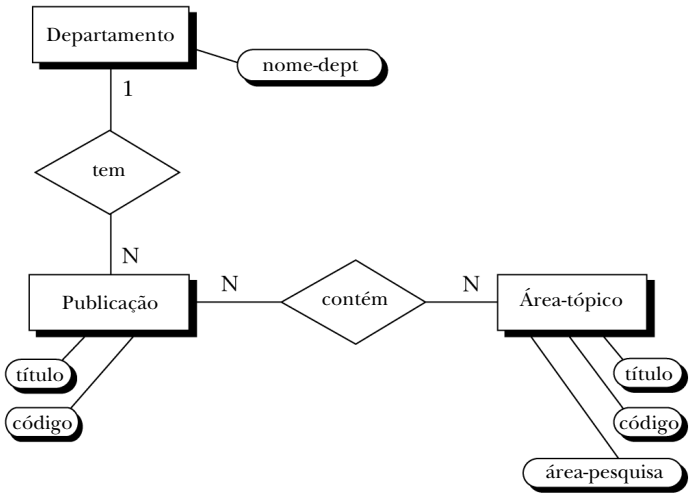
podem ser consolidados. Isso aparece na Figura 4.6a, que apresenta um esquema revisado, o esquema 2.1. No esquema 2.1, Palavra-chave foi substituída por Área-tópico.

Em seguida, procuramos conflitos estruturais entre os esquemas. Um conflito de tipo existe entre a entidade Departamento no esquema 1 e o atributo "nome-dept" no esquema 2.1. O conflito é resolvido mantendo-se o tipo de entidade mais forte, Departamento, e movendo-se o tipo de atributo "nome-dept" de Publicação no esquema 2 para a nova entidade, Departamento, no esquema 2.2 (ver Figura 4.6b).

Neste ponto, temos semelhança suficiente entre esquemas para tentar uma unificação. Nos esquemas 1 e 2.2, temos dois conjuntos de entidades comuns,



(a) Esquema 2.1, em que Palavra-chave foi trocada para Área-tópico



(b) Esquema 2.2, em que o atributo nome-dept foi trocado por um atributo e uma entidade.

Figura 4.6 Integração de visão: conflito de tipo.

Departamento e Área-tópico. Outras entidades não se sobrepõem e precisam aparecer intactas no esquema superposto, ou unificado. O esquema unificado, o esquema 3, aparece na Figura 4.7a. Como as entidades comuns são verdadeiramente equivalentes, não existem efeitos colaterais ruins da unificação devido a relacionamentos existentes envolvendo essas entidades em um esquema e não no outro. (Um relacionamento desse tipo que permanece intacto existe no esquema 1, entre Área-tópico e Relatório, por exemplo.) Se a verdadeira equivalência não puder ser estabelecida, a unificação pode não ser possível na forma existente.

Na Figura 4.7, existe alguma redundância entre Publicação e Relatório nos termos dos relacionamentos com Departamento e Área-tópico. Essa re-

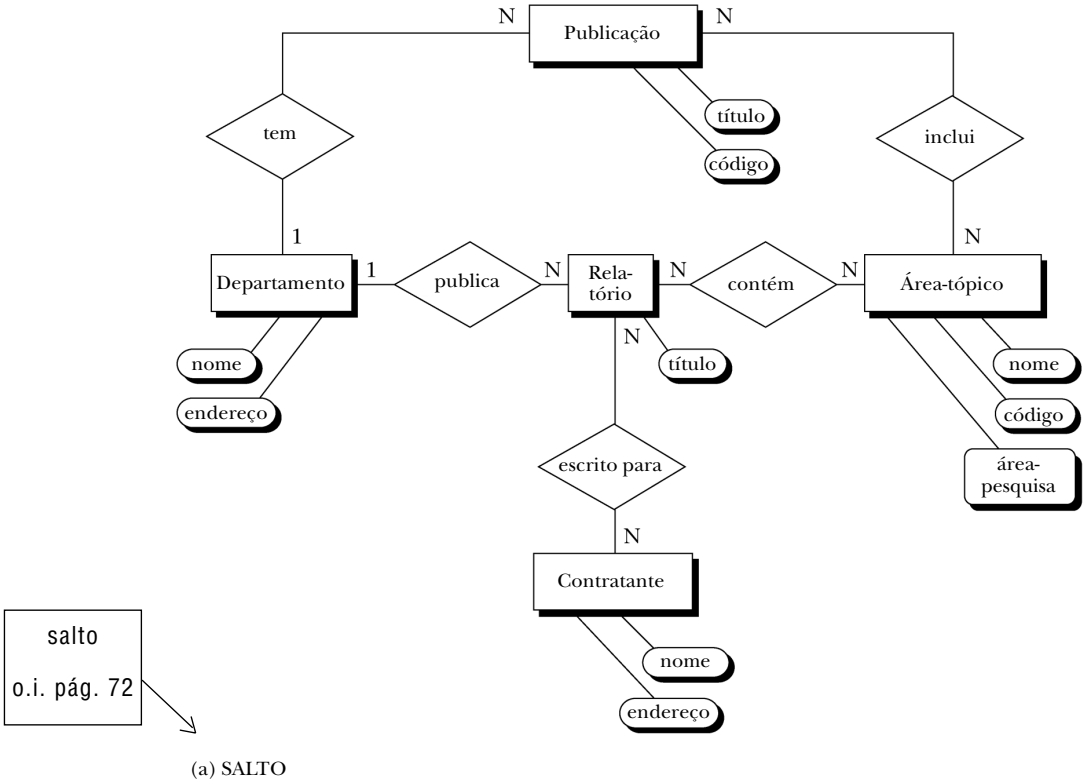
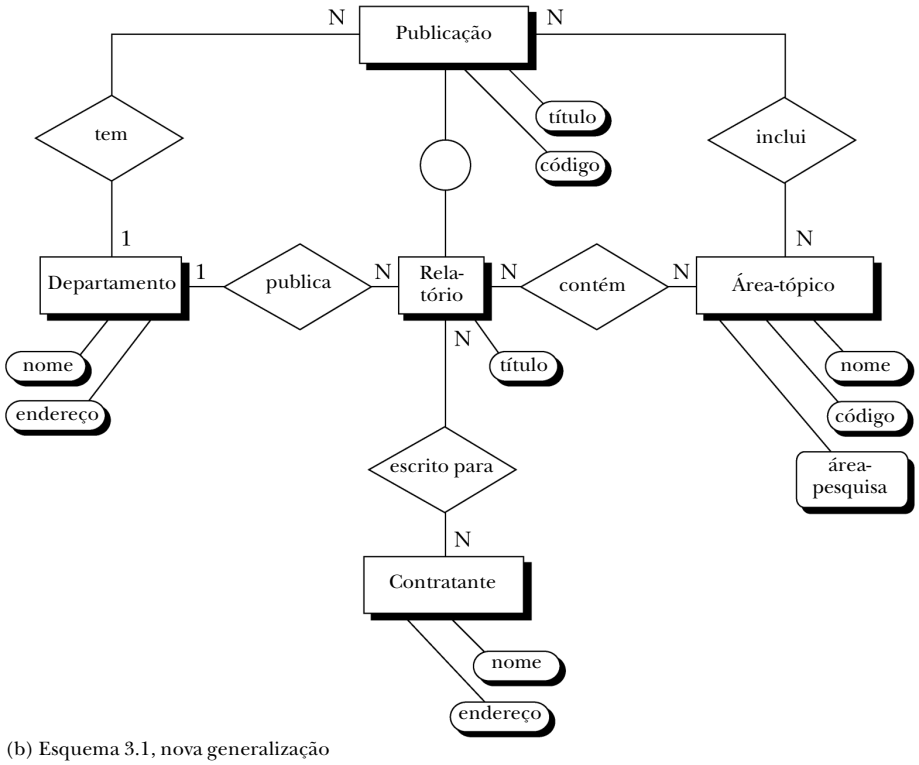


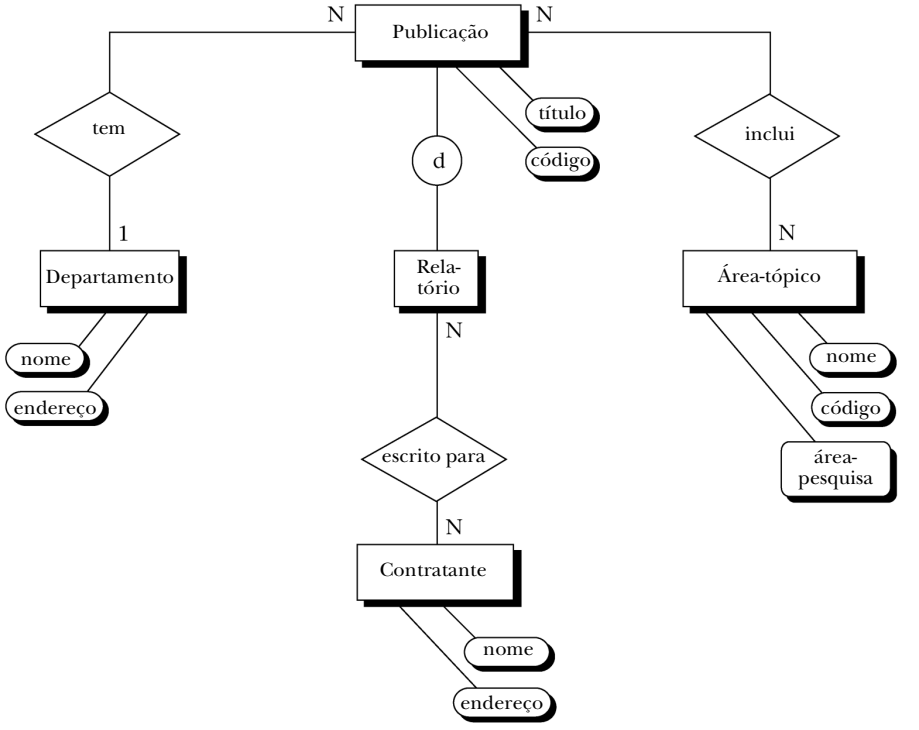
Figura 4.7 Integração de visão: o esquema de unificação.

dundância poderá ser eliminada se houver um relacionamento de supertipo/subtipo entre Publicação e Relatório, que de fato ocorre nesse caso porque Publicação é uma generalização de Relatório. No esquema 3.1 (Figura 4.7b), vemos a introdução dessa generalização de Relatório para Publicação. Depois, no esquema 3.2 (Figura 4.7c), vemos que os relacionamentos redundantes entre Relatório e Departamento e Área-tópico foram descartados. O atributo "título" foi eliminado como um atributo de Relatório na Figura 4.7c, pois "título" já aparece como um atributo de Publicação em um nível de abstração mais alto; "título" é herdado pelo subtipo Relatório.

O esquema final, na Figura 4.7c, expressa completude, pois todos os conceitos originais (relacionamento, publicação, área de assunto, departamento e contratante) são mantidos intactos. Ele expressa minimalidade devido à transformação de "nome-dept" de atributo, no esquema 1, para entidade e atribu-



(b) Esquema 3.1, nova generalização



(c) Esquema 3.2, eliminação de relacionamentos redundantes

Figura 4.7 *(continuação)*

to, no esquema 2.2, e a unificação entre o esquema 1 e o esquema 2.2 para formar o esquema 3, e por causa da eliminação de "título" como um atributo de Relatório e dos relacionamentos de Relatório com Área-tópico e Departamento. Finalmente, ele expressa inteligibilidade porque o esquema final realmente tem mais significado do que os esquemas originais individuais.

A integração de visão é um processo de refinamento e reavaliação contínua. Também deve ser observado que a minimalidade nem sempre é o modo mais eficiente de proceder. Por exemplo, se a eliminação dos relacionamentos redundantes "publica" e/ou "contém" do esquema 3.1 para 3.2 fizer com que o tempo necessário para realizar certas consultas seja excessivamente longo, poderá ser melhor (do ponto de vista do desempenho) mantê-los. Essa decisão poderia ser tomada durante a análise das transações sobre o banco de dados ou durante a fase de testes do banco de dados totalmente implementado.

4.5 Agrupamento de entidades* do modelo ER

Esta seção apresenta o conceito de agrupamento de entidades, que separa o esquema ER a tal ponto que o esquema inteiro possa aparecer em uma única folha de papel ou em uma única tela de computador. Isso traz boas conseqüências para o usuário final e para o projetista de banco de dados, pois permite desenvolver um conhecimento mútuo do conteúdo do banco de dados e documentar formalmente o modelo conceitual.

Um grupo (ou *cluster*) de entidades é o resultado da operação de agrupamento sobre uma coleção de entidades e relacionamentos. O agrupamento de entidades é potencialmente útil para o projeto de grandes bancos de dados. Quando a escala de um banco de dados ou estrutura de informação é grande e inclui um grande número de interconexões entre seus diferentes componentes, pode ser muito difícil entender a semântica dessa estrutura e gerenciá-la, especialmente pelos usuários finais e gerentes. Em um diagrama ER com 1.000 entidades, a estrutura geral provavelmente não será muito clara, até mesmo para um analista de banco de dados bem treinado. O agrupa-

---

* *Nota dos Revisores Técnicos:* o termo Entity Clustering foi traduzido como agrupamento de entidade. Alguns autores, na área de banco de dados, denominam o agrupamento de entidade como módulo de entidades. Um exemplo disso pode ser encontrado em Ferreira J.E.; Finger, M.: *Controle de concorrência e distribuição de dados*: a teoria clássica, suas limitações e extensões modernas, Coleção de textos especialmente preparada para a Escola de Computação, 12a, São Paulo, 2000; ou acesso *http://www.ime.usp.br/~jef/ec2000.ps*

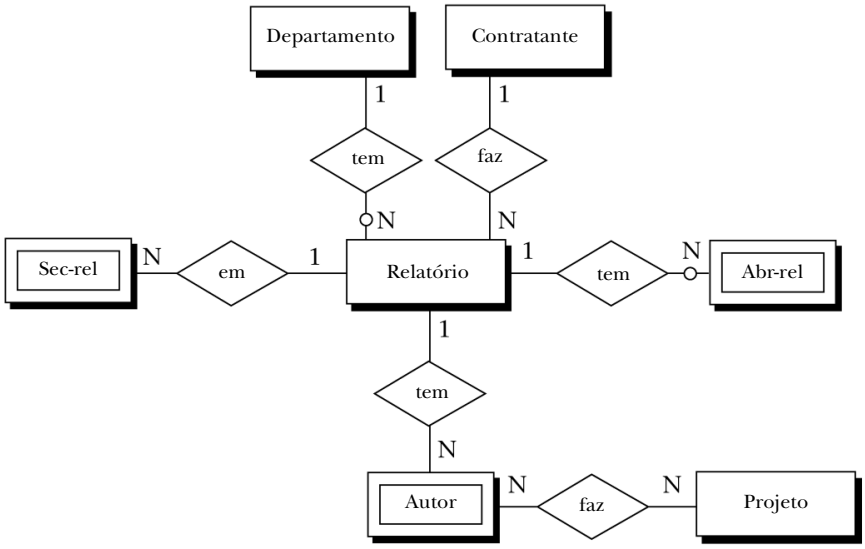
mento, portanto, é importante porque fornece um método de organização para um esquema de banco de dados conceitual em camadas de abstração e fornece visões diferentes dos vários usuários finais.

4.5.1 Conceitos de agrupamento

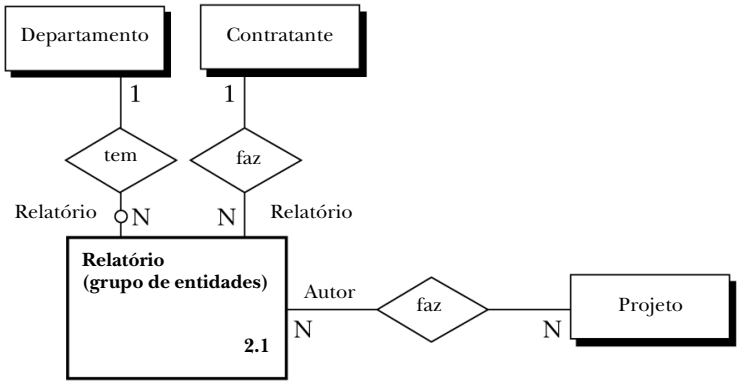
Deve-se pensar no agrupamento como uma operação que combina entidades e seus relacionamentos para formar uma construção de nível superior. O resultado de uma operação de agrupamento sobre entidades simples é chamado de *grupo de entidades*. Uma operação de agrupamento sobre grupos de entidades, ou sobre combinações de entidades elementares e grupos de entidades, resulta em um grupo de entidades de nível superior. O grupo de entidades de nível mais alto, representando o esquema conceitual do banco de dados inteiro, é chamado de *grupo de entidades raiz*.

A Figura 4.8a ilustra o conceito de agrupamento de entidades em um caso simples, em que entidades (elementares) Sec-rel (seção de relatório), Abr-rel (abreviação de relatório) e Autor estão naturalmente vinculadas à (dominadas pela) entidade Relatório; e as entidades Departamento, Contratante e Projeto não são dominadas. (Observe que, para evitar detalhes desnecessários, não incluímos os atributos das entidades nos diagramas.) Na Figura 4.8b, a caixa de borda escura em torno da entidade Relatório e as entidades que ela domina definem o grupo de entidades Relatório. A caixa de borda escura é chamada de caixa EC (*Entity Cluster*), para dar a idéia de ser um grupo de entidades. Em geral, o nome do grupo de entidades não precisa ser o mesmo que o nome de qualquer entidade interna; porém, quando existe uma única entidade dominante, os nomes normalmente são iguais. O número da caixa EC no canto inferior direito é um número de nível de agrupamento usado para registrar a seqüência em que o agrupamento é realizado. O número 2.1 significa que o grupo de entidades Relatório é o primeiro grupo de entidades no nível 2. Observe que todas as entidades originais são consideradas no nível 1.

A abstração de nível superior, o agrupamento, precisa manter os mesmos relacionamentos entre as entidades que estão dentro e fora do grupo de entidades, como ocorre entre as mesmas entidades no diagrama de nível inferior. Assim, os nomes de entidade pertencentes ao agrupamento devem aparecer do lado de fora da caixa EC, no caminho do seu relacionamento direto com entidades devidamente relacionadas fora da caixa, mantendo interfaces (relacionamentos) consistentes, como mostra a Figura 4.8b. Para simplificar,



(a) Modelo ER antes do agrupamento



(b) Modelo ER depois do agrupamento

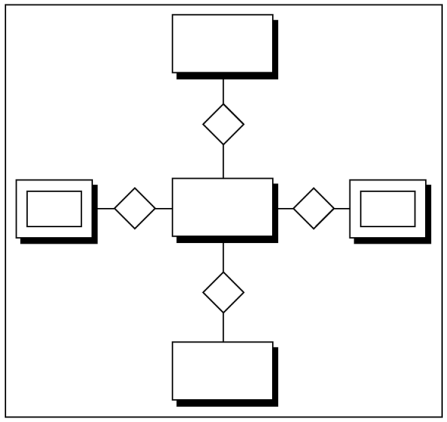
Figura 4.8 Conceitos de agrupamento de entidades.

modificamos essa regra ligeiramente: se o relacionamento for entre uma entidade externa e a entidade interna dominante (com a qual o grupo de entidades é nomeado), o nome do grupo de entidades não precisa ser repetido fora da caixa EC. Assim, na Figura 4.8b, poderíamos descartar o nome Relatório nos dois locais em que ele aparece fora da caixa Relatório, mas precisamos reter o nome Autor, que não é o nome do grupo de entidades.

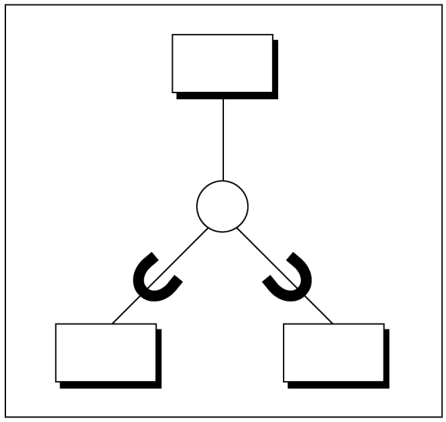
4.5.2 Operações de agrupamento

As operações de agrupamento são os componentes fundamentais da técnica de agrupamento de entidades. Elas definem quais coleções de entidades e relacionamentos comporão os objetos de nível superior, os grupos de entidades. As operações são heurísticas por natureza e incluem (ver Figura 4.9):

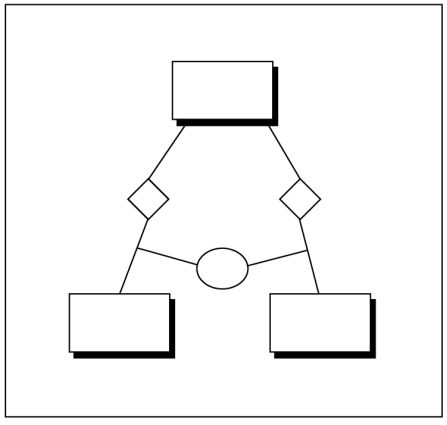
- Agrupamento de dominância
- Agrupamento de abstração
- Agrupamento de restrição
- Agrupamento de relacionamento



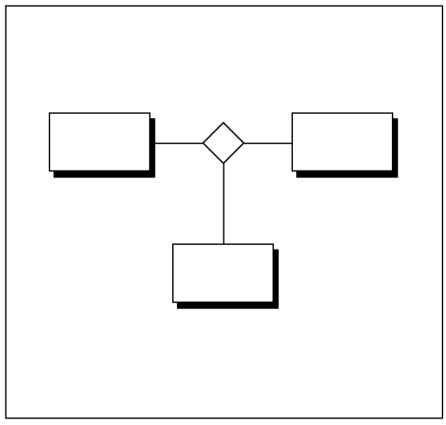
(a) Agrupamento de dominância



(b) Agrupamento de abstração



(c) Agrupamento de restrição



(d) Agrupamento de relacionamento

Figura 4.9 Operações de agrupamento.

Essas operações de agrupamento podem ser aplicadas recursivamente ou usadas em diversas combinações para produzir grupos de entidade de nível superior, ou seja, grupos em qualquer nível de abstração. Uma entidade ou grupo de entidades pode ser um objeto que está sujeito a combinações com outros objetos para formar o próximo nível superior. Ou seja, os grupos de entidades possuem as propriedades das entidades e podem ter relacionamentos com quaisquer outros objetos em qualquer nível igual ou inferior. Os relacionamentos originais entre as entidades são preservados após todas as operações de agrupamento, conforme ilustrado na Figura 4.8.

Os objetos ou entidades dominantes normalmente se tornam óbvios a partir do diagrama ER ou das definições de relacionamentos. Cada objeto dominante é agrupado com todos os seus objetos não dominantes relacionados para formar um grupo. Entidades fracas podem ser conectadas a uma entidade para criar um grupo. Objetos de dados multinível, usando abstrações como generalização e agregação, podem ser agrupados em um grupo de entidades. O supertipo ou nome da entidade agregada é usado como nome do grupo de entidades. Objetos relacionados às restrições, que estendem o modelo ER para incorporar restrições de integridade, como o OR exclusivo, podem ser agrupados em um grupo de entidades. Além disso, relacionamentos de grau ternário ou maior potencialmente podem ser agrupados em um grupo de entidades. O grupo representa o relacionamento como um todo.

4.5.3 Técnica de agrupamento

As operações de agrupamento e sua ordem ou precedência determinam as atividades individuais necessárias para o agrupamento. Agora, podemos aprender como criar um grupo de entidades raiz a partir das entidades e relacionamentos elementares definidos no processo de modelagem ER. Essa técnica considera que uma análise *top-down* foi realizada como parte da análise de requisitos do banco de dados e que a análise foi documentada para que as principais áreas funcionais e subáreas fossem identificadas. As áreas funcionais normalmente são definidas pelas unidades organizacionais importantes de uma empresa, atividades comerciais ou, possivelmente, por aplicações dominantes para processamento de informações. Como exemplo, lembre-se da Figura 4.3 (reconstruída na Figura 4.10), que pode ser imaginada como tendo três áreas funcionais principais: organização da empresa (divisão, departamento), gerenciamento de projetos (projeto, habilidade, local, funcionário) e dados do funcionário (funcionário, gerente, secretário, engenheiro, técni-

co, assoc-prof e desktop). Observe que as áreas funcionais podem se sobrepor. A Figura 4.10 usa um diagrama ER resultante da análise de requisitos do banco de dados para mostrar como o agrupamento envolve uma série de etapas *bottom-up* usando as operações de agrupamento básicas. A lista a seguir explica essas etapas.

1. *Defina pontos de agrupamento dentro de áreas funcionais.* Localize as entidades dominantes em uma área funcional por meio de relacionamentos naturais, relacionamentos *n*-ários locais, restrições de integridade, abstrações ou apenas o foco central de muitos relacionamentos simples. Se esses pontos de agrupamento não existirem dentro de uma área, considere um agrupamento funcional de uma área inteira.
2. *Forme grupos de entidades.* Use as operações básicas de agrupamento sobre entidades elementares e seus relacionamentos para formar objetos de nível superior, ou grupos de entidades. Como as entidades podem pertencer a vários grupos em potencial, precisamos ter um conjunto de prioridades para formar grupos de entidades. O conjunto de regras a seguir, listadas em ordem de prioridade, define o conjunto que mais provavelmente preservará a clareza do modelo conceitual:
   a. As entidades a serem colocadas em um grupo de entidades devem existir dentro da mesma área funcional; ou seja, o grupo de entidades como um todo deve ocorrer dentro dos limites de uma área funcional. Por exemplo, na Figura 4.10, o relacionamento entre Departamento e Funcionário não deve ser agrupado a menos que Funcionário seja incluído na área funcional de organização da empresa com Departamento e Divisão. Em outro exemplo, o relacionamento entre o supertipo Funcionário e seus subtipos poderia ser agrupado dentro da área funcional de dados do funcionário.
   b. Se um conflito na escolha entre dois ou mais grupos de entidades em potencial não puder ser resolvido (por exemplo, entre duas restrições de agrupamentos no mesmo nível de precedência), deixe esses grupos de entidades não-agrupados dentro de sua área funcional. Se essa área funcional permanecer repleta de escolhas não resolvidas, defina subáreas funcionais para agrupar as entidades não resolvidas, grupos de entidades e seus relacionamentos.
3. *Forme grupos de entidades de nível mais alto.* Aplique as operações de agrupamento recursivamente a qualquer combinação de entidades elementares e grupos de entidades para formar novos níveis de grupos de entidades

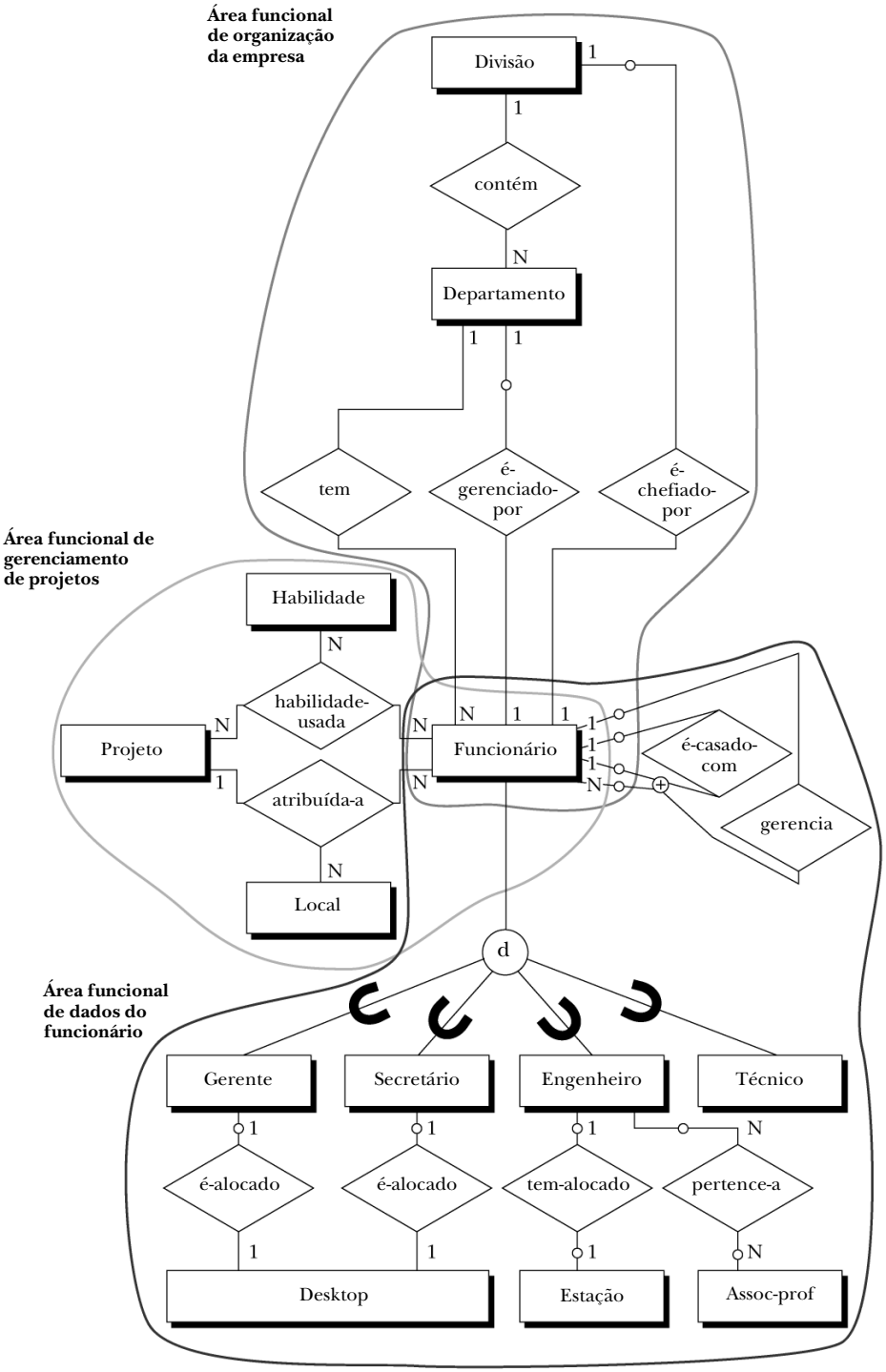


Figura 4.10 Diagrama ER: técnica de agrupamento.

(objetos de nível mais alto). Resolva os conflitos usando o mesmo conjunto de regras de prioridade dado na etapa 2. Continue com as operações de agrupamento até que todas as representações de entidade se ajustem em uma única página sem complexidade indevida. O grupo de entidades raiz é definido em seguida.

4. *Valide o diagrama de grupos.* Verifique a consistência das interfaces (relacionamentos) entre objetos em cada nível do diagrama. Verifique o significado de cada nível com os usuários finais.

O resultado de uma rodada de agrupamento aparece na Figura 4.11, em que cada um dos grupos aparece no nível 2.

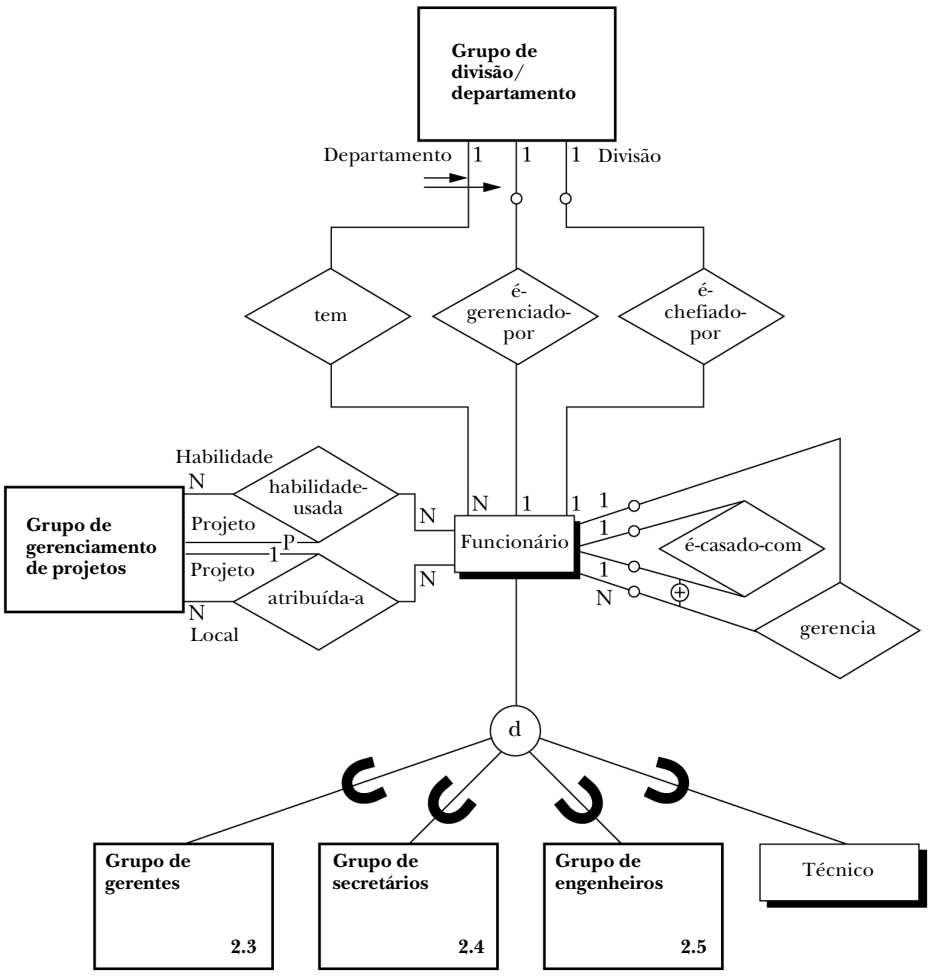


Figura 4.11 Resultados do agrupamento.

4.6 Resumo

A modelagem de dados conceitual, usando a abordagem ER ou UML, é particularmente útil nas primeiras etapas do ciclo de vida do banco de dados, que envolvem a análise de requisitos e o projeto lógico. Essas duas etapas costumam ser feitas simultaneamente, em especial quando os requisitos são determinados por entrevistas com os usuários finais e modelados em termos de relacionamentos de dados-para-dados e relacionamentos de processos-para-dados. A etapa de modelagem de dados conceitual (abordagem ER) envolve, inicialmente, a classificação de entidades e atributos, depois a identificação de hierarquias de generalização e outras abstrações e finalmente a definição de todos os relacionamentos entre entidades. Os relacionamentos podem ser binários (o mais comum), ternários e *n*-ários de nível mais alto. A modelagem de dados de requisitos individuais normalmente envolve a criação de uma visão diferente para cada um dos requisitos do usuário final. Depois, o projetista precisa integrar essas visões em um esquema global, para que o banco de dados completo seja representado como um todo integrado. Isso ajuda a eliminar a redundância desnecessária — essa eliminação é particularmente importante no projeto lógico. A redundância controlada pode ser criada mais tarde, no nível de projeto físico, para melhorar o desempenho do banco de dados. Finalmente, um grupo de entidades é um agrupamento de entidades e seus relacionamentos correspondentes em um objeto abstrato de nível superior. O agrupamento promove a simplicidade, vital para agilizar a compreensão do usuário final. No próximo capítulo, utilizaremos o esquema global produzido a partir da modelagem de dados conceitual e o transformaremos em tabelas SQL. O formato SQL é o produto final do projeto lógico, que ainda é independente de qualquer sistema de gerenciamento de banco de dados em particular.

4.7 Resumo da literatura

A modelagem de dados conceitual é definida em Tsichritzis e Lochovsky [1982], Brodie, Mylopoulos e Schmidt [1984], Nijssen e Halpin [1989], Batini, Ceri e Navathe [1992]. A discussão sobre o processo de coleta de dados de requisitos pode ser encontrada em Martin [1982], Teorey e Fry [1982] e Yao [1985]. A integração da visão progrediu de uma ferramenta de representação [Smith e Smith, 1977] para algoritmos heurísticos [Batin, Lenzerini e Navathe, 1986; Elmasri e Navathe, 2003]. Esses algoritmos normalmente são interativos,

permitindo que o projetista de banco de dados tome decisões com base em ações de integração alternativas sugeridas. Diversos modelos de agrupamento de entidade foram definidos para fornecer um alicerce útil para a técnica de agrupamento apresentada aqui [Feldman e Miller, 1986; Dittrich, Gotthard e Lockemann, 1986; Teorey *et al.*, 1989].

5 Transformando o modelo de dados conceitual em SQL

Este capítulo focaliza a etapa de ciclo de vida do banco de dados que é de particular interesse durante o projeto de bancos de dados relacionais: a transformação do modelo de dados conceitual em tabelas candidatas e sua definição na SQL [etapa II(c)]. Existe uma evolução natural dos modelos de dados ER e UML para um esquema relacional. A evolução é tão natural, na verdade, que se admite a afirmação de que essa modelagem de dados conceitual é uma etapa inicial efetiva no desenvolvimento do banco de dados relacional. Essa afirmação tem sido provada até certo ponto pela comercialização e uso das ferramentas genéricas de projeto de software que dão apoio não apenas à modelagem de dados conceitual, mas também à conversão automática desses modelos para definições de tabelas SQL e restrições de integridade específicas do fornecedor.

Neste capítulo, consideramos que as aplicações são transacionais, ou seja, OLTP (*Online Transaction Processing*). Observe que aplicações OLAP (*Online Analytical Processing*) é o assunto do Capítulo 8.

5.1 Regras de transformação e construtores SQL

Vamos primeiro examinar os construtores de modelagem ER e UML com detalhes, para ver como as regras sobre transformação do modelo de dados conceitual em tabelas SQL são definidas e aplicadas. Nosso exemplo é baseado nos esquemas conceituais de pessoas e projetos de uma empresa, ilustrados na Figura 4.3 (ver Capítulo 4).

As transformações básicas podem ser descritas em termos dos três tipos de tabelas que elas produzem:

- *Tabela SQL com o mesmo conteúdo de informação da entidade original da qual é derivada.* Essa transformação sempre ocorre para entidades com relacionamentos binários (associações) que são muitos-para-muitos, um-para-muitos no lado "um" (pai), ou um-para-um em qualquer lado; entidades com relacionamentos recursivos binários que são muitos-para-muitos; e entidades com qualquer relacionamento ternário ou de maior grau ou uma hierarquia de generalização.
- *Tabela SQL com a inclusão da chave estrangeira da entidade pai.* Essa transformação sempre ocorre para entidades com relacionamentos binários que são um-para-muitos para a entidade no lado "muitos" (filho), para relacionamentos um-para-um para uma das entidades, e para cada entidade com um relacionamento recursivo binário que seja um-para-um ou um-para-muitos. Essa é uma das duas maneiras mais comuns de como as ferramentas de projeto tratam os relacionamentos, pedindo ao usuário para definir uma chave estrangeira na tabela filha que corresponda a uma chave primária na tabela pai.
- *Tabela SQL derivada de um relacionamento, contendo as chaves estrangeiras de todas as entidades no relacionamento.* Essa transformação sempre ocorre para relacionamentos que são binários muitos-para-muitos, recursivos ou não e com todos os relacionamentos que são de grau ternário ou maior. Essa é a outra maneira mais comum de como as ferramentas de projeto tratam os relacionamentos nos modelos ER e UML. Um relacionamento muitos-para-muitos só pode ser definido em termos de uma tabela que contém chaves estrangeiras correspondentes às chaves primárias das duas entidades associadas. Essa nova tabela também pode conter os atributos do relacionamento original - por exemplo, um relacionamento "matriculado-em" entre duas entidades Aluno e Curso pode ter os atributos "semestre" e "nota", que estão associados a uma inscrição em particular de um aluno em um determinado curso.

As regras a seguir se aplicam ao tratamento de valores nulos da SQL nestas transformações:

- Nulos são permitidos em uma tabela SQL para chaves estrangeiras de entidades opcionais (referenciadas) associadas.

- Nulos não são permitidos em uma tabela SQL para chaves estrangeiras de entidades obrigatórias (referenciadas) associadas.
- Nulos não são permitidos para qualquer chave em uma tabela SQL derivada de um relacionamento muitos-para-muitos, pois somente entradas de linha completas fazem sentido na tabela.

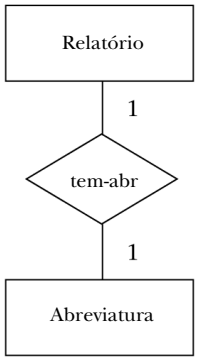
As Figuras de 5.1 a 5.8 mostram as instruções create table da SQL que podem ser derivadas de cada tipo construtor do modelo ER ou UML. Observe que os nomes de tabela aparecem em negrito por questão de legibilidade. Observe também que, em cada definição de tabela SQL, o termo "chave primária" representa a chave da tabela que deve ser usada para indexação e consulta dos dados.

5.1.1 Relacionamentos binários

Um relacionamento binário um-para-um entre duas entidades é ilustrado na Figura 5.1, partes de a até c. Observe que a associação binária equivalente em UML é dada na Figura 5.2, partes de a até c.

Quando as duas entidades são obrigatórias (Figura 5.1a), cada entidade se torna uma tabela, e a chave de qualquer entidade pode aparecer na tabela da outra entidade como uma chave estrangeira. Uma das entidades em um relacionamento opcional (ver Departamento, na Figura 5.1b) deve conter a chave estrangeira da outra entidade em sua tabela transformada. Funcionário, a outra entidade na Figura 5.1b, também pode conter uma chave estrangeira (dept_no) com nulos permitidos, mas isso exigiria mais espaço de armazenamento, devido ao número muito maior de instâncias da entidade Funcionário que instâncias de Departamento. Quando as duas entidades forem opcionais (Figura 5.1c), qualquer entidade pode conter a chave estrangeira embutida da outra entidade, com nulos permitidos nas chaves estrangeiras.

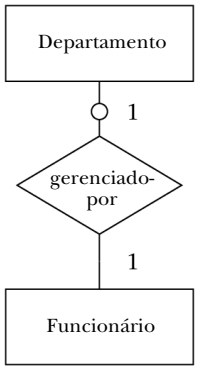
O relacionamento um-para-muitos pode se apresentar como obrigatório ou opcional no lado "muitos", sem afetar a transformação. No lado "um", ele pode ser obrigatório (Figura 5.1d) ou opcional (Figura 5.1e). Em todos os casos, a chave estrangeira precisa aparecer no lado "muitos", que representa a entidade filha, com nulos permitidos para chaves estrangeiras apenas nos casos em que no lado "um" é opcional. As restrições de chave estrangeira são definidas de acordo com o significado específico do relacionamento e podem variar de um relacionamento para outro.



Cada relatório tem uma abreviatura, e cada abreviatura representa exatamente um relatório.

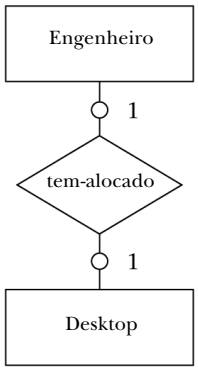
salto
o.i. pág. 86

(a) Um-para-um, ambas as entidades obrigatórias



Cada departamento precisa ter um gerente, mas um funcionário pode ser um gerente de no máximo um departamento.

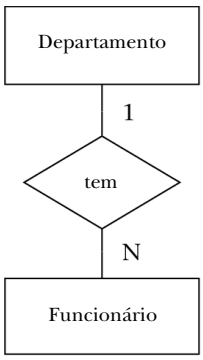
(b) Um-para-um, uma entidade opcional, uma obrigatória



Alguns computadores de *desktop* são alocados a engenheiros, mas não necessariamente a todos os engenheiros.

(c) Um-para-um, ambas as entidades opcionais

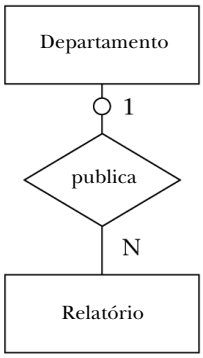
Figura 5.1 Modelo ER: relacionamento binário um-para-um entre duas entidades.



Cada funcionário trabalha em exatamente um departamento, e cada departamento tem pelo menos um funcionário.

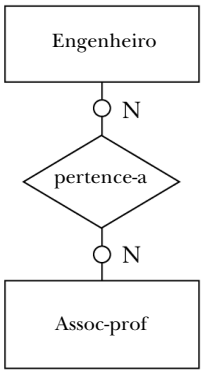
salto
o.i. pág. 87

(d) Um-para-muitos, ambas as entidades obrigatórias



Cada departamento publica um ou mais relatórios. Determinado relatório pode não necessariamente ser publicado por um departamento.

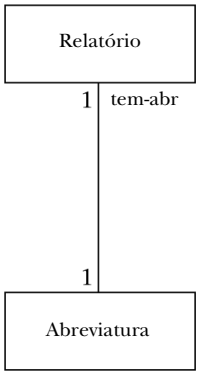
(e) Um-para-muitos, uma entidade opcional, uma obrigatória



Cada associação profissional pode ter nenhum, um ou muitos membros engenheiros. Cada engenheiro poderia ser membro de nenhuma, uma ou muitas associações profissionais.

(f) Muitos-para-muitos, ambas as entidades opcionais

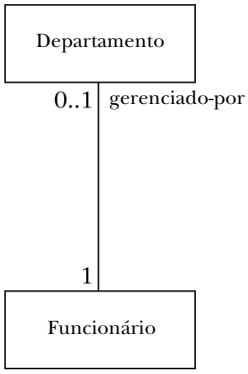
Figura 5.1 *(continuação)*



Cada relatório tem uma abreviatura, e cada abreviatura representa exatamente um relatório.

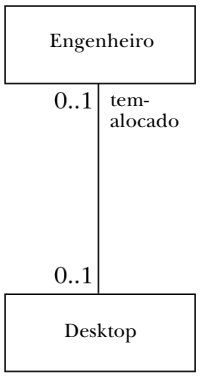
salto
o.i. pág. 88

(a) Um-para-um, ambas as entidades obrigatórias



Cada departamento precisa ter um gerente, mas um funcionário pode ser um gerente de no máximo um departamento.

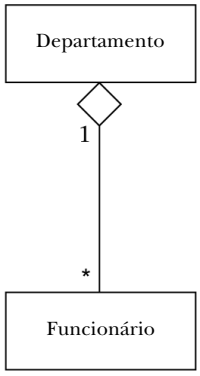
(b) Um-para-um, uma entidade opcional, uma obrigatória



Alguns computadores de desktop são alocados a engenheiros, mas não necessariamente a todos os engenheiros.

(c) Um-para-um, ambas as entidades opcionais

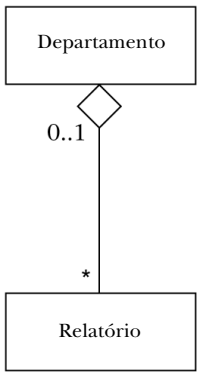
Figura 5.2 UML: relacionamento binário um-para-um entre duas entidades.



Cada funcionário trabalha em exatamente um departamento, e cada departamento tem pelo menos um funcionário.

salto
o.i. pág. 89

(d) Um-para-muitos, ambas as entidades obrigatórias



Cada departamento publica um ou mais relatórios. Determinado relatório pode não necessariamente ser publicado por um departamento.

(e) Um-para-muitos, uma entidade opcional, uma obrigatória

Cada associação profissional pode ter nenhum, um ou muitos membros engenheiros. Cada engenheiro poderia ser membro de nenhuma, uma ou muitas associações profissionais.

(f) Muitos-para-muitos, ambas as entidades opcionais

Figura 5.2 *(continuação)*

O relacionamento muitos-para-muitos, mostrado na Figura 5.1f como opcional para as duas entidades, exige uma nova tabela contendo as chaves primárias das duas entidades. A mesma transformação se aplica ao caso opcional ou obrigatório, incluindo o fato de a cláusula *not null* precisar aparecer para as chaves estrangeiras nos dois casos. Observe também que a tabela SQL derivada de uma entidade opcional pode ter zero linhas para esse relacionamento em particular. Isso não afeta a definição de "null" ou "not null" da tabela.

5.1.2 Relacionamentos binários recursivos

Um relacionamento um-para-um com uma única entidade indica alguma forma de acoplamento entre ocorrências dessa entidade, conforme indicado pelo nome do relacionamento. Esse acoplamento pode ser completamente opcional, completamente obrigatório ou nenhum deles. Em todos esses casos (Figura 5.3a para ER e Figura 5.4a para UML), a chave da entidade de acoplada aparece como chave estrangeira na tabela resultante. Os dois atributos de chave são retirados do mesmo domínio, mas recebem diferentes nomes para indicar o seu uso exclusivo. O relacionamento um-para-muitos exige uma chave estrangeira na tabela resultante (Figura 5.3b). As restrições de chave estrangeira podem variar de acordo com a especificidade do relacionamento.

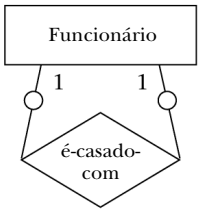
O relacionamento muitos-para-muitos é mostrado como opcional (Figura 5.3c) e resulta em uma nova tabela; ele também poderia ser definido como obrigatório (usando a palavra "deve" no lugar de "pode"); nos dois casos, as chaves estrangeiras são definidas como "not null". Nos relacionamentos muitos-para-muitos, as restrições de chave estrangeira sobre delete e update sempre precisam ser propagadas, pois cada entrada na tabela SQL depende do valor atual ou da existência da chave primária referenciada.

5.1.3 Relacionamentos ternários e n-ários

Relacionamentos n-ários possuem $(n + 1)$ variações possíveis de conectividade: todos os n lados com conectividade "um"; $(n - 1)$ lados com conectividade "um" e um lado com conectividade "muitos"; $(n - 2)$ lados com conectividade "um" e dois lados com "muitos"; e assim por diante, até que todos os lados sejam "muitos".

As quatro variedades possíveis de um relacionamento ternário aparecem na Figura 5.5 para o modelo ER e na Figura 5.6 para UML. Todas as variações são transformadas pela criação de uma tabela SQL contendo as chaves primá-

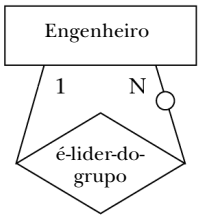
rias de todas as entidades; porém, em cada caso, o significado das chaves é diferente. Quando todos os três relacionamentos são "um" (Figura 5.5a), a tabela SQL resultante consiste em três chaves distintas possíveis. Esse arranjo representa o fato de que três DFs são necessárias para descrever esse relacionamento. A restrição de opcionalidade não é usada aqui porque todas as n entidades precisam participar de cada instância do relacionamento para satisfazer as restrições da DF. (Veja, no Capítulo 6, uma discussão mais profunda sobre dependências funcionais.)



Um funcionário pode ser casado com outro funcionário nesta empresa.

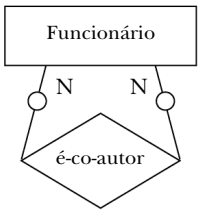
salto
o.i. pág. 91

(a) Um-para-um, ambos os lados opcionais



Os engenheiros são divididos em grupos para certos projetos. Cada grupo possui um líder.

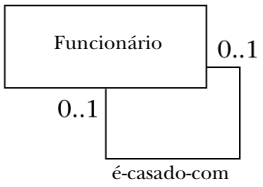
(b) Um-para-muitos, lado um obrigatório, lado muitos opcional



Cada funcionário tem a oportunidade de ser co-autor de um relatório com um ou mais outros funcionários, ou de escrever o relatório sozinho.

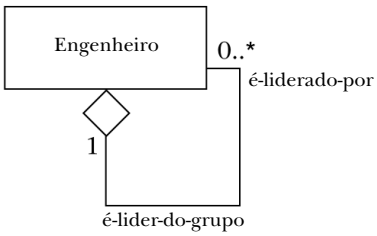
(c) Muitos-para-muitos, ambos os lados opcionais

Figura 5.3 Modelo ER: relacionamento binário recursivo.



Um funcionário pode ser casado com outro funcionário nesta empresa.

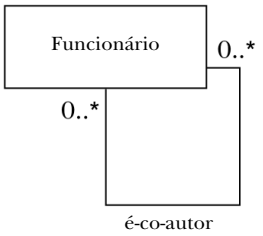
(a) Um-para-um, ambos os lados opcionais



Os engenheiros são divididos em grupos para certos projetos. Cada grupo possui um líder.

salto
o.i. pág. 92

(b) Um-para-muitos, lado um obrigatório, lado muitos opcional

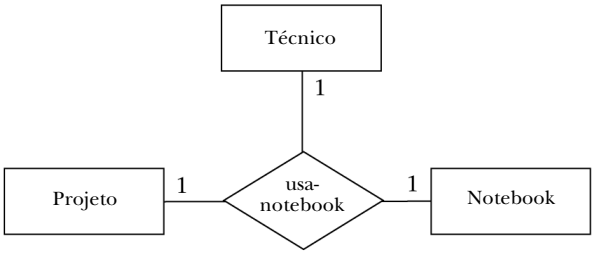


Cada funcionário tem a oportunidade de ser co-autor de um relatório com um ou mais outros funcionários, ou de escrever o relatório sozinho.

(c) Muitos-para-muitos, ambos os lados opcionais

Figura 5.4 UML: relacionamento binário recursivo.

Em geral, o número de entidades com conectividade "um" determina o limite inferior sobre o número de DFs. Assim, na Figura 5.3b, que é um-para-um-para-muitos, existem duas DFs; na Figura 5.5c, que é um-para-muitos-para-muitos, existe apenas uma DF. Quando todos os relacionamentos são "muitos" (Figura 5.5d), a tabela de relacionamentos é uma única chave composta, sem considerar os atributos próprios do relacionamento. Nesse caso, a chave é o composto de todas as três chaves correspondente às três entidades associadas.



Um técnico usa exatamente um notebook para cada projeto. Cada notebook pertence a um técnico para cada projeto. Observe que um técnico ainda pode trabalhar em muitos projetos e manter diferentes notebooks para diferentes projetos.

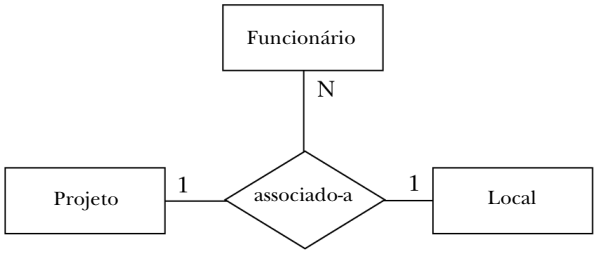
salto
o.i. pág. 93

35	alpha	5001
35	gamma	2008
42	delta	1004
42	epsilon	3005
81	gamma	1007
93	alpha	1009
93	beta	5001

(a) Relacionamento ternário um-para-um-para-um

Figura 5.5 Modelo ER: relacionamentos ternário e *n*-ário.

As restrições de chave estrangeira sobre delete e update para relacionamentos ternários transformados em tabelas SQL sempre precisam ser propagadas, pois cada entrada na tabela SQL depende do valor atual (ou da existência) da chave primária referenciada.



Cada funcionário associado para um projeto trabalha em apenas um local para esse projeto, mas pode estar em um local diferente para um projeto diferente. Em determinado local, um funcionário trabalha em apenas um projeto. Em um local em particular, pode haver muitos funcionários associados a um determinado projeto.

salto
o.i. pág. 94

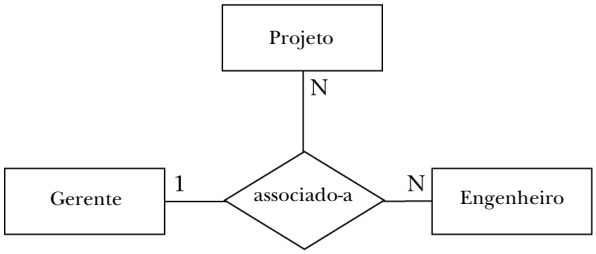
48101		B66
48101		E71
20702		A12
20702		D54
51266		G14
51266		A12
76323		B66

(b) Relacionamentos ternários um-para-um-para-muitos

Figura 5.5 *(continuação)*

5.1.4 Generalização e agregação

A transformação de uma abstração de generalização pode produzir tabelas SQL separadas para a entidade genérica ou supertipo e para cada um dos subtipos (Figura 5.7 em ER e Figura 5.8 em UML). A tabela derivada de uma entidade supertipo contém a chave da entidade supertipo e todos os atributos



Cada engenheiro trabalhando em um determinado projeto tem exatamente um gerente, mas um projeto pode ter muitos gerentes e um engenheiro pode ter muitos gerentes e muitos projetos. Um gerente pode gerenciar vários projetos.

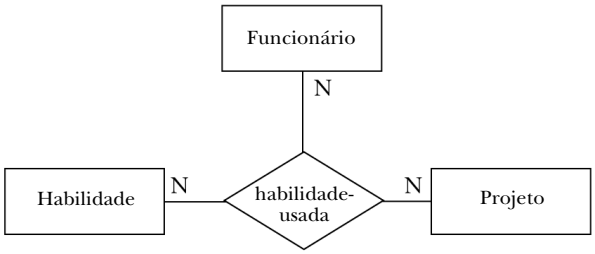
salto
o.i. pág. 95

alpha	4106	27
alpha	4200	27
beta	7033	32
beta	4200	14
gamma	4106	71
delta	7033	55
delta	4106	39
iota	4106	27

(c) Relacionamentos ternários um-para-muitos-para-muitos

Figura 5.5 *(continuação)*

comuns. Cada tabela derivada das entidades subtipo contém a chave da entidade supertipo e apenas os atributos que são específicos a esse subtipo. A integridade de atualização é mantida exigindo-se que todas as inserções e exclusões ocorram na tabela do supertipo e na tabela do subtipo relevante - ou seja, a propagação da restrição de chave estrangeira precisa ser usada. Se a



Os funcionários podem usar diferentes habilidades em qualquer um dos vários projetos, e cada projeto possui muitos funcionários com diversas habilidades.

salto
o.i. pág. 96

101		
101		
101		
101		
102		
102		
102		
105		

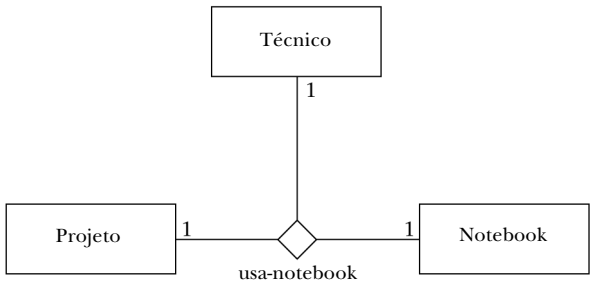
(d) Relacionamentos ternários muitos-para-um-para-muitos

Dependências funcionais

Nenhuma

Figura 5.5 *(continuação)*

atualização é na chave primária da tabela supertipo, então todas as tabelas subtipo, além da tabela supertipo, precisam ser atualizadas. Uma atualização em um atributo não-chave afeta o supertipo ou uma tabela subtipo, mas não ambos. As regras de transformação (e regras de integridade) são iguais para as generalizações de subtipo disjuntas e sobrepostas.



Um técnico usa exatamente um notebook para cada projeto. Cada notebook pertence a um técnico para cada projeto. Observe que um técnico ainda pode trabalhar em muitos projetos e manter diferentes notebooks para diferentes projetos.

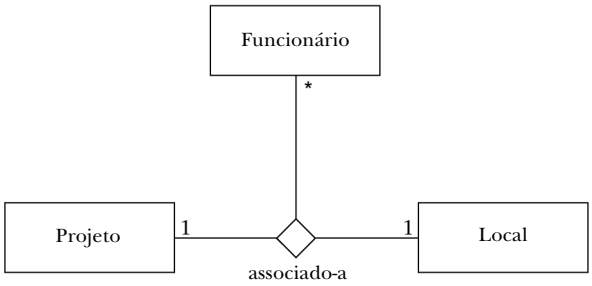
salto
o.i. pág. 97

35	alpha	5001
35	gamma	2008
42	delta	1004
42	epsilon	3005
81	gamma	1007
93	alpha	1009
93	beta	5001

(a) Associação ternária um-para-um-para-um

Figura 5.6 UML: relacionamentos ternários e *n*-ários.

Uma outra abordagem é ter uma única tabela que inclui todos os atributos do supertipo e dos subtipos (a hierarquia inteira em uma tabela), com os nulos sendo usados quando for necessário. Uma terceira possibilidade é uma tabela para cada subtipo, levando-se os atributos comuns para os subtipos específicos. Existem vantagens e desvantagens em cada uma dessas três abor-



Cada funcionário associado a um projeto trabalha em apenas um local para esse projeto, mas pode estar em um local diferente para um projeto diferente. Em determinado local, um funcionário trabalha em apenas um projeto. Em um local em particular, pode haver muitos funcionários associados a um determinado projeto.

salto
o.i. pág. 98

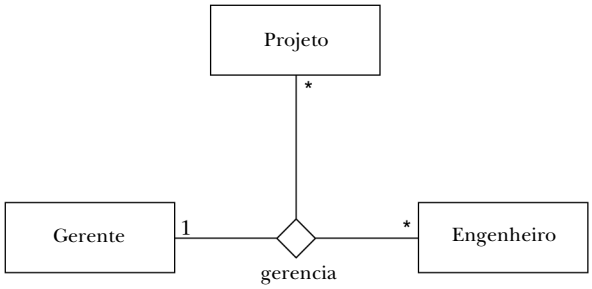
48101		B66
48101		E71
20702		A12
20702		D54
51266		G14
51266		A12
76323		B66

(b) Associações ternárias um-para-um-para-muitos

Figura 5.6 *(continuação)*

dagens. Atualmente, existem várias ferramentas de software que admitem todas as três opções [Fowler 2003; Ambler, 2003].

Os profissionais de banco de dados normalmente acrescentam um discriminador ao supertipo quando implementam a generalização. O discriminador é um atributo que possui um valor separado para cada supertipo



Cada engenheiro trabalhando em um determinado projeto tem exatamente um gerente, mas um projeto pode ter muitos gerentes, e um engenheiro pode ter muitos gerentes e muitos projetos. Um gerente pode gerenciar vários projetos.

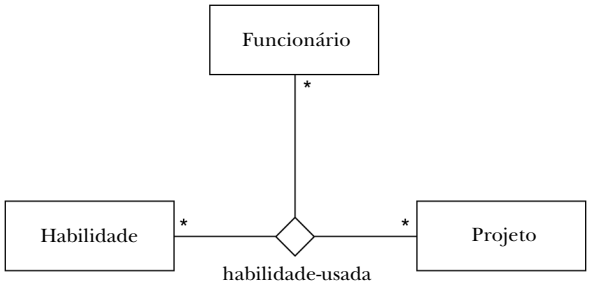
salto
o.i. pág. 99

alpha	4106	27
alpha	4200	27
beta	7033	32
beta	4200	14
gamma	4106	71
delta	7033	55
delta	4106	39
iota	4106	27

(c) Associação ternária um-para-muitos-para-muitos

Figura 5.6 *(continuação)*

e indica qual subtipo usar para obter mais informações. Essa abordagem funciona, até certo ponto. Entretanto, existem situações que exigem vários níveis de supertipos e subtipos, em que mais de um discriminador pode ser necessário.



Funcionários podem usar diferentes habilidades em qualquer um dos vários projetos; e cada projeto possui muitos funcionários com diversas habilidades.

salto
o.i. pág. 100

101		
101		
101		
101		
102		
102		
102		
105		

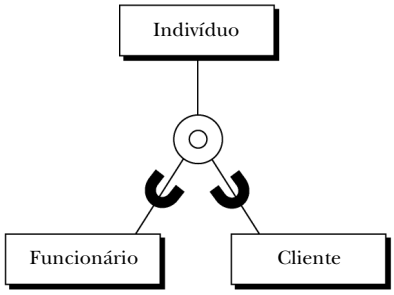
(d) Associações ternárias muitos-para-um-para-muitos

Dependências funcionais

Nenhuma

Figura 5.6 *(continuação)*

A transformação de uma abstração de agregação também produz uma tabela separada para a entidade supertipo e cada entidade subtipo. Contudo, não existem atributos comuns e restrições de integridade a serem mantidas. A principal função da agregação é oferecer uma abstração para auxiliar o pro-



salto
o.i. pág. 101

Figura 5.7 Modelo ER: generalização e agregação.

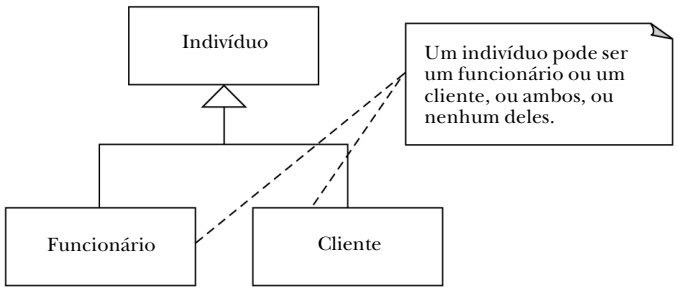
cesso de integração de visão. Na UML, a agregação é um relacionamento de composição, que em ER corresponde a uma entidade fraca [Muller, 1999].

5.1.5 Relacionamentos múltiplos

Múltiplos Relacionamentos entre *n* entidades sempre são considerados completamente independentes. Relacionamentos binários um-para-um, um-para-muitos ou recursivos que resultem em tabelas equivalentes ou que difiram apenas no acréscimo de uma chave estrangeira podem simplesmente ser unificados em uma única tabela contendo todas as chaves estrangeiras. Relacionamentos muitos-para-muitos ou ternários que resultam em tabelas SQL costumam ser exclusivos e não podem ser unificados.

5.1.6 Entidades fracas

Entidades fracas diferem das entidades apenas em sua necessidade de chaves de outras entidades para estabelecer sua unicidade. Fora isso, possuem as



salto
o.i. pág. 102

Figura 5.8 UML: generalização e agregação.

mesmas propriedades de transformação que as entidades, e nenhuma regra especial é necessária. Quando uma entidade fraca for derivada de duas ou mais entidades no diagrama ER, ela poderá ser transformada diretamente em uma tabela sem nenhuma alteração.

5.2 Etapas de transformação

A lista a seguir resume as etapas básicas de transformação do diagrama ER para tabelas SQL:

- Transformar cada entidade em uma tabela contendo os atributos-chave e não-chave da entidade.
- Transformar cada relacionamento binário muitos-para-muitos, recursivo ou não, em uma tabela com as chaves das entidades e os atributos do relacionamento.
- Transformar cada relacionamento ternário ou *n*-ário de nível superior em uma tabela.

Agora, vamos estudar cada etapa individualmente.

5.2.1 Transformação de entidade

Se houver um relacionamento um-para-muitos entre duas entidades, acrescente a chave da entidade do lado "um" (o pai) à tabela filho como uma chave estrangeira. Se houver um relacionamento um-para-um entre duas entidades, acrescente a chave de uma das entidades à tabela de outra entidade como uma chave estrangeira. A adição de uma chave estrangeira devido a um relacionamento um-para-um pode ser feito em qualquer direção. Uma estratégia é manter o relacionamento pai-filho mais natural colocando a chave do pai na tabela filho. Outra estratégia é baseada na eficiência: acrescente uma chave estrangeira à tabela com menos linhas.

Cada entidade em uma hierarquia de generalização é transformada em uma tabela. Cada uma dessas tabelas contém uma chave da entidade supertipo; na realidade, as chaves primárias do subtipo também são chaves estrangeiras. A tabela supertipo também contém valores não-chaves que são comuns a todas as entidades relevantes; as outras tabelas contêm valores não–chaves específicos de cada entidade subtipo.

Os construtores da SQL para essas transformações podem incluir restrições not null, unique e foreign key. Uma chave primária precisa ser especificada para cada tabela, seja explicitamente a partir das chaves no diagrama ER ou pela composição de todos os atributos como chave default. Observe que a designação de chave primária implica que o atributo é not null e unique. No entanto, é importante observar que nem todos os SGBDs seguem o padrão ANSI em relação a essa questão — em alguns sistemas, pode ser possível criar uma chave primária que pode ser nula. Assim, recomendamos que você especifique "not null" explicitamente para todos os atributos-chave.

5.2.2 Transformação do relacionamento binário muitos-para-muitos

Nesta etapa, cada relacionamento binário muitos-para-muitos é transformado em uma tabela contendo as chaves das entidades e os atributos do relacionamento. A tabela resultante mostrará a correspondência entre instâncias específicas de uma entidade com as de outra entidade. Qualquer atributo dessa correspondência, como o escritório eleito que um engenheiro tem em uma associação profissional (Figura 5.1f), é considerado dado da interseção e é acrescentado à tabela como um atributo não-chave.

Os construtores SQL para essa transformação podem incluir restrições not null. A restrição unique não é usada aqui, pois todas as chaves são compo-

sições das chaves primárias das entidades participantes associadas ao relacionamento. As restrições para chave primária e chave estrangeira são obrigatórias, pois a tabela é definida como uma composição das chaves primárias das entidades associadas.

5.2.3 Transformação de relacionamento ternário

Nesta etapa, cada relacionamento ternário (ou *n*-ário mais alto) é transformado em uma tabela. Os relacionamentos ternários (ou *n*-ário mais altos) são definidos como uma coleção das *n* chaves primárias das entidades associadas nesse relacionamento, possivelmente com alguns atributos não-chave que são dependentes da chave formada pela composição dessas *n* chaves primárias.

Os construtores SQL para essa transformação precisam incluir restrições not null, pois a opcionalidade não é permitida. A restrição unique não é usada para atributos individuais, pois todas as chaves são composições das chaves primárias das entidades participantes associadas ao relacionamento. As restrições para chave primária e chave estrangeira são exigidas porque a tabela é definida como uma composição das chaves primárias das entidades associadas. A cláusula unique também precisa ser usada para definir chaves alternativas que normalmente ocorrem em relacionamentos ternários. Observe que uma tabela derivada de um relacionamento *n*-ário possui *n* chaves estrangeiras.

5.2.4 Exemplo de transformação ER-para-SQL

Os diagramas ER do banco de dados pessoa e projeto da empresa (Capítulo 4) podem ser transformados em tabelas SQL. Um resumo da transformação de entidades e relacionamentos em tabelas SQL é ilustrado na lista a seguir.

Tabelas SQL derivadas diretamente das entidades (ver Figura 4.3d):

divisão	**secretário**	**projeto**
departamento	**engenheiro**	**local**
funcionário	**técnico**	**assoc_prof**
gerente	**habilidade**	**desktop**

Tabelas SQL derivadas de relacionamentos binários muitos-para-muitos ou binários recursivos muitos-para-muitos:

- **pertence_a**

Tabelas SQL transformadas de relacionamentos ternários:

- **habilidade_usada**
- **associada_a**

5.3 Resumo

Entidades, atributos e relacionamentos do modelo ER e classes, atributos e associações da UML podem ser transformados diretamente em definições de tabela SQL (SQL-99) com algumas regras simples. Entidades são transformadas em tabelas, com todos os atributos mapeados, um a um, como atributos de tabela. As tabelas que representam entidades filhas (lado "muitos") de um relacionamento pai-filho (um-para-muitos ou um-para-um) também precisam incluir, como chave estrangeira, a chave primária da entidade pai. Um relacionamento muitos-para-muitos é transformado em uma tabela e conterá as chaves primárias das entidades associadas como chave primária composta; os componentes dessa chave também são indicados como chaves estrangeiras em SQL. Um relacionamento ternário ou *n*-ário mais alto é transformado em uma tabela que contém as chaves primárias das entidades associadas; essas chaves são indicadas como chaves estrangeiras em SQL. Um subconjunto dessas chaves pode ser designado como chave primária, conforme as dependências funcionais associadas ao relacionamento. As regras de generalização exigem a herança da chave primária do supertipo para as entidades subtipo quando transformadas em tabelas SQL. As restrições de opcionalidade nos diagramas ER ou UML se traduzem em nulos permitidos no modelo relacional quando aplicadas ao lado "um" de um relacionamento. Em SQL, a falta de uma restrição de opcionalidade determina a designação not null na definição create table.

5.4 Resumo da literatura

A definição das transformações básicas do modelo ER para tabelas é explicada em McGee [1974], Wong e Katz [1979], Sakai [1983], Martin [1983], Hawryszkiewyck [1984], Jajodia e Ng [1984]; e para UML, em Muller [1999].

6 Normalização

Este capítulo aborda os fundamentos das formas normais para bancos de dados relacionais e a etapa de projeto de banco de dados que normaliza as tabelas candidatas (etapa II(d) do ciclo de vida de projeto de banco de dados). Ele também investiga a equivalência entre o modelo de dados conceitual (por exemplo, o modelo ER) e as formas normais de tabelas. À medida que apresentamos os exemplos neste capítulo, deve-se tornar óbvio que um projeto bom e bem elaborado de um modelo conceitual resultará em bancos de dados que já estão normalizados ou que podem ser facilmente normalizados com pequenas mudanças. Essa verdade ilustra a beleza da abordagem de modelagem conceitual para o projeto de banco de dados, uma vez que o projetista experiente em banco de dados relacional poderá desenvolver desde o início uma atração natural para um modelo normalizado.

Para a maioria dos profissionais de banco de dados, as Seções de 6.1 a 6.4 abordam a normalização crucial e necessária para o uso diário, até a Forma Normal de Boyce-Codd (FNBC). A Seção 6.5 aborda as formas normais mais altas e de maior interesse teórico; porém, para o leitor interessado, mostramos a equivalência entre as formas normais mais altas e os relacionamentos ternários no modelo ER e na UML.

6.1 Fundamentos da normalização

As tabelas de banco de dados relacional, sejam elas derivadas dos modelos ER ou UML, às vezes sofrem com alguns problemas bastante sérios em termos de desempenho, integridade e facilidade de manutenção. Por exemplo, quan-

do o banco de dados como um todo é definido como uma única e grande tabela, isso pode resultar em uma grande quantidade de dados redundantes e pesquisas demoradas mesmo para pequenas quantidades de linhas. Isso também pode resultar em atualizações longas e dispendiosas, e as exclusões em particular podem provocar a eliminação de dados úteis como um efeito colateral indesejado.

Essa situação pode ser demonstrada na Figura 6.1, na qual produtos, vendedores, clientes e pedidos são todos armazenados em uma única tabela, chamada Sales (Vendas). Nessa tabela, notamos que certas informações de produto e cliente são armazenadas de forma redundante, desperdiçando espaço de armazenamento. Certas consultas, como "Quais clientes pediram vacuum cleaners (aspiradores de pó) no mês passado?" exigiriam uma pesquisa na tabela inteira. Além disso, atualizações como alterar o endereço do cliente Dave Bachmann exigiriam a alteração de muitas linhas. Finalmente, a exclusão de um pedido de um cliente, por exemplo, Qiang Zhu (que comprou um computador caro), se esse for seu único pedido pendente, excluirá a única cópia do seu endereço e da sua avaliação de crédito como um efeito colateral. Essa informação pode ser difícil (ou às vezes impossível) de ser recuperada. Esses problemas também ocorrem em um banco de dados contendo tabelas com tamanhos significativos.

Se tivéssemos um método de desmembrar essa enorme tabela em tabelas menores, para que esses tipos de problemas fossem eliminados, o banco de dados seria muito mais eficiente e confiável. Classes de esquemas de bancos de dados relacionais, ou definições de tabelas, chamadas formas normais,

Sales

product_name	order_no	cust_name	cust_addr	credit	date	sales_name
vacuum cleaner	1458	Dave Bachmann	Austin	6	1-3-03	Carl Bloch
computer	2730	Qiang Zhu	Plymouth	10	4-15-05	Ted Hanss
refrigerator	2460	Mike Stolarchuck	Ann Arbor	8	9-12-04	Dick Phillips
DVD player	519	Peter Honeyman	Detroit	3	12-5-04	Fred Remley
radio	1986	Charles Antonelli	Chicago	7	5-10-05	R. Metz
CD player	1817	C.V. Ravishankar	Mumbai	8	8-3-02	Paul Basile
vacuum cleaner	1865	Charles Antonelli	Chicago	7	10-1-04	Carl Bloch
vacuum cleaner	1885	Betsy Karmeisool	Detroit	8	4-19-99	Carl Bloch
refrigerator	1943	Dave Bachmann	Austin	6	1-4-04	Dick Phillips
television	2315	Sakti Pramanik	East Lansing	6	3-15-04	Fred Remley

Verificar se haverá tradução

o.i. pág. 108

Figura 6.1 Banco de dados de uma única tabela.

normalmente são usadas para realizar esse objetivo. A criação de tabelas de banco de dados em uma dada forma normal é chamada de normalização. A normalização é feita analisando-se as interdependências entre atributos individuais associados a essas tabelas e tomando-se projeções (subconjuntos de colunas) de tabelas maiores para formar tabelas menores.

Vamos rever primeiro as formas normais básicas, que já foram bem estabelecidas na literatura de banco de dados relacional e na prática.

6.1.1 Primeira forma normal

Definição. Uma tabela estará na *primeira forma normal* (1FN) se, e somente se, todas as colunas tiverem apenas valores atômicos, ou seja, se cada coluna só puder ter um valor para cada linha na tabela.

Tabelas de bancos de dados relacionais, como a tabela **Sales** ilustrada na Figura 6.1, possui apenas valores atômicos em cada coluna de cada linha. Essas tabelas são consideradas na primeira forma normal, o nível mais básico de tabelas normalizadas.

Para entender melhor a definição da 1FN, é importante conhecer a diferença entre um domínio, um atributo e uma coluna. Um *domínio* é o conjunto de todos os valores possíveis para determinado tipo de atributo, mas pode ser usado para mais de um atributo. Por exemplo, o domínio dos nomes de pessoas é o conjunto básico de todos os possíveis nomes que poderiam ser usados para nome-cliente ou nome-vendedor na tabela de banco de dados da Figura 6.1. Cada coluna em uma tabela relacional representa um único atributo, mas, em alguns casos, mais de uma coluna pode se referir a atributos diferentes do mesmo domínio. Quando isso ocorre, a tabela ainda está na 1FN, pois os valores nela ainda são atômicos. Na verdade, o padrão SQL assume apenas valores atômicos, e uma tabela relacional está, por padrão, na 1FN. Uma boa explicação sobre isso é dada em Muller [1999].

6.1.2 Superchaves, chaves candidatas e chaves primárias

Uma tabela na 1FN normalmente sofre de problemas de duplicação de dados, desempenho e integridade na atualização, como já observamos anteriormente. Contudo, para entendermos melhor, temos de definir o conceito de chave no contexto das tabelas normalizadas. Uma *superchave* é um conjunto de um ou mais atributos que, quando tomados coletivamente, nos permite identificar exclusivamente uma ocorrência de uma entidade ou linha de uma

tabela. Qualquer subconjunto de atributos de uma superchave que também seja uma superchave, e que não possa ser reduzido a outras superchaves, é chamado de *chave candidata*. Uma *chave primária* é selecionada arbitrariamente, a partir do conjunto de chaves candidatas, para que ela seja usada como índice dessa tabela.

Como exemplo, na Figura 6.2, a composição de todos os atributos da tabela forma uma superchave, pois linhas duplicadas não são permitidas no modelo relacional. Assim, uma superchave trivial é formada a partir da composição de todos os atributos de uma tabela. Supondo que cada endereço de departamento (dept_addr) nessa tabela tenha um único valor, podemos concluir que a composição de todos os atributos, exceto dept_addr, também seja uma superchave. Examinando composições cada vez menores dos atributos e fazendo suposições realistas sobre quais atributos possuem valor exclusivo, descobrimos que a composição (report_no, author_id) determina exclusivamente todos os outros atributos da tabela e, portanto, é uma superchave. Entretanto, nem report_no nem author_id isolados pode determinar uma linha exclusivamente; assim, a composição desses dois atributos não pode ser reduzida em superchaves menores. Assim, a composição (report_no, author_id) torna-se uma chave candidata. Como essa é a única chave candidata nessa tabela, ela também se torna a chave primária.

Uma tabela pode ter mais de uma chave candidata. Se, por exemplo, na Figura 6.2, tivéssemos uma coluna adicional para author_ssn,* e a composi-

Report

report_no	editor	dept_no	dept_name	dept_addr	author_id	author_name	author_addr
4216	woolf	15	design	argus1	53	mantei	cs-tor
4216	woolf	15	design	argus1	44	bolton	mathrev
4216	woolf	15	design	argus1	71	koenig	mathrev
5789	koenig	27	analysis	argus2	26	fry	folkstone
5789	koenig	27	analysis	argus2	38	umar	prise
5789	koenig	27	analysis	argus2	71	koenig	mathrev

Verificar se haverá tradução

o.i. pág. 108

Figura 6.2 Tabela **Report**.

* *Nota dos Revisores Técnicos*: ssn (social security numbers): Nos Estados Unidos, o ssn (Social Security Number — número do seguro social) é um número único associado aos seus cidadãos com o principal propósito de identificar contribuintes para cobrança de impostos e garantir os benefícios da previdência social. Apesar disso, o ssn estende seu propósito original e é utilizado como o principal número de identificação da população norte-americana.

ção de report_no e author_ssn determinar exclusivamente todos os outros atributos da tabela, então tanto (report_no, author_id) quanto (report_no, author_ssn) seriam chaves candidatas. A chave primária, então, seria uma escolha arbitrária entre essas duas chaves candidatas.

Outros exemplos de múltiplas chaves candidatas podem ser vistos na Figura 5.5 (ver Capítulo 5). Na Figura 5.5a, a tabela **uses_notebook** possui três chaves candidatas: (emp_id, project_name), (emp_id, notebook_no) e (project_name, notebook_no); e na Figura 5.5b, a tabela **assigned_to** possui duas chaves candidatas: (emp_id, loc_name) e (emp_id, project_name). As Figuras 5.5c e 5.5d possuem cada uma apenas uma única chave candidata.

6.1.3 Segunda forma normal

Para explicar o conceito da segunda forma normal (2FN) e outras mais altas, apresentamos o conceito de dependência funcional, que já foi descrito rapidamente no Capítulo 2. A propriedade de um ou mais atributos determinarem unicamente o valor de um ou mais outros atributos é chamada de *dependência funcional*. Dada uma tabela (R), um conjunto de atributos (B) é dependente funcionalmente de outro conjunto de atributos (A) se, em qualquer momento do tempo, cada valor A estiver associado a apenas um valor B. Essa dependência funcional é indicada por A -> B.* No exemplo anterior da Figura 6.2, vamos supor que recebemos as seguintes dependências funcionais para a tabela **report:**

report: report_no -> editor, dept_no
dept_no -> detp_name, dept_addr
author_id -> author_name, author_addr

Definição. Uma tabela está na segunda forma normal (2FN) se, e somente se, ela estiver na 1FN e os atributos não-chaves forem totalmente dependentes da chave primária. Um atributo será totalmente dependente da chave primária se estiver no lado direito de uma DF que tem no lado esquerdo a própria chave primária ou algo que possa ser derivado da chave primária usando a transitividade das DFs.

**Nota dos Revisores Técnicos*: a notação A à B também pode ser utilizada. Veja exemplos na Figura 2.6.

Um exemplo de uma DF transitiva em **report** é o seguinte:

report_no -> dept_no
dept_no -> dept_name

Portanto, podemos derivar a DF (report_no -> dept_name), pois dept_name é transitivamente dependente de report_no.

Continuando nosso exemplo, a chave composta na Figura 6.2, (report_no, author_id), é a única chave candidata e, portanto, é a chave primária. Todavia, existe uma DF (dept_no -> dept_name, dept_addr) que não possui componente da chave primária no lado esquerdo, e duas DFs (report_no -> editor, dept_no e author_id -> author_name, author_addr) que contêm um componente da chave primária no lado esquerdo, mas não ambos os componentes. Dessa forma, **report** não satisfaz a condição para a 2FN para qualquer uma das DFs.

Considere as desvantagens da tabela **report** que só está na 1FN. Os valores dos atributos Report_no, editor e dept_no estão duplicados para cada autor de um mesmo relatório. Portanto, se o editor do relatório mudar, por exemplo, várias linhas terão de ser atualizadas. Isso é conhecido como *anomalia de atualização* e representa uma degradação em potencial do desempenho devido a atualizações redundantes. Se um novo editor tiver de ser incluído na tabela, isso só poderá ser feito se ele estiver editando um relatório: tanto o número do relatório quanto o número do editor precisam ser conhecidos para se incluir uma linha na tabela, pois você não pode ter uma chave primária com um valor nulo na maioria dos bancos de dados relacionais. Isso é conhecido como *anomalia de inserção*. Finalmente, se um relatório for retirado, todas as linhas associadas a ele precisam ser excluídas. Isso tem o efeito colateral de excluir a informação que associa um author_id a author_name e author_addr. Os efeitos colaterais da exclusão dessa natureza são conhecidos como *anomalias de exclusão*. Elas representam uma perda de integridade em potencial, pois a única maneira possível de restaurar as informações perdidas é encontrar esses mesmos dados em algum lugar fora do banco de dados e inseri-los de volta no banco de dados. Todas essas três anomalias representam problemas enfrentados pelos projetistas de banco de dados, mas a anomalia de exclusão é, com certeza, a mais séria, pois você pode perder dados que não poderão ser recuperados.

Essas desvantagens podem ser contornadas transformando-se a tabela 1FN em duas ou mais tabelas 2FN, usando o operador de projeção sobre o subconjunto de atributos da tabela 1FN. Neste exemplo, projetamos **report**

sobre report_no, editor, dept_no, dept_name e dept_addr para formar **report1**; projetamos **report** sobre author_id, author_name e author_addr para formar **report2**; e finalmente projetamos **report** sobre report_no e author_id para formar **report3**. A projeção de **report** em três tabelas menores preservou as DFs e a associação entre report_no e author_no, que era importante na tabela original. Os dados para as três tabelas aparecem na Figura 6.3. As DFs para essas tabelas 2FN são:

report1: report_no -> editor, dept_no
dept_no -> dept_name, dept_addr
report2: author_id -> author_name, author_addr
report3: report_no, author_id é uma chave candidata (sem DFs)

Report 1

report_no	editor	dept_no	dept_name	dept_addr
4216	woolf	15	design	argus 1
5789	koenig	27	analysis	argus 2

Report 2

author_id	author_name	author_addr
53	mantei	cs-tor
44	bolton	mathrev
71	koenig	mathrev
26	fry	folkstone
38	umar	prise
71	koenig	mathrev

Report 3

report_no	author_id
4216	53
4216	44
4216	71
5789	26
5789	38
5789	71

Figura 6.3 Tabelas 2FN.

Agora, temos três tabelas que satisfazem as condições para a 2FN e eliminamos os piores problemas da 1FN, especialmente a integridade (a anomalia de exclusão). Primeiro, editor, dept_no, dept_name e dept_addr não estão mais duplicados para cada autor de um relatório. Segundo, uma mudança de editor resulta em apenas uma atualização em uma linha para **report1**. E terceiro, o mais importante, a exclusão do relatório não tem o efeito colateral de excluir a informação do autor.

No entanto, nem toda a degradação de desempenho é eliminada; report_no ainda é duplicado para cada autor, e a exclusão de um relatório exige atualizações em duas tabelas (**report1** e **report3**), em vez de uma. Contudo, esses são problemas menores em comparação com aqueles na tabela **report**, na 1FN.

Observe que essas três tabelas de relacionamento na 2FN poderiam ter sido geradas diretamente por um diagrama ER (ou UML) que modelasse essa situação de forma equivalente com entidades Author e Report e um relacionamento muitos-para-muitos entre elas.

6.1.4 Terceira forma normal

As tabelas 2FN que estabelecemos na seção anterior representam uma melhoria significativa em relação às tabelas 1FN. Entretanto, elas ainda sofrem dos mesmos tipos de anomalias das tabelas 1FN, embora por motivos diferentes associados a dependências transitivas. Se existir uma dependência transitiva (funcional) em uma tabela, isso significa que dois fatos separados são representados nessa tabela, um fato para cada dependência funcional envolvendo um lado esquerdo diferente. Por exemplo, se excluirmos um relatório do banco de dados, o que envolve excluir as linhas apropriadas de **report1** e **report3** (ver Figura 6.3), temos o efeito colateral de excluir também a associação entre dept_no, dept_name e dept_addr. Se pudéssemos projetar a tabela **report1** sobre report_no, editor e dept_no para formar a tabela **report11**, e projetar **report1** sobre dept_no, dept_name e dept_addr para formar a tabela **report12**, poderíamos eliminar esse problema. Exemplos de tabelas **report11** e **report12** são apresentados na Figura 6.4.

> ***Definição.*** Uma tabela está na *terceira forma normal (3FN)* se, e somente se, para cada dependência funcional não trivial X->A, onde X e A são atributos simples ou compostos, uma das duas condições precisam ser mantidas: ou o atributo X é uma superchave, ou o atributo A é membro de uma chave candidata. Se o atributo A é membro de uma chave

candidata, A é chamado de atributo primo. Nota: uma DF trivial tem a forma YZ->Z.

No exemplo anterior, depois de projetar **report1** em **report11** e **report12** para eliminar a dependência transitiva report_no -> dept_no -> dept_name, dept_addr, temos as seguintes tabelas 3FN e suas dependências funcionais (e dados do exemplo na Figura 6.4):

report11: report_no -> editor, dept_no
report12: dept_no -> dept_name, dept_addr
report2: author_id -> author_name, author_addr
report3: report_no, author_id é uma chave candidata (sem DFs)

6.1.5 Forma Normal Boyce-Codd

A 3FN, que elimina a maioria das anomalias conhecidas nos bancos de dados de hoje, é o padrão mais comum para normalização nos bancos de dados comerciais e ferramentas CASE. As poucas anomalias restantes podem ser eliminadas pela Forma Normal de Boyce-Codd (FNBC) e formas normais mais altas, definidas aqui e na Seção 6.5. A FNBC é considerada uma variação forte da 3FN.

Report 11

report_no	editor	dept_no
4216	woolf	15
5789	koenig	27

Report 12

dept_no	dept_name	dept_addr
15	design	argus 1
27	analysis	argus 2

Report 2

author_id	author_name	author_addr
53	mantei	cs-tor
44	bolton	mathrev
71	koenig	mathrev
26	fry	folkstone
38	umar	prise
71	koenig	mathrev

Report 3

report_no	author_id
4216	53
4216	44
4216	71
5789	26
5789	38
5789	71

Verificar se haverá tradução
o.i. pág. 114

Figura 6.4 Tabelas 3FN.

Definição. Uma tabela **R** está na *Forma Normal de Boyce-Codd (FNBC)* se, para cada DF não trivial X->A, X for uma superchave.

A FNBC é uma forma de normalização mais forte do que a 3FN, pois elimina a segunda condição para a 3FN, que permitia que o lado direito da DF fosse um atributo primo. Assim, cada lado esquerdo de uma DF em uma tabela precisa ser uma superchave. Cada tabela que está na FNBC também está na 3FN, 2FN e 1FN, pelas definições anteriores.

O exemplo a seguir mostra uma tabela 3FN que não está na FNBC. Esse tipo de tabela possui anomalias de exclusão semelhantes àquelas nas formas normais inferiores.

Afirmação 1. Para determinada equipe, cada funcionário é dirigido por apenas um líder. Uma equipe pode ser dirigida por mais de um líder.

emp_name, team_name -> leader_name

Afirmação 2. Cada líder dirige apenas uma equipe.

leader_name -> team_name

Essa tabela está na 3FN com uma chave candidata composta emp_id, team_id:

team:	emp_name	team_name	leader_name
	Sutton	Hawks	Wei
	Sutton	Conders	Bachmann
	Niven	Hawks	Wei
	Niven	Eagles	Makowski
	Wilson	Eagles	DeSmith

A tabela **team** tem a seguinte anomalia de exclusão: se Sutton sair da equipe Condors, então perderemos o registro de Bachmann liderando a equipe Condors. Conforme mostrado por Date [1999], esse tipo de anomalia não pode ter uma decomposição sem perdas e preservar todas as DFs. Uma decomposição sem perdas exige que, quando você decompõe a tabela em duas tabelas menores projetando a tabela original sobre os dois subconjuntos superpostos do esquema, a junção natural dessas tabelas de subconjunto precisa resultar na tabela original sem quaisquer linhas indesejadas extras. O

modo mais simples de evitar a anomalia de exclusão para esse tipo de situação é criar uma tabela separada para cada uma das duas afirmações*[3]. Essas duas tabelas são particularmente redundantes, o suficiente para evitar a anomalia de exclusão. Essa decomposição é sem perdas (trivialmente) e preserva dependências funcionais, mas também diminui o desempenho da atualização devido à redundância, e necessita de espaço de armazenamento adicional. A troca normalmente compensa, pois a anomalia de exclusão é evitada.

6.2 O projeto de tabelas normalizadas: um exemplo simples

O exemplo nesta seção é baseado no diagrama ER da Figura 6.5 e nas DFs indicadas a seguir. Em geral, as DFs podem ser dadas explicitamente, derivadas do diagrama ER, ou derivadas da intuição (ou seja, da experiência com o domínio do problema).

1. emp_id, start_date -> job_title, end_date
2. emp_id -> emp_name, phone_no, office_no, proj_no, proj_name, dept_no
3. phone_no -> office_no
4. proj_no -> proj_name, proj_start_date, proj_end_date
5. dept_no -> dept_name, mgr_id
6. mgr_id -> dept_no

Nosso objetivo é projetar um esquema de banco de dados relacional que seja normalizado para pelo menos a 3FN e, se possível, minimizar o número de tabelas exigidas. Nossa técnica é aplicar a definição da terceira forma normal (3FN) da Seção 6.1.4 a essas DFs e criar tabelas que satisfaçam a definição.

Se tentarmos colocar as DFs de 1 a 6 em uma única tabela com a chave candidata composta (e chave primária) (emp_id, start_date), violaremos a definição da 3FN, pois as DFs de 2 a 6 envolvem nos lados esquerdos das DFs subconjuntos de atributos que não são superchaves. Conseqüentemente, precisamos separar a 1ª DF do restante. Se depois tentarmos combinar de 2 a 6, teremos muitas transitividades. Intuitivamente, sabemos que 2, 3, 4 e 5 precisam ser separadas em diferentes tabelas, devido às dependências transitivas.

---

* *Nota dos Revisores Técnicos*: na realidade, as tabelas devem ser team1(leader_name, team_name) e team2(emp_name, leader_name)]

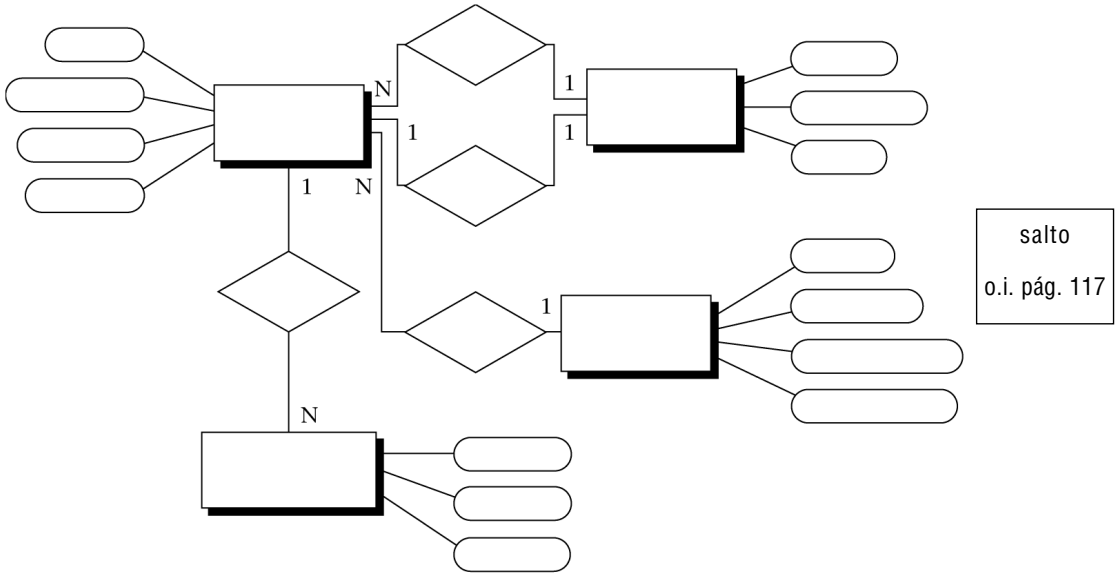


Figura 6.5 SALTO

Depois, temos de decidir se 5 e 6 podem ser combinadas sem perda da 3FN; isso pode ser feito porque mgr_id e dept_no são mutuamente dependentes e ambos os atributos são superchaves em uma tabela combinada. Assim, podemos definir as seguintes tabelas por projeções apropriadas de 1 a 6.

emp_hist:	emp_id, start_date -> job_title, end_date
employee:	emp_id -> emp_name, phone_no, proj_no, dept_no
phone:	phone_no -> office_no
project:	proj_no -> proj_name, proj_start_date, proj_end_date
department:	dept_no -> dept_name, mgr_id
	mgr_id -> dept_no

Essa solução, que está na FNBC e também na 3FN, mantém todas as DFs originais. Esse também é um conjunto mínimo de tabelas normalizadas. Na Seção 6.4, veremos um método formal de determinar um conjunto mínimo que podemos aplicar às situações muito mais complexas.

Projetos alternativos podem envolver a divisão de tabelas em partições para dados voláteis (freqüentemente atualizados) e passivos (raramente atualizados), a consolidação de tabelas para obter melhor desempenho na

consulta, ou duplicação de dados em diferentes tabelas para obter melhor desempenho na consulta sem perder a integridade. Em resumo, as medidas que usamos para avaliar as compensações em nosso projeto são:

- Desempenho da consulta (tempo)
- Desempenho da atualização (tempo)
- Desempenho do armazenamento (espaço)
- Integridade (impedimento de anomalias de exclusão)

6.3 Normalização de tabelas candidatas derivadas de diagramas ER

A normalização de tabelas candidatas [etapa II(d) do ciclo de vida do banco de dados] é realizada analisando-se as DFs associadas a essas tabelas: DFs explícitas da análise de requisitos do banco de dados (Seção 6.2), DFs derivadas do diagrama ER e DFs derivadas da intuição.

As *DFs primárias* representam as dependências entre os elementos de dados que são chaves de entidades, ou seja, as dependências inter-entidades. As *DFs secundárias*, por outro lado, representam dependências entre elementos de dados que compreendem uma única entidade, ou seja, as dependências intra-entidades. Normalmente, as DFs primárias são derivadas do diagrama ER, e as DFs secundárias são obtidas explicitamente da análise de requisitos. Se os construtores da ER não incluírem atributos não-chaves usados nas DFs secundárias, a especificação de requisitos de dados ou dicionário de dados precisam ser consultados. A Tabela 6.1 mostra os tipos de DFs primárias deriváveis de cada tipo de construtor da ER.

Cada tabela candidata normalmente terá várias DFs primárias e secundárias exclusivamente associadas a ela, as quais determinam o grau atual de normalização da tabela. Qualquer uma das técnicas bem conhecidas de elevar o grau de normalização pode ser aplicada a cada tabela até o grau desejado indicado na especificação de requisitos. A integridade é mantida exigindo-se que o esquema da tabela normalizada inclua todas as dependências de dados existentes no esquema da tabela candidata.

Qualquer tabela **B** que seja substituída por outra tabela **A** pode ser potencialmente eliminada. A tabela **B** é substituída por outra tabela **A** quando todos os atributos de **B** também estiverem contidos em **A**, e todas as dependências de dados em **B** também ocorrerem em **A**. Como um caso trivial, qualquer tabela contendo apenas uma chave composta e nenhum atributo não-chave é

Tabela 6.1 DFs primárias deriváveis de construções de relacionamento ER

Grau	Conectividade	DF primária
Binário ou Binário Recursivo	um-para-um	2 vias: chave(lado um) -> chave (lado um)
	um-para-muitos	chave(lado muitos) -> chave(lado um)
	muitos-para-muitos	nenhuma (chave composta dos dois lados)
Ternário	um-para-um-para-um	3 vias: chave(um), chave(um) -> chave(um)
	um-para-um-para-muitos	2 vias: chave(um), chave(muitos) -> chave(um)
	um-para-muitos-para-muitos	1 via: chave(muitos), chave(muitos) -> chave(um)
	muitos-para-muitos-para-muitos	nenhuma (chave composta de todos os 3 lados)
Generalização	nenhuma	nenhuma (somente DF secundária)

automaticamente substituída por qualquer outra tabela contendo os mesmos atributos chaves, pois a chave composta é a forma mais fraca de dependência de dados. Entretanto, se as tabelas **A** e **B** representarem os casos de supertipo e subtipo, respectivamente, das entidades definidas pela abstração de generalização, e **A** substituir **B** porque **B** não possui atributos específicos adicionais, o projetista precisará coletar e analisar informações adicionais para decidir se eliminará **B** ou não.

Uma tabela também pode ser substituída pela construção de uma junção de duas outras tabelas (uma tabela "join"). Quando isso ocorre, a eliminação de uma tabela substituída pode resultar em perda de eficiência na recuperação, embora os custos de armazenamento e atualização tendam a ser reduzidos. Essa compensação precisa ser mais bem analisada durante o projeto físico em relação aos requisitos de processamento para determinar se a eliminação da tabela substituída é razoável.

Para continuar nosso exemplo de banco de dados de pessoa e projeto da empresa, queremos obter as DFs primárias aplicando as regras da Tabela 6.1 a cada relacionamento no diagrama ER da Figura 4.3. Os resultados aparecem na Tabela 6.2.

Tabela 6.2 DFs primárias derivadas do diagrama ER da Figura 4.3

dept_no -> div_no	em Departamento do relacionamento "contém"
emp_id -> dept_no	em Funcionário do relacionamento "tem"
div_no -> emp_id	em Divisão do relacionamento "é-chefiado-por"
dept_no -> empt_id	do relacionamento binário "é-gerenciado-por"
emp_id -> desktop_no	do relacionamento binário "é-alocado"
desktop_no -> emp_no	do relacionamento binário "é-alocado"
emp_ip -> spouse_id	do relacionamento binário recursivo "é-casado-com"
Spouse_id -> emp_id	do relacionamento binário recursivo "é-casado-com"
emp_id, loc_name -> project_name	do relacionamento ternário "atribuída-a"

Em seguida, queremos determinar as DFs secundárias. Vamos considerar que as dependências na Tabela 6.3 são derivadas da especificação de requisitos e da intuição.

Tabela 6.3 DFs secundárias derivadas da especificação de requisitos

div_no -> div_name, div_addr	da entidade Divisão
dept_no -> dept_name, dept_addr, mgr_id	da entidade Departamento
emp_id -> emp_name, emp_addr, office_no, phone_no	da entidade Funcionário
skill_type -> skill_descrip	da entidade Habilidade
project_name -> start_date, end_date, head_id	da entidade Projeto
loc_name -> loc_county, loc_state, zip	da entidade Localização
mgr_id -> mgr_start_date, beeper_phone_no	da entidade Gerente
assoc_name -> assoc_addr, phone_no, start_date	da entidade Assoc-prof
desktop_no -> computer_type, serial_no	da entidade Desktop

A normalização das tabelas candidatas é realizada em seguida. Na Tabela 6.4, reunimos as DFs primárias e secundárias que se aplicam a cada tabela candidata. Observamos que, para cada tabela, exceto **employee**, todos os atributos são funcionalmente dependentes da chave primária (indicada pelo lado esquerdo das DFs) e, assim, estão na FNBC. No caso da tabela **employee**,

observamos que spouse_id determina emp_id e emp_id é a chave primária; assim, spouse_id pode ser mostrado como sendo uma superchave (ver a Regra 2 de Superchave na Seção 6.4). Portanto, descobrimos que **employee** está na FNBC.

Em geral, observamos que as tabelas candidatas, como as mostradas na Tabela 6.4, são indicadores muito bons do esquema final, e normalmente exigem muito pouco refinamento para chegar à 3FN ou FNBC. Essa observação é importante — um bom projeto conceitual inicial normalmente resulta em tabelas que já são normalizadas ou estão muito próximas de serem

Tabela 6.4 Tabelas candidatas (e DFs) da transformação do diagrama ER

division	div_no -> div_name, div_addr
	div_no -> emp_id
department	dept_no -> dept_name, dept_addr, mgr_id
	dept_no -> div_no
	dept_no -> emp_id
employee	emp_id -> emp_name, emp_addr, office_no, phone_no
	emp_id -> dept_no
	emp_id -> spouse_id
	spouse_id -> emp_id
manager	mgr_id -> mgr_start_date, beeper_phone_no
secretary	None
engineer	emp_id -> desktop_no
technician	None
skill	skill_type -> skill_descrip
project	Project_name -> start_date, end_date, head_id
location	loc_name -> loc_county, loc_state, zip
prof_assoc	assoc_name -> assoc_addr, phone_no, start_date
desktop	Desktop_no -> computer_type, serial_no
	Desktop_no -> emp_no
assigned_to	emp_id, loc_name -> project_name
skill_used	None

normalizadas, e por isso o processo de normalização normalmente é uma tarefa simples.

6.4 Determinando o conjunto mínimo de tabelas 3FN

Um conjunto mínimo de tabelas 3FN pode ser obtido a partir de um determinado conjunto de DFs, usando o algoritmo de síntese bem conhecido desenvolvido por Bernstein [1976]. Esse processo é particularmente útil quando você é confrontado com uma lista de centenas ou milhares de DFs que descrevem a semântica de um banco de dados. Na prática, o processo de modelagem ER decompõe automaticamente esse problema em subproblemas menores: os atributos e DFs de interesse são restritos aos atributos dentro de uma entidade (e sua tabela equivalente) e quaisquer chaves estrangeiras que poderiam ser impostas a essa tabela. Assim, um projetista de banco de dados raramente terá de lidar com mais de dez ou doze atributos de cada vez, e, de fato, a maioria das entidades já está definida inicialmente na 3FN. Para as tabelas que ainda não estão na 3FN, somente alguns pequenos ajustes serão necessários na maioria dos casos.

No texto a seguir, descrevemos rapidamente o algoritmo de síntese para as situações em que o modelo ER não é útil para a decomposição. Para aplicar o algoritmo, utilizamos os axiomas de Armstrong bem conhecidos, que definem os relacionamentos básicos entre as DFs.

Regras de inferência (axiomas de Armstrong)

Reflexividade	Se Y é um subconjunto dos atributos de X, então X -> Y (ou seja, se X é ABCE e Y é ABC, então X -> Y. Trivialmente, X -> X).
Aumento	Se X -> Y e Z é um subconjunto da tabela R (ou seja, Z é qualquer atributo em R), então XZ -> YZ.
Transitividade	Se X->Y e Y->Z, então X->Z.
Pseudotransitividade	Se X->Y e YW->Z, então XW->Z. (A transitividade é um caso especial de pseudotransitividade quando W = null.)
União	Se X->Y e X->Z, então X->YZ (ou, de forma equivalente, X->Y,Z).
Decomposição	Se X->YZ, então X->Y e X->Z.

Esses axiomas podem ser usados para derivar duas regras práticas e derivar superchaves de tabelas onde pelo menos uma superchave já é conhecida.

Regra de superchave 1. Qualquer DF que envolva todos os atributos de uma tabela define uma superchave como sendo o lado esquerdo da DF.
Dado: Uma DF contendo todos os atributos na tabela **R**(W,X,Y,Z), ou seja, XY -> WZ.
Prova:

1. XY -> WZ conforme dado no enunciado.
2. XY -> XY aplicando o axioma de reflexividade.
3. XY -> XYWZ aplicando o axioma de união.
4. XY determina exclusivamente cada atributo na tabela **R**, conforme mostramos em 3.
5. XY define exclusivamente a tabela **R**, pela definição de uma tabela como não tendo linhas duplicadas.
6. XY é, portanto, uma superchave, por definição.

Regra de superchave 2. Qualquer atributo que determina funcionalmente uma superchave de uma tabela também é uma superchave para essa tabela.
Dado: O atributo A é uma superchave para a tabela **R**(A,B,C,D,E), e E -> A.
Prova:

1. O atributo A define de forma única cada linha da tabela **R**, pela definição de uma superchave.
2. A -> ABCDE aplicando a definição de uma superchave e uma tabela relacional.
3. E -> A, conforme indicado no enunciado.
4. E -> ABCDE aplicando o axioma de transitividade.
5. E é uma superchave para a tabela **R**, por definição.

Antes de podermos descrever o algoritmo de síntese, é preciso definir alguns conceitos importantes. Considere que H seja um conjunto de DFs que representa pelo menos parte da semântica conhecida de um banco de dados. O fechamento de H, especificado por H+, é o conjunto de todas as DFs deriváveis de H usando os axiomas de Armstrong ou regras de inferência. Por exemplo, podemos aplicar a regra de transitividade às seguintes DFs no conjunto H:

A -> B, B -> C, A -> C e C -> D

para derivar as DFs A -> D e B -> D. Todas as seis DFs constituem o fechamento H+. Uma cobertura de H, chamada H', é qualquer conjunto de DFs da qual H+ pode ser derivado. Possíveis coberturas para esse exemplo são:

1. A->B, B->C, C->D, A->C, A->D, B->D (caso trivial em que H' e H+ são iguais)
2. A->B, B->C, C->D, A->C, A->D
3. A->B, B->C, C->D, A->C (esse é o conjunto original H)
4. A->B, B->C, C->D

Uma cobertura não-redundante de H é uma cobertura de H que não contém o subconjunto próprio de DFs, que também é uma cobertura. O algoritmo de síntese exige coberturas não-redundantes.

Algoritmo de síntese 3FN

Dado um conjunto de DFs, H, determinamos um conjunto mínimo de tabelas na 3FN.

H:	AB->C	DM->NP
	A->DEFG	D->M
	E->G	L->D
	F->DJ	PQR->ST
	G->DI	PR->S
	D->KL	

A partir deste ponto, o processo de chegar ao conjunto mínimo de tabelas 3FN consiste em cinco etapas:

1. Eliminar atributos estranhos nos lados esquerdos das DFs.
2. Procurar uma cobertura não-redundante, G de H
3. Dividir G em grupos de modo que todas as DFs com o mesmo lado esquerdo estejam num mesmo grupo.
4. Unificar chaves equivalentes.
5. Definir o conjunto mínimo de tabelas normalizadas.

Agora, discutimos cada etapa individualmente, em termos do conjunto anterior de DFs, H.

Etapa 1. Eliminar atributos estranhos

A primeira tarefa é eliminar atributos estranhos nos lados esquerdos das DFs.

Os dois relacionamentos a seguir (regras) entre os atributos no lado esquerdo de uma DF oferecem meios para reduzir a quantidade de atributos no lado esquerdo.

Regra de redução 1. XY->Z e X->Z => Y é estranho no lado esquerdo (aplicando os axiomas de reflexividade e transitividade).
Regra de redução 2. XY->Z e X->Y => Y é estranho; portanto, X->Z (aplicando o axioma de pseudotransitividade).
Aplicando essas **regras de redução** ao conjunto de DFs em H, obtemos*[4]:
DM->NP e D->M => D->NP
PQR->ST e PR->S => PQR->T

Etapa 2. Procurar uma cobertura não-redundante

Temos de eliminar qualquer DF derivável de outras em H usando as regras de inferência.

DFs transitivas a serem eliminadas:

A->E e E->G => eliminar A->G
A->F e F->D => eliminar A->D

Etapa 3. Dividir a cobertura não-redundante

Para dividir a cobertura não-redundante em grupos de modo que todas as DFs com o mesmo lado esquerdo estejam em um mesmo grupo, temos de separar as dependências funcionais não completas e as dependências transitivas em tabelas separadas. Neste ponto, temos uma solução viável para tabelas 3FN, mas esse não é necessariamente o conjunto mínimo.

Essas dependências funcionais não completas precisam ser colocadas em tabelas separadas:

AB->C
A->EF

---

* *Nota dos Revisores Técnicos*: PQR->ST ó PRQ->ST. PRQ->ST e PR->S => SQ->ST => Q->T. Assim, quaisquer atributos justapostos no lado esquerdo junto com Q manterão a DF válida; em especial PQR->T. No entanto, neste caso, não houve redução no lado esquerdo dos atributos estranhos como os autores pretendiam. Assim preferimos o Q->T

Grupos com o mesmo lado esquerdo:

G1: AB->C	G6: D->KLMNP
G2: A->EF	G7: L->D
G3: E->G	G8: PQR->T
G4: G->DI	G9: PR->S
G5: F->DJ	

Etapa 4. Unificar chaves equivalentes (unificação de tabelas)

Nesta etapa, unificamos os grupos com os lados esquerdos equivalentes (por exemplo, X->Y e Y->X implicam que X e Y são equivalentes). Essa etapa produz um conjunto mínimo.

1. Escreva o fechamento de todos os atributos do lado esquerdo a partir da Etapa 3, com base nas transitividades.
2. Usando os fechamentos, encontre tabelas que são subconjuntos de outros grupos e tente unificá-los. Use a regra de superchave 1 e a regra de superchave 2 para estabelecer se a unificação resultará em DFs com superchaves no lado esquerdo. Se não, tente usar os axiomas para modificar as DFs e adequar à definição das superchaves.
3. Depois que os subconjuntos forem esgotados, procure quaisquer sobreposições entre tabelas e aplique as regras de superchave 1 e 2 (e os axiomas) novamente.

Neste exemplo, observe que G7 (L->D) possui um subconjunto de G6 (D->KLMNP). Portanto, mesclamos a uma única tabela, com DFs D->KLMNP e L->D, pois isso satisfaz a 3FN: D é uma superchave pela regra de superchave 1, e L é uma superchave pela regra de superchave 2.

Etapa 5. Definição do conjunto mínimo de tabelas normalizadas

O conjunto mínimo de tabelas normalizadas agora foi calculado. Nós as definimos a seguir em termos do nome da tabela, os atributos da tabela, as DFs da tabela e as chaves candidatas para essa tabela.

R1: ABC (AB->C com chave AB)	R5: DFJ (F->DJ com chave F)
R2: AEF (A->EF com chave A)	R6: DKLMNP (D->KLMNP, L->D, com chaves D, L)

R3: EG (E->G com chave E)

R4: DGI (G->DI com chave G)

R7: PQRT (PQR->T com chave PQR)

R8: PRS (PR->S com chave PR)

Observe que esse resultado não está apenas 3FN, mas também está na FNBC, o que acontece com muita freqüência. Esse fato sugere um algoritmo prático para um conjunto mínimo (próximo) de tabelas FNBC: use o algoritmo de Bernstein para conseguir um conjunto mínimo de tabelas 3FN; depois inspecione cada tabela a fim de decompô-la para a FNBC (ou para incluir redundância parcial, como discutido na Seção 6.1.5).

6.5 Quarta e quinta formas normais

As formas normais até a FNBC foram definidas unicamente sobre DFs e, para a maioria dos profissionais de banco de dados, a 3FN ou a FNBC é um nível de normalização suficiente. Contudo, de fato existem mais duas formas normais necessárias para eliminar o restante das anomalias atualmente conhecidas. Nesta seção, veremos diferentes tipos de restrições sobre as tabelas: dependências multivaloradas e dependências de junção. Se essas restrições não existirem em uma tabela, que é a situação mais comum, então qualquer tabela em FNBC estará automaticamente na quarta forma normal (4FN) e também na quinta forma normal (5FN). No entanto, quando essas restrições existirem, poderá haver mais anomalias de atualização (especialmente de exclusão) que precisam ser corrigidas. Primeiro, temos de definir o conceito de dependência multivalorada.

6.5.1 Dependências multivaloradas

Definição. Em uma *dependência multivalorada (DMV)*, X->>Y está presente sobre a tabela **R** com esquema RS se, sempre que nas instâncias válidas da tabela **R**(X,Y,Z) tiver um par de linhas que contêm valores duplicados de X, então as instâncias também conterão um par de linhas obtidas pela troca dos valores Y no par original. Isso inclui situações em que existem apenas pares de linhas. Observe que X e Y podem conter atributos únicos ou compostos.

Uma DMV X ->> Y é trivial se Y for um subconjunto de X, ou se X união Y = RS. Finalmente, uma DF implica uma DMV, pois uma única

linha com determinado valor de X também é uma DMV, apesar de ser uma forma trivial.

Os exemplos a seguir mostram onde uma DMV existe e não existe em uma tabela. Na **R1**, as quatro primeiras linhas satisfazem a todas as condições para as DMVs X->>Y e X->>Z. Observe que as DMVs aparecem em pares, devido ao tipo de produto cruzado de relacionamento entre Y e Z=RS-Y como dois lados direitos das duas DMVs. A quinta e sexta linhas de **R1** (quando o valor de X é 2) satisfazem às condições de troca de linha na definição anterior. Em ambas as linhas, o valor Y é 2, de modo que a troca de valores Y é trivial. A sétima linha (3,3,3) satisfaz à definição de forma trivial.

Na Tabela **R2**, porém, os valores Y na quinta e sexta linhas são diferentes (1 e 2), e a troca dos valores 1 e 2 para Y resulta em uma linha (2,2,2) que não aparece na tabela. Assim, em **R2**, não existe DMV entre X e Y ou entre X e Z, embora as quatro primeiras linhas satisfaçam à definição da DMV. Observe que, para a DMV existir, todas as linhas precisam satisfazer ao critério para uma DMV.

A Tabela **R3** contém as três primeiras linhas que não satisfazem ao critério para uma DMV, pois a mudança de Y de 1 para 2 na segunda linha resulta em uma linha que não aparece na tabela. De modo semelhante, a mudança de Z de 1 para 2 na terceira linha resulta em uma linha que não aparece. Assim, **R3** não tem quaisquer DMVs entre X e Y ou entre X e Z.

R1:

X	Y	Z
1	1	1
1	1	2
1	2	1
1	2	2
2	2	1
2	2	2
3	3	3

R2:

X	Y	Z
1	1	1
1	1	2
1	2	1
1	2	2
2	2	1
2	1	2

R3:

X	Y	Z
1	1	1
1	1	2
1	2	1
2	2	1
2	2	2

Pelo mesmo argumento, na tabela **R1** temos as DMVs Y->> X e Y->>Z, mas nada com Z no lado esquerdo. As tabelas **R2** e **R3** não possuem DMV alguma.

As regras de inferência a seguir para DMVs são um tanto semelhantes às regras de inferência para dependências funcionais, dadas na Seção 6.4 [Beeri, Fagin e Howard, 1977]. Elas são muito úteis na análise e decomposição de tabelas para a 4FN.

Regras de inferência de dependência multivalorada

Reflexividade	X —>> X
Aumento	Se X —>> Y, então XZ —>> Y.
Transitividade	Se X —>> Y e Y —>> Z, então X —>> (Z-Y).
Pseudotransitividade	*Se* X —>> Y e YW —>> Z, então XW —>> (Z-YW). (Transitividade é um caso especial de pseudotransitividade quando W é nulo.)
União	Se X —>> Y e X —>> Z, então X —>> YZ.
Decomposição	Se X —>> Y e X —>> Z, então X —>> Y intersecção Z e X —>> (Z-Y).
Complemento	Se X —>> Y e Z=R-X-Y, então X —>> Z.
DF implica DMV	Se X -> Y, então X —>> Y.
DF, DMV Mix	Se X —>> Z e Y —>> Z' (onde Z' está contido em Z, e Y e Z são disjuntos), então X->Z'.

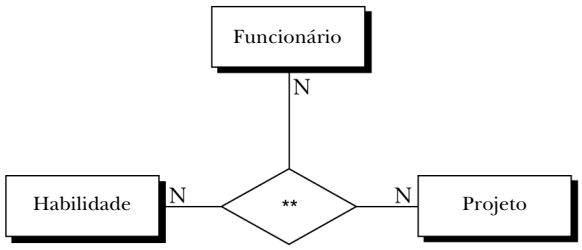
6.5.2 Quarta forma normal

O objetivo da 4FN é eliminar as DMVs não triviais de uma tabela projetando-as para tabelas menores separadas, e assim, eliminar as anomalias de atualização associadas às DMVs. Esse tipo de forma normal é razoavelmente fácil de alcançar se você souber onde estão as DMVs. Em geral, as DMVs precisam ser definidas a partir da semântica do banco de dados; elas não podem ser determinadas apenas examinando-se os dados. O conjunto atual de dados só pode verificar se a sua suposição sobre uma DMV é atualmente verdadeira ou não, mas isso pode mudar toda vez que os dados são atualizados.

> ***Definição.*** Uma tabela **R** está na *quarta forma normal (4FN)* se, e somente se, estiver na FNBC e, sempre que existir uma DMV em **R** (digamos, X->>Y), pelo menos uma das seguintes condições for verdadeira: a DMV é trivial, ou X é uma superchave para **R**.

Aplicando essa definição às três tabelas no exemplo da seção anterior, vemos que **R1** não está na 4FN porque existe pelo menos uma DMV não trivial e nenhuma coluna isolada é uma superchave. Nas tabelas **R2** e **R3**, porém, não existem DMVs. Assim, essas duas tabelas estão pelo menos na 4FN.

Como exemplo da transformação de uma tabela que não está na 4FN em duas tabelas que estão na 4FN, observamos o relacionamento ternário skill_required (habilidade_exigida), mostrado na Figura 6.6. O relacionamento



** (1) skill-required (habilidade-exigida)
(2) skill-in-common (habilidade-em-comum)
(3) skill-used (habilidade-usada)

Figura 6.6 Relacionamento ternário com múltiplas interpretações.

skill_required é definido da seguinte maneira: "Um funcionário precisa ter todas as habilidades exigidas necessárias para um projeto funcionar nesse projeto". Por exemplo, na Tabela 6.5, o projeto com proj_no = 3 exige os tipos de habilidade A e B por todos os funcionários (ver funcionários 101 e 102). A tabela **skill_required** não possui DFs, mas possui várias DMVs não triviais, e portanto está apenas na FNBC. Nesse caso, ela pode ter uma decomposição sem perdas para dois relacionamentos binários muitos-para-muitos entre as entidades Funcionário e Projeto, e Projeto e Habilidade. Cada um desses dois novos relacionamentos representa uma tabela na 4FN. Ela também pode ter uma decomposição sem perdas resultante de um relacionamento binário muitos-para-muitos entre as entidades Funcionário e Habilidade, e Projeto e Habilidade.

Uma decomposição sem perdas bidirecional (em duas vias) ocorre quando **skill_required** é projetada sobre (emp_ip, proj_no) para formar skill_req1 e projetada sobre (proj_no, skill_type) para formar skill_req3. Porém, a projeção sobre (emp_id, proj_no) para formar skill_req1 e sobre (emp_id, skill_type) para formar skill_req2 não é sem perdas. Uma decomposição sem perdas de três vias ocorre quando **skill_required** é projetada sobre (emp_id, proj_no), (emp_id, skill_type) e (proj_no, skill_type).

As tabelas em 4FN evitam certas anomalias de atualização (ou ineficiências). Por exemplo, existe uma anomalia de exclusão quando dois fatos independentes são reunidos de modo não natural, de modo que pode haver efeitos colaterais em certas exclusões. Por exemplo, em **skill_required**, a última linha de um skill_type pode ser perdida se um funcionário temporariamente não estiver trabalhando em quaisquer projetos. Uma ineficiência de atualiza-

Tabela 6.5 A tabela **skill_required** e suas três projeções

skill_required	emp_id	proj_no	Skill_type	*DMVs(nontrivial)*
	101	3	A	proj_no ->> skill_type
	101	3	B	proj_no ->> emp_id
	101	4	A	
	101	4	C	
	102	3	A	
	102	3	B	
	103	5	D	

skill_req1	skill_req2	skill_req3			
emp_id	proj_no	emp_id	Skill_type	proj_no	skill_type
101	3	101	A	3	A
101	4	101	B	3	B
102	3	101	C	4	A
103	5	102	A	4	C
		102	B	5	D
		103	D		

ção pode ocorrer quando se inclui um novo projeto em **skill_required**, que exige que as inserções para muitas linhas incluam todas as habilidades exigidas para esse novo projeto. De modo semelhante, a perda de um projeto exige muitas exclusões. Essas ineficiências são evitadas quando **skill_required** é decomposto em **skill_req1** e **skill_req3**. Em geral, mas nem sempre, a decomposição de uma tabela em tabelas 4FN resulta em menos redundância de dados.

6.5.3 Decomposição de tabelas para 4FN

Os algoritmos para decompor tabelas para 4FN são difíceis de desenvolver. Vejamos algumas técnicas simples para 4FN a partir da FNBC e formas normais inferiores. Primeiro, se uma tabela estiver na FNBC, ela ou não tem DFs ou cada DF é caracterizada por seu lado esquerdo sendo uma superchave. Assim, se as únicas DMVs nessa tabela forem derivadas de suas DFs, elas têm apenas superchaves como seus lados esquerdos, e a tabela é 4FN por definição. Todavia, se houver outras DMVs não triviais cujos lados esquerdos não

são superchaves, a tabela estará apenas na FNBC e precisará ser decomposta para atingir a normalização mais alta.

O processo básico de decomposição de uma tabela FNBC é definido selecionando a DMV mais importante (ou, se isso não for possível, então selecionando uma DMV qualquer), definindo sua DMV de complemento e decompondo a tabela em duas tabelas contendo os atributos tanto do lado esquerdo quanto do direito dessa DMV e de seu complemento. Esse tipo de decomposição é sem perdas porque cada nova tabela é baseada no mesmo atributo, que é o lado esquerdo de ambas as DMVs. As mesmas DMVs nessas novas tabelas agora são triviais, pois contêm apenas os seus atributos na tabela. Entretanto, outras DMVs ainda podem estar presentes, e mais decomposições por DMVs e seus complementos podem ser necessárias. Esse processo de seleção arbitrária de DMVs para decomposição é continuado até que existam somente DMVs triviais, deixando as tabelas finais na 4FN.

Como exemplo, considere **R**(A,B,C,D,E,F) sem DFs, e com as DMVs A ->> B e CD ->> EF. A primeira decomposição de **R** é realizada em duas tabelas **R1**(A,B) e **R2**(A,C,D,E,F), aplicando a DMV A ->> B e seu complemento A ->> CDEF. A Tabela **R1** agora está na 4FN, pois A ->> B é trivial e é a única DMV na tabela. A Tabela **R2**, porém, ainda está apenas na FNBC, devido à DMV não trivial CD ->> EF. Depois, decompomos **R2** em **R21**(C,D,E,F) e **R22**(C,D,A) aplicando a DMV CD ->> EF e seu complemento CD ->> A. Tanto **R21** quanto **R22** agora estão na 4FN. Se tivéssemos aplicado a regra de complemento de DMV na ordem oposta, usando CD ->> EF e seu complemento CD ->> AB primeiro, as mesmas três tabelas 4FN resultariam desse método. Porém, isso não ocorre em todos os casos; só ocorre nas tabelas em que as DMVs não possuem atributos de interseção.

Esse método, em geral, infelizmente, tem o efeito colateral de potencialmenteperder algumas ou todas as DFs e DMVs. Portanto, qualquer decisão de transformar tabelas de FNBC para 4FN precisa levar em conta a escolha entre normalização e a eliminação de anomalias de exclusão, e a preservação de DFs e possivelmente DMVs. Também devemos observar que essa técnica deriva um conjunto viável, mas não necessariamente mínimo, de tabelas em 4FN.

Uma segunda técnica para a decomposição de tabelas FNBC é ignorar completamente as DMVs e dividir cada tabela FNBC em um conjunto de tabelas menores, com a chave candidata de cada tabela FNBC sendo a chave candidata de uma nova tabela e os atributos não-chaves distribuídos entre as novas tabelas de alguma maneira semanticamente significativa. Essa forma de decomposição por chave candidata (ou seja, superchave) é sem perdas por-

que as chaves candidatas se juntam de forma exclusiva; isso, em geral, resulta na forma mais simples de tabelas 5FN, aquelas com uma chave candidata e um atributo não-chave, e sem DMVs. Contudo, se uma ou mais DMVs ainda existirem, mais decomposição deverá ser feita com a técnica de complemento DMV/DMV dada anteriormente. A decomposição por chaves candidatas preserva as DFs, mas a técnica complemento DMV/DMV não preserva as DFs e nem DMVs.

As tabelas que ainda não estão em FNBC também podem ser decompostas diretamente em 4FN usando a técnica de complemento DMV/DMV. Essas tabelas normalmente podem ser decompostas em conjuntos mínimos menores do que aqueles derivados inicialmente pela transformação para FNBC e depois 4FN, mas com um custo maior de DFs pedidas. Na maioria das situações de projeto de banco de dados, é preferível desenvolver primeiro as tabelas na FNBC e depois avaliar a necessidade de normalizar ainda mais enquanto se preserva as DFs.

6.5.4 Quinta forma normal

Definição. Uma tabela **R** está na *quinta forma normal (5FN)* ou forma normal projeção-junção (FN/PJ) se, e somente se, cada dependência de junção em **R** for inferida pelas chaves de **R**.

Como você pode se lembrar, uma decomposição sem perdas de uma tabela implica que ela pode ser decomposta por duas ou mais projeções, seguidas por uma junção natural dessas projeções (em qualquer ordem), o que resulta na tabela original, sem quaisquer linhas falsas ou em falta. A restrição geral da decomposição sem perdas, envolvendo qualquer quantidade de projeções, também é conhecida como *dependência de junção* (DJ). Uma dependência de junção é ilustrada pelo seguinte exemplo: em uma tabela **R** com *n* subconjuntos arbitrários do conjunto de atributos de **R**, **R** satisfaz a uma dependência de junção por esses *n* subconjuntos se, e somente se, **R** for igual à junção natural de suas projeções. Uma DJ é trivial se um dos subconjuntos for a própria tabela **R**.

A 5FN ou FN/PJ exige que se satisfaça ao algoritmo de participação [Fagin, 1979], o qual determina se uma DJ é membro do conjunto de conseqüências lógicas do (pode ser derivado do) conjunto de dependências chave conhecidas para essa tabela. Com efeito, para qualquer tabela 5FN, cada dependência (DF, DMV, DJ) é determinada pelas chaves. Como uma questão prática, observamos que, como as DJs são muito difíceis de determinar em bancos de dados

Tabela 6.6 A tabela **skill_in_common** e suas três projeções

skill_in_common	emp_id	proj_no	skill_type
	101	3	A
	101	3	B
	101	4	A
	101	4	B
	102	3	A
	102	3	B
	103	3	A
	103	4	A
	103	5	A
	103	5	C

skill_in_com1		**skill_in_com2**		**skill_in_com3**	
emp_id	proj_no	emp_id	skill_type	proj_no	skill_type
101	3	101	A	3	A
101	4	101	B	3	B
102	3	102	A	4	A
103	3	102	B	4	B
103	4	103	A	5	A
103	5	103	C	5	C

grandes com muitos atributos, as tabelas 5FN não são facilmente deriváveis, e o projeto lógico de banco de dados normalmente produz tabelas FNBC.

Também devemos observar que, pelas definições anteriores, só porque uma tabela é passível de decomposição não necessariamente significa que ela não esteja na 5FN. Por exemplo, considere uma tabela simples com quatro atributos (A,B,C,D), uma DF (A->BCD) e nenhuma DMV ou DJ não inferida por essa DF. Ela pode ser decomposta em três tabelas, A->B, A->C e A->D, todas baseadas na mesma superchave A; porém, ela já está na 5FN sem a decomposição. Assim, a decomposição não é exigida para a normalização. Por outro lado, a decomposição pode ser uma ferramenta útil em alguns casos para melhorar o desempenho.

O exemplo a seguir demonstra que uma tabela representando um relacionamento ternário pode não ter quaisquer decomposições sem perdas de duas

vias; porém, ele pode ter uma decomposição sem perdas de três vias, o que é equivalente a três relacionamentos binários, com base nas três projeções possíveis dessa tabela. Essa situação ocorre no relacionamento skill-in-common (Figura 6.6), que é definido como "O funcionário precisa aplicar a interseção de suas habilidades disponíveis com as habilidades necessárias para trabalhar em certos projetos". Nesse exemplo, skill-in-common é menos restritivo do que skill-required, pois permite que um funcionário trabalhe em um projeto mesmo que não tenha todas as habilidades exigidas para esse projeto.

Conforme mostra a Tabela 6.6, as três projeções de **skill_in_common** resultam em uma decomposição sem perdas de três vias. Não existem decomposições sem perdas de duas vias nem DMVs; assim, a tabela **skill-in-common** está na 4FN.

O relacionamento ternário na Figura 6.6 ainda pode ser interpretado de outra maneira. O significado do relacionamento skill-used é "Podemos registrar seletivamente diferentes habilidades que cada funcionário aplica para trabalhar em projetos individuais". Isso é equivalente a uma tabela na 5FN que não pode ser decomposta em duas ou três tabelas binárias. Observe, estudando a Tabela 6.7, que a tabela associada, **skill_used**, não possui DMVs ou DJs.

Uma tabela pode ter restrições que são DFs, DMVs ou DJs. Uma DMV é um caso especial de uma DJ. Para determinar o nível de normalização da tabela, analise as DFs primeiro para determinar a normalização pela FNBC; depois, analise as DMVs para determinar quais tabelas FNBC também são 4FN; depois, finalmente, analise as DJs para determinar quais tabelas 4FN também são 5FN.

Tabela 6.7 A tabela **skill_used**, suas três projeções e junções naturais de suas projeções

skill_used	emp_id	proj_no	skill_type
	101	3	A
	101	3	B
	101	4	A
	101	4	C
	102	3	A
	102	3	B
	102	4	A
	102	4	B

Tabela 6.7 A tabela skill_used, suas três projeções e junções naturais de suas projeções *(continuação)*

Três projeções sobre **skill_used** resultam em:

skill_used1

emp_id	proj_no
101	3
101	4
102	3
102	4

skill_used2

proj_no	skill_type
3	A
3	B
4	A
4	B
4	C

skill_used3

emp_id	skill_type
101	A
101	B
101	C
102	A
102	B

junte **skill_used1** com **skill_used2** para formar:

skill_used_12

emp_id	proj_no	skill_type
101	3	A
101	3	B
101	4	A
101	4	B
101	4	C
102	3	A
102	3	B
102	4	A
102	4	B
102	4	C

junte **skill_used12** com **skill_used3** para formar:

skill_used_123

emp_id	proj_no	skill_type
101	3	A
101	3	B
101	4	A
101	4	B (falso)
101	4	C
102	3	A
102	3	B
102	4	A
102	4	B

Um relacionamento ternário muitos-para-muitos-para-muitos está na:

- FNBC se puder ser substituído por dois relacionamentos binários
- 4FN se só puder ser substituído por três relacionamentos binários
- 5FN se não puder ser substituído de maneira alguma (e, portanto, se for um relacionamento ternário verdadeiro)

Observamos a equivalência entre certos relacionamentos ternários e as tabelas em formas normais mais elevadas transformadas a partir desses relaci-

onamentos. Os relacionamentos ternários que têm pelo menos uma entidade "um" não podem ser decompostos (ou desmembrados) em relacionamentos binários, pois isso destruiria as DFs (uma ou mais) exigidas na definição, como mostramos anteriormente. No entanto, um relacionamento ternário com todas as entidades "muitos" não possui DFs, mas, em alguns casos, pode ter DMVs, e por isso têm uma decomposição sem perdas para relacionamentos binários equivalentes.

Resumindo, os três casos comuns que ilustram a correspondência entre uma decomposição sem perdas em uma tabela de relacionamento ternário muitos-para-muitos-para-muitos e formas normais mais elevadas no modelo relacional são mostrados na Tabela 6.8.

Tabela 6.8 Resumo das formas normais mais altas

Nome da tabela	*Forma normal*	*Decomp/junção sem perdas de duas vias?*	*Decomp/junção sem perdas de três vias?*	*DMVs não triviais*
skill_required	FNBC	sim	sim	2
skill_in_common	4FN	não	sim	0
skill_used	5FN	não	não	0

6.6 Resumo

Neste capítulo, definimos as restrições normalmente impostas sobre tabelas — as dependências funcionais, ou DFs. Com base nessas restrições, as formas normais aplicáveis às tabelas de banco de dados foram definidas: 1FN, 2FN, 3FN e FNBC. Todas foram baseadas nos tipos de DFs existentes. Neste capítulo, foi dado um algoritmo prático para localizar o conjunto mínimo de tabelas na 3FN.

As seguintes afirmações resumem a equivalência funcional entre o modelo ER e as tabelas normalizadas:

1. *Interna a uma entidade.* O nível de normalização é totalmente dependente dos relacionamentos entre os atributos-chave e não-chave. Poderia ser qualquer forma, desde a não normalizada até a FNBC ou maior.
2. *Relacionamento binário (ou binário recursivo) um-para-um ou um-para-muitos.* Interna à entidade "filho", a chave estrangeira (uma cópia da chave pri-

mária do "pai") é funcionalmente dependente da chave primária do filho. Isso garante ao menos a FNBC, supondo que a entidade por si só, sem a chave estrangeira, já esteja na FNBC.

3. *Relacionamento binário (ou binário recursivo) muitos-para-muitos.* A tabela de interseção tem uma chave composta e possivelmente alguns atributos não-chave funcionalmente dependentes dela. Isso garante a FNBC.
4. *Relacionamento ternário.*
   a. um-para-um-para-um => três chaves compostas sobrepostas, garante pelo menos a FNBC
   b. um-para-um-para-muitos => duas chaves compostas sobrepostas, garante pelo menos a FNBC
   c. um-para-muitos-para-muitos => uma chave composta, garante pelo menos a FNBC
   d. muitos-para-muitos-para-muitos => uma chave composta com três atributos, garante pelo menos a FNBC; em alguns casos, também pode garantir a 4FN, ou mesmo a 5FN

Resumindo, observamos que um procedimento de projeto conceitual bom e metódico normalmente resulta em tabelas de banco de dados já normalizadas (FNBC) ou que podem ser normalizadas com pouquíssimas mudanças.

6.7 Resumo da literatura

Bons resumos de formas normais podem ser encontrados em Date [1999], Kent [1983], Dutka e Hanson [1989] e Smith [1985]. Algoritmos para técnicas de decomposição e síntese de forma normal são apresentados em Bernstein [1976], Fagin [1977] e Maier [1983]. O trabalho mais antigo sobre formas normais foi feito por Codd [1970, 1974].

7 Exemplo de projeto lógico de banco de dados

O exemplo a seguir ilustra como prosseguir, de forma prática, com as etapas de análise de requisitos e projeto lógico do ciclo de vida do banco de dados para um banco de dados relacional.

7.1 Especificação de requisitos

A gerência de uma grande loja comercial gostaria que um banco de dados acompanhasse as atividades de vendas. A análise de requisitos desse banco de dados levou a seis entidades e seus identificadores exclusivos, mostrados na Tabela 7.1.

As seguintes afirmações descrevem os relacionamentos dos dados:

- Cada cliente tem um cargo, mas diferentes clientes podem ter o mesmo cargo.
- Cada cliente pode fazer muitos pedidos, mas somente um cliente pode ter feito um pedido específico.
- Cada departamento possui muitos vendedores, mas cada vendedor precisa trabalhar em somente um departamento.
- Cada departamento possui muitos itens à venda, mas cada item é vendido em apenas um departamento ("Item" significa tipo de item, como IBM PC).
- Para cada pedido, os itens pedidos em diferentes departamentos precisam envolver diferentes vendedores, mas todos os itens pedidos dentro de um departamento precisam ser tratados por exatamente um vende-

Tabela 7.1 Resultados da análise de requisitos

Entidade	*Chave da entidade em caracteres*	*Tamanho da chave (máx.)*	*Número de ocorrências*
Customer	Cust-no	6	80.000
Job	job-no	24	80
Order	order-no	9	200.000
Salesperson	sales-id	20	150
Department	Dept-no	2	10
Item	Item-no	6	5.000

dor. Em outras palavras, para cada pedido, cada item tem exatamente um vendedor; e para cada pedido, cada departamento tem exatamente um vendedor.

Para o projeto físico (por exemplo, métodos de acesso etc.), é necessário determinar que tipo de processamento precisa ser feito sobre os dados; isto é, quais são as consultas e atualizações necessárias para satisfazer aos requisitos do usuário e quais são suas freqüências? Além disso, a análise de requisitos deverá determinar se haverá crescimento substancial do banco de dados (ou seja, volumétrico); em que espaço de tempo esse crescimento ocorrerá; e se a freqüência e o tipo das consultas e atualizações mudarão também. O decaimento e o crescimento devem ser estimados, pois cada um terá efeito significativo nos estágios posteriores do projeto do banco de dados.

7.1.1 Problemas de projeto

1. Usando a informação dada e, em particular, as cinco afirmações, derive um modelo de dados conceitual e um conjunto de dependências funcionais (DFs) que representem todos os relacionamentos de dados conhecidos.
2. Transforme o modelo de dados conceitual em um conjunto de tabelas SQL candidatas. Apresente as tabelas, suas chaves primárias e seus atributos.
3. Encontre o conjunto mínimo de tabelas normalizadas (FNBC) que sejam funcionalmente equivalentes às tabelas candidatas.

7.2 Projeto lógico

Nosso primeiro passo é desenvolver um diagrama do modelo de dados conceitual e um conjunto de DFs correspondentes a cada uma das afirmações dadas. A Figura 7.1 apresenta o diagrama do modelo ER, e a Figura 7.2 mostra o diagrama equivalente em UML. Normalmente, o modelo de dados conceitual é desenvolvido sem que se conheçam todas as DFs, mas, neste exemplo, os atributos não-chave foram omitidos, para que o banco de dados possa ser representado com apenas algumas declarações e DFs. O resultado dessa análise, relativo a cada uma das afirmações dadas, aparece na Tabela 7.2.

As tabelas candidatas exigidas na representação da semântica desse problema podem ser derivadas com facilidade a partir dos construtores de entidades e relacionamentos. As chaves primárias e as chaves estrangeiras são definidas explicitamente.

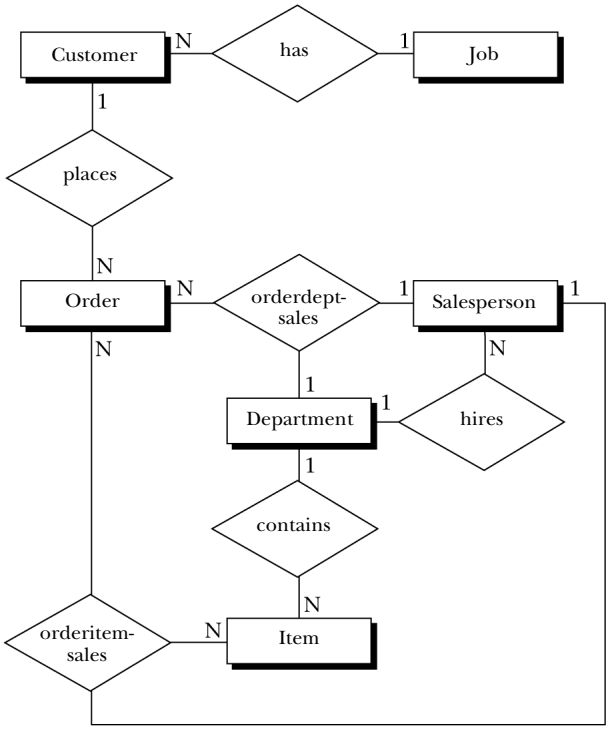


Figura 7.1 Diagrama do modelo de dados conceitual para o modelo ER

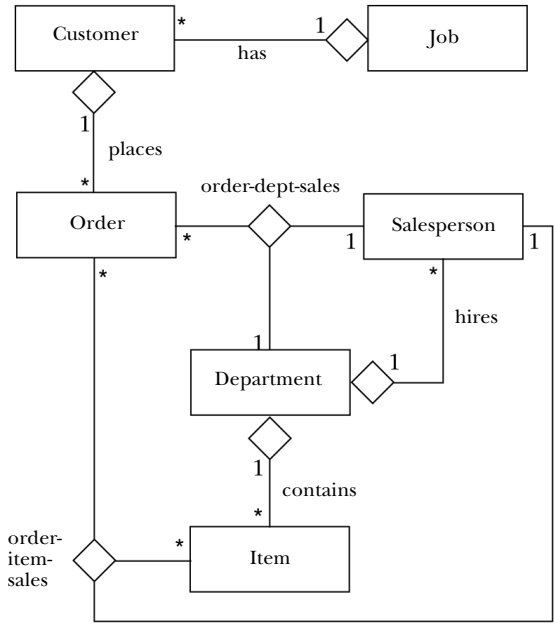


Figura 7.2 Diagrama do modelo de dados conceitual para UML

Tabela 7.2 Resultados da análise do modelo de dados conceitual

Construção ER	*DFs*
Customer(muitos): Job(um)	cust-no -> job-title
Order(muitos): Customer(um)	order-no -> cust-no
Salesperson(muitos): Department(um)	sales-id -> dept-no
Item(muitos): Department(um)	item-no -> dept-no
Order(muitos): Item(muitos): Salesperson(um)	order-no,item-no->sales-id
Order(muitos): Department(muitos): Salesperson(um)	order-no,dept-no-> sales-id

```
create table customer
   (cust_no        char(6),
   job_title       varchar(256),
   primary key (cust_no),
   foreign key (job_title) references job
       on delete set null on update cascade);

create table job
   (job_no         char(6),
   job_title       varchar(256),
   primary key (job_no));

create table order
   (order_no       char(9),
   cust_no         char(6) not null,
   primary key (order_no),
   foreign key (cust_no) references customer
       on delete set null on update cascade);

create table salesperson
   (sales_id       char(10)
   sales_name      varchar(256),
   dept_no         char(2),
   primary key (sales_id),
   foreign key (dept_no) references department
       on delete set null on update cascade);

create table department
   (dept_no        char(2),
   dept_name       varchar(256),
   manager_name    varchar(256),
   primary key (dept_no));

create table item
   (item_no        char(6),
   dept_no         char(2),
   primary key (item_no),
   foreign key (dept_no) references department
       on delete set null on update cascade);
```

```
create table order_item_sales
   (order_no       char(9),
   item_no         char(6),
   sales_id        varchar(256) not null,
   primary key (order_no, item_no),
   foreign key (order_no) references order
       on delete cascade on update cascade,
   foreign key (item_no) references item
       on delete cascade on update cascade,
   foreign key (sales_id) references salesperson
       on delete cascade on update cascade);

create table order_dept_sales
   (order_no       char(9),
   dept_no         char(2),
   sales_id        varchar(256) not null,
   primary key (order_no, dept_no),
   foreign key (order_no) references order
       on delete cascade on update cascade,
   foreign key (dept_no) references department
       on delete cascade on update cascade,
   foreign key (sales_id) references salesperson
       on delete cascade on update cascade);
```

Observe que normalmente é melhor colocar definições de chave estrangeira em instruções separadas (alter). Isso impede a possibilidade de obter definições circulares em esquemas muito grandes.

Esse processo de decomposição e redução de tabelas nos leva para mais perto de um conjunto mínimo de tabelas normalizadas (FNBC), conforme mostra a Tabela 7.3.

As reduções apresentadas nesta seção diminuíram o espaço de armazenamento e o custo de atualização e mantiveram a normalização na FNBC (e, portanto, na 3FN). Por outro lado, o custo de recuperação é potencialmente mais alto – por exemplo, na transação "listar todos os job_titles" – e aumentamos o potencial de perda de integridade, pois eliminamos tabelas simples, com apenas atributos-chave. A resolução dessas decisões depende de suas prioridades em relação ao seu banco de dados.

Tabela 7.3 Decomposição e redução de tabelas

Tabela	*Chave primária*	*Prováveis não-chaves*
customer	cust_no	job_title, cust_name, cust_address
order	order_no	cust_no, item_no, date_of_purchase, price
salesperson	sales_id	dept_no, sales_name, phone_no
item	item_no	dept_no, color, model_no
order_item_sales	order_no,item_no	sales_id
order_dept_sales	order_no,dept_no	sales_id

Os detalhes de indexação não serão abordados aqui. Contudo, durante a fase de projeto lógico da definição de tabelas SQL, faz sentido começar a considerar onde os índices serão criados. No mínimo, todas as chaves primárias e todas as chaves estrangeiras devem ser indexadas. Os índices são relativamente fáceis de implementar e armazenar e fazem uma diferença significativa na redução do tempo de acesso aos dados armazenados.

7.3 Resumo

Neste capítulo, desenvolvemos um esquema conceitual global e um conjunto de tabelas SQL para um banco de dados relacional, dada a especificação de requisitos de um banco de dados comercial. O exemplo ilustra as etapas do ciclo de vida do banco de dados: modelagem de dados conceitual, projeto do esquema global, transformação para tabelas SQL e normalização dessas tabelas. Ele resume as técnicas apresentadas nos Capítulos de 1 a 6.