Elmasri • Navathe

Sistemas de banco de dados

6ª edição

ALWAYS LEARNING

PEARSON

Elmasri • Navathe

Sistemas de banco de dados

6ª edição

Ramez Elmasri
Departamento de Ciência da Computação e Engenharia
Universidade do Texas em Arlington
Shamkant B. Navathe
Faculdade de Computação
Georgia Institute of Technology

Tradução:
Daniel Vieira
Revisão técnica:
Enzo Seraphim
Thatyana de Faria Piola Seraphim

Professores Doutores do Instituto de Engenharia de Sistemas e Tecnologias da Informação — Universidade Federal de Itajubá

PEARSON

Tradução autorizada a partir da edição original, em inglês, *Fundamentals of Database Systems*, 6th edition, publicada pela Pearson Education, Inc., sob o selo Addison-Wesley.

Diretor editorial: Roger Trimer
Gerente editorial: Sabrina Cairo
Supervisor de produção editorial: Marcelo Françozo
Editora plena: Thelma Babaoka
Editora de texto: Sabrina Levensteinas
Preparação: Paula Brandão Perez Mendes
Revisão: Elisa Andrade Buzzo
Capa: Thyago Santos sobre o projeto original de Lou Gibbs/Getty Images
Projeto gráfico e diagramação: Globaltec Artes Gráficas Ltda.

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

Elmasri, Ramez

Sistemas de banco de dados / Ramez Elmasri e Shamkant B. Navathe ; tradução Daniel Vieira ; revisão técnica Enzo Seraphim e Thatyana de Faria Piola Seraphim. -- 6. ed. -- São Paulo : Pearson Addison Wesley, 2011.

Título original: Fundamentals of database systems.

ISBN 978-85-4301-381-7

1. Banco de dados I. Navathe, Shamkant B..II. Título.

10-11462 CDD-005.75

Índices para catálogo sistemático:
1. Banco de dados : Sistemas : Processamento de dados 005.75
2. Banco de dados : Fundamentos : Processamento de dados 005.75

3ª reimpressão – Julho 2014

Rua Nelson Francisco, 26
CEP 02712-100 – São Paulo – SP – Brasil
Fone: (11) 2178-8686 – Fax: (11) 2178-8688
vendas@pearson.com

capítulo 4

SQL básica

A linguagem SQL pode ser considerada um dos principais motivos para o sucesso dos bancos de dados relacionais comerciais. Como ela se tornou um padrão para esse tipo de bancos de dados, os usuários ficaram menos preocupados com a migração de suas aplicações de outros tipos de sistemas de banco de dados — por exemplo, sistemas de rede e hierárquicos — para sistemas relacionais. Isso aconteceu porque, mesmo que os usuários estivessem insatisfeitos com o produto de SGBD relacional em particular que estavam usando, a conversão para outro produto de SGBD relacional não seria tão cara ou demorada, pois os dois sistemas seguiam os mesmos padrões de linguagem. Na prática, é óbvio, existem muitas diferenças entre diversos pacotes de SGBD relacionais comerciais. Porém, se o usuário for cuidadoso em usar apenas dos recursos que fazem parte do padrão, e se os dois sistemas relacionais admitirem fielmente o padrão, então a conversão entre ambos deverá ser bastante simplificada. Outra vantagem de ter esse padrão é que os usuários podem escrever comandos em um programa de aplicação de banco de dados que pode acessar dados armazenados em dois ou mais SGBDs relacionais sem ter de mudar a sublinguagem de banco de dados (SQL) se os sistemas admitirem o padrão SQL.

Este capítulo apresenta os principais recursos do padrão SQL para SGBDs relacionais *comerciais*, enquanto o Capítulo 3 apresentou os conceitos mais importantes por trás do modelo de dados relacional *formal*. No Capítulo 6 (seções 6.1 a 6.5), discutiremos as operações da *álgebra relacional*, que são muito importantes para entender os tipos de solicitações que podem ser especificadas em um banco de dados relacional. Elas também são importantes para processamento e otimização de consulta em um SGBD relacional, conforme veremos no Capítulo 19. No entanto, as operações da álgebra relacional são consideradas muito técnicas para a maioria dos usuários de SGBD comercial, pois uma consulta em álgebra relacional é escrita como uma sequência de operações que, quando executadas, produz o resultado exigido. Logo, o usuário precisa especificar como — ou seja, *em que ordem* — executar as operações de consulta. Por sua vez, a linguagem SQL oferece uma interface de linguagem *declarativa* de nível mais alto, de modo que o usuário apenas especifica *qual* deve ser o resultado, deixando a otimização real e as decisões sobre como executar a consulta para o SGBD. Embora a SQL inclua alguns recursos da álgebra relacional, ela é baseada em grande parte no *cálculo relacional de tupla*, que descrevemos na Seção 6.6. Porém, a sintaxe SQL é mais fácil de ser utilizada do que qualquer uma das duas linguagens formais.

O nome **SQL** hoje é expandido como Structured Query Language (Linguagem de Consulta Estruturada). Originalmente, SQL era chamada de SEQUEL (Structured English QUEry Language) e foi criada e implementada na IBM Research como a interface para um sistema de banco de dados relacional experimental, chamado SYSTEM R. A SQL agora é a linguagem padrão para SGBDs relacionais comerciais. Um esforço conjunto entre o American National Standards Institute (ANSI) e a International Standards Organization (ISO) levou a uma versão-padrão da SQL (ANSI, 1986), chamada SQL-86 ou SQL1. Um padrão revisado e bastante expandido, denominado SQL-92 (também conhecido como SQL2) foi desenvolvido mais tarde. O próximo padrão reconhecido foi SQL:1999, que começou como SQL3. Duas atualizações posteriores ao padrão são SQL:2003 e SQL:2006, que acrescentaram recursos de XML (ver Capítulo 12) entre outras atualizações para a linguagem. Outra atualização em 2008 incorporou mais

recursos de banco de dados de objeto na SQL (ver Capítulo 11). Tentaremos abordar a última versão da SQL ao máximo possível.

SQL é uma linguagem de banco de dados abrangente: tem instruções para definição de dados, consultas e atualizações. Logo, ela é uma DDL *e* uma DML. Além disso, ela tem facilidades para definir visões sobre o banco de dados, para especificar segurança e autorização, para definir restrições de integridade e para especificar controles de transação. Ela também possui regras para embutir instruções SQL em uma linguagem de programação de uso geral, como Java, COBOL ou C/C++.[1]

Os padrões SQL mais recentes (começando com **SQL:1999**) são divididos em uma especificação **núcleo** mais **extensões** especializadas. O núcleo deve ser implementado por todos os fornecedores de SGBDR que sejam compatíveis com SQL. As extensões podem ser implementadas como módulos opcionais a serem adquiridos independentemente para aplicações de banco de dados específicas, como mineração de dados, dados espaciais, dados temporais, data warehousing, processamento analítico on-line (OLAP), dados de multimídia e assim por diante.

Como a SQL é muito importante (e muito grande), dedicamos dois capítulos para explicar seus recursos. Neste capítulo, a Seção 4.1 descreve os comandos de DDL da SQL para criação de esquemas e tabelas, e oferece uma visão geral dos tipos de dados básicos em SQL. A Seção 4.2 apresenta como restrições básicas, chave e integridade referencial, são especificadas. A Seção 4.3 descreve as construções básicas da SQL para especificar recuperação de consultas, e a Seção 4.4 descreve os comandos SQL para inserção, exclusão e alteração de dados.

No Capítulo 5, descreveremos recuperação de consultas SQL mais complexas, bem como comandos ALTER para alterar o esquema. Também descreveremos a instrução CREATE ASSERTION, que permite a especificação de restrições mais gerais no banco de dados. Também apresentamos o conceito de triggers, que será abordado com mais detalhes no Capítulo 26, e descreveremos a facilidade SQL para definir views no banco de dados no Capítulo 5. As views também são conhecidas como *tabelas virtuais* ou *tabelas derivadas* porque apresentam ao usuário o que parecem ser tabelas; porém, a informação nessas tabelas é derivada de tabelas previamente definidas.

A Seção 4.5 lista alguns dos recursos da SQL que serão apresentados em outros capítulos do livro; entre eles estão o controle de transação no Capítulo 21, segurança/autorização no Capítulo 24, bancos de dados ativos (triggers) no Capítulo 26, recursos orientados a objeto no Capítulo 11 e recursos de processamento analítico on-line (OLAP) no Capítulo 29. No final deste capítulo há um resumo. Os capítulos 13 e 14 discutem as diversas técnicas de programação em banco de dados para programação com SQL.

4.1 Definições e tipos de dados em SQL

A SQL usa os termos **tabela**, **linha** e **coluna** para os termos do modelo relacional formal *relação*, *tupla* e *atributo*, respectivamente. Usaremos os termos correspondentes para indicar a mesma coisa. O principal comando SQL para a definição de dados é o CREATE, que pode ser usado para criar esquemas, tabelas (relações) e domínios (bem como outras construções como views, assertions e triggers). Antes de descrevermos a importância das instruções CREATE, vamos discutir os conceitos de esquema e catálogo na Seção 4.1.1 para contextualizar nossa discussão. A Seção 4.1.2 descreve como as tabelas são criadas, e a Seção 4.1.3, os tipos de dados mais importantes disponíveis para especificação de atributo. Como a especificação SQL é muito grande, oferecemos uma descrição dos recursos mais importantes. Outros detalhes poderão ser encontrados em diversos documentos dos padrões SQL (ver bibliografia selecionada ao final do capítulo).

4.1.1 Conceitos de esquema e catálogo em SQL

As primeiras versões da SQL não incluíam o conceito de um esquema de banco de dados relacional; todas as tabelas (relações) eram consideradas parte do mesmo esquema. O conceito de um esquema SQL foi incorporado inicialmente com SQL2 a fim de agrupar tabelas e outras construções que pertencem à mesma aplicação de banco de dados. Um **esquema SQL** é identificado por um **nome de esquema**, e inclui um **identificador de autorização** para indicar o usuário ou conta proprietário do esquema, bem como **descritores** para *cada elemento*. Esses **elementos** incluem tabelas, restrições, views, domínios e outras construções (como concessões — *grants* — de autorização) que descrevem o esquema que é criado por meio da instrução CREATE SCHEMA. Esta pode incluir todas as definições dos elementos do esquema. Como alternativa, o esquema pode receber um identificador de nome e autorização, e os elementos podem ser definidos mais tarde. Por exemplo, a instrução a seguir cria um esque-

[1] Originalmente, a SQL tinha instruções para criar e remover índices nos arquivos que representam as relações, mas estes foram retirados do padrão SQL por algum tempo.

ma chamado EMPRESA, pertencente ao usuário com identificador de autorização 'Jsilva'. Observe que cada instrução em SQL termina com um ponto e vírgula.

CREATE SCHEMA EMPRESA **AUTHORIZATION** 'Jsilva';

Em geral, nem todos os usuários estão autorizados a criar esquemas e elementos do esquema. O privilégio para criar esquemas, tabelas e outras construções deve ser concedido explicitamente às contas de usuário relevantes pelo administrador do sistema ou DBA.

Além do conceito de um esquema, a SQL usa o conceito de um **catálogo** — uma coleção nomeada de esquemas em um ambiente SQL. Um **ambiente** SQL é basicamente uma instalação de um SGBDR compatível com SQL em um sistema de computador.[2] Um catálogo sempre contém um esquema especial, chamado INFORMATION_SCHEMA, que oferece informações sobre todos os esquemas no catálogo e todos os seus descritores de elemento. As restrições de integridade, como a integridade referencial, podem ser definidas entre as relações somente se existirem nos esquemas dentro do mesmo catálogo. Os esquemas dentro do mesmo catálogo também podem compartilhar certos elementos, como definições de domínio.

4.1.2 O comando CREATE TABLE em SQL

O comando **CREATE TABLE** é usado para especificar uma nova relação, dando-lhe um nome e especificando seus atributos e restrições iniciais. Os atributos são especificados primeiro, e cada um deles recebe um nome, um tipo de dado para especificar seu domínio de valores e quaisquer restrições de atributo, como NOT NULL. As restrições de chave, integridade de entidade e integridade referencial podem ser especificadas na instrução CREATE TABLE, depois que os atributos forem declarados, ou acrescentadas depois, usando o comando ALTER TABLE (ver Capítulo 5). A Figura 4.1 mostra exemplos de instruções de definição de dados em SQL para o esquema de banco de dados relacional EMPRESA da Figura 3.7.

Em geral, o esquema SQL em que as relações são declaradas é especificado implicitamente no ambiente em que as instruções CREATE TABLE são executadas. Como alternativa, podemos conectar explicitamente o nome do esquema ao nome da relação, separados por um ponto. Por exemplo, escrevendo

CREATE TABLE EMPRESA.FUNCIONARIO ...

em vez de

CREATE TABLE FUNCIONARIO ...

como na Figura 4.1, podemos explicitamente (ao invés de implicitamente) tornar a tabela FUNCIONARIO parte do esquema EMPRESA.

As relações declaradas por meio das instruções CREATE TABLE são chamadas de **tabelas da base** (ou relações da base); isso significa que a relação e suas tuplas são realmente criadas e armazenadas como um arquivo pelo SGBD. As relações da base são distintas das **relações virtuais**, criadas por meio da instrução CREATE VIEW (ver Capítulo 5), que podem ou não corresponder a um arquivo físico real. Em SQL, os atributos em uma tabela da base são considerados *ordenados na sequência em que são especificados* no comando CREATE TABLE. No entanto, as linhas (tuplas) não são consideradas ordenadas dentro de uma relação.

É importante observar que, na Figura 4.1, existem algumas *chaves estrangeiras que podem causar erros*, pois são especificadas por referências circulares ou porque dizem respeito a uma tabela que ainda não foi criada. Por exemplo, a chave estrangeira Cpf_supervisor na tabela FUNCIONARIO é uma referência circular, pois se refere à própria tabela. A chave estrangeira Dnr na tabela FUNCIONARIO se refere à tabela DEPARTAMENTO, que ainda não foi criada. Para lidar com esse tipo de problema, essas restrições podem ser omitidas inicialmente do comando CREATE TABLE, e depois acrescentadas usando a instrução ALTER TABLE (ver Capítulo 5). Apresentamos todas as chaves estrangeiras na Figura 4.1, para mostrar o esquema EMPRESA completo em um só lugar.

4.1.3 Tipos de dados de atributo e domínios em SQL

Os **tipos de dados** básicos disponíveis para atributos são numérico, cadeia ou sequência de caracteres, cadeia ou sequência de bits, booleano, data e hora.

- Os tipos de dados **numerico** incluem números inteiros de vários tamanhos (INTEGER ou INT e SMALLINT) e números de ponto flutuante (reais) de várias precisões (FLOAT ou REAL e DOUBLE PRECISION). O formato dos números pode ser declarado usando DECIMAL(*i*, *j*) — ou DEC(*i*, *j*) ou NUMERIC(*i*, *j*) — onde *i*, a *precisão*, é o número total de dígitos decimais e *j*, a *escala*, é o número de dígitos após o ponto decimal. O valor padrão para a escala é zero, e para a precisão, é definido pela implementação.
- Tipos de dados de **cadeia de caracteres** são de tamanho fixo — CHAR(*n*) ou CHARACTER(*n*), onde *n* é o número de caracteres — ou de tamanho variável — VARCHAR(*n*) ou CHAR

[2] A SQL também inclui o conceito de um grupo (*cluster*) de catálogos dentro de um ambiente.

```
CREATE TABLE FUNCIONARIO
        (Pnome              VARCHAR(15)             NOT NULL,
        Minicial            CHAR,
        Unome               VARCHAR(15)             NOT NULL,
        Cpf                 CHAR(11),               NOT NULL,
        Datanasc            DATE,
        Endereço            VARCHAR(30),
        Sexo                CHAR,
        Salario             DECIMAL(10,2),
        Cpf_supervisor      CHAR(11),               NOT NULL,
        Dnr                 INT
      PRIMARY KEY (Cpf),
      FOREIGN KEY (Cpf_supervisor) REFERENCES FUNCIONARIO(Cpf),
      FOREIGN KEY (Dnr) REFERENCES DEPARTAMENTO(Dnumero) );
CREATE TABLE DEPARTAMENTO
        ( Dnome             VARCHAR(15)             NOT NULL,
        Dnumero             INT                     NOT NULL,
        Cpf_gerente         CHAR(11),               NOT NULL,
        Data_inicio_gerente DATE,
      PRIMARY KEY (Dnumero),
      UNIQUE (Dnome),
      FOREIGN KEY (Cpf_gerente) REFERENCES FUNCIONARIO(Cpf) );
CREATE TABLE LOCALIZACAO_DEP
        ( Dnumero           INT                     NOT NULL,
        Dlocal              VARCHAR(15)             NOT NULL,
      PRIMARY KEY (Dnumero, Dlocal),
      FOREIGN KEY (Dnumero) REFERENCES DEPARTAMENTO(Dnumero) );
CREATE TABLE PROJETO
        ( Projnome          VARCHAR(15)             NOT NULL,
        Projnumero          INT                     NOT NULL,
        Projlocal           VARCHAR(15),
        Dnum                INT                     NOT NULL,
      PRIMARY KEY (Projnumero),
      UNIQUE (Projnome),
      FOREIGN KEY (Dnum) REFERENCES DEPARTAMENTO(Dnumero) );
CREATE TABLE TRABALHA_EM
        ( Fcpf              CHAR(9)                 NOT NULL,
        Pnr                 INT                     NOT NULL,
        Horas               DECIMAL(3,1)            NOT NULL,
      PRIMARY KEY (Fcpf, Pnr),
      FOREIGN KEY (Fcpf) REFERENCES FUNCIONARIO(Cpf),
      FOREIGN KEY (Pnr) REFERENCES PROJETO(Projnumero) );
CREATE TABLE DEPENDENTE
        ( Fcpf              CHAR(11),               NOT NULL,
        Nome_dependente     VARCHAR(15)             NOT NULL,
        Sexo                CHAR,
        Datanasc            DATE,
        Parentesco          VARCHAR(8),
      PRIMARY KEY (Fcpf, Nome_dependente),
      FOREIGN KEY (Fcpf) REFERENCES FUNCIONARIO(Cpf) );
```

Figura 4.1
Instruções CREATE TABLE da SQL para definição do esquema EMPRESA da Figura 3.7.

VARYING(*n*) ou CHARACTER VARYING(*n*), onde *n* é o número máximo de caracteres. Ao especificar um valor literal de cadeia de caracteres, ele é colocado entre aspas simples (apóstrofos), e é *case sensitive* (diferencia maiúsculas de minúsculas).[3] Para cadeias de caracteres de tamanho fixo, uma cadeia mais curta é preenchida com caracteres em branco à direita. Por exemplo, se o valor 'Silva' for para um atributo do tipo CHAR(10), ele é preenchido com cinco caracteres em branco para se tornar 'Silva ', se necessário. Os espaços preenchidos geralmente são ignorados quando as cadeias são comparadas. Para fins de comparação, as cadeias de caracteres são consideradas ordenadas em ordem alfabética (ou lexicográfica); se uma cadeia *str1* aparecer antes de outra cadeia *str2* em ordem alfabética, então *str1* é considerada menor que *str2*.[4] Há também um operador de concatenação indicado por || (barra vertical dupla), que pode concatenar duas cadeias de caracteres em SQL. Por exemplo, 'abc' || 'XYZ' resulta em uma única cadeia 'abcXYZ'. Outro tipo de dado de cadeia de caracteres de tamanho variável, chamado CHARACTER LARGE OBJECT ou CLOB, também está disponível para especificar colunas que possuem grandes valores de texto, como documentos. O tamanho máximo de CLOB pode ser especificado em kilobytes (K), megabytes (M) ou gigabytes (G). Por exemplo, CLOB(20M) especifica um tamanho máximo de 20 megabytes.

- Tipos de dados de **cadeia de bits** podem ser de tamanho fixo *n* — BIT(*n*) — ou de tamanho variável — BIT VARYING(*n*), onde *n* é o número máximo de bits. O valor padrão para *n*, o tamanho de uma cadeia de caracteres ou cadeia de bits, é 1. Os literais de cadeia de bits literais são colocados entre apóstrofos, mas precedidos por um B para distingui-los das cadeias de caracteres; por exemplo, B'10101'.[5] Outro tipo de dados de cadeia de bits de tamanho variável, chamado BINARY LARGE OBJECT ou BLOB, também está disponível para especificar colunas que possuem grandes valores binários, como imagens. Assim como para CLOB, o tamanho máximo de um BLOB pode ser especificado em kilobits (K), megabits

[3] Isso não acontece com palavras-chave da SQL, como CREATE ou CHAR. Com as palavras-chave, a SQL é *case insensitive*, significando que ela trata letras maiúsculas e minúsculas como equivalentes nessas palavras.

[4] Para caracteres não alfabéticos, existe uma ordem definida.

[5] Cadeias de bits cujo tamanho é um múltiplo de 4 podem ser especificadas em notação *hexadecimal*, em que a cadeia literal é precedida por X e cada caractere hexadecimal representa 4 bits.

(M) ou gigabits (G). Por exemplo, BLOB(30G) especifica um tamanho máximo de 30 gigabits.

- Um tipo de dado **booleano** tem os valores tradicionais TRUE (verdadeiro) ou FALSE (falso). Em SQL, devido à presença de valores NULL (nulos), uma lógica de três valores é utilizada, de modo que um terceiro valor possível para um tipo de dado booleano é UNKNOWN (indefinido). Discutiremos a necessidade de UNKNOWN e a lógica de três valores no Capítulo 5.
- O tipo de dados **DATE** possui dez posições, e seus componentes são DAY (dia), MONTH (mês) e YEAR (ano) na forma DD-MM-YYYY. O tipo de dado TIME (tempo) tem pelo menos oito posições, com os componentes HOUR (hora), MINUTE (minuto) e SECOND (segundo) na forma HH:MM:SS. Somente datas e horas válidas devem ser permitidas pela implementação SQL. Isso implica que os meses devem estar entre 1 e 12 e as datas devem estar entre 1 e 31; além disso, uma data deve ser uma data válida para o mês correspondente. A comparação < (menor que) pode ser usada com datas ou horas — uma data *anterior* é considerada menor que uma data posterior, e da mesma forma com o tempo. Os valores literais são representados por cadeias com apóstrofos precedidos pela palavra-chave DATE ou TIME; por exemplo, DATE '27-09-2008' ou TIME '09:12:47'. Além disso, um tipo de dado TIME(*i*), onde *i* é chamado de *precisão em segundos fracionários de tempo*, especifica *i* + 1 posições adicionais para TIME — uma posição para um caractere separador de período adicional (.), e *i* posições para especificar as frações de um segundo. Um tipo de dados TIME WITH TIME ZONE inclui seis posições adicionais para especificar o *deslocamento* com base no fuso horário universal padrão, que está na faixa de +13:00 a -12:59 em unidades de HOURS:MINUTES. Se WITH TIME ZONE não for incluído, o valor padrão é o fuso horário local para a sessão SQL.

Alguns tipos de dados adicionais são discutidos a seguir. A lista apresentada aqui não está completa; diferentes implementações acrescentaram mais tipos de dados à SQL.

- Um tipo de dado **timestamp** (TIMESTAMP) inclui os campos DATE e TIME, mais um mínimo de seis posições para frações decimais de segundos e um qualificador opcional WITH TIME ZONE. Valores literais são representados por cadeias entre apóstrofos precedidos pela palavra-chave TIMESTAMP, com um espaço em branco entre data e hora; por exemplo, TIMESTAMP '27-09-2008 09:12:47.648302'.
- Outro tipo de dado relacionado a DATE, TIME e TIMESTAMP é o INTERVAL. Este especifica um **intervalo** — um *valor relativo* que pode ser usado para incrementar ou decrementar um valor absoluto de uma data, hora ou timestamp. Os intervalos são qualificados para serem de YEAR/MONTH ou de DAY/TIME.

O formato de DATE, TIME e TIMESTAMP pode ser considerado um tipo especial de cadeia. Logo, eles geralmente podem ser usados em comparações de cadeias sendo **convertidos** (**cast**) em cadeias equivalentes.

É possível especificar o tipo de dado de cada atributo diretamente, como na Figura 4.1; como alternativa, um domínio pode ser declarado e seu nome, usado com a especificação de atributo. Isso torna mais fácil mudar o tipo de dado para um domínio que é usado por diversos atributos em um esquema, e melhora a legibilidade do esquema. Por exemplo, podemos criar um domínio TIPO_CPF com a seguinte instrução:

```
CREATE DOMAIN TIPO_CPF AS CHAR(11);
```

Podemos usar TIPO_CPF no lugar de CHAR(11) na Figura 4.1 para os atributos Cpf e Cpf_supervisor de FUNCIONARIO, Cpf_gerente de DEPARTAMENTO, Fcpf de TRABALHA_EM e Fcpf de DEPENDENTE. Um domínio também pode ter uma especificação padrão opcional por meio de uma cláusula DEFAULT, conforme discutiremos mais adiante para os atributos. Observe que os domínios podem não estar disponíveis em algumas implementações da SQL.

4.2 Especificando restrições em SQL

Esta seção descreve as restrições básicas que podem ser especificadas em SQL como parte da criação de tabela. Estas incluem restrições de chave e integridade referencial, restrições sobre domínios de atributo e NULLs e restrições sobre tuplas individuais dentro de uma relação. Discutiremos a especificação de restrições mais gerais, chamadas asserções (ou *assertions*) no Capítulo 5.

4.2.1 Especificando restrições de atributo e defaults de atributo

Como a SQL permite NULLs como valores de atributo, uma *restrição* NOT NULL pode ser especificada se o valor NULL não for permitido para determinado atributo. Isso sempre é especificado de maneira implícita para os atributos que fazem parte da *chave primária* de cada relação, mas pode ser especificado para quaisquer outros atributos cujos valores não podem ser NULL, como mostra a Figura 4.1.

```
CREATE TABLE FUNCIONARIO
    ( . . . ,
    Dnr             INT             NOT NULL      DEFAULT 1,
    CONSTRAINT CHPFUNC
        PRIMARY KEY (Cpf),
    CONSTRAINT CHESUPERFUNC
        FOREIGN KEY (Cpf_supervisor) REFERENCES FUNCIONARIO(Cpf)
        ON DELETE SET NULL    ON UPDATE CASCADE,
    CONSTRAINT CHEDEPFUNC
    FOREIGN KEY(Dnr) REFERENCES DEPARTAMENTO(Dnumero)
            ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE DEPARTAMENTO
    ( . . . ,
    Cpf_gerente     CHAR(11)        NOT NULL      DEFAULT '88866555576',
    . . . ,
    CONSTRAINT CHPDEP
        PRIMARY KEY(Dnumero),
    CONSTRAINT CHSDEP
        UNIQUE (Dnome),
    CONSTRAINT CHEGERDEP
    FOREIGN KEY (Cpf_gerente) REFERENCES FUNCIONARIO(Cpf)
        ON DELETE SET DEFAULT ON UPDATE CASCADE);
CREATE TABLE LOCALIZACAO_DEP
    ( . . . ,
    PRIMARY KEY (Dnumero, Dlocal),
    FOREIGN KEY (Dnumero) REFERENCES DEPARTAMENTO(Dnumero)
        ON DELETE CASCADE     ON UPDATE CASCADE);
```

Figura 4.2
Exemplo ilustrando como os valores de atributo default e as ações disparadas por integridade referencial são especificadas em SQL.

Também é possível definir um *valor padrão* para um atributo anexando a cláusula **DEFAULT** <valor> a uma definição de atributo. O valor padrão está incluído em qualquer nova tupla se um valor explícito não for fornecido para esse atributo. A Figura 4.2 ilustra um exemplo de especificação de um gerente default para um novo departamento e um departamento default para um novo funcionário. Se nenhuma cláusula default for especificada, o *valor padrão* será NULL para atributos *que não possuem* a restrição NOT NULL.

Outro tipo de restrição pode limitar valores de atributo ou domínio usando a cláusula **CHECK** (verificação) após uma definição de atributo ou domínio.[6] Por exemplo, suponha que números de departamento sejam restritos a números inteiros entre 1 e 20; então, podemos mudar a declaração de atributo de Dnumero na tabela DEPARTAMENTO (ver Figura 4.1) para o seguinte:

```
Dnumero INT NOT NULL CHECK (Dnumero > 0 AND
Dnumero < 21);
```

A cláusula CHECK também pode ser usada em conjunto com a instrução CREATE DOMAIN. Por exemplo, podemos escrever a seguinte instrução:

```
CREATE DOMAIN D_NUM AS INTEGER
CHECK (D_NUM > 0 AND D_NUM < 21);
```

Depois, podemos usar o domínio criado D_NUM como o tipo de atributo para todos os atributos que se referem aos números de departamento na Figura 4.1, como Dnumero de DEPARTAMENTO, Dnum de PROJETO, Dnr de FUNCIONARIO, e assim por diante.

4.2.2 Especificando restrições de chave e integridade referencial

Como chaves e restrições de integridade referencial são muito importantes, existem cláusulas especiais dentro da instrução CREATE TABLE para especificá-las. Alguns exemplos para ilustrar a especificação de chaves e integridade referencial aparecem na Figura 4.1.[7] A cláusula PRIMARY KEY especifica um ou mais atributos que compõem a chave primária de uma relação. Se uma chave primária tiver um *único* atributo, a cláusula pode acompanhar o atributo diretamente. Por exemplo, a chave primária de DEPARTAMENTO pode ser especificada da seguinte forma (em vez do modo como ela é especificada na Figura 4.1):

```
Dnumero INT PRIMARY KEY;
```

A cláusula **UNIQUE** especifica chaves alternativas (secundárias), conforme ilustramos nas declarações da tabela DEPARTAMENTO e PROJETO na Figura 4.1. A cláusula **UNIQUE** também pode ser especificada diretamente para uma chave secundária se esta for um único atributo, como no exemplo a seguir:

```
Dnome VARCHAR(15) UNIQUE;
```

A integridade referencial é especificada por meio da cláusula **FOREIGN KEY** (chave estrangeira), como mostra a Figura 4.1. Conforme discutimos na Seção 3.2.4, uma restrição de integridade referencial pode ser violada quando tuplas são inseridas ou excluídas, ou quando um valor de atributo de chave estrangeira ou chave primária é modificado. A ação default que a SQL toma para uma violação de integridade é **rejeitar** a operação de atualização que causará uma violação, o que é conhecido como opção RESTRICT. Porém, o projetista do esquema pode especificar uma ação alternativa

[6] A cláusula CHECK também pode ser usada para outras finalidades, conforme veremos.

[7] As restrições de chave e integridade referencial não estavam incluídas nas primeiras versões da SQL. Em algumas implementações iniciais, as chaves eram especificadas implicitamente no nível interno por meio do comando CREATE INDEX.

para ser tomada conectando uma cláusula de **ação de disparo referencial** a qualquer restrição de chave estrangeira. As opções incluem SET NULL, CASCADE e SET DEFAULT. Uma opção deve ser qualificada com ON DELETE ou ON UPDATE. Ilustramos isso com os exemplos mostrados na Figura 4.2. Aqui, o projetista de banco de dados escolhe ON DELETE SET NULL e ON UPDATE CASCADE para a chave estrangeira Cpf_supervisor de FUNCIONARIO. Isso significa que, se a tupla para um *funcionário supervisor* é *excluída*, o valor de Cpf_supervisor será automaticamente definido como NULL para todas as tuplas de funcionários que estavam referenciando a tupla do funcionário excluído. Por sua vez, se o valor de Cpf para um funcionário supervisor é *atualizado* (digamos, porque foi inserido incorretamente), o novo valor será *propagado em cascata* de Cpf_supervisor para todas as tuplas de funcionário que referencia a tupla de funcionário atualizada.[8]

Em geral, a ação tomada pelo SGBD para SET NULL ou SET DEFAULT é a mesma para ON DELETE e ON UPDATE: o valor dos atributos de referência afetados é mudado para NULL em caso de SET NULL e para o valor padrão especificado do atributo de referência em caso de SET DEFAULT. A ação para CASCADE ON DELETE é excluir todas as tuplas de referência, enquanto a ação para CASCADE ON UPDATE é mudar o valor do(s) atributo(s) de chave estrangeira de referência para o (novo) valor de chave primária atualizado para todas as tuplas de referência. É responsabilidade do projetista de banco de dados escolher a ação apropriada e especificá-la no esquema do banco de dados. Como uma regra geral, a opção CASCADE é adequada para relações de 'relacionamento' (ver Seção 9.1), como TRABALHA_EM; para relações que representam atributos multivalorados, como LOCALIZACAO_DEP; e para relações que representam tipos de entidades fracas, como DEPENDENTE.

4.2.3 Dando nomes a restrições

A Figura 4.2 também ilustra como uma restrição pode receber um **nome de restrição**, seguindo a palavra-chave **CONSTRAINT**. Os nomes de todas as restrições dentro de um esquema em particular precisam ser exclusivos. Um nome de restrição é usado para identificar uma restrição em particular caso ela deva ser removida mais tarde e substituída por outra, conforme discutiremos no Capítulo 5. Dar nomes a restrições é algo opcional.

4.2.4 Especificando restrições sobre tuplas usando CHECK

Além das restrições de chave e integridade referencial, que são especificadas por palavras-chave especiais, outras *restrições de tabela* podem ser especificadas por meio de cláusula adicional CHECK ao final de uma instrução CREATE TABLE. Estas podem ser chamadas de restrições **baseadas em tupla**, pois se aplicam a cada tupla *individualmente* e são verificadas sempre que uma tupla é inserida ou modificada. Por exemplo, suponha que a tabela DEPARTAMENTO da Figura 4.1 tivesse um atributo adicional Dep_data_criacao, que armazena a data em que o departamento foi criado. Então, poderíamos acrescentar a seguinte cláusula CHECK ao final da instrução CREATE TABLE para a tabela DEPARTAMENTO para garantir que a data de início de um gerente seja posterior à data de criação do departamento.

```
CHECK (Dep_data_criacao <= Data_inicio_gerente);
```

A cláusula CHECK também pode ser usada para especificar restrições mais gerais usando a instrução CREATE ASSERTION da SQL. Discutiremos isso no Capítulo 5 porque exige o entendimento completo sobre consultas, que são discutidas nas seções 4.3 e 5.1.

4.3 Consultas de recuperação básicas em SQL

A SQL tem uma instrução básica para recuperar informações de um banco de dados: a instrução **SELECT**. A instrução SELECT *não é o mesmo que* a operação SELECT da álgebra relacional, que discutiremos no Capítulo 6. Existem muitas opções e tipos de instrução SELECT em SQL, de modo que introduziremos seus recursos gradualmente. Usaremos consultas de exemplo especificadas no esquema da Figura 3.5 e vamos nos referir ao exemplo estado de banco de dados mostrado na Figura 3.6 para mostrar os resultados de alguns exemplos de consultas. Nesta seção, apresentamos os recursos da SQL para *consultas de recuperação simples*. Os recursos da SQL para especificar consultas de recuperação mais complexas serão apresentados na Seção 5.1.

Antes de prosseguir, devemos apontar uma *distinção importante* entre a SQL e o modelo relacional formal discutido no Capítulo 3: a SQL permite que uma tabela (relação) tenha duas ou mais tuplas

[8] Observe que a chave estrangeira Cpf_supervisor na tabela FUNCIONARIO é uma referência circular e, portanto, pode ter de ser incluída mais tarde como uma restrição nomeada, usando a instrução ALTER TABLE, conforme discutimos no final da Seção 4.1.2.

que são idênticas em todos os seus valores de atributo. Assim, em geral, uma tabela **SQL** não é um *conjunto de tuplas*, pois um conjunto não permite dois membros idênticos; em vez disso, ela é um **multiconjunto** (também chamado de *bag*) de tuplas. Algumas relações SQL são *restritas a serem conjuntos* porque uma restrição de chave foi declarada ou porque a opção DISTINCT foi usada com a instrução SELECT (descrita mais adiante nesta seção). Precisamos estar cientes dessa distinção à medida que discutirmos os exemplos.

4.3.1 A estrutura SELECT-FROM-WHERE das consultas SQL básicas

As consultas em SQL podem ser muito complexas. Começaremos com consultas simples, e depois passaremos para as mais complexas de maneira passo a passo. A forma básica do comando SELECT, às vezes chamada de **mapeamento** ou **bloco select-from-where**, é composta pelas três cláusulas SELECT, FROM e WHERE, e tem a seguinte forma:[9]

```
SELECT  <lista atributos>
FROM    <lista tabelas>
WHERE   <condição>;
```

onde

- <lista atributos> é uma lista de nomes de atributo cujos valores devem ser recuperados pela consulta.
- <lista tabelas> é uma lista dos nomes de relação exigidos para processar a consulta.
- <condição> é uma expressão condicional (booleana) que identifica as tuplas a serem recuperadas pela consulta.

Em SQL, os operadores básicos de comparação lógicos para comparar valores de atributo entre si e com constantes literais são =, <, <=, >, >= e <>. Estes correspondem aos operadores da álgebra relacional =, <, ≤, >, ≥ e ≠, respectivamente, e aos operadores da linguagem de programação C/C++ =, <, <=, >, >= e !=. A principal diferença sintática é o operador *diferente*. A SQL possui operadores de comparação adicional que apresentaremos de maneira gradual.

Ilustramos a instrução SELECT básica em SQL com alguns exemplos de consultas. As consultas são rotuladas aqui com os mesmos números de consulta usados no Capítulo 6 para facilitar a referência cruzada.

Consulta 0. Recuperar a data de nascimento e o endereço do(s) funcionário(s) cujo nome seja 'João B. Silva'.

```
C0: SELECT  Datanasc, Endereco
    FROM    FUNCIONARIO
    WHERE   Pnome='João' AND Minicial='B' AND
            Unome='Silva';
```

Esta consulta envolve apenas a relação FUNCIONARIO listada na cláusula FROM. A consulta *seleciona* as tuplas FUNCIONARIO individuais que satisfazem a condição da cláusula WHERE, depois *projeta* o resultado nos atributos Datanasc e Endereco listados na cláusula SELECT.

A cláusula SELECT da SQL especifica os atributos cujos valores devem ser recuperados, que são chamados **atributos de projeção**, e a cláusula WHERE especifica a condição booleana que deve ser verdadeira para qualquer tupla recuperada, que é conhecida como **condição de seleção**. A Figura 4.3(a) mostra o resultado da consulta C0 sobre o banco de dados da Figura 3.6.

Podemos pensar em uma **variável de tupla** implícita (ou *iterator*) na consulta SQL variando ou *repetindo* sobre cada tupla individual na tabela FUNCIONARIO e avaliando a condição na cláusula WHERE. Somente as tuplas que satisfazem a condição — ou seja, aquelas tuplas para as quais a condição é avaliada como TRUE após substituir seus valores de atributo correspondentes — são selecionadas.

Consulta 1. Recuperar o nome e endereço de todos os funcionários que trabalham para o departamento 'Pesquisa'.

```
C1: SELECT  Pnome, Unome, Endereco
    FROM    FUNCIONARIO, DEPARTAMENTO
    WHERE   Dnome='Pesquisa' AND Dnumero=Dnr;
```

Na cláusula WHERE de C1, a condição Dnome = 'Pesquisa' é uma **condição de seleção** que escolhe a tupla de interesse em particular na tabela DEPARTAMENTO, pois Dnome é um atributo de DEPARTAMENTO. A condição Dnumero = Dnr é chamada **condição de junção**, pois combina duas tuplas: uma de DEPARTAMENTO e uma de FUNCIONARIO, sempre que o valor de Dnumero em DEPARTAMENTO é igual ao valor de Dnr em FUNCIONARIO. O resultado da consulta C1 é mostrado na Figura 4.3(b). Em geral, qualquer número de condições de seleção e junção pode ser especificado em uma única consulta SQL.

[9] As cláusulas SELECT e FROM são necessárias em todas as consultas SQL. O WHERE é opcional (ver Seção 4.3.3).

(a)

Datanasc	Endereco
09-01-1965	Rua das Flores, 751, São Paulo, SP

(b)

Pnome	Unome	Endereco
João	Silva	Rua das Flores, 751, São Paulo, SP
Fernando	Wong	Rua da Lapa, 34, São Paulo, SP
Ronaldo	Lima	Rua Rebouças, 65, Piracicaba, SP
Joice	Leite	Av. Lucas Obes, 74, São Paulo, SP

(c)

Projnumero	Dnum	Unome	Endereco	Datanasc
10	4	Souza	Av. Artur de Lima, 54, Santo André, SP	20-06-1941
30	4	Souza	Av. Artur de Lima, 54, Santo André, SP	20-06-1941

(d)

F.Pnome	F.Unome	S.Pnome	S.Unome
João	Silva	Fernando	Wong
Fernando	Wong	Jorge	Brito
Alice	Zelaya	Jennifer	Souza
Jennifer	Souza	Jorge	Brito
Ronaldo	Lima	Fernando	Wong
Joice	Leite	Fernando	Wong
André	Pereira	Jennifer	Souza

(e)

F.Pnome
12345678966
33344555587
99988777767
98765432168
66688444476
45345345376
98798798733
88866555576

(f)

Cpf	Dnome
12345678966	Pesquisa
33344555587	Pesquisa
99988777767	Pesquisa
98765432168	Pesquisa
66688444476	Pesquisa
45345345376	Pesquisa
98798798733	Pesquisa
88866555576	Pesquisa
12345678966	Administração
33344555587	Administração
99988777767	Administração
98765432168	Administração
66688444476	Administração
45345345376	Administração
98798798733	Administração
88866555576	Administração
12345678966	Matriz
33344555587	Matriz
99988777767	Matriz
98765432168	Matriz
66688444476	Matriz
45345345376	Matriz
98798798733	Matriz
88866555576	Matriz

(g)

Pnome	Minicial	Unome	Cpf	Datanasc	Endereco	Sexo	Salario	Cpf_supervisor	Dnr
João	B	Silva	12345678966	09-01-1965	Rua das Flores, 751, São Paulo, SP	M	30.000	33344555587	5
Fernando	T	Wong	33344555587	08-12-1955	Rua da Lapa, 34, São Paulo, SP	M	40.000	88866555576	5
Ronaldo	K	Lima	66688444476	15-09-1962	Rua Rebouças, 65, Piracicaba, SP	M	38.000	33344555587	5
Joice	A	Leite	45345345376	31-07-1972	Av. Lucas Obes, 74, São Paulo, SP	F	25.000	33344555587	5

Figura 4.3
Resultados das consultas SQL quando aplicados ao estado do banco de dados EMPRESA mostrado na Figura 3.6. (a) C0. (b) C1. (c) C2. (d) C8. (e) C9. (f) C10. (g) C1C.

Uma consulta que envolve apenas condições de seleção e junção mais atributos de projeção é conhecida como uma consulta **seleção-projeção-junção**. O próximo exemplo é uma consulta seleção-projeção-junção com *duas* condições de junção.

Consulta 2. Para cada projeto localizado em 'Mauá', liste o número do projeto, o número do departamento que o controla e o sobrenome, endereço e data de nascimento do gerente do departamento.

```
C2: SELECT  Projnumero, Dnum, Unome, Endereco,
            Datanasc
    FROM    PROJETO, DEPARTAMENTO,
            FUNCIONARIO
    WHERE   Dnum=Dnumero AND
            Cpf_gerente=Cpf AND
            Projlocal='Mauá';
```

A condição de junção Dnum = Dnumero relaciona uma tupla de projeto a sua tupla de departamento que o controla, enquanto a condição de junção Cpf_gerente = Cpf relaciona a tupla do departamento que o controla à tupla de funcionário que gerencia esse departamento. Cada tupla no resultado será uma *combinação* de um projeto, um departamento e um funcionário, que satisfaz as condições de junção. Os atributos de projeção são usados para escolher os atributos a serem exibidos com base em cada tupla combinada. O resultado da consulta C2 aparece na Figura 4.3(c).

4.3.2 Nomes de atributos ambíguos, apelido, renomeação e variáveis de tupla

Em SQL, o mesmo nome pode ser usado para dois (ou mais) atributos, desde que estes estejam em *relações diferentes*. Se isso acontecer, e uma consulta em múltiplas tabelas se referir a dois ou mais atributos com o mesmo nome, é *preciso* **qualificar** o nome do atributo com o nome da relação, para evitar ambiguidade. Isso é feito *prefixando* o nome da relação ao nome do atributo e separando os dois por um ponto. Para ilustrar isso, suponha que, nas figuras 3.5 e 3.6, os atributos Dnr e Unome da relação FUNCIONARIO fossem chamados de Dnumero e Nome, e o atributo Dnome de DEPARTAMENTO também fosse chamado Nome. Então, para evitar ambiguidade, a consulta C1 seria reformulada como mostramos em C1A. Devemos prefixar os nomes de atributo Nome e Dnumero em C1A para especificar a quais estamos nos referindo, pois os mesmos nomes de atributo são usados nas duas relações:

```
C1A: SELECT  Pnome, FUNCIONARIO.Nome,
             Endereco
     FROM    FUNCIONARIO, DEPARTAMENTO
     WHERE   DEPARTAMENTO.Nome='Pesquisa' AND
             DEPARTAMENTO.
             Dnumero=FUNCIONARIO.Dnumero;
```

Nomes de atributo totalmente qualificados podem ser usados por clareza mesmo que não haja ambiguidade nos nomes de atributo. C1 aparece dessa maneira como C1' a seguir. Também podemos criar um *apelido* para cada nome de tabela, para evitar a digitação repetida de nomes de tabela longos (ver C8, a seguir).

```
C1': SELECT  FUNCIONARIO.Pnome, FUNCIONARIO.
             UNome FUNCIONARIO.Endereco,
     FROM    FUNCIONARIO, DEPARTAMENTO
     WHERE   DEPARTAMENTO.DNome='Pesquisa' AND
             DEPARTAMENTO.Dnumero=
             FUNCIONARIO.Dnr;
```

A ambiguidade dos nomes de atributo também surge no caso de consultas que se referem à mesma relação duas vezes, como no exemplo a seguir.

Consulta 8. Para cada funcionário, recupere o primeiro e o último nome do funcionário e o primeiro e o último nome de seu supervisor imediato.

```
C8: SELECT  F.Pnome, F.Unome, S.Pnome, S.Unome
    FROM    FUNCIONARIO AS F, FUNCIONARIO
            AS S
    WHERE   F.Cpf_supervisor=S.Cpf;
```

Neste caso, precisamos declarar nomes de relação alternativos F e S, chamados **apelidos** ou **variáveis de tupla**, para a relação FUNCIONARIO. Um apelido pode vir após a palavra-chave AS, como mostramos em C8, ou pode vir diretamente após o nome da relação — por exemplo, escrevendo FUNCIONARIO F, FUNCIONARIO S na cláusula FROM de C8. Também é possível **renomear** os atributos da relação dentro da consulta em SQL, dando-lhe apelidos. Por exemplo, se escrevermos

```
FUNCIONARIO AS F(Pn, Mi, Un, Cpf, Dn, End, Sexo,
Sal, Scpf, Dnr)
```

na cláusula FROM, Pn torna-se um apelido para Pnome, Mi para Minicial, Un para Unome, e assim por diante.

Em C8, podemos pensar em F e S como duas *cópias diferentes* da relação FUNCIONARIO; a primei-

ra, F, representa funcionários no papel de supervisionados ou subordinados; a segunda, S, representa os funcionários no papel de supervisores. Agora, podemos juntar as duas cópias. Naturalmente, na realidade existe *apenas uma* relação FUNCIONARIO, e a condição de junção serve para juntar a própria relação, combinando as tuplas que satisfazem a condição de junção F.Cpf_supervisor = S.Cpf. Observe que este é um exemplo de uma consulta recursiva de um nível, conforme discutiremos na Seção 6.4.2. Nas versões anteriores da SQL, não era possível especificar uma consulta recursiva geral, com um número desconhecido de níveis, em uma única instrução SQL. Uma construção para especificar consultas recursivas foi incorporada na SQL:1999 (ver Capítulo 5).

O resultado da consulta C8 aparece na Figura 4.3(d). Sempre que um ou mais apelidos são dados a uma relação, podemos usar esses nomes para representar diferentes referências a essa mesma relação. Isso permite múltiplas referências à mesma relação dentro de uma consulta.

Podemos usar esse mecanismo de nomeação de apelidos em qualquer consulta SQL para especificar variáveis de tupla para cada tabela na cláusula WHERE, não importando se a mesma relação precisa ser referenciada mais de uma vez. De fato, essa prática é recomendada, pois resulta em consultas mais fáceis de compreender. Por exemplo, poderíamos especificar a consulta C1 como em C1B:

```
C1B: SELECT F.Pnome, F.Unome, F.Endereco
     FROM   FUNCIONARIO F, DEPARTAMENTO D
     WHERE  D.DNome='Pesquisa' AND
            D.Dnumero=F.Dnr;
```

4.3.3 Cláusula WHERE não especificada e uso do asterisco

Vamos discutir aqui mais dois recursos da SQL. A falta de uma cláusula WHERE indica que não há condição sobre a seleção de tuplas; logo, *todas as tuplas* da relação especificada na cláusula FROM se qualificam e são selecionadas para o resultado da consulta. Se mais de uma relação for especificada na cláusula FROM e não houver uma cláusula WHERE, então o PRODUTO CARTESIANO — *todas as combinações de tuplas possíveis* — dessas relações será selecionado. Por exemplo, a Consulta 9 seleciona todos os Cpfs de FUNCIONARIO (Figura 4.3(e)) e a Consulta 10 seleciona todas as combinações de um Cpf de FUNCIONARIO e um Dnome de DEPARTAMENTO, independentemente de o funcionário trabalhar ou não para o departamento (Figura 4.3(f)).

Consultas 9 e 10. Selecionar todos os Cpfs de FUNCIONARIO (C9) e todas as combinações de Cpf de FUNCIONARIO e Dnome de DEPARTAMENTO (C10) no banco de dados.

```
C9:  SELECT Cpf
     FROM   FUNCIONARIO;
C10: SELECT Cpf, Dnome
     FROM   FUNCIONARIO, DEPARTAMENTO;
```

É extremamente importante especificar cada condição de seleção e junção na cláusula WHERE. Se alguma condição desse tipo for esquecida, o resultado poderá ser relações incorretas e muito grandes. Observe que C10 é semelhante a uma operação de PRODUTO CARTESIANO seguida por uma operação PROJECAO na álgebra relacional (ver Capítulo 6). Se especificarmos todos os atributos de FUNCIONARIO e DEPARTAMENTO em C10, obteremos o PRODUTO CARTESIANO real (exceto pela eliminação de duplicatas, se houver).

Para recuperar todos os valores de atributo das tuplas selecionadas, não precisamos listar os nomes de atributo explicitamente em SQL; basta especificar um *asterisco* (*), que significa *todos os atributos*. Por exemplo, a consulta C1C recupera todos os valores de atributo de qualquer FUNCIONARIO que trabalha no DEPARTAMENTO número 5 (Figura 4.3(g)), a consulta C1D recupera todos os atributos de um FUNCIONARIO e os atributos do DEPARTAMENTO em que ele ou ela trabalha para todo funcionário no departamento 'Pesquisa', e C10A especifica o PRODUTO CARTESIANO das relações FUNCIONARIO e DEPARTAMENTO.

```
C1C:  SELECT *
      FROM   FUNCIONARIO
      WHERE  Dnr=5;
C1D:  SELECT *
      FROM   FUNCIONARIO, DEPARTAMENTO
      WHERE  Dnome='Pesquisa' AND Dnr=Dnumero;
C10A: SELECT *
      FROM   FUNCIONARIO, DEPARTAMENTO;
```

4.3.4 Tabelas como conjuntos em SQL

Conforme já dissemos, a SQL normalmente trata uma tabela não como um conjunto, mas como um **multiconjunto**; *tuplas duplicadas podem aparecer mais de uma vez* em uma tabela, e no resultado de uma consulta. A SQL não elimina automaticamente tuplas duplicadas nos resultados das consultas, pelos seguintes motivos:

- A eliminação de duplicatas é uma operação dispendiosa. Um modo de implementá-la é classificar as tuplas primeiro e depois eliminar as duplicatas.
- O usuário pode querer ver as tuplas duplicadas no resultado de uma consulta.
- Quando uma função agregada (ver Seção 5.1.7) é aplicada às tuplas, na maioria dos casos não queremos eliminar duplicatas.

Uma tabela SQL com uma chave é restrita a ser um conjunto, uma vez que o valor de chave precisa ser distinto em cada tupla.[10] Se *quisermos* eliminar tuplas duplicadas do resultado de uma consulta SQL, usamos a palavra-chave **DISTINCT** na cláusula SELECT, significando que apenas as tuplas distintas deverão permanecer no resultado. Em geral, uma consulta com SELECT DISTINCT elimina duplicatas, enquanto uma consulta com SELECT ALL não elimina. A especificação de SELECT sem ALL ou DISTINCT — como em nossos exemplos anteriores — é equivalente a SELECT ALL. Por exemplo, a C11 recupera o salário de cada funcionário; se vários funcionários tiverem o mesmo salário, esse valor de salário aparecerá muitas vezes no resultado da consulta, como mostra a Figura 4.4(a). Se estivermos interessados apenas em valores de salário distintos, queremos que cada valor apareça apenas uma vez, independentemente de quantos funcionários ganham esse salário. Usando a palavra-chave **DISTINCT**, como em C11A, conseguimos isso, como mostra a Figura 4.4(b).

Consulta 11. Recuperar o salário de cada funcionário (C11) e todos os valores de salário distintos (C11A).

```
C11:   SELECT   ALL Salario
       FROM     FUNCIONARIO;
C11A:  SELECT   DISTINCT Salario
       FROM     FUNCIONARIO;
```

A SQL incorporou diretamente algumas das operações de conjunto da *teoria de conjuntos* da matemática, que também fazem parte da álgebra relacional (ver Capítulo 6). Existem operações de união de conjunto (UNION), diferença de conjunto (**EXCEPT**),[11] e interseção de conjunto (**INTERSECT**). As relações resultantes dessas operações de conjunto são conjuntos de tuplas; ou seja, *tuplas duplicadas são eliminadas do resultado*. Essas operações de conjunto se aplicam apenas a *relações compatíveis com união*, de modo que precisamos garantir que as duas relações em que aplicamos a operação tenham os mesmos atributos e que os atributos apareçam na mesma ordem nas duas relações. O próximo exemplo ilustra o uso de UNION.

(a)

Salário
30.000
40.000
25.000
43.000
38.000
25.000
25.000
55.000

(b)

Salário
30.000
40.000
25.000
43.000
38.000
55.000

(c)

Pnome	Unome

(d)

Pnome	Unome
Fernando	Wong

Figura 4.4
Resultados de consultas SQL adicionais, quando aplicadas ao estado de banco de dados EMPRESA, mostrados na Figura 3.6. (a) C11. (b) C11A. (c) C12. (d) C12A.

Consulta 4. Fazer uma lista de todos os números de projeto para aqueles que envolvam um funcionário cujo último nome é 'Silva', seja como um trabalhador ou como um gerente do departamento que controla o projeto.

```
C4A:  ( SELECT    DISTINCT Projnumero
        FROM      PROJETO, DEPARTAMENTO,
                  FUNCIONARIO
        WHERE     Dnum=Dnumero AND
                  Cpf_gerente=Cpf
                  AND Unome='Silva' )
        UNION
      ( SELECT    DISTINCT Projnumero
        FROM      PROJETO, TRABALHA_EM,
                  FUNCIONARIO
        WHERE     Projnumero=Pnr AND Fcpf=Cpf
                  AND Unome='Silva' );
```

A primeira consulta SELECT recupera os projetos que envolvem um 'Silva' como gerente do de-

[10] Em geral, uma tabela SQL não precisa ter uma chave, embora, na maioria dos casos, exista uma.

[11] Em alguns sistemas, a palavra-chave MINUS é usada para a operação de diferença de conjunto, em vez de EXCEPT.

(a)

R

A
a1
a2
a2
a3

S

A
a1
a2
a4
a5

(b)

T

A
a1
a1
a2
a2
a2
a3
a4
a5

(c)

T

A
a2
a3

(d)

T

A
a1
a2

Figura 4.5
Os resultados das operações de multiconjunto da SQL. (a) Duas tabelas, R(A) e S(A). (b) R(A) UNION ALL S(A). (c) R(A) EXCEPT ALL S(A). (d) R(A) INTERSECT ALL S(A).

partamento que controla o projeto, e a segunda, recupera os projetos que envolvem um 'Silva' como um trabalhador no projeto. Observe que, se todos os funcionários tiverem o último nome 'Silva', os nomes de projeto envolvendo qualquer um deles seriam recuperados. A aplicação da operação UNION às duas consultas SELECT gera o resultado desejado.

A SQL também possui operações multiconjuntos correspondentes, que são acompanhadas da palavra-chave ALL (UNION ALL, EXCEPT ALL, INTERSECT ALL). Seus resultados são multiconjuntos (duplicatas não são eliminadas). O comportamento dessas operações é ilustrado pelos exemplos da Figura 4.5. Basicamente, cada tupla — seja ela uma duplicata ou não — é considerada uma tupla diferente ao aplicar essas operações.

4.3.5 Combinação de padrão de subcadeias e operadores aritméticos

Nesta seção, discutimos vários outros recursos da SQL. O primeiro recurso permite condições de comparação apenas sobre partes de uma cadeia de caracteres, usando o operador de comparação **LIKE**. Isso pode ser usado para **combinação de padrão** de cadeia. Cadeias parciais são especificadas usando dois caracteres reservados: % substitui um número qualquer de zero ou mais caracteres, e o sublinhado (_) substitui um único caractere. Por exemplo, considere a seguinte consulta.

Consulta 12. Recuperar todos os funcionários cujo endereço esteja em São Paulo, SP.

```
C12: SELECT  Pnome, Unome
     FROM    FUNCIONARIO
     WHERE   Endereco LIKE '%SaoPaulo,SP%';
```

Para recuperar todos os funcionários que nasceram durante a década de 1950, podemos usar a Consulta C12A. Aqui, '5' precisa ser o nono caractere da cadeia (de acordo com nosso formato para data), de modo que usamos o valor '_ _ _ _ _ _ _ _ 5 _', com cada sublinhado servindo como um marcador de lugar para um caractere qualquer.

Consulta 12A. Encontrar todos os funcionários que nasceram durante a década de 1950.

```
C12: SELECT  Pnome, Unome
     FROM    FUNCIONARIO
     WHERE   Datanasc LIKE '_ _ _ _ _ _ _ _ 5 _';
```

Se um sublinhado ou % for necessário como um caractere literal na cadeia, este deve ser precedido por um *caractere de escape*, que é especificado após a cadeia usando a palavra-chave ESCAPE. Por exemplo, 'AB_CD\%EF' ESCAPE '\' representa a cadeia literal 'AB_CD%EF', pois \ é especificado como o caractere de escape. Qualquer caractere não usado na cadeia pode ser escolhido como caractere de escape. Além disso, precisamos de uma regra para especificar apóstrofos ou aspas simples (' ') se eles tiverem de ser incluídos em uma cadeia, pois são usados para iniciar e terminar cadeias. Se um apóstrofo (') for necessário, ele será representado como dois apóstrofos consecutivos (''), de modo que não será interpretado como o término da cadeia. Observe que a comparação de subcadeia implica que os valores de atributo não sejam valores atômicos (indivisíveis), conforme assumimos no modelo relacional formal (ver Seção 3.1).

Outro recurso permite o uso de aritmética nas consultas. Os operadores aritméticos padrão para adição (+), subtração (–), multiplicação (*) e divisão (/) podem ser aplicados a valores ou atributos numéricos com domínios numéricos. Por exemplo, suponha que queiramos ver o efeito de dar a todos os funcionários que trabalham no projeto 'ProdutoX' um aumento de 10 por cento; podemos fazer a Consulta 13 para ver quais seriam seus novos salários. Este exemplo também mostra como podemos renomear um atributo no resultado da consulta usando AS na cláusula SELECT.

Consulta 13. Mostrar os salários resultantes se cada funcionário que trabalha no projeto 'ProdutoX' receber um aumento de 10 por cento.

```
C13: SELECT  F.Pnome, F.Unome, 1,1 * F.Salario AS
             Aumento_salario
     FROM    FUNCIONARIO AS F, TRABALHA_EM
             AS T, PROJETO AS P
     WHERE   F.Cpf=T.Fcpf AND T.Pnr=P.Projnumero
             AND P.Projnome='ProdutoX';
```

Para os tipos de dados de cadeia, o operador de concatenação || pode ser usado em uma consulta para anexar dois valores de cadeia. Para tipos de dados data, hora, timestamp e intervalo, os operadores incluem incremento (+) ou decremento (–) de uma data, hora ou timestamp por um intervalo. Além disso, um valor de intervalo é o resultado da diferença entre dois valores de data, hora ou timestamp. Outro operador de comparação, que pode ser usado por conveniência, é **BETWEEN**, que está ilustrado na Consulta 14.

Consulta 14. Recuperar todos os funcionários no departamento 5 cujo salário esteja entre R$ 30.000 e R$ 40.000.

```
C14: SELECT  *
     FROM    FUNCIONARIO
     WHERE   (Salario BETWEEN 30.000 AND
             40.000) AND Dnr = 5;
```

A condição (Salario **BETWEEN** 30.000 **AND** 40.000) em C14 é equivalente à condição ((Salario >= 30.000) **AND** (Salario <= 40.000)).

4.3.6 Ordem dos resultados da consulta

A SQL permite que o usuário ordene as tuplas no resultado de uma consulta pelos valores de um ou mais dos atributos que aparecem, usando a cláusula **ORDER BY**. Isso é ilustrado pela Consulta 15.

Consulta 15. Recuperar uma lista dos funcionários e dos projetos em que estão trabalhando, ordenada por departamento e, dentro de cada departamento, ordenada alfabeticamente pelo sobrenome, depois pelo nome.

```
C15: SELECT   D.Dnome, F.Unome, F.Pnome,
              P.Projnome
     FROM     DEPARTAMENTO D, FUNCIONARIO
              F, TRABALHA_EM T, PROJETO P
     WHERE    D.Dnumero= F.Dnr AND F.Cpf=
              T.Fcpf AND T.Pnr= P.Projnumero
     ORDER BY D.Dnome, F.Unome, F.Pnome;
```

A ordem padrão está em ordem crescente de valores. Podemos especificar a palavra-chave **DESC** se quisermos ver o resultado em uma ordem decrescente de valores. A palavra-chave **ASC** pode ser usada para especificar a ordem crescente explicitamente. Por exemplo, se quisermos a ordem alfabética decrescente de Dnome e ordem crescente de Unome, Pnome, a cláusula ORDER BY da C15 pode ser escrita como

```
ORDER BY D.Dnome DESC, F.Unome ASC, F.Pnome
ASC
```

4.3.7 Discussão e resumo das consultas de recuperação da SQL básica

Uma *simples* consulta em SQL pode consistir em até quatro cláusulas, mas apenas as duas primeiras — SELECT e FROM — são obrigatórias. As cláusulas são especificadas na seguinte ordem, com aquelas entre colchetes [...] sendo opcionais:

```
SELECT      <lista atributos>
FROM        <lista tabelas>
[ WHERE     <condição> ]
[ ORDER BY  <lista atributos> ];
```

A cláusula SELECT lista os atributos a serem recuperados, e a cláusula FROM especifica todas as relações (tabelas) necessárias na consulta simples. A cláusula WHERE identifica as condições para selecionar as tuplas dessas relações, incluindo condições de junção, se necessário. ORDER BY especifica uma ordem para exibir os resultados de uma consulta. Duas cláusulas adicionais, GROUP BY e HAVING, serão descritas na Seção 5.1.8.

No Capítulo 5, apresentaremos recursos mais complexos das consultas SQL. Estes incluem os seguintes: consultas aninhadas, que permitem que uma consulta seja incluída como parte de outra consulta; funções de agregação, que são usadas para fornecer resumos da informação nas tabelas; duas cláusulas adicionais (GROUP BY e HAVING), que podem ser usadas para fornecer mais poder para as funções agregadas; e vários tipos de junções (*joins*) que podem combinar registros de várias tabelas de diferentes maneiras.

4.4 Instruções INSERT, DELETE e UPDATE em SQL

Em SQL, três comandos podem ser usados para modificar o banco de dados: INSERT, DELETE e UPDATE. Discutiremos cada um deles.

4.4.1 O comando INSERT

Em sua forma mais simples, INSERT é usado para acrescentar uma única tupla a uma relação. Temos de especificar o nome da relação e uma lista de valores para a tupla. Os valores devem ser listados *na mesma ordem* em que os atributos correspondentes foram especificados no comando CREATE TABLE. Por exemplo, para acrescentar uma nova tupla à relação FUNCIONARIO mostrada na Figura 3.5 e especificada no comando CREATE TABLE FUNCIONARIO... da Figura 4.1, podemos usar U1:

```
U1:  INSERT INTO  FUNCIONARIO
     VALUES       ( 'Ricardo', 'K', 'Marini',
                    '65329865388', '30-12-
                    1962', 'Rua Itapira, 44,
                    Santos, SP', 'M', 37.000,
                    '65329865388', 4 );
```

Uma segunda forma da instrução INSERT permite que o usuário especifique nomes de atributo explícitos que correspondem aos valores fornecidos no comando INSERT. Isso é útil se uma relação tiver muitos atributos, mas apenas alguns deles recebem valores em uma nova tupla. Porém, os valores precisam incluir todos os atributos com a especificação NOT NULL *e* nenhum valor padrão. Os atributos com NULL permitido ou com valores DEFAULT são aqueles que podem ser *omitidos*. Por exemplo, para inserir uma tupla para um novo FUNCIONARIO do qual conhecemos apenas os atributos Pnome, Unome, Dnr e Cpf, podemos usar U1A:

```
U1A: INSERT INTO  FUNCIONARIO (Pnome,
                  Unome, Dnr, Cpf)
     VALUES       ('Ricardo', 'Marini', 4,
                  '65329865388');
```

Os atributos não especificados em U1A são definidos como seu valor DEFAULT ou NULL, e os valores são listados na mesma ordem que os *atributos são listados no próprio comando INSERT*. Também é possível inserir em uma relação *múltiplas tuplas* separadas por vírgulas em um único comando INSERT. Os valores de atributo que formam *cada tupla* ficam entre parênteses.

Um SGBD que implementa totalmente a SQL deve aceitar e impor todas as restrições de integridade que podem ser especificadas na DDL. Por exemplo, se emitirmos o comando em U2 sobre o banco de dados mostrado na Figura 3.6, o SGBD deve *rejeitar* a operação, pois não existe uma tupla DEPARTAMENTO no banco de dados com Dnumero = 2. De modo semelhante, U2A seria *rejeitado* porque nenhum valor de Cpf é fornecido e essa é a chave primária, que não pode ser NULL.

```
U2:  INSERT INTO  FUNCIONARIO (Pnome,
                  Unome, Cpf, Dnr)
     VALUES       ('Roberto', 'Gomes',
                  '98076054011', 2);
```

(U2 é rejeitado se a verificação da integridade referencial for oferecida pelo SGBD.)

```
U2A: INSERT INTO  FUNCIONARIO (Pnome,
                  Unome, Dnr)
     VALUES       ('Roberto', 'Gomes', 5);
```

(U2 é rejeitado se a verificação de NOT NULL for oferecida pelo SGBD.)

Uma variação do comando INSERT inclui várias tuplas em uma relação em conjunto com a criação da relação e sua carga com o *resultado de uma consulta*. Por exemplo, para criar uma tabela temporária que possui o sobrenome do funcionário, o nome do projeto e as horas por semana para cada funcionário que trabalha em um projeto, podemos escrever as instruções em U3A e U3B:

```
U3A: CREATE TABLE  TRABALHA_EM_INFO
     ( Func_nome       VARCHAR(15),
       Proj_nome       VARCHAR(15),
       Horas_semanal   DECIMAL(3,1) );
U3B: INSERT INTO   TRABALHA_EM_INFO
                   ( Func_nome, Proj_nome,
                   Horas_por_semana )
     SELECT        F.Unome, P.Projnome,
                   T.Horas
     FROM          PROJETO P, TRABALHA_
                   EM T, FUNCIONARIO F
     WHERE         P.Projnumero=T.Pnr AND
                   T.Fcpf=F.Cpf;
```

Uma tabela TRABALHA_EM_INFO é criada por U3A e é carregada com a informação da junção recuperada do banco de dados pela consulta em U3B. Agora, podemos consultar TRABALHA_EM_INFO como faríamos com qualquer outra relação; quando não precisarmos mais dela, poderemos removê-la usando o comando DROP TABLE (ver Capítulo 5). Observe que a tabela TRABALHA_EM_INFO pode não estar atualizada; ou seja, se atualizarmos qualquer uma das relações PROJETO, TRABALHA_EM ou FUNCIONARIO depois de emitir U3B, a informação em TRABALHA_EM_INFO *pode ficar desatualizada*. Temos de criar uma visão, ou view (ver Capítulo 5), para manter essa tabela atualizada.

4.4.2 O comando DELETE

O comando **DELETE** remove tuplas de uma relação. Ele inclui uma cláusula WHERE, semelhante a que é usada em uma consulta SQL, para selecionar as tuplas a serem excluídas. As tuplas são explicitamente excluídas de apenas uma tabela por vez. No entanto, a exclusão pode se propagar para as tuplas

em outras relações, se *ações de disparo referencial* forem especificadas nas restrições de integridade referencial da DDL (ver Seção 4.2.2).[12] Dependendo do número de tuplas selecionadas pela condição na cláusula WHERE, zero, uma ou várias tuplas podem ser excluídas por um único comando DELETE. Uma cláusula WHERE inexistente especifica que todas as tuplas na relação deverão ser excluídas; porém, a tabela permanece no banco de dados como uma tabela vazia. Temos de usar o comando DROP TABLE para remover a definição da tabela (ver Capítulo 5). Os comandos DELETE em U4A a U4D, se aplicados de maneira independente ao banco de dados da Figura 3.6, excluirão zero, uma, quatro e todas as tuplas, respectivamente, da relação FUNCIONARIO:

```
U4A:  DELETE FROM   FUNCIONARIO
      WHERE         Unome='Braga';
U4B:  DELETE FROM   FUNCIONARIO
      WHERE         Cpf='12345678966';
U4C:  DELETE FROM   FUNCIONARIO
      WHERE         Dnr=5;
U4D:  DELETE FROM   FUNCIONARIO;
```

4.4.3 O comando UPDATE

O comando **UPDATE** é usado para modificar valores de atributo de uma ou mais tuplas selecionadas. Assim como no comando DELETE, uma cláusula WHERE no comando UPDATE seleciona as tuplas a serem modificadas em uma única relação. No entanto, a atualização de uma chave primária pode ser propagada para os valores de chave estrangeira das tuplas em outras relações se tal *ação de disparo referencial* for especificada nas restrições de integridade referencial da DDL (ver Seção 4.2.2). Uma cláusula **SET** adicional no comando UPDATE especifica os atributos a serem modificados e seus novos valores. Por exemplo, para alterar o local e número de departamento que controla o número de projeto 10 para 'Santo André' e 5, respectivamente, usamos U5:

```
U5:  UPDATE   PROJETO
     SET      Projlocal = 'Santo André', Dnum
              = 5
     WHERE    Projnumero=10;
```

Várias tuplas podem ser modificadas com um único comando UPDATE. Um exemplo é dar a todos os funcionários no departamento 'Pesquisa' um aumento de 10 por cento no salário, como mostra U6. Nesta solicitação, o valor de Salario modificado depende do valor de Salario em cada tupla, de modo que duas referências ao atributo Salario são necessárias. Na cláusula SET, a referência ao atributo Salario à direita refere-se ao antigo valor de Salario, *antes da modificação*, e aquele à esquerda refere-se ao novo valor de Salario, *após a modificação*:

```
U6:  UPDATE   FUNCIONARIO
     SET      Salario = Salario * 1,1
     WHERE    Dnr = 5;
```

Também é possível especificar NULL ou DEFAULT como o novo valor do atributo. Observe que cada comando UPDATE refere-se explicitamente a apenas uma única relação. Para modificar várias relações, precisamos emitir vários comandos UPDATE.

4.5 Recursos adicionais da SQL

A SQL possui uma série de recursos adicionais que não descrevemos neste capítulo, mas que discutiremos em outras partes do livro. São eles:

- No Capítulo 5, que é uma continuação deste capítulo, apresentaremos os seguintes recursos da SQL: diversas técnicas para especificar consultas de recuperação complexas, incluindo consultas aninhadas, funções de agregação, agrupamento, tabelas com junções (*join*), junções externas (*outer joins*) e consultas recursivas; visões (*views*), gatilhos (*triggers*) e asserções (*assertions*) da SQL; e comandos para modificação de esquema.
- A linguagem SQL possui diversas técnicas para escrever programas em várias linguagens de programação, que incluem instruções SQL para acessar um ou mais bancos de dados. Estas incluem SQL embutida (e dinâmica), SQL/CLI (Call Level Interface) e seu predecessor ODBC (Open Data Base Connectivity) e SQL/PSM (Persistent Stored Modules). Discutiremos essas técnicas no Capítulo 13. Também discutiremos como acessar bancos de dados SQL por meio da linguagem de programação Java usando JDBC e SQLJ.
- Cada SGBDR comercial terá, além dos comandos SQL, um conjunto de comandos para especificar parâmetros de projeto do

[12] Outras ações podem ser aplicadas automaticamente por meio de disparos (ver Seção 26.1) e outros mecanismos.

banco de dados físico, estruturas de arquivo para relações e caminhos de acesso, como índices. Chamamos esses comandos de *linguagem de definição de armazenamento* (*SDL*) no Capítulo 2. As primeiras versões da SQL tinham comandos para **criar índices**, mas estes foram removidos da linguagem porque não estavam no nível de esquema conceitual. Muitos sistemas ainda têm comandos CREATE INDEX.

- A SQL possui comandos de controle de transação. Estes são usados para especificar unidades de processamento de banco de dados para fins de controle de concorrência e recuperação. Discutiremos esses comandos no Capítulo 21, depois de discutirmos o conceito de transações com mais detalhes.
- A SQL possui construções da linguagem para especificar a *concessão* e *revogação de privilégios* aos usuários. Os privilégios normalmente correspondem ao direito de usar certos comandos SQL para acessar determinadas relações. Cada relação recebe um owner (proprietário), e este ou o DBA pode conceder a usuários selecionados o privilégio de usar uma instrução SQL — como SELECT, INSERT, DELETE ou UPDATE — para acessar a relação. Além disso, o DBA pode conceder os privilégios para criar esquemas, tabelas ou views a certos usuários. Esses comandos SQL — chamados de **GRANT** e **REVOKE** — serão discutidos no Capítulo 24, onde falaremos sobre segurança e autorização no banco de dados.
- A SQL possui construções de linguagem para a criação de triggers. Estas geralmente são conhecidas como técnicas de **banco de dados ativo**, pois especificam ações que são disparadas automaticamente por eventos, como atualizações no banco de dados. Discutiremos esses recursos na Seção 26.1, na qual abordaremos os conceitos de banco de dados ativo.
- A SQL incorporou muitos recursos dos modelos orientados a objeto para ter capacidades mais poderosas, levando a sistemas relacionais avançados, conhecidos como **objeto-relacional**. Capacidades como criar atributos complexos (também chamadas **relações aninhadas**), especificar tipos de dados abstratos (chamados **UDT**s ou tipos definidos pelo usuário) para atributos e tabelas, criar **identificadores de objeto** para referenciar tuplas e especificar **operações** sobre tipos serão discutidas no Capítulo 11.
- SQL e bancos de dados relacionais podem interagir com novas tecnologias, como XML (ver Capítulo 12) e OLAP (Capítulo 29).

Resumo

Neste capítulo, apresentamos a linguagem de banco de dados SQL. Essa linguagem e suas variações têm sido implementadas como interfaces para muitos SGBDs relacionais comerciais, incluindo Oracle e Rdb, da Oracle;[13] DB2, Informix Dynamic Server e SQL/DS, da IBM; SQL Server e Access, da Microsoft; e INGRES. Alguns sistemas de código aberto também fornecem SQL, como MySQL e PostgreSQL. A versão original da SQL foi implementada no SGBD experimental chamado SYSTEM R, que foi desenvolvido na IBM Research. A SQL foi projetada para ser uma linguagem abrangente, que inclui instruções para definição de dados, consultas, atualizações, especificação de restrição e definição de view. Discutimos os seguintes recursos da SQL neste capítulo: os comandos de definição de dados para criar tabelas, comandos para especificação de restrição, consultas de recuperação simples e comandos de atualização de banco de dados. No próximo capítulo, apresentaremos os seguintes recursos da SQL: consultas de recuperação complexas; views; triggers e assertions; e comandos de modificação de esquema.

Perguntas de revisão

4.1. Como as relações (tabelas) em SQL diferem das relações definidas formalmente no Capítulo 3? Discuta as outras diferenças na terminologia. Por que a SQL permite tuplas duplicadas em uma tabela ou em um resultado de consulta?

4.2. Liste os tipos de dados que são permitidos para atributos SQL.

4.3. Como a SQL permite a implementação das restrições de integridade de entidade e de integridade referencial descritas no Capítulo 3? E as ações de disparo referencial?

4.4. Descreva as quatro cláusulas na sintaxe de uma consulta de recuperação SQL simples. Mostre que tipo de construções pode ser especificado em cada uma das cláusulas. Quais são obrigatórias e quais são opcionais?

[13] O Rdb foi produzido originalmente pela Digital Equipment Corporation. Ele foi comprado da Digital pela Oracle em 1994, e está recebendo suporte e sendo melhorado.

Exercícios

4.5. Considere o banco de dados mostrado na Figura 1.2, cujo esquema aparece na Figura 2.1. Quais são as restrições de integridade referencial que devem ser mantidas no esquema? Escreva instruções DDL da SQL apropriadas para definir o banco de dados.

4.6. Repita o Exercício 4.5, mas use o esquema de banco de dados COMPANHIA AEREA da Figura 3.8.

4.7. Considere o esquema de banco de dados relacional BIBLIOTECA mostrado na Figura 4.6. Escolha a ação apropriada (rejeitar, propagar, SET NULL, SET DEFAULT) para cada restrição de integridade referencial, tanto para a *exclusão* de uma tupla referenciada quanto para a *atualização* de um valor de atributo de chave primária em uma tupla referenciada. Justifique suas escolhas.

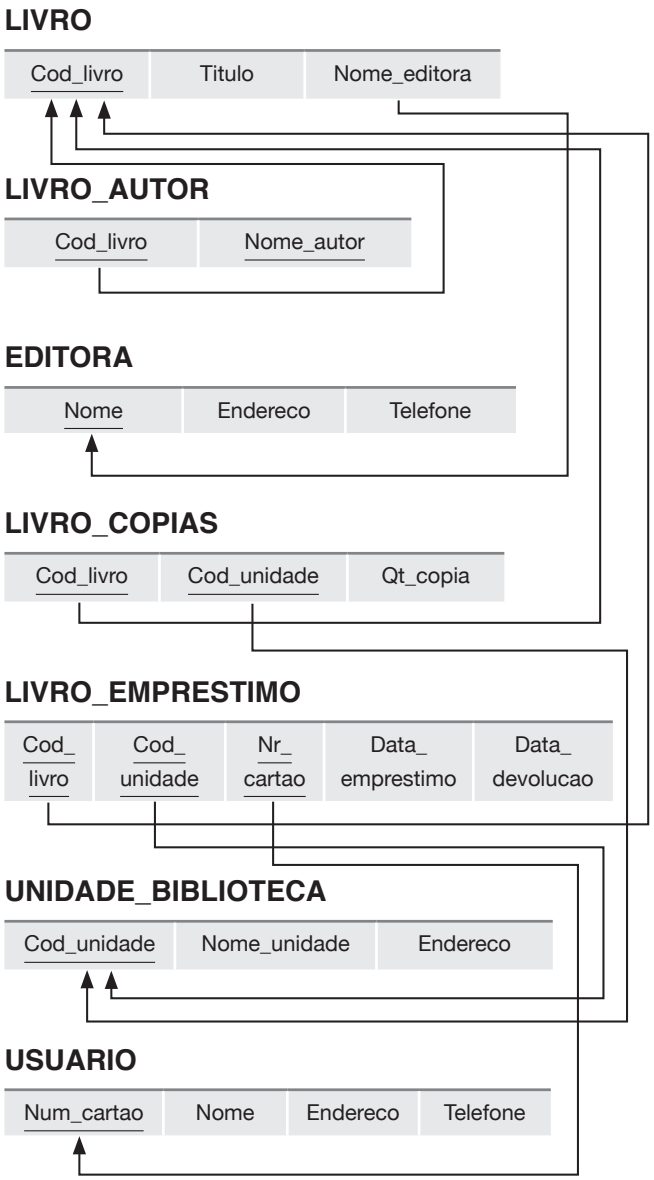

Figura 4.6
Um esquema de banco de dados relacional para um banco de dados BIBLIOTECA.

4.8. Escreva as instruções DDL da SQL apropriadas para declarar o esquema de banco de dados relacional BIBLIOTECA da Figura 4.6. Especifique as chaves e ações de disparo referencial.

4.9. Como as restrições de chave e chave estrangeira podem ser impostas pelo SGBD? A técnica de imposição que você sugere é difícil de implementar? As verificações de restrição podem ser executadas de modo eficiente quando as atualizações são aplicadas ao banco de dados?

4.10. Especifique as seguintes consultas em SQL sobre o esquema de banco de dados relacional EMPRESA mostrado na Figura 3.5. Mostre o resultado de cada consulta se ela for aplicada ao banco de dados EMPRESA na Figura 3.6.

- **a.** Recupere os nomes de todos os funcionários no departamento 5 que trabalham mais de 10 horas por semana no projeto ProdutoX.
- **b.** Liste os nomes de todos os funcionários que possuem um dependente com o mesmo primeiro nome que seu próprio.
- **c.** Ache os nomes de todos os funcionários que são supervisionados diretamente por 'Fernando Wong'.

4.11. Especifique as atualizações do Exercício 3.11 usando comandos de atualização da SQL.

4.12. Especifique as consultas a seguir em SQL no esquema de banco de dados da Figura 1.2.

- **a.** Recupere os nomes de todos os alunos sênior se formando em 'CC' (Ciência da computação).
- **b.** Recupere os nomes de todas as disciplinas lecionadas pelo Professor Kleber em 2007 e 2008.
- **c.** Para cada matéria lecionada pelo Professor Kleber, recupere o número da disciplina, semestre, ano e número de alunos que realizaram a matéria.
- **d.** Recupere o nome e o histórico de cada aluno sênior (Tipo_aluno = 4) formando em CC. Um histórico inclui nome da disciplina, número da disciplina, crédito, semestre, ano e nota para cada disciplina concluída pelo aluno.

4.13. Escreva instruções de atualização SQL para realizar ações sobre o esquema de banco de dados mostrado na Figura 1.2.

- **a.** Inserir um novo aluno <'Alves, 25, 1, 'MAT'>, no banco de dados.
- **b.** Alterar a turma do aluno 'Silva' para 2.
- **c.** Inserir uma nova disciplina, <'Engenharia do conhecimento', 'CC4390', 3, 'CC'>.
- **d.** Excluir o registro para o aluno cujo nome é 'Silva' e cujo número de aluno é 17.

4.14. Crie um esquema de banco de dados relacional para uma aplicação de banco de dados a sua escolha.

- **a.** Declare suas relações, usando a DDL da SQL.
- **b.** Especifique algumas consultas em SQL que sejam necessárias para sua aplicação de banco de dados.
- **c.** Com base em seu uso esperado do banco de dados, escolha alguns atributos que deverão ter índices específicos.
- **d.** Implemente seu banco de dados, se você tiver um SGBD que suporta SQL.

4.15. Considere que a restrição CHESUPERFUNC da tabela FUNCIONARIO, conforme especificado na Figura 4.2, seja mudada para:

```
CONSTRAINT CHESUPERFUNC
    REFERENCES FUNCIONARIO(Cpf)
                ON DELETE CASCADE ON
UPDATE CASCADE,
    FOREIGN KEY (Cpf_supervisor)
```

Responda às seguintes questões:

- **a.** O que acontece quando o comando a seguir é executado no estado de banco de dados mostrado na Figura 3.6?

```
DELETE FUNCIONARIO WHERE Unome = 'Brito'
```

- **b.** É melhor usar CASCADE ou SET NULL no caso da restrição ON DELETE de CHESUPERFUNC?

4.16. Escreva instruções SQL para criar uma tabela FUNCIONARIO_BACKUP para fazer o backup da tabela FUNCIONARIO mostrada na Figura 3.6.

Bibliografia selecionada

A linguagem SQL, originalmente chamada SEQUEL, foi baseada na linguagem SQUARE (Specifying Queries as Relational Expressions), descrita por Boyce et al. (1975). A sintaxe da SQUARE foi modificada para a SEQUEL (Chamberlin e Boyce, 1974) e depois para SEQUEL 2 (Chamberlin et al., 1976), na qual a SQL é baseada. A implementação original da SEQUEL foi feita na IBM Research, em San Jose, Califórnia. Mostraremos outras referências aos vários aspectos da SQL ao final do Capítulo 5.