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.