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: