Nota: Este material representa uma cópia resumida do capítulo 3 do livro Sistemas de Gerenciamento Sistemas de Gerenciamento de Banco de Dados - Ramakrishnan Gehrke (3ª edição), versão disponível em pdf na internet.

Modelo de Dados Relacional

Introdução ao Modelo Relacional
- Criando e Modificando Relações Usando SQL

Restrições de Integridade Sobre Relações
- Restrições de Chave
  - Especificando Restrições de Chave em SQL
- Restrições de Chave Estrangeira
  - Especificando Restrições de Chave Estrangeira em SQL
- Restrições Gerais
- Verificando Restrições de Integridade
  - Transações e Restrições

Consultando Dados Relacionais

Projeto Lógico do Banco de Dados Usando o Modelo ER
- Conjuntos de Entidades para Tabelas
- Conjuntos de Relacionamentos (sem Restrições) para Tabelas
- Mapeando Conjuntos de Relacionamentos com Restrições de Chave
- Mapeando Conjuntos de Relacionamentos com Restrições de Participação
- Mapeando Conjuntos de Entidades Fracas
- ??Mapeando Hierarquias de Classe
- ??Mapeando Diagramas ER com Agregação

Introdução a Visões
- Visões, Independência de Dados, Segurança
- Atualizações nas Visões
- Necessidade de Restringir Atualizações de Visões

Destruindo/Alterando Tabelas e Visões

Codd propôs o modelo de dados relacional em 1970. Naquela época, a maioria dos sistemas de banco de dados era baseada em um de dois modelos de dados mais antigos (o modelo hierárquico e o modelo de rede). Atualmente, o modelo relacional é de longe o modelo de dados dominante e é a base dos SGBDs líderes do mercado, incluindo a família DB2 da IBM, o Informix, o Oracle, o Sybase, o Access e o SQLServer, da Microsoft, o FoxBase e o Paradox.

- **SQL**: Originalmente desenvolvida como linguagem de consulta do SGBD relacional pioneiro da IBM, o System-R, a linguagem de consulta estruturada **(SQL, de Structured Query Language**) tornou-se a mais usada para criar, manipular e consultar SGBDs relacionais.
- **SQL:2023** ou ISO/IEC 9075:2023 (sob o título geral "Tecnologia da informação – Linguagens de banco de dados – SQL") é a nona edição do padrão ISO (1987) e ANSI (1986) para a linguagem de consulta de banco de dados SQL. Foi formalmente adotado em junho de 2023.

Um **banco de dados** é uma **coleção** de uma ou mais **relações**, em que cada **relação** é uma **tabela com linhas e colunas**.

Introdução ao Modelo Relacional

Um **banco de dados** é uma **coleção** de uma ou mais **relações**, em que cada **relação** é uma **tabela com linhas e colunas**. Um **banco de dados relacional** é uma **coleção** de relações com nomes distintos. O **esquema de banco de dados relacional** é a **coleção** de esquemas das relações presentes no banco de dados.

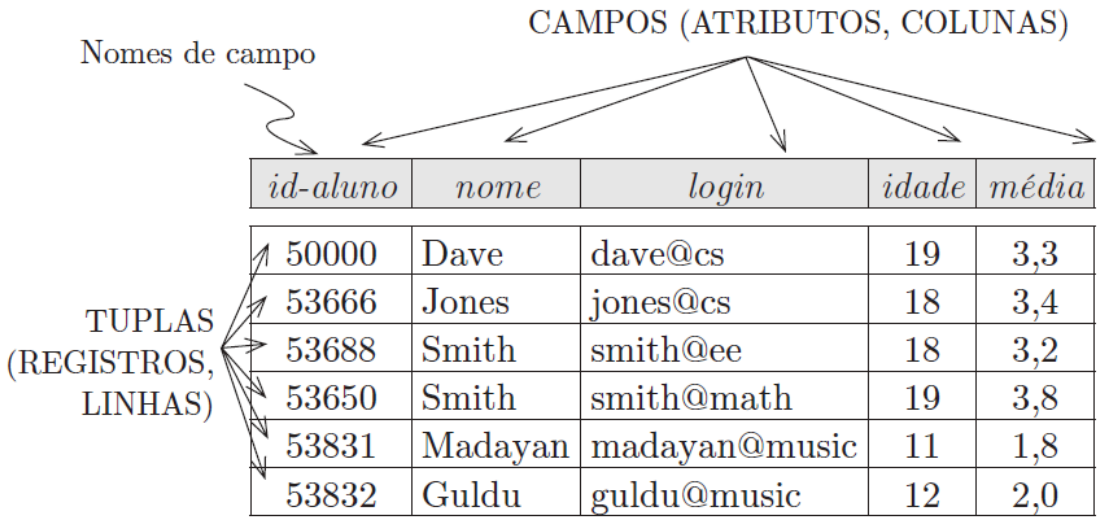
O principal **construtor** para representar dados no **modelo relacional** é a **relação**. Uma relação consiste em um **esquema de relação** e em uma **instância de relação**.

- a instância de relação é uma **tabela**.
- O esquema de relação descreve os cabeçalhos de **coluna (atributos, campo)** da tabela.
- O campo chamado *id-aluno* tem um **domínio (tipo)** denominado string. Em termos de linguagem de programação, o domínio de um campo é basicamente o tipo desse campo e restringe os valores que podem aparecer no campo.
- O **grau**, também chamado **aridade**, de uma relação é o número de **campos (colunas)**.
- A **cardinalidade** de uma instância de relação é o **número de tuplas (linhas)** que ela contém.

Primeiro, descreveremos o **esquema de relação** e depois a **instância de relação**. O esquema especifica o nome da relação, o nome de cada campo (ou coluna ou atributo) e o domínio de cada campo.

Alunos (id-aluno: string, nome: string, login: string,idade: integer, média: real)

Uma **instância de uma relação** é um conjunto de **tuplas**, também chamadas **registros (linhas)**, no qual cada tupla tem o mesmo número de campos que o esquema de relação. A instância A1 contém seis tuplas e, conforme esperávamos, com base no esquema, ela tem cinco campos. Note que não existem duas linhas idênticas. Isso é um requisito do modelo relacional — cada relação é definida como um conjunto de tuplas ou linhas únicas (Na prática podem existir nos sistemas comerciais).



id-aluno	*nome*	*login*	*idade*	*média*
50000	Dave	dave@cs	19	3,3
53666	Jones	jones@cs	18	3,4
53688	Smith	smith@ee	18	3,2
53650	Smith	smith@math	19	3,8
53831	Madayan	madayan@music	11	1,8
53832	Guldu	guldu@music	12	2,0

Figura 3.1 Uma instância *A1* da relação Alunos.

Criando e Modificando Relações Usando SQL

A linguagem SQL padrão usa a palavra tabela para denotar relação e frequentemente seguimos essa convenção ao discutirmos a SQL. A instrução CREATE TABLE é usada para definir uma nova tabela. Para criarmos a relação Alunos, podemos usar a seguinte instrução:

```
CREATE TABLE Alunos (id-aluno  CHAR(20),
                     nome      CHAR(30),
                     login     CHAR(20),
                     idade     INTEGER,
                     média     REAL )
```

A instrução CREATE TABLE é usada para definir uma nova tabela. Para criarmos a relação Alunos, podemos usar a seguinte instrução:

```
INSERT
INTO     Alunos   (id-aluno, nome, login, idade, média)
VALUES   (53688, 'Smith', 'smith@ee', 18, 3,2)
```

Podemos excluir tuplas usando o comando DELETE. Podemos excluir todas as tuplas de Alunos com nome igual a Smith, usando o comando:

```
DELETE
FROM     Alunos A
WHERE    A.nome = 'Smith'
```

Podemos modificar os valores de coluna em uma linha existente usando o comando UPDATE. Por exemplo, podemos incrementar a idade e decrementar a média do aluno com id-aluno 53688:

```
UPDATE   Alunos A
SET      A.idade = A.idade + 1, A.média = A.média – 1
WHERE    A.id-aluno = 53688
```

A cláusula WHERE é aplicada primeiro e determina quais linhas devem ser modificadas. Então, a cláusula SET determina como essas linhas devem ser modificadas. Se a coluna que está sendo modificada também é usada para determinar o novo valor, o valor usado na expressão no lado direito do sinal de igualdade (=) é o valor antigo; ou seja, antes da modificação. Para ilustrar melhor esses pontos, considere a seguinte variação da consulta anterior:

```
UPDATE   Alunos A
SET      A.média = A.média – 0,1
WHERE    A.média >=  3,3
```

Se essa consulta for aplicada na instância A1 de Alunos mostrada na Figura 3.1, obteremos a instância que aparece na Figura 3.3.

id-aluno	*nome*	*login*	*idade*	*média*
50000	Dave	dave@cs	19	3,2
53666	Jones	jones@cs	18	3,3
53688	Smith	smith@ee	18	3,2
53650	Smith	smith@math	19	3,7
53831	Madayan	madayan@music	11	1,8
53832	Guldu	guldu@music	12	2,0

Figura 3.3 Instância *A*1 de Alunos após a atualização.

Restrições de Integridade Sobre Relações

Uma **restrição de integridade (RI)** é uma condição especificada sobre um esquema de banco de dados e limita os dados que podem ser armazenados em uma instância do banco de dados. Já vimos um exemplo de restrição de integridade nas restrições de domínio associadas a um esquema de relação. Em geral, outros tipos de restrições também podem ser especificados; por exemplo, dois alunos não podem ter o mesmo valor de id-aluno.

Restrições de Chave

Garante que não haja identificações duplicadas de registros na relação.

- **Chave candidata e superchave:** Define conjuntos mínimos de campos que identificam de forma única as tuplas.
- **Chave primária:** Identifica a chave preferencial para referenciar eficientemente as tuplas e permite otimizações no banco de dados, como a criação de **índices**.

Especificando Restrições de Chave em SQL

Na SQL, podemos declarar que um subconjunto das colunas de uma tabela constituem uma chave, usando a restrição UNIQUE. No máximo **uma** dessas chaves candidatas pode ser declarada como **chave primária**, usando-se a restrição PRIMARY KEY.

```
CREATE TABLE Alunos ( id-aluno  CHAR(20),
                      nome      CHAR(30),
                      login     CHAR(20),
                      idade     INTEGER,
                      média     REAL,
                      UNIQUE    (nome, idade),
                      CONSTRAINT Chave Alunos PRIMARY KEY (id-aluno) )
```

Essa definição diz que id-aluno é a chave primária e a combinação de nome e idade também é uma chave. A definição da chave primária também ilustra como podemos nomear uma restrição, precedendo-a com CONSTRAINT nome-da-restrição. Se a restrição for violada, seu nome será retornado e poderá ser usado para identificar o erro.

Restrições de Chave Estrangeira

Restrições de **Chave Estrangeira** garantem consistência entre relações, exigindo que valores em uma relação estejam presentes em outra. A violação pode ocorrer durante inserções ou exclusões, afetando a integridade dos dados. O uso de **null** em chaves estrangeiras é permitido, mas não em chaves primárias.

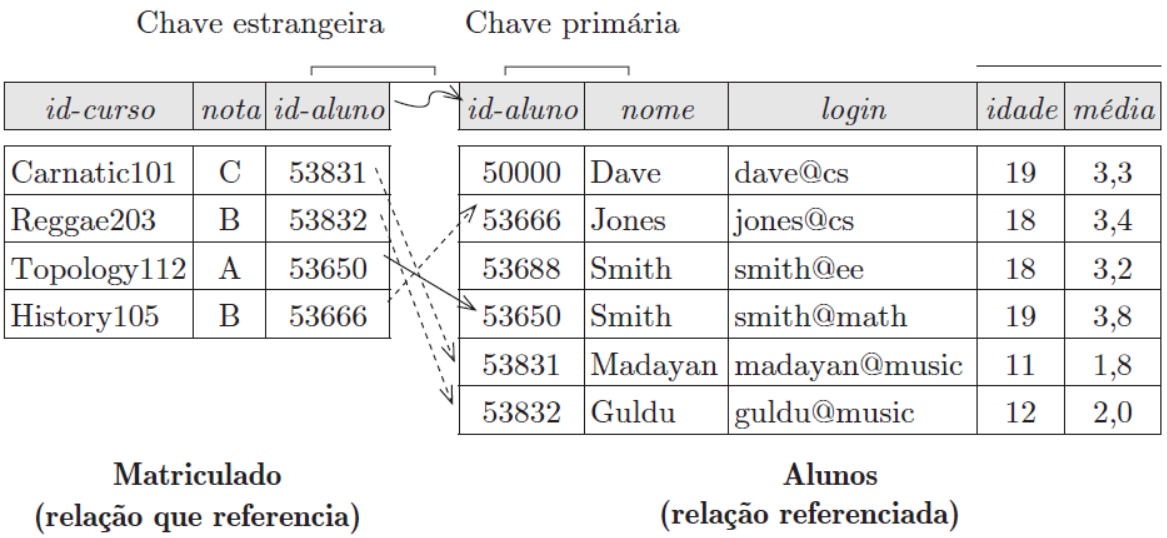


Figura 3.4 Integridade referencial.

O campo id-aluno de Matriculado é chamado **chave estrangeira** e se refere a Alunos. A **chave estrangeira** na relação de referência (Matriculado, em nosso exemplo) deve corresponder à **chave primária** da relação referenciada (Alunos); Se tentarmos inserir a tupla <55555, Art104, A> em M1, a RI será violada, pois não há nenhuma tupla em A1 com id-aluno 55555; Analogamente, se excluirmos a tupla <53666, Jones, jones@cs, 18, 3,4> de A1, violaremos a restrição de chave estrangeira, pois a tupla <53666,History105, B> em M1 contém o valor de id-aluno 53666, o id-aluno da tupla de Alunos excluída.

Especificando Restrições de Chave Estrangeira em SQL

Vamos definir Matriculado(id-aluno: string, id-curso: string, nota: string):

```
CREATE TABLE Matriculado ( id-aluno   CHAR(20),
                           id-curso   CHAR(20),
                           nota       CHAR(10),
                           PRIMARY KEY   (id-aluno, id-curso),
                           FOREIGN KEY   (id-aluno) REFERENCES Alunos )
```

A restrição de chave estrangeira diz que todo valor de id-aluno em Matriculado também deve aparecer em Alunos; ou seja, id-aluno em Matriculado é uma chave estrangeira referenciando Alunos.

Restrições Gerais

Restrições gerais, como limites de idade para alunos, são essenciais para garantir a integridade dos dados em sistemas de banco de dados relacionais. Elas vão além das **restrições de chave primária** e **estrangeira**.Os SGBDs relacionais atuais suportam restrições gerais na forma de **restrições de tabela** e **assertivas**. As restrições de tabela são associadas a uma única tabela e verificadas quando essa tabela é

modificada. Em contraste, as assertivas envolvem várias tabelas e são verificadas quando qualquer uma dessas tabelas é modificada.

A RI que diz que os alunos devem ser maiores de 16 anos pode ser considerada uma **restrição de domínio estendida**, pois estamos basicamente definindo o conjunto de valores de idade permitidos de maneira mais restrita do que é possível usando simplesmente um domínio padrão, como integer.

Verificando Restrições de Integridade

Considere a instância A1 de Alunos mostrada na Figura 3.1. A inserção a seguir viola a restrição de chave primária, porque já existe uma tupla com id-aluno 53688, e será rejeitada pelo SGBD:

```
INSERT
INTO    Alunos (id-aluno, nome, login, idade, media)
VALUES  (53688, 'Mike', 'mike@ee', 17, 3,4)
```

A inserção a seguir viola a restrição de que a chave primária não pode conter null:

```
INSERT
INTO    Alunos (id-aluno, nome, login, idade, media)
VALUES  (null, 'Mike', 'mike@ee', 17, 3,4)
```

Essa atualização viola a restrição de chave primária, pois já existe uma tupla com id-aluno 50000.

```
UPDATE  Alunos A
SET     A.id-aluno = 50000
WHERE   A.id-aluno = 53688
```

Além da instância A1 de Alunos, considere a instância de Matriculado mostrada na Figura 3.4. As exclusões de tuplas de Matriculado não violam a integridade referencial, mas as inserções de tuplas de Matriculado poderiam violar. A inserção a seguir é inválida, pois não há nenhuma tupla de Alunos com id-aluno 51111:

```
INSERT
INTO    Matriculado (id-curso, nota, id-aluno)
VALUES  ('Hindi101', 'B', 51111)
```

Por outro lado, inserções de tuplas em Alunos não violam a integridade referencial, e as exclusões poderiam causar violações. A SQL fornece várias maneiras alternativas de tratar de violações de chave estrangeira. Devemos considerar três perguntas básicas:

1. O que devemos fazer se uma linha de Matriculado é inserida, com um valor na coluna id-aluno que não aparece em nenhuma linha da tabela Alunos?
    - Nesse caso, o comando INSERT é simplesmente rejeitado.
2. O que devemos fazer se uma linha de Alunos é excluída? As opções são:
    - Excluir todas as linhas de Matriculado que referenciam a linha de Alunos excluída.
    - Proibir a exclusão da linha de Alunos, caso uma linha de Matriculado a referencie.
    - Configurar a coluna id-aluno com o valor de id-aluno de algum aluno (existente) “padrão”, para cada linha de Matriculado que referenciam a linha de Alunos excluída.

- Para cada linha de Matriculado que referencia a linha excluída, configurar a coluna id-aluno como null. Em nosso exemplo, esta opção entra em conflito com o fato de que id-aluno faz parte da chave primária de Matriculado e, portanto, não pode ser configurada como null. Assim, em nosso exemplo estamos limitados às três primeiras opções, embora esta quarta opção (configurar a chave estrangeira como null) esteja disponível em geral.

3. O que devemos fazer se o valor da chave primária de uma linha de Alunos for atualizada?
   - As opções aqui são semelhantes às do caso anterior. A SQL nos permite escolher qualquer uma das quatro opções em DELETE e UPDATE.
     Por exemplo, podemos especificar que, quando uma linha de Alunos é excluída, todas as linhas de Matriculado que se referem a ela também devem ser excluídas, mas que, quando a coluna id-aluno de uma linha de Alunos é modificada, essa atualização deve ser rejeitada, caso uma linha de Matriculado se refira à linha de Alunos modificada:

```
CREATE TABLE Matriculado ( id-aluno  CHAR(20),
                           id-curso  CHAR(20),
                           nota      CHAR(10),
                           PRIMARY KEY (id-aluno, id-curso),
                           FOREIGN KEY (id-aluno) REFERENCES Alunos
                                 ON DELETE CASCADE
                                 ON UPDATE NO ACTION )
```

As opções são especificadas como parte da declaração da chave estrangeira. A opção padrão é NO ACTION, que significa que a ação (DELETE ou UPDATE) deve ser rejeitada. A palavra-chave CASCADE diz que, se uma linha de Alunos for excluída, todas as linhas de Matriculado que se referem a ela também serão excluídas. Se a cláusula UPDATE especificasse CASCADE e a coluna id-aluno de uma linha de Alunos fosse atualizada, essa atualização também seria executada em cada linha de Matriculado que se referisse à linha de Alunos atualizada. Se uma linha de Alunos for excluída, podemos trocar a matrícula para um aluno ‘padrão’, usando ON DELETE SET DEFAULT. A solução correta nesse exemplo é excluir também todas as tuplas de matrícula do aluno excluído (isto é, CASCADE) ou rejeitar a atualização. A SQL também permite o uso de null como valor padrão, especificando-se ON DELETE SET NULL.

Transações e Restrições

Por padrão, uma restrição é verificada no final de cada instrução SQL que possa levar a uma violação e, se houver violação, a instrução será rejeitada. Às vezes essa estratégia é inflexível. Considere as variantes das relações Alunos e Cursos a seguir; todo aluno é obrigado a ter um curso de distinção, e todo curso é obrigado a ter um monitor, que é algum aluno.

```
CREATE TABLE Alunos ( id-aluno  CHAR(20),
                      nome      CHAR(30),
                      login     CHAR(20),
                      idade     INTEGER,
                      distinção CHAR(10)  NOT NULL,
                      media     REAL )
                      PRIMARY KEY (id-aluno),
                      FOREIGN KEY (distinção) REFERENCES Cursos (id-curso)

CREATE TABLE Cursos ( id-curso CHAR(10),
                      nomec    CHAR(10),
                      créditos INTEGER,
                      monitor CHAR(20)  NOT NULL,
                      PRIMARY KEY (id-curso)
                      FOREIGN KEY (monitor) REFERENCES Alunos (id-aluno))
```

Como vamos inserir a primeira tupla de curso ou de aluno? Uma não pode ser inserida sem a outra. A única maneira de realizar essa inserção é **adiando** a verificação da restrição, que normalmente seria feita no final de uma instrução INSERT. A SQL permite que uma restrição esteja no modo DEFERRED ou IMMEDIATE.

SET CONSTRAINT nome-restrição DEFERRED

Uma restrição no modo adiado (**deferred**) é verificada no momento da efetivação da transação (commit). Em nosso exemplo, as restrições de chave estrangeira sobre Alunos e Cursos podem ambas ser declaradas no modo adiado. Podemos então inserir um aluno com um curso de distinção inexistente (tornando o banco de dados temporariamente inconsistente), inserir o curso (restaurando a consistência) e depois efetivar e verificar se as duas restrições são satisfeitas.

Consultando Dados Relacionais

Uma linguagem de consulta é uma linguagem especializada para escrever consultas. A SQL é a linguagem de consulta comercial mais popular para um SGBD relacional. Considere a instância da relação Alunos mostrada na Figura 3.1.

```
SELECT  *
FROM    Alunos A
WHERE   A.idade < 18
```

id-aluno	*nome*	*login*	*idade*	*média*
53831	Madayan	madayan@music	11	1,8
53832	Guldu	guldu@music	12	2,0

Figura 3.6 Alunos com *idade* < 18 na instância *A1*.

O símbolo ‘*’ significa que manteremos no resultado todos os campos das tuplas selecionadas. Considere A uma variável que assume o valor de cada tupla em Alunos, uma tupla após a outra. A condição A.idade < 18 na cláusula WHERE especifica que queremos selecionar apenas as tuplas nas quais o campo idade tem um valor menor do que 18.

Também podemos combinar informações das relações Alunos e Matriculado. Se quiséssemos obter os nomes de todos os alunos que obtiveram A e a identificação do curso em que tiraram A, poderíamos escrever a seguinte consulta:

```
SELECT  A.nome, M.id-curso
FROM    Alunos A, Matriculado M
WHERE   A.id-aluno = M.id-aluno AND M.nota = 'A'
```

Essa consulta pode ser entendida como segue: "Se houver uma tupla A de Alunos e uma tupla M de Matriculado, tal que A.id-aluno = M.id-aluno (de modo que A descreve o aluno matriculado em M) e M.nota = 'A', então, imprima o nome do aluno e a identificação do curso". Quando avaliada nas instâncias de Alunos e Matriculado da Figura 3.4, essa consulta retorna uma única tupla, <Smith, Topology112>.

Projeto Lógico do Banco de Dados Usando o Modelo ER

Descreveremos agora como se faz para transformar um diagrama ER em uma coleção de tabelas com restrições associadas; ou seja, um esquema de banco de dados relacional.

Conjuntos de Entidades para Tabelas

Um conjunto de entidades é mapeado em uma relação de maneira direta: cada atributo do conjunto de entidades torna-se um atributo da tabela. Considere o conjunto de entidades Funcionários com atributos cpf, nome e vaga, mostrado na Figura 3.8.

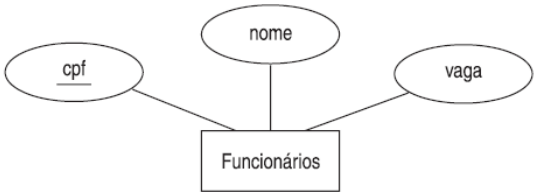


Figura 3.8 O conjunto de entidades Funcionários.

cpf	*nome*	*vaga*
123-22-3666	Attishoo	48
231-31-5368	Smiley	22
131-24-3650	Smethurst	35

Figura 3.9 Uma instância do conjunto de entidades Funcionários.

Note que conhecemos os domínios de cada atributo e a chave (primária) de um conjunto de entidades.
A instrução SQL a seguir captura as informações anteriores, incluindo as restrições de domínio e as informações de chave:

```
CREATE TABLE Funcionários ( cpf      CHAR(11),
                            nome     CHAR(30),
                            vaga     INTEGER,
                            PRIMARY  KEY (cpf) )
```

Conjuntos de Relacionamentos (sem Restrições) para Tabelas

Para representarmos um relacionamento, devemos identificar cada entidade participante e fornecer valores para os atributos descritivos do relacionamento. Assim, os atributos da relação incluem:

- Os atributos da chave primária de cada conjunto de entidades participantes, como campos de chave estrangeira.
- Os atributos descritivos do conjunto de relacionamentos.

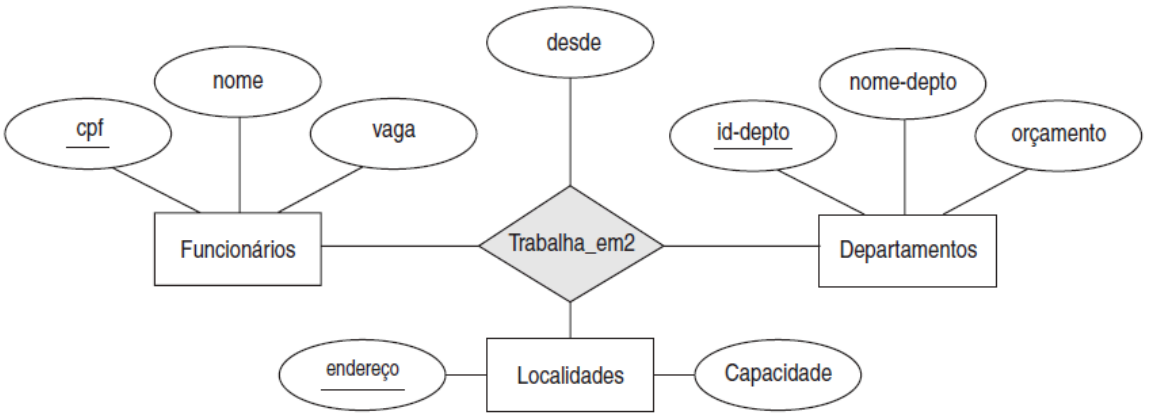


Figura 3.10 Um conjunto de relacionamentos ternário.

Todas as informações disponíveis sobre a tabela Trabalha_em2 são capturadas pela seguinte definição SQL:

```
CREATE TABLE Trabalha_em2 (cpf          CHAR(11),
                           id-depto     INTEGER,
                           endereço     CHAR(20) ,
                           desde        DATE,
                           PRIMARY KEY (cpf, id-depto, endereço),
                           FOREIGN KEY (cpf) REFERENCES Funcionários,
                           FOREIGN KEY (endereço) REFERENCES Localidades,
                           FOREIGN KEY (id-depto) REFERENCES  Departamentos )
```

Note que os campos endereço, id-depto e cpf não podem receber valores null. Como esses campos fazem parte da chave primária de Trabalha_em2, uma restrição NOT NULL está implícita para cada um deles. Também podemos especificar que uma ação em particular é desejada quando uma tupla **referenciada** de Funcionários, Departamentos ou Localidades é excluída, conforme explicado na discussão sobre restrições de integridade

Finalmente, considere o conjunto de relacionamentos Reporta_a, mostrado na Figura 3.11. Os indicadores de papel supervisor e subordinado são usados para criar nomes de campo significativos na instrução CREATE da tabela Reporta_a:

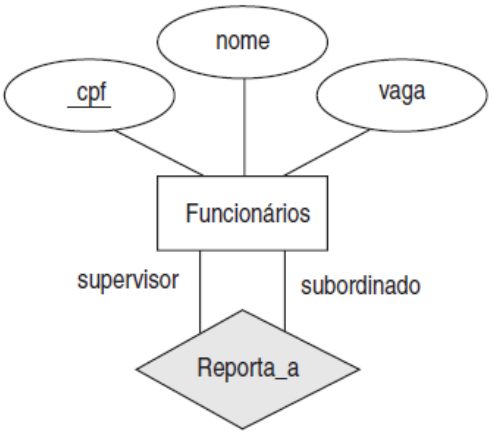


Figura 3.11 O conjunto de relacionamentos Reporta_a.

Observe que precisamos nomear explicitamente o campo referenciado de Funcionários, pois o nome do campo difere do(s) nome(s) do(s) campo(s) de referência.

```
CREATE TABLE Reporta_a (
        supervisor_cpf      CHAR(11),
        subordinado_cpf     CHAR(11),
        PRIMARY KEY   (supervisor_cpf, subordinado_cpf),
        FOREIGN KEY   (supervisor_cpf) REFERENCES Funcionários(cpf),
        FOREIGN KEY   (subordinado_cpf) REFERENCES Funcionários(cpf) )
```

Mapeando Conjuntos de Relacionamentos com Restrições de Chave

Se um conjunto de relacionamentos envolve n conjuntos de entidades e alguns m deles são vinculados por meio de setas no diagrama ER, a chave de qualquer um desses m conjuntos de entidades constitui uma chave para a relação na qual o conjunto de relacionamentos é mapeado. Considere o conjunto de relacionamentos Gerencia mostrado na Figura 3.12. A tabela correspondente a Gerencia tem os atributos cpf, id-depto, desde. Entretanto, como cada departamento tem no máximo um gerente, duas tuplas não podem ter o mesmo valor de id-depto e valores diferentes de cpf.

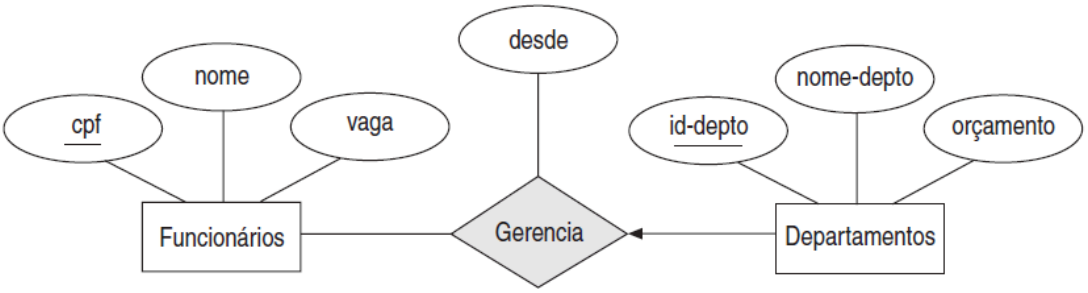


Figura 3.12 Restrição de chave em Gerencia.

```
CREATE TABLE Gerencia ( cpf        CHAR(11),
                        id-depto   INTEGER,
                        desde      DATE,
                        PRIMARY KEY (id-depto),
                        FOREIGN KEY (cpf) REFERENCES Funcionários,
                        FOREIGN KEY (id-depto) REFERENCES Departamentos)
```

Para evitar a criação de uma tabela distinta para o conjunto de relacionamentos é possível incluir as informações sobre o conjunto de relacionamentos na tabela correspondente ao conjunto de entidades que têm a chave, tirando proveito da restrição de chave. Essa estratégia elimina a necessidade de uma relação Gerencia separada. O único inconveniente dessa estratégia é que espaço poderia ser desperdiçado, caso vários departamentos não tivessem gerentes. Nesse caso, os campos adicionados teriam de ser preenchidos com valores null. A instrução SQL a seguir, definindo uma relação Depto_Gerencia que captura as informações sobre Departamentos e sobre Gerencia. Note que cpf pode receber valores null.

```
CREATE TABLE Depto_Gerencia ( id-depto     INTEGER,
                              nome-depto   CHAR(20),
                              orçamento    REAL,
                              cpf          CHAR(11),
                              desde        DATE,
                              PRIMARY      KEY (id-depto),
                              FOREIGN      KEY (cpf) REFERENCES Funcionários)
```

Mapeando Conjuntos de Relacionamentos com Restrições de Participação

Considere o diagrama ER da Figura 3.13, que mostra dois conjuntos de relacionamentos, Gerencia e Trabalha_em.

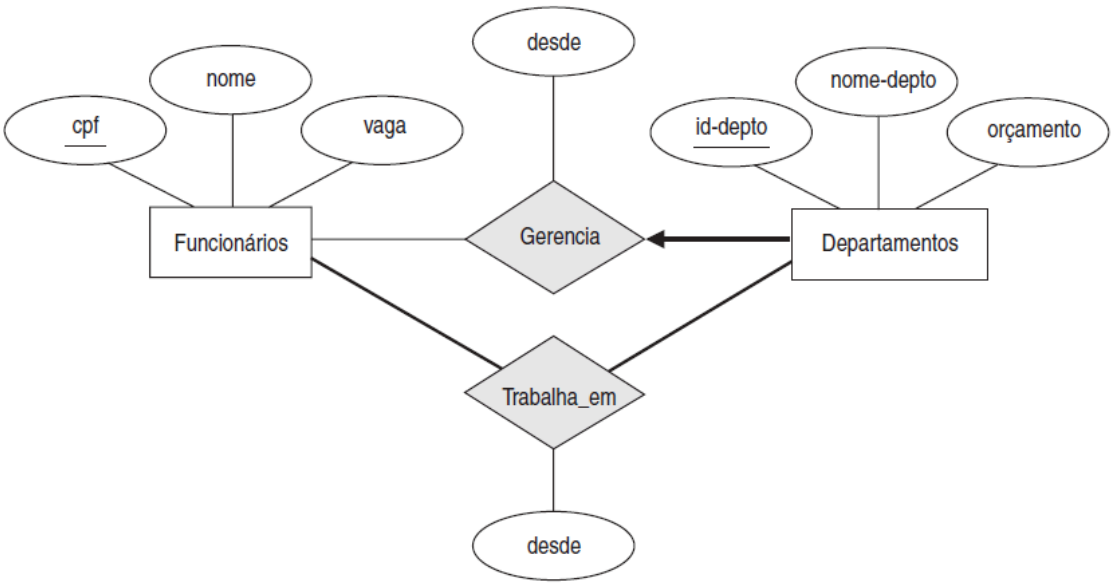


Figura 3.13 Gerencia e Trabalha_em.

Todo departamento é obrigado a ter um gerente, pela restrição de participação, e no máximo um gerente, pela restrição de chave. A instrução SQL a seguir reflete a segunda estratégia de mapeamento discutida na Seção anterior e utiliza a restrição de chave:

```
CREATE TABLE Depto_Gerencia ( id-depto      INTEGER,
                              nome-depto    CHAR(20),
                              orçamento     REAL,
                              cpf           CHAR(11) NOT NULL,
                              desde         DATE,
                              PRIMARY       KEY (id-depto),
                              FOREIGN       KEY (cpf) REFERENCES Funcionários
                                            ON DELETE NO ACTION)
```

A instrução SQL também captura a restrição de participação de que todo departamento deve ter um gerente: como cpf não pode receber valores null, cada tupla de Depto_Gerencia identifica uma tupla em Funcionários (que é o gerente). A especificação NO ACTION, que é o padrão e não precisa ser explícita, garante que uma tupla de Funcionários não pode ser excluída enquanto for apontada por uma tupla de Depto_Gerencia. Se quisermos excluir tal tupla de Funcionários, devemos primeiro alterar a tupla de Depto_Gerencia para termos um novo funcionário como gerente. (Poderíamos ter especificado CASCADE, em vez de NO ACTION, mas excluir todas as informações sobre um departamento apenas porque seu gerente foi demitido parece um pouco extremo!)

Infelizmente, existem muitas restrições de participação que não podemos capturar usando SQL sem usar **restrições de tabela** ou **assertivas**. As restrições de tabela e as assertivas podem ser especificadas usando-se o poder total da linguagem de consulta SQL e são muito expressivas, mas também muito dispendiosas para verificar e garantir.

Considere a relação Trabalha_em obtida pelo mapeamento do diagrama ER em relações. Ela contém os campos cpf e id-depto, os quais são chaves estrangeiras referindo-se a Funcionários e a Departamentos. Para garantirmos a participação total de Departamentos em Trabalha_em, precisamos garantir que cada valor de id-depto em Departamentos apareça em uma tupla de Trabalha_em. Poderíamos tentar garantir essa condição declarando que id-depto em Departamentos é uma chave estrangeira referindo-se a Trabalha_em, mas essa não é uma restrição de chave estrangeira válida, porque id-depto não é uma chave candidata para Trabalha_em.

Para garantirmos a participação total de Departamentos em Trabalha_em usando SQL, precisamos de uma assertiva. Precisamos garantir que cada valor de id-depto em Departamentos apareça em uma tupla de Trabalha_em; além disso, essa tupla de Trabalha_em também não deve ter valores null nos campos que são chaves estrangeiras referenciando outros conjuntos de entidades envolvidos no relacionamento (neste exemplo, o campo cpf). Podemos garantir a segunda parte dessa restrição impondo o requisito mais forte de que cpf em Trabalha_em não pode conter valores null. (Garantir que a participação de Funcionários em Trabalha_em seja total e simétrica.)

Mapeando Conjuntos de Entidades Fracas

Um conjunto de entidades fracas sempre participa de um relacionamento binário de um-para-muitos e tem uma restrição de chave e participação total

Considere o conjunto de entidades fracas Dependentes, mostrado na Figura 3.14, com **chave parcial** nomed. Uma entidade de Dependentes pode ser identificada univocamente apenas se tomarmos a chave da entidade de Funcionários proprietária e a chave nomed da entidade de Dependentes deve ser excluída se a entidade de Funcionários proprietária for excluída.

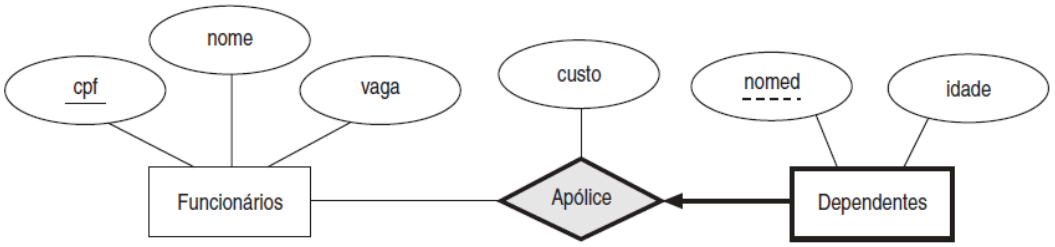


Figura 3.14 O conjunto de entidades fracas Dependentes.

Podemos capturar a semântica desejada com a seguinte definição da relação Apólice_Dep:

```
CREATE TABLE Apólice_Dep ( nomed      CHAR(20),
                           idade      INTEGER,
                           custo      REAL,
                           cpf        CHAR(11),
                           PRIMARY KEY (nomed, cpf),
                           FOREIGN KEY (cpf) REFERENCES Funcionários
                           ON DELETE CASCADE )
```

Observe que a chave primária é <nomed, cpf>, pois Dependentes é uma entidade fraca. Precisamos garantir que cada entidade de Dependentes seja associada a uma entidade de Funcionários (a proprietária), de acordo com a restrição de participação total em Dependentes. Isto é, cpf não pode ser null. Isso é garantido, pois cpf faz parte da chave primária. A opção CASCADE garante que as informações sobre a apólice e os dependentes de um funcionário sejam excluídas, caso a tupla de Funcionários correspondente seja removida.

??Mapeando Hierarquias de Classe

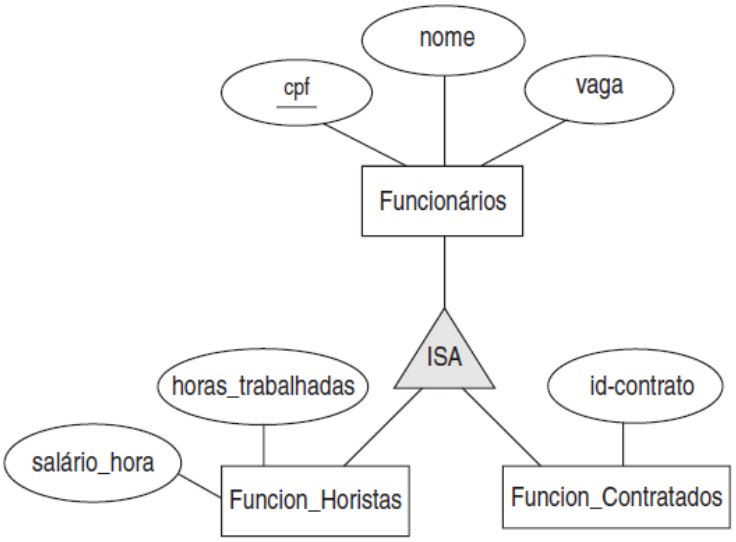


Figura 3.15 Hierarquia de classes.

??Mapeando Diagramas ER com Agregação

Os conjuntos de entidades Funcionários, Financia e Departamentos e o conjunto de relacionamentos Financia são mapeados conforme descrito nas seções anteriores. Para o conjunto de relacionamentos Monitora, criamos uma relação com os seguintes atributos: os atributos de chave de Funcionários (cpf), os atributos de chave de Financia (id-depto, id-projeto) e os atributos descritivos de Monitora (até). Essa transformação é basicamente o mapeamento padrão para um conjunto de relacionamentos

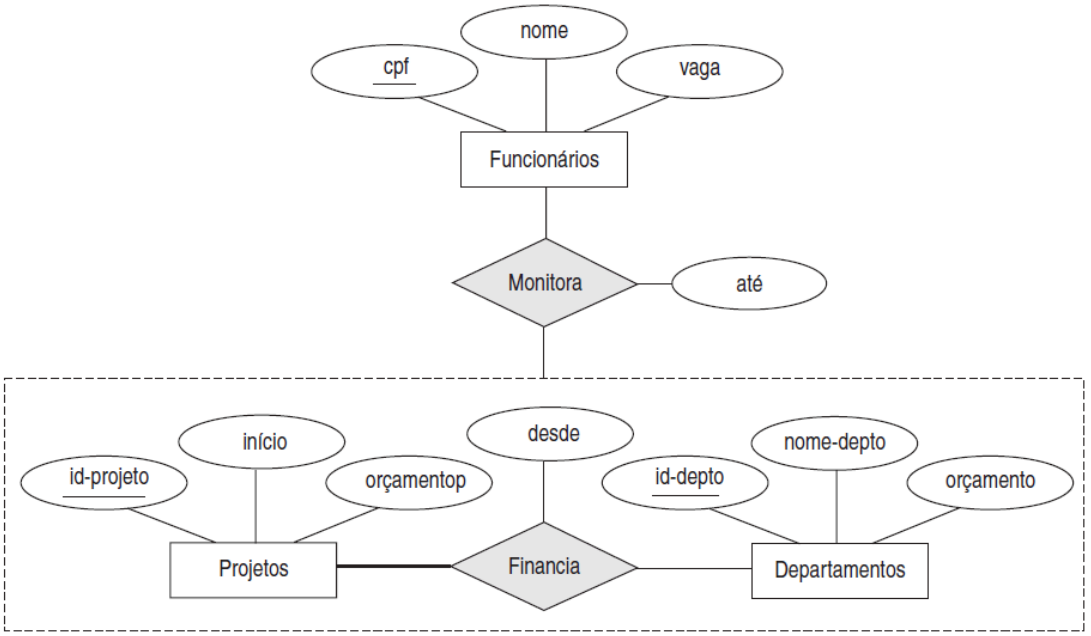


Figura 3.16 Agregação.

Existe um caso especial no qual esse mapeamento pode ser refinado, eliminando-se a relação Financia. Considere a relação Financia. Ela tem os atributos id-projeto, iddepto e desde; e, em geral, precisamos dela (além de Monitora) por duas razões:

- 1. Precisamos registrar os atributos descritivos (em nosso exemplo, desde) do relacionamento Financia.
- 2. Nem todo financiamento tem um monitor e, assim, alguns pares <id-projeto, id-depto> na relação Financia podem não aparecer na relação Monitora.

Entretanto, se Financia não tem atributos descritivos e tem participação total em Monitora, toda instância possível da relação Financia pode ser obtida a partir das colunas <id-projeto, id-depto> de Monitora; Financia pode ser eliminada.

Introdução a Visões

Uma **visão (views)** é uma tabela cujas linhas não são armazenadas explicitamente no banco de dados, mas são calculadas conforme for necessário. Suponha que estejamos frequentemente interessados em localizar os nomes e identificadores de aluno dos estudantes que tiraram nota B em algum curso, junto com o identificador de curso.

```
CREATE VIEW Estudantes-B (nome, id-aluno, curso)
       AS SELECT A.nome, A.id-aluno, M.id-curso
       FROM   Alunos A, Matriculado M
       WHERE  A.id-aluno = M.id-aluno AND M.nota = 'B'
```

A visão Estudantes-B tem três campos, chamados nome, id-aluno e curso, com os mesmos domínios dos campos nome e id-aluno em Alunos e id-curso em Matriculado. (Se os argumentos opcionais nome, id-aluno e curso forem omitidos do comando CREATE VIEW, serão herdados os nomes de coluna nome, id-aluno e id-curso.)

Essa visão pode ser usada exatamente como uma **tabela base**, ou tabela armazenada explicitamente, na definição de novas consultas ou visões. Estudantes-B contém as tuplas que aparecem na Figura 3.18. Conceitualmente, quando Estudantes-B é usada em uma consulta, a definição da visão é primeiramente avaliada para obter a instância de Estudantes-B correspondente e, depois, o restante da consulta é avaliado, tratando Estudantes-B como qualquer outra relação referida na consulta.

nome	*id-aluno*	*curso*
Jones	53666	History105
Guldu	53832	Reggae203

Figura 3.18 Uma instância da visão Estudantes-B.

Visões, Independência de Dados, Segurança

O esquema físico de um banco de dados relacional descreve como as relações do esquema conceitual são armazenadas. Este esquema conceitual é a coleção de esquemas das relações armazenadas no banco de dados. O mecanismo de visão fornece suporte para independência lógica de dados no modelo relacional, permitindo definir relações no esquema externo que mascaram as alterações no esquema conceitual. As visões também são úteis para segurança, permitindo definir restrições de acesso para grupos de usuários. Por exemplo, podemos criar uma visão que permite aos alunos verem o nome e a idade de outros alunos, mas não sua média, restringindo o acesso à tabela base Alunos.

Atualizações nas Visões

A motivação existente por trás do mecanismo de visões é personalizar a maneira como os usuários vêem os dados. Os usuários não devem ter de se preocupar com a distinção entre visão e tabela base. Esse objetivo é atingido no caso de consultas em visões; uma visão pode ser usada exatamente como qualquer outra relação na definição de uma consulta. Entretanto, é natural querer especificar também atualizações em visões.
O padrão SQL-92 permite que atualizações sejam especificadas apenas em visões definidas em uma única tabela base, usando apenas seleção e projeção, sem nenhum uso de operações agregadas. Tais visões são chamadas de visões atualizáveis. Essa definição é bastante simplificada, mas captura o espírito das restrições. Uma atualização em uma visão restrita sempre pode ser implementada pela atualização da tabela base subjacente, de maneira não ambígua.

```
CREATE VIEW EstudantesBons (id-aluno, media)
       AS SELECT A.id-aluno, A.media
          FROM   Alunos A
          WHERE  A.media > 3,0
```

Uma observação importante é a de que um comando INSERT ou UPDATE pode alterar a tabela base subjacente, de modo que a linha resultante (isto é, inserida ou modificada) não esteja na visão! Por exemplo, se tentarmos inserir uma linha <51234, 2,8 > na visão, essa linha poderá ser (preenchida com valores null nos outros campos de Alunos e então) adicionada na tabela Alunos subjacente, mas ela não aparecerá na visão EstudantesBons, pois não satisfaz a condição da visão media > 3,0. A ação padrão da SQL é permitir essa inserção, mas podemos proibir isso adicionando a cláusula WITH CHECK OPTION na definição da visão. Nesse caso, apenas as linhas que realmente aparecem na visão são inserções permitidas.

Necessidade de Restringir Atualizações de Visões

Embora as regras da SQL sobre visões atualizáveis sejam mais rigorosas do que o necessário, existem alguns problemas fundamentais nas atualizações especificadas em visões e há um bom motivo para limitar a classe de visões atualizáveis. Considere a relação Alunos e uma nova relação chamada Clubes:

Clubes(nomec: string, ano_filiação: date, nomem: string)

Suponha que estejamos frequentemente interessados em localizar os nomes e logins dos alunos com média maior do que 3, que pertençam a pelo menos um clube, junto com o nome do clube e a data em que entraram nele. Podemos definir uma visão para esse propósito:

```
CREATE VIEW EstudantesAtivos (nome, login, club, desde)
       AS SELECT  A.nome, A.login, C.nomec, C.ano_filiação
          FROM    Alunos A, Clubes C
          WHERE   A.snome = C.nomem AND A.media > 3
```

Considere as instâncias de Alunos e Clubes mostradas nas Figuras 3.19 e 3.20. Quando avaliada usando as instâncias C e A3, EstudantesAtivos contém as linhas mostradas na Figura 3.21.

nomec	*ano_filiação*	*nomem*
Sailing	1996	Dave
Hiking	1997	Smith
Rowing	1998	Smith

Figura 3.19 Uma instância C de Clubes.

id-aluno	*nome*	*login*	*idade*	*média*
50000	Dave	dave@cs	19	3,3
53666	Jones	jones@cs	18	3,4
53688	Smith	smith@ee	18	3,2
53650	Smith	smith@math	19	3,8

Figura 3.20 Uma instância $A3$ de Alunos.

nome	*login*	*clube*	*desde*
Dave	dave@cs	Sailing	1996
Smith	smith@ee	Hiking	1997
Smith	smith@ee	Rowing	1998
Smith	smith@math	Hiking	1997
Smith	smith@math	Rowing	1998

Figura 3.21 Instância de EstudantesAtivos.

Agora, suponha que queremos excluir a linha <Smith, smith@ee, Hiking, 1997> de EstudantesAtivos. Como fazemos isso? As linhas de EstudantesAtivos não estão armazenadas fisicamente, mas são calculadas conforme o necessário. Assim, precisamos alterar Alunos ou Clubes (ou ambas) de tal maneira que a avaliação da definição da visão na instância modificada não produza a linha <Smith, smith@ee, Hiking, 1997 >. Essa tarefa pode ser realizada de duas maneiras: excluindo-se a linha <53688, Smith, smith@ee, 18, 3,2 > de Alunos ou excluindo-se a linha <Hiking, 1997, Smith> de Clubes. Mas nenhuma das duas soluções é satisfatória. Na verdade,a única solução razoável é proibir essas atualizações nas visões.

Destruindo/Alterando Tabelas e Visões

Se decidirmos que não precisamos mais de uma tabela base e quisermos destruí-la (isto é, excluir todas as linhas e remover as informações de definição da tabela), podemos usar o comando DROP TABLE. Por exemplo, DROP TABLE Alunos RESTRICT destrói a tabela Alunos, a menos que alguma visão ou restrição de integridade se refira a Alunos; se assim for, o comando falhará. Se a palavra-chave RESTRICT for substituída por CASCADE, Alunos será eliminada e todas as visões ou restrições de integridade associadas também serão eliminadas (recursivamente). Uma visão pode ser eliminada usando-se o comando DROP VIEW, que é exatamente como DROP TABLE.

ALTER TABLE modifica a estrutura de uma tabela existente. Para adicionarmos uma coluna chamada nome-família em Alunos, por exemplo, usaríamos o seguinte comando:

```
ALTER TABLE Alunos
        ADD COLUMN nome-família CHAR(10)
```

A definição de Alunos é modificada para adicionar essa coluna, e todas as linhas existentes são preenchidas com valores null nessa coluna. ALTER TABLE também pode ser usado para excluir colunas e adicionar ou eliminar restrições de integridade em uma tabela;

Referências:

- Fundamentos de Sistemas de Gerenciamento de Banco de Dados - Elmasri & Navathe (6ª edição)
- Sistemas de Gerenciamento Sistemas de Gerenciamento de Banco de Dados - Ramakrishnan Gehrke (3ª edição)