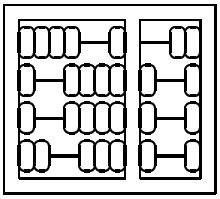
Fundamentos de Bancos de Dados

Modelagem, Projeto e Linguagem SQL

Célio Cardoso Guimarães



Instituto de Computação

IC – UNICAMP

Julho 2003

Sumário

Capítulo 2

Modelagem de Dados

2.1 Projeto "top down" de Bases de Dados

Como em muitas aplicações de engenharia, o projeto de uma Base de Dados através da técnica "top down" ou de refinamentos sucessivos, é largamente utilizado. Ele começa pela análise dos requisitos dos usuários finais da BD e da "visão externa" que eles têm sobre os dados. Esta visão e requisitos dependem da aplicação pretendida da BD, variam de um usuário para outro dentro da organização, e refletem suas necessidades para o trabalho diário. Ela é comumente informal e incompleta, em graus que variam com o nível de informatização da aplicação (ou da organização). É comum procedimentos manuais se intercalarem ou se sobreporem a procedimentos automatizados via computador. A descrição desses procedimentos aparece em documentos, relatórios e muitas vezes de forma verbal. Se a BD for corporativa as interfaces com outras áreas ou departamentos da organização são muitas vezes incompletas ou ineficientes. Um exemplo de ineficiência é a emissão de um relatório por um departamento A para um departamento B, onde B redigita os dados do relatório para integração com seus próprios dados e processamento posterior.

Os objetivos finais dessa análise são: (i) obter uma visão unificada de todos os dados da aplicação e, (ii) definir os "procedimentos funcionais" para operar com os dados. É, portanto, uma sistemática similar à de "análise de sistemas" convencional. Esta visão unificada dos dados é comumente chamada de "modelagem de dados" e corresponde a uma abstração do mundo real contendo o conjunto de informações sobre o mesmo que julgamos importante armazenar e manipular. O projeto "top down" da BD através da "modelagem de dados" consiste em especificar os dados através de refinamentos sucessivos, mapeando os dados definidos num nível mais alto e abstrato para o nível seguinte, menos abstrato e mais detalhado.

No nível mais alto a visão e requisitos da BD ainda é informal e é normalmente apresentada sob a forma de documentos textuais. Vamos denominá-la de "Visão Externa dos Dados" ou "Modelagem Externa de Dados".

O próximo passo consiste em mapear esta visão externa dos dados para um "modelo conceitual de dados" de nível suficientemente alto para esconder detalhes de implementação, mas ao mesmo tempo suficientemente detalhado para descrever os diversos tipos de dados requeridos pela aplicação, os seus inter-relacionamentos e algumas regras de consistência. Esse nível é chamado de "Modelagem ou Projeto Conceitual de Dados".

Em 1976 uma forma gráfica e sucinta de Modelagem Conceitual foi proposta por [Chen76] e denominada de **Modelo Entidade - Relacionamento (MER)**. Ela teve grande aceitação por ser um meio de comunicação do Projeto Conceitual de fácil compreensão por usuários finais da BD (possivelmente leigos), e ao mesmo tempo ser capaz de formalizar vários aspectos do projeto. A Modelagem Conceitual utilizando o MER consiste em se projetar uma série de "Diagramas Entidade - Relacionamento" que descrevem os dados da BD e os seus inter-relacionamentos. Seu estudo será o objetivo desse capítulo.

O próximo nível consiste na "especificação lógica" dos dados num formato adequado ao SGBD escolhido para a implementação da BD. Ele é chamado de "Projeto Lógico de Dados" e, dependendo do SGBD escolhido, pode ser de nível suficientemente alto para esconder a maioria dos detalhes de implementação. Fazendo uma analogia com linguagens de programação, ele corresponde, aproximadamente, ao projeto das estruturas de dados de um programa, onde os tipos de dados são completamente definidos.

O mapeamento de "Diagramas Entidade - Relacionamento" para o nível lógico pode ser feito através de ferramentas CASE sofisticadas ou manualmente. Quando estudarmos o Modelo Relacional veremos que este mapeamento é relativamente simples de ser feito manualmente.

O nível lógico se situa na fronteira da modelagem de dados que é independente do SGBD daquela que é específica ao SGBD.

O último nível é denominado de "projeto físico dos dados" e corresponde à organização interna do armazenamento dos dados pelo SGBD e à definição de estruturas de dados auxiliares visando uma maior eficiência na recuperação e manipulação dos dados. Dependendo do SGBD uma parte considerável do nível físico fica escondida das aplicações.

A Modelagem de Dados normalmente não contempla o mapeamento dos "procedimentos funcionais" (vistos na visão externa acima) em programas para manipular a BD, embora esta seja uma das atividades mais complexas do projeto completo de uma aplicação de BD.

A especificação dos dados de uma BD é comumente chamada de "esque-

ma dos dados" e também se pode chamar os níveis vistos acima de "esquema externo", "esquema conceitual", "esquema lógico" e "esquema físico" ou "esquema interno" da BD. É importante distinguir o "esquema de uma BD", que reflete o projeto e a especificação dos dados (em geral fixos), do conteúdo específico da BD que é chamado de " instância da BD", normalmente variável no tempo.

A Fig. 2.1 mostra os níveis da abordagem "top down" no projeto de uma BD.

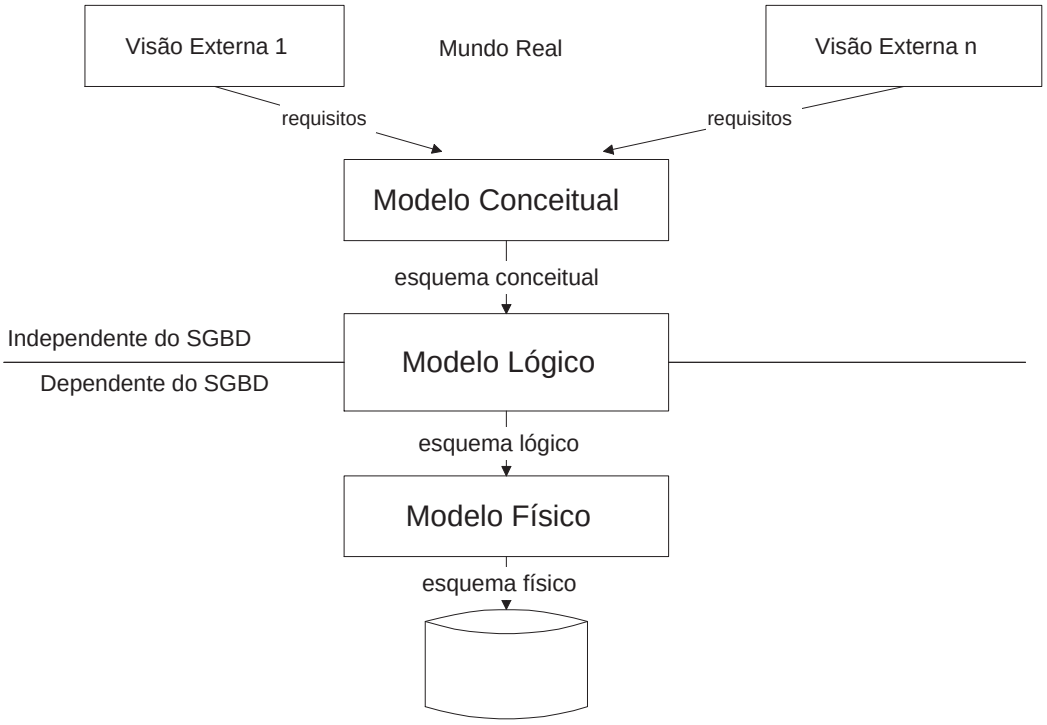


Figura 2.1: Projeto "Top Down" de uma Base de Dados

2.2 O Modelo Entidade - Relacionamento

Uma "**entidade**" é um objeto ou ente do mundo real que possui existência própria e cujas características ou propriedades desejamos registrar. Ela pode ter uma existência física (uma pessoa, um carro, um livro, uma peça) ou abstrata (um departamento, um projeto, um curso).

Uma entidade é caracterizada por algumas propriedades específicas que achamos importante registrar (para os objetivos principais da Base de Dados) e que serão denominadas de **atributos**. Cada atributo de uma entidade

possui um nome e um "valor" específico para a entidade. Por exemplo o atributo "peso" da entidade "Ana Paula" poderia ter o valor 55Kg.

Entidades podem ter uma miríade de propriedades, especialmente as que têm existência física, mas apenas algumas são relevantes para a aplicação pretendida da Base de Dados. Por exemplo, um indivíduo que é funcionário de uma empresa, tem características físicas distintivas (peso, altura, sexo, cor da pele, cor do cabelo, idade), sociais (ascendência, descendência, riqueza, posição social), mentais (QI, formação escolar, linguas, etc), habilidades esportivas, culturais, psicológicas, etc. Para o setor de RH da empresa onde ele trabalha apenas algumas dessas características são importantes (nome, RG, CPF, endereço, profissão, sexo, etc). Se esse indivíduo passar a ser paciente de uma clínica ou hospital, os atributos importantes para os registros da clínica ou hospital serão muito distintos (peso, altura, doenças pregressas, cirurgias, tratamentos, alergias, pressão arterial, etc, etc) e pouca interseção terão com os atributos registrados no setor de RH da empresa onde ele trabalha.

Em geral queremos registrar os atributos de diversas entidades similares entre si, ou seja, do "mesmo tipo"; por exemplo, os funcionários de uma empresa ou os livros de uma biblioteca. Para este fim agrupamos entidades similares em um "**Conjunto de Entidades**" para o qual damos um nome significativo como "Funcionários" e "Livros" no exemplo acima. Cada entidade de um Conjunto de Entidades possui a mesma lista de atributos, embora os valores desses atributos variem de uma entidade para outra.

Um **Conjunto de Entidades** (**CE**) é um conjunto matemático no sentido de que todos os seus elementos são distintos, e não existe nenhuma ordem intrínseca entre eles. Isto implica que "valores correspondentes" dos atributos de duas entidades não podem ser **todos** iguais. Em outras palavras, a lista de atributos de um CE deve ser suficiente para caracterizar completamente qualquer entidade do conjunto.

Exemplo: suponhamos que queremos registrar para um conjunto de pérolas as seguintes informações: cor, diâmetro, peso, lote; elas podem não ser suficientes para distinguir duas pérolas que podem ter os mesmos valores para cor, diametro, peso e lote. Se quisermos que essas pérolas façam parte de um Conjunto de Entidades algumas outras propriedades teriam que ser incluídas como local da extração, empresa, pescador, etc. Um exemplo mais realista é o caso de itens fabricados em série cujos atributos mensuráveis são idênticos; nesse caso é comum distingui-los através de um "número de série" unívoco impresso no item.

Estas considerações nos levam ao ponto principal na especificação de um CE que é o conceito de **Atributo Determinante** (ou "Atributo Chave"). Um "Atributo Determinante" de um CE é um dos atributos do CE especi-

almente projetado para identificar de forma única qualquer entidade do CE, ou seja, dadas duas entidades quaisquer e_1 e e_2 do CE em questão, os valores do Atributo Determinante são distintos para e_1 e e_2. Em alguns casos práticos é conveniente estender esta definição de forma que um subconjunto dos atributos do CE seja o "Atributo Determinante" do CE como no exemplo das pérolas visto acima. Outros exemplos serão vistos posteriormente. É importante enfatizar a expressão acima "especialmente projetado", porque a unicidade do valor do atributo determinante deve ser garantida para qualquer conteúdo futuro do CE e não apenas para a instância atual do CE.

Resumindo: um CE fica especificado no Modelo Entidade - Relacionamento, dado o nome do CE, os nomes dos atributos do CE e dentre esses o nome do atributo determinante. Uma forma textual de definir um CE poderia ser por exemplo:

Funcionários(numf, RG, CPF, nome, endereço, salário)

No exemplo acima o atributo determinante *numf* aparece sublinhado. A forma gráfica de representar um Conjunto de Entidades consiste em colocar o seu nome dentro de um retângulo e os nomes de cada atributo no final de uma aresta saindo do retângulo ou dentro de uma oval no final da aresta, [1] conforme mostra a figura 2.2:

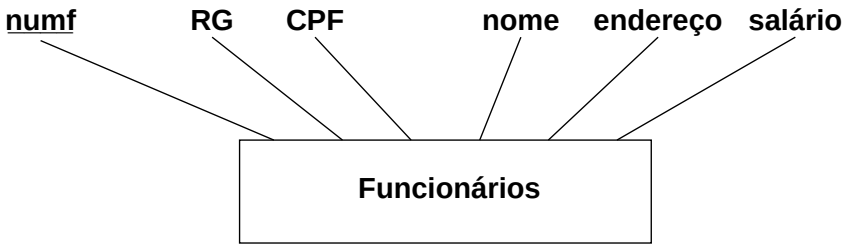


Figura 2.2: Representação de um Conjunto de Entidades no MER

2.2.1 Tipos de Atributos

Um atributo pode conter vários sub-atributos. Nesse caso ele se diz "composto". Por exemplo, o atributo "endereço" pode conter os sub-atributos Local, Cidade e CEP. Local, por sua vez poderia conter os sub-atributos Rua, Num

[1] Por economia de notação (e de esforço!) usaremos nos exemplos a primeira opção; no entanto, diversos livros da área preferem a opção com ovais.

e Compl. O valor de endereço é a lista dos valores dos sub-atributos Rua, Num, Compl, Cidade e CEP. A representação gráfica desse atributo composto é mostrada na figura 2.3:

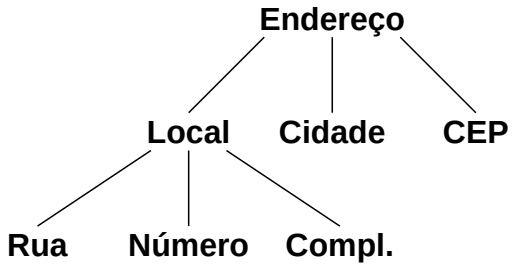


Figura 2.3: Exemplo de Atributo Composto

Se um atributo de uma entidade pode tomar diversos valores então ele se diz "multivalorado". Por exemplo, um funcionário pode ter diversos telefones que constituem um atributo multivalorado do CE Funcionários. Outro exemplo: um livro pode ter diversos autores e tratar de diversos assuntos. Esta propriedade pode ser indicada colocando-se um asterisco * após os nomes dos atributos autores e assuntos. O CE Livros poderia então, ser especificado assim:

Livros (num-livro, autores, assuntos*)*

Atributos multivalorados podem ser representados num diagrama ER de forma compacta, colocando-se um asterisco após o seu nome. Outra forma comumente usada quando se usa a notação oval, é colocar o nome do atributo dentro de uma oval com borda dupla.

2.2.2 Relacionamentos

Um **relacionamento** entre dois CEs é uma lista (conjunto) de pares de entidades, onde cada par representa uma associação entre uma entidade de um CE com outra entidade do outro CE e que estabelece uma inter-dependência entre essas entidades. Por exemplo, podemos representar as "Lotações" de Funcionários em Departamentos através de uma lista de pares ordenados $(f_i, d_j), (f_k, d_l), \ldots$, com o significado: o funcionário f_i está lotado no departamento d_j, f_k está lotado no departamento d_l e assim por diante. Sob o ponto de vista matemático esse relacionamento é representado por um subconjunto do produto cartesiano Funcionários x Departamentos, que contém todos os possíveis pares (funcionário, departamento).

Neste exemplo, vários funcionários podem estar lotados num único departamento. Dizemos então que o relacionamento "Lotações" de Funcionários com Departamentos é de **classe** ou **multiplicidade** N para 1 ou N : 1. Usaremos, por economia, o termo classe no que se segue.

A representação gráfica desse relacionamento é feita através de um losango contendo o nome do relacionamento e unindo ao losango, via segmentos de reta, os CEs Funcionários e Departamentos conforme mostra a figura 2.4. A classe do relacionamento está indicada pelos símbolos N e 1. Para determinar sua posição correta pode-se usar o seguinte argumento: "dado um funcionário existe apenas um departamento no qual ele pode estar lotado (indicado pelo símbolo "1" ao lado do CE Departamentos, e dado um departamento, existem N funcionários nele lotados (indicado pelo símbolo "N" ao lado do CE Funcionários)".

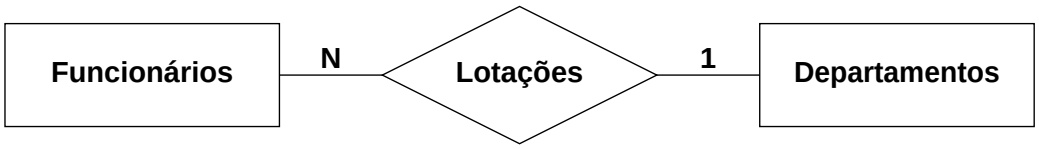


Figura 2.4: Relacionamento N : 1

Se uma entidade de um CE só pode estar associada a uma única entidade de outro CE e vice-versa, dizemos que o relacionamento é de classe 1 para 1. Por exemplo, Departamentos são gerenciados por Funcionários de tal forma que dado um departamento existe um único funcionário que o gerencia e dado um funcionário ele pode ser o gerente de um único departamento (ou de nenhum). Essas regras estão explicitadas na figura 2.5.

Figura 2.5: Relacionamento 1 : 1

A classe de um relacionamento é dita N : N (ou muitos para muitos, ou N : M) quando uma entidade de um CE pode estar associada a várias outras entidades de outro CE e vice-versa. Por exemplo, um funcionário

pode participar de diversos projetos e um projeto pode ter a participação de diversos funcionários. Esse relacionamento está representado na figura 2.6 e tem o nome "Participações".

Figura 2.6: Relacionamento N : N (muitos para muitos)

Muitas vezes queremos explicitar que **todas** as entidades de um CE devem fazer parte de um dado relacionamento com outro CE. Nesse caso diremos que o relacionamento é **total** entre o primeiro CE e o segundo CE. Por exemplo, no relacionamento Lotações visto acima, podemos requerer que todo funcionário esteja lotado em um departamento. Isto seria indicado por uma marca oval ou circular conforme mostra a figura 2.7. A ausência desta marca no ângulo do losango voltado para Departamentos indica que o relacionamento de Departamentos com Funcionários é **parcial**. Isto permitiria, por exemplo, um novo departamento ser criado sem ter funcionários. Este é um exemplo de uma "regra de consistência" ou "regra de negócio" suportada pelo MER.

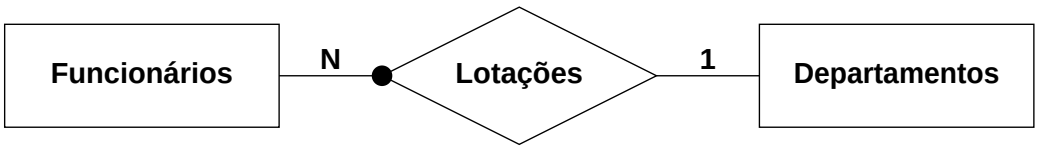


Figura 2.7: Relacionamento total

Relacionamentos podem ter atributos. Por exemplo, o relacionamento N : N para indicar a associação de Materiais com seus Fornecedores pode indicar para cada par (m_i, f_j) do relacionamento, o preço, o prazo e o lote (quantidade) que o fornecedor f_j estabelece para fornecer o material m_i. A figura 2.8 mostra a representação gráfica desse relacionamento.

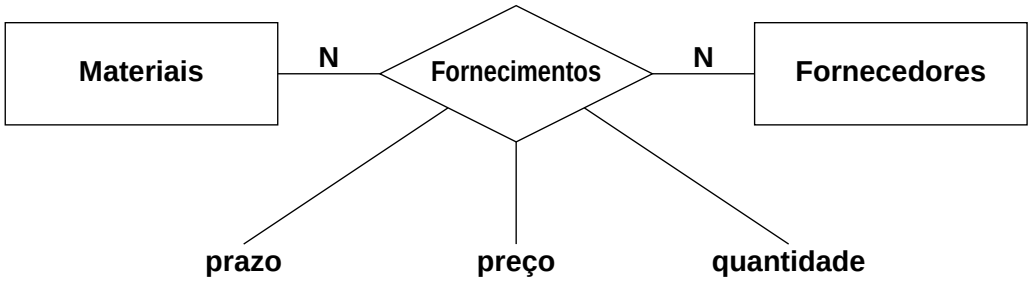


Figura 2.8: Relacionamento com Atributos

Poderíamos questionar nesse exemplo, se um dado atributo, digamos, preço, pertence de fato ao relacionamento em vez de a um dos Conjuntos de Entidades envolvidos. Um teste simples é capaz de esclarecer a dúvida. Fixe o material e varie o fornecedor: se o preço varia então o atributo não é do material; em seguida, fixe o fornecedor e varie o material: se o preço varia então o atributo não é do fornecedor. Fica claro nesse caso, que o atributo preço é do relacionamento, isto é, cada par (material, fornecedor) possui um preço. Idem para os atributos prazo e lote.

2.2.3 Conjuntos de Entidades Fracos

Há casos em que a existência de um CE está vinculada à existência de outro CE. Um exemplo típico é o registro, para fins de seguro-saúde ou imposto de renda, dos dependentes de um funcionário. Nesse caso o registro só faz sentido para a empresa porque o dependente está "ligado" ao funcionário. Se diz então, que o CE "Dependentes" é um "Conjunto de Entidades Fraco". O CE Funcionários é ás vezes chamado de 'conjunto pai" ou "conjunto mestre" e Dependentes é às vezes chamado de "conjunto detalhe".

CEs fracos não possuem um atributo determinante "per se", mas em geral utilizam o atributo determinante do "conjunto pai" para construir o seu atributo determinante. Por exemplo, o atributo determinante de Dependentes poderia ser o par (numf, nome-dependente).

CEs fracos são representados no MER por retângulos de borda dupla e seu relacionamento com o "conjunto pai" por um losango também de borda dupla. A figura 2.9 mostra o diagrama ER para esse exemplo:

2.2.4 Auto-relacionamentos

Muitas vezes queremos fazer o relacionamento de um CE consigo mesmo. Por exemplo, dado o CE Peças queremos saber quais peças são componentes

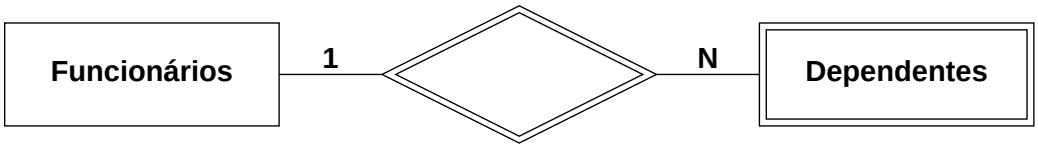


Figura 2.9: Exemplo de Conjunto de Entidades Fraco

de uma dada peça ou, dada uma peça quais peças a têm como componente. Esses dois relacionamentos podem ser representados pelo diagrama da figura 2.10. Observe que cada uma das ligações do losango com o CE Peças recebeu um rótulo. O primeiro rótulo significa: "uma peça é componente de outra peça", e o segundo rótulo significa "uma peça tem como componente" outra peça. Os rótulos das ligações explicitam "o papel" (em inglês: "role") que a peça desempenha no relacionamento. Este papel é normalmente evidente nos relacionamentos normais, mas precisa ser explicitado nos auto-relacionamentos. Observe também que o relacionamento é da classe N : N, isto é, uma peça pode ter vários componentes e uma dada peça pode ser componente de várias peças, conforme mostra a figura 2.10.

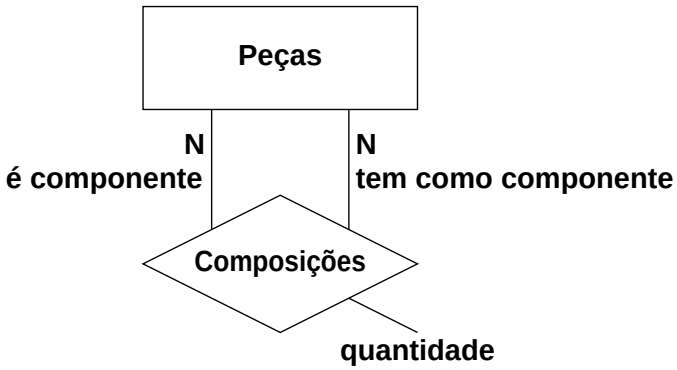


Figura 2.10: Exemplo de Auto-relacionamento

Consideremos um outro exemplo onde temos dois auto-relacionamentos entre funcionários: o primeiro, "Gerencia", de classe 1 : N, significa que um funcionário pode ser o gerente de vários funcionários e o segundo, "É gerenciado", de classe N : 1, significa que vários funcionários podem ser gerenciados por 1 gerente (figura 2.11).

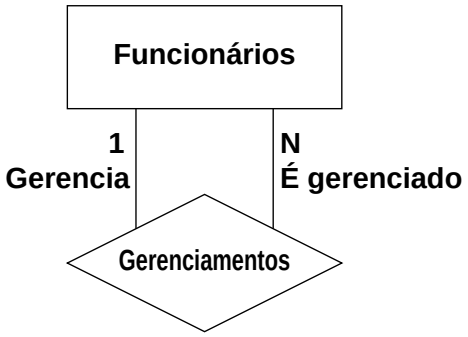


Figura 2.11: Outro exemplo de Auto-relacionamento

Os exemplos vistos até agora são de relacionamentos envolvendo dois CEs. Eles são ditos binários ou de grau 2 e são os mais comuns na prática. O **grau** de um relacionamento é o número de CEs envolvidos no relacionamento. A figura 2.12 mostra um relacionamento de grau 3, ou triplo, entre Professores, Alunos e Disciplinas.

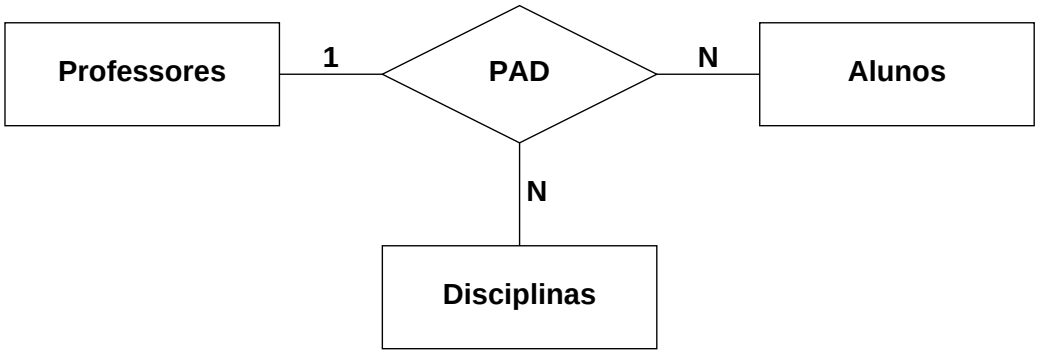


Figura 2.12: Exemplo de Relacionamento Triplo

A classe desse relacionamento, 1 : N : N, pode ser interpretada da seguinte forma:

- dada uma disciplina temos diversos alunos e 1 professor,
- dado um aluno ele pode tomar diversas disciplinas com um professor,
- dado um professor ele pode ministrar diversas disciplinas para diversos alunos.

A figura 2.13 mostra um relacionamento triplo chamado MRP entre Materiais, Pedidos e Requisições, típico de um sistema de compras de materiais, onde requisições vindas de diversos setores são agrupadas em pedidos

ou ordens de compra e se deseja relacionar cada pedido com as requisições originais. O relacionamento possui dois atributos: quantidade requisitada e quantidade pedida. O relacionamento tem classe N : N : N e é total do lado de Pedidos e de Requisições (toda requisição está associada a um ou mais pedidos e pode conter vários materiais, idem para pedidos), mas é parcial do lado de Materiais (um dado material pode não estar sendo comprado no momento).

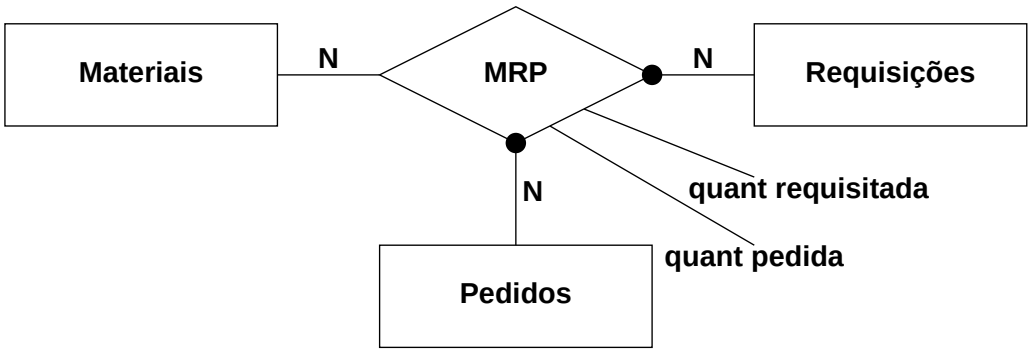


Figura 2.13: Outro exemplo de Relacionamento Triplo

2.2.5 Extensões ao Modelo Entidade - Relacionamento

Agregações

Há casos em que relacionamentos de grau superior a 2 não capturam as "regras de negócio" desejadas. Por exemplo, no relacionamento triplo MRP visto anteriormente entre Materiais, Requisições e Pedidos (Ordens de Compra), uma requisição está relacionada com um ou mais materiais e com um ou mais pedidos (ou com nenhum deles), o que é artificial. Requisições precedem a pedidos no tempo e originalmente estão relacionadas apenas com Materiais, já que toda requisição é criada para comprar um ou mais materiais. Elas se originam em diversos setores da empresa e são encaminhadas ao setor de compras. O setor de compras associa um pedido de compra ao par (requisição, material), portanto, ao relacionamento, especificando a quantidade a ser comprada (que pode ser diferente da requisitada, por exemplo, para obter um desconto maior ou devido a alguma restrição de tamanho de lote de venda). Esta separação de funções implica na existência de dois relacionamentos distintos; o segundo é chamado de "agregação" porque o relacionamento de Materiais com Requisições é "agregado" em um pseudo CE que por sua vez

se relaciona com Pedidos através do relacionamento "Itens de pedidos". A figura 2.14 mostra como representar esse conceito num diagrama ER:

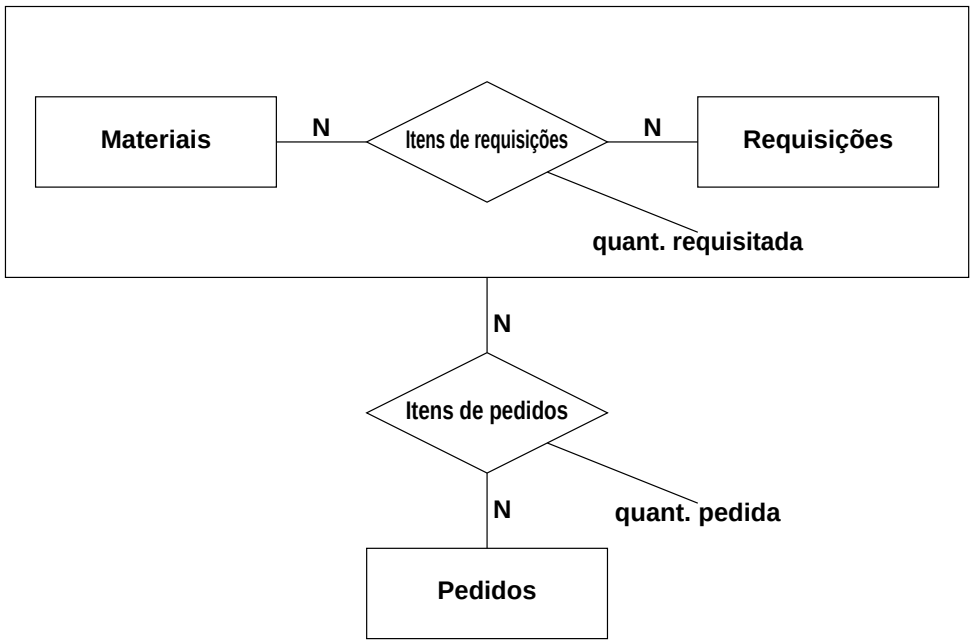


Figura 2.14: Exemplo de Agregação

Posteriormente, quando estudarmos o mapeamento do Modelo Conceitual para o Modelo Relacional, veremos que relacionamentos N : N serão representados por "tabelas auxiliares" e, portanto, se comportam como CEs, o que torna muito simples representar esse relacionamento especial no Modelo Relacional.

Especialização

As técnicas de Orientação a Objetos tiveram várias influências sobre o projeto de Bases de Dados. Uma delas é o conceito de sub-classe e herança. Muitas vezes queremos registrar características especiais de certos subconjuntos de um CE. Por exemplo, no CE Funcionários temos secretárias, técnicos, engenheiros, gerentes, etc, e para cada uma dessas categorias queremos guardar alguns atributos específicos como habilidades das secretárias em digitação, informática, línguas, etc. Outros atributos seriam requeridos para engenheiros e assim por diante. Seria ineficiente estender o CE Funcionários com todos esses atributos que só teriam valores para cada grupo específico de funcionários ficando vazios os não correspondentes ao grupo. Para este fim criamos os CEs denominados Secretárias, Técnicos, Engenheiros, etc, e um

tipo especial de relacionamento desses CEs com Funcionários chamado "é-um" (do inglês "is-a") e representado por um triângulo conforme mostra a figura 2.15.

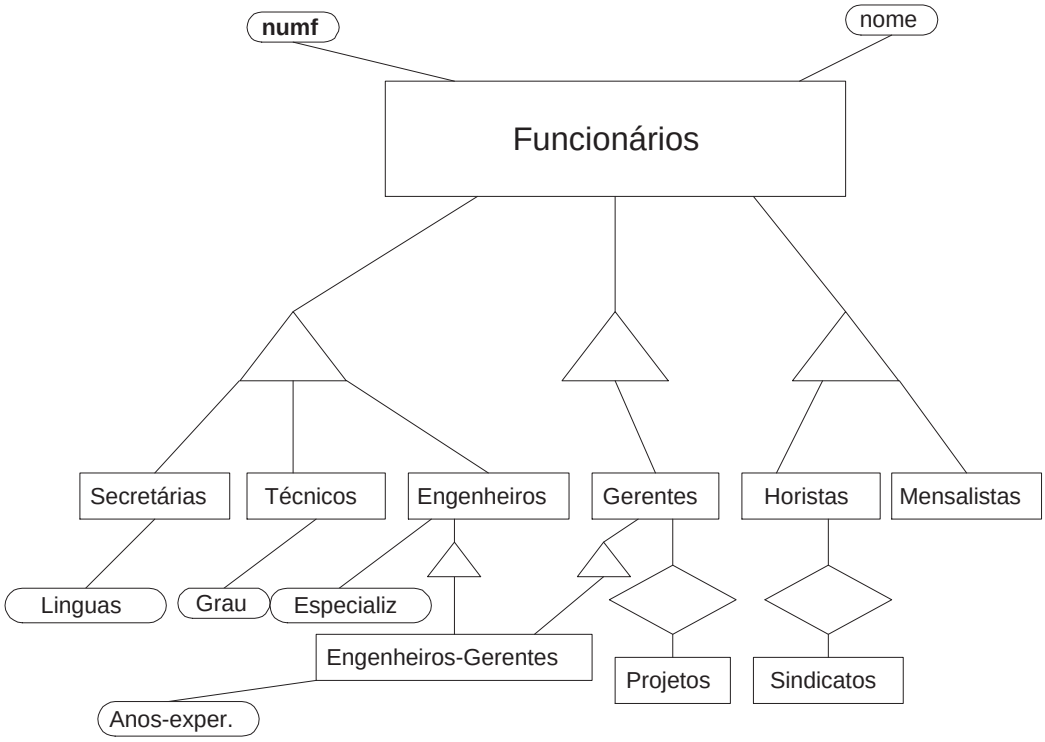


Figura 2.15: Exemplo de Especialização

Esses novos CEs são também chamados de "especializações" do CE Funcionários. Como o nome "é-um" indica, o relacionamento significa que uma secretária "é uma" funcionária, isto é, possui todos os atributos do CE Funcionários. Além disso secretárias possuem atributos específicos que aparecem no diagrama ligados ao CE Secretárias. Este é o mesmo conceito de herança em sistemas Orientados a Objeto.

Este processo de especialização pode ser estendido recursivamente, se preciso; por exemplo, Engenheiros poderiam ser subdivididos em "Engenheiros-Gerentes". É possível também a um funcionário pertencer a mais de uma especialização como por exemplo a Engenheiros e Gerentes. Poderíamos até ter herança múltipla como é o caso de Engenheiros-Gerentes no diagrama.

Observe também que certos relacionamentos podem se aplicar apenas a certas especializações como é o caso do relacionamento de Horistas com Sindicatos e o de Gerentes com Projetos.

Quando fizermos o mapeamento do MER para o Modelo Relacional veremos que é bastante simples representar os relacionamentos "é-um" através

de "tabelas auxiliares".

Existem outras extensões ao MER relacionadas a especialização: em vez de especializarmos um CE poderíamos fazer uma "generalização", isto é, poderíamos agrupar vários CEs em um CE mais abrangente.

Em outros casos pode ser conveniente criar um CE formado por diferentes CEs, por exemplo um CE de "Proprietários" de veículos que poderiam ser "Pessoas", "Bancos" ou "Empresas". Esse relacionamento especial foi chamado de "categorização". "Proprietários" teriam um relacionamento comum com o CE "Veículos". Estas extensões são, ao nosso ver, menos úteis, por serem menos claras ao usuário leigo em Bancos de Dados. Por este motivo, e por outros a serem expostos depois, não cremos que o Projeto Conceitual deva ser excessivamente detalhado com essas extensões.

2.2.6 Um exemplo de projeto conceitual usando o MER: a BD Torneios de Tênis da ATP

Vamos agora detalhar melhor a visão externa da BD "Torneios de Tênis da ATP" vista anteriormente. Após muita interação com o "gerente do portal" que está contratando o projeto, chegamos à conclusão de que inicialmente serão colocadas na BD informações sobre jogadores que venceram Torneios do "Grand Slam" a partir de 1992, e de alguns jogadores brasileiros. Deseja-se também registrar dados cadastrais sobre todos os torneios do "Grand Slam" e do "Master Series", e informações sobre os jogadores que venceram esses torneios a partir de 1992. Chegamos à conclusão de que apenas 2 CEs e um relacionamento são necessários: um CE contendo dados cadastrais dos jogadores e que vamos chamar de *Jogadores*. Ele corresponde ao que se costuma chamar informalmente de "arquivo mestre" em projetos voltados para arquivos. Contém informações típicas de um cadastro de pessoas. O atributo determinante será denominado *numj* . Outro CE corresponde ao registro de dados cadastrais dos torneios do "Grand Slam" e do "Master Series" e que chamaremos de *Lista_torneios*, com atributo determinante *numt*, e atributos nome do torneio, país hospedeiro, categoria do torneio, tipo de quadra e número inicial de participantes. Finalmente, queremos representar num relacionamento entre *Jogadores* e *Lista_torneios*, dados históricos das realizações dos torneios, ou seja, para cada torneio e ano, queremos registrar o jogador que venceu o torneio e o valor do prêmio recebido. Vamos chamar esse relacionamento de *Vencedores_Torneios*.

A figura 2.16 mostra o Projeto Conceitual através de um diagrama Entidade-Relacionamento. Observe que um jogador pode ter ganho um certo torneio mais de uma vez, mas dado um torneio e um ano só existe um vencedor. O CE "Anos_realização" foi colocado no Projeto Conceitual unicamente

para refletir esta restrição, o relacionamento N : N : 1 entre Anos_realização, Lista_torneios e Jogadores. O diagrama foi intencionalmente simplificado, omitindo os atributos dos CE's envolvidos.

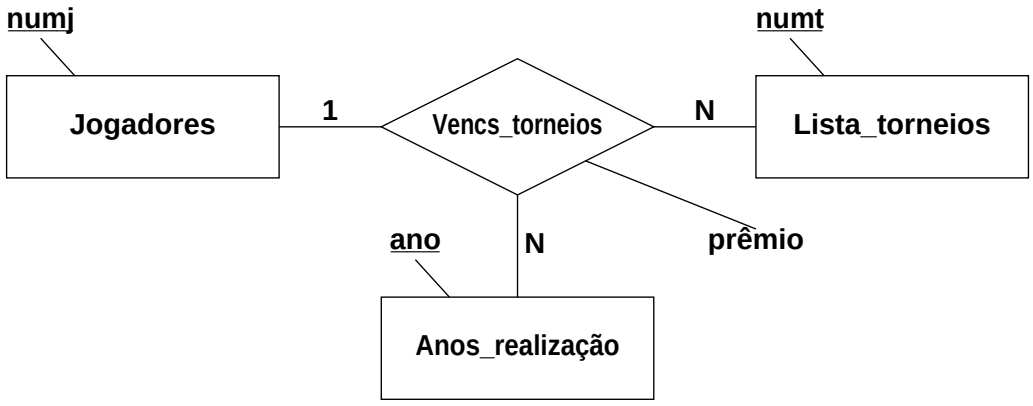


Figura 2.16: Projeto Conceitual da BD Torneios de Tênis da ATP utilizando o Modelo Entidade-Relacionamento

Embora extremamente simples, várias dúvidas colocadas anteriormente na visão externa foram esclarecidas nesse Projeto Conceitual, com exceção da que diz respeito a quais dados manter quando um jogador se aposenta e que relatórios devem ser emitidos pelo sistema. A última corresponde a procedimentos funcionais não cobertos nesse nível do projeto. Posteriormente faremos o mapeamento desse projeto conceitual para o "Projeto Lógico",voltado para o Modelo Relacional de BD, onde a 1^a dúvida será esclarecida e informações adicionais serão colocadas, como por exemplo, o tipo e unidade monetária do prêmio concedido ao vencedor de um torneio, os tipos de dados para cada um dos atributos dos diversos CEs, etc.

2.2.7 Convenções para nomes - considerações práticas

A atribuição de nomes para CEs e relacionamentos deve ser feita de forma a exprimir a semântica (ou seja o sinificado) dos objetos sendo definidos. Nomes apropriados para CEs aparecem naturalmente na modelagem externa de dados como *substantivos* na sua descrição. Relacionamentos em geral aparecem na descrição de requisitos como *verbos* que descrevem o inter-relacionamento entre as entidades. Verbos no entanto, expressam geralmente o "papel" de um CE em relação ao outro no relacionamento, quando seria preferível um substantivo que expressasse os dois papeis, mas nem sempre é fácil de se conseguir.

No exemplo anterior, a alocação de funcionários em departamentos foi expressa através do termo simétrico ou neutro, "Lotações", enquanto os papeis

seriam expressos por: funcionário "está lotado no" departamento e departamento "emprega" ou "lota" funcionário.

Observe que adotamos a convenção de expressar nomes de CEs e relacionamentos sempre no plural (para enfatizar que são conjuntos contendo vários elementos similares), embora alguns livros adotem a convenção contrária (para enfatizar que o relacionamento se dá entre *uma* entidade de um CE com *uma* entidade do outro CE!). De qualquer forma, a uniformidade da notação é a consideração mais importante para o leitor da documentação do projeto.

No caso de relacionamentos triplos ou quádruplos, na prática menos freqüentes do que relacionamentos binários, dificilmente se consegue um nome significativo e uma solução de compromisso é usar as iniciais dos CEs participantes e incluir uma boa descrição textual do seu significado, nem sempre óbvio simplesmente olhando o diagrama. Felizmente, quando passarmos do Projeto Conceitual para o Projeto Lógico tais dificuldades de interpretação vão desaparecer, pois, como veremos, a representação do relacionamento no Modelo Lógico esclarece a maior parte das dúvidas sobre o seu significado.

Um último comentário sobre o MER: a principal utilidade da Modelagem Conceitual utilizando o MER é ser um meio de comunicação do projeto da BD inteligível a usuários leigos, especialmente aqueles que têm a responsabilidade de definir os objetivos e usos da BD, como gerentes, supervisores, etc. Esses usuários devem, portanto, ter uma visão clara do projeto e, principalmente, uma visão **consistente** com a visão do projetista da BD. Caso contrário, o projetista corre o perigo de implantar a aplicação que "não era bem aquela que o usuário queria". Isto, infelizmente, ocorre na prática com muito mais freqüência do que seria desejável, daí a nossa ênfase nesse ponto.

2.2.8 Exercícios

Utilize o Modelo Entidade-Relacionamento (MER) para o projeto conceitual das Bases de Dados dos problemas abaixo, satisfazendo:

- para cada Conjunto de Entidades especifique os atributos relevantes (atributos podem ser simples, compostos ou multivalorados) incluindo um atributo determinante,

- para cada relacionamento inclua a multiplicidade do mesmo, se a participação de cada CE envolvido é total ou parcial, atributos e restrições adicionais se houverem,

- nos auto-relacionamentos especifique adicionalmente o "papel" de cada entidade.

Você pode, opcionalmente, justificar suas decisões de projeto através de documentação textual.

1. Projete uma BD que registra dados de times, jogadores e torcedores incluindo: para cada time, o seu nome, seus jogadores, o capitão do time (um dos jogadores) e as cores dos seus uniformes. Para cada jogador o seu nome, idade, e gols que fez. Para cada torcedor, registre seu nome, seu time favorito, seus jogadores favoritos, e seu uniforme favorito. Deseja-se manter um histórico dos times em que um jogador já jogou (no presente um jogador joga em apenas um time), incluindo datas de início e fim. Inclua um relacionamento "Capitaneado-por" com informações do tipo (jogador1, jogador2, time, ano), onde jogador1 foi capitaneado por jogador2.

   Observe que cada item acima indica o que se deseja registrar e não necessariamente especifica um conjunto de entidades ou relacionamento. Observe também que a Base de Dados pretendida tem muito pouco em comum com a BD exemplo "Torneios de Tênis da ATP" do texto.

2. Deseja-se registrar os seguintes tipos de relacionamentos entre indivíduos de um conjunto de entidades "Pessoas": mãe, pai, filho, cônjuge. Para cada pessoa inclua um atributo educação cujo valor é um conjunto de diplomas obtidos pela pessoa incluindo o nome/tipo do diploma, a escola e a data de obtenção.

3. Projete uma BD para a seguinte descrição de "Alunos, Professores e Disciplinas" com os seguintes requisitos:

   (a) alunos têm um identificador único e um nome;

   (b) professores têm um identificador único, um nome e um departamento;

   (c) disciplinas têm uma sigla única, um nome e uma descrição;

   (d) um aluno pode cursar uma ou mais disciplinas;

   (e) um professor pode ministrar zero ou mais disciplinas, porém, cada disciplina é ministrada por um único professor.

   Obs: o relacionamento triplo PAD, visto no exemplo da figura 2.12 não é um bom esboço de projeto! As razões para isso serão vistas no capítulo 5.

4. Projete uma BD para a seguinte descrição do problema das "Cervejas, Bares, Freqüentadores" [Ullman97]:

(a) "Freqüentadores" têm nomes e endereços unívocos. Eles apreciam uma ou mais cervejas e freqüentam um ou mais bares, mas cada freqüentador tem sua cerveja favorita. Eles possuem telefones, usualmente um, mas às vezes vários ou nenhum.

(b) Bares têm nomes e endereços unívocos. Eles servem uma ou mais cervejas e são freqüentados por um ou mais "freqüentadores". Eles cobram um preço para cada cerveja que servem, e que pode variar de cerveja para cerveja.

(c) Cervejas têm nomes e fabricantes unívocos. São servidas por um ou mais bares e são apreciadas por um ou mais freqüentadores.

(d) Fabricantes têm nomes e endereços unívocos e podem fabricar mais de uma cerveja.

5. Projete uma BD para fins de registro acadêmico de estudantes semelhante à da sua universidade, incluindo informações sobre: estudantes, departamentos, cursos, disciplinas, oferecimentos de disciplinas (turmas) no semestre corrente, salas de aula, estudantes matriculados em oferecimentos de disciplinas, quais disciplinas cada professor está ministrando, notas dos estudantes, etc. Inclua, opcionalmente, o histórico de disciplinas de cada estudante. Note que esta questão não está claramente delimitada e muitas variantes e extensões são possíveis. Exemplos: Curso: 34; disciplina: MC526; oferecimento de disciplina: MC526B (no 2^0 semestre de 200X).

6. Considere uma BD de esportes envolvendo Ligas, Times e Jogadores, onde se deseja utilizar Conjuntos de Entidades Fracos [Ullman97] para representar as seguintes condições:

   (a) nomes de ligas são unívocos,

   (b) nenhuma liga tem dois times com o mesmo nome,

   (c) nenhum time tem dois jogadores com o mesmo número, porém, pode haver jogadores com números iguais em times diferentes,

   (d) pode haver times com o mesmo nome em ligas diferentes.

   Faça o diagrama de ER correspondente e indique como um atributo chave pode ser obtido para cada conjunto fraco de entidades.

7. *Especialização* [Ullman97]: Considere uma BD de navios de guerra onde cada navio possui os atributos nome, tonelagem e tipo (encouraçado, destroier, cruzador, porta-aviões, etc.) Além disso existem as seguintes sub-classes de navios:

(a) gunships: são navios com grandes canhões como encouraçados e cruzadores. Para tais navios queremos registrar a quantidade e o diametro dos canhões,

(b) porta-aviões: queremos registrar o comprimento da pista e os conjuntos de "equipes de apoio ao pouso/decolagem".

(c) submarinos: registrar a profundidade máxima e o tamanho da tripulação,

(d) "porta-aviões de combate": são ao mesmo tempo porta-aviões e gunships e além disso levam mísseis cujo número e alcance deseja-se registrar.

8. Projete uma BD envolvendo trens e estações, consistente com os seguintes requisitos [Ullman97]:

   (a) Trens são ou trens locais ou trens expressos, mas nunca ambos.

   (b) Cada trem tem um número unívoco e um engenheiro.

   (c) Estações são ou "paradas expressas" ou "paradas locais", mas nunca ambas.

   (d) Uma estação tem um nome unívoco e um endereço.

   (e) Todos os trens locais param em todas as estações.

   (f) Trens expressos param unicamente em estações expressas.

   (g) Para cada trem e cada estação, existem dois horários em que o trem para (um para cada sentido do trajeto).

9. Projete uma BD para a seguinte descrição de "Produtos, Revendedores e Cidades", consistente com as seguintes restrições:

   (a) Um revendedor pode atuar em mais de uma cidade e revender mais de um produto.

   (b) Cada produto possui apenas 1 ou nenhum revendedor.

   (c) Cada cidade concede permissão para apenas 1 ou nenhum revendedor.

   (d) Se uma cidade dá permissão de revenda de um produto a um revendedor, deseja-se registrar a data da concessão.

Capítulo 3

Projeto Lógico de Dados

3.1 Modelos Lógicos Históricos

Os primeiros SGBDs comerciais se basearam em dois principais "modelos lógicos de dados" desenvolvidos no início dos anos 70. O primeiro, chamado "modelo de redes" foi definido pelo comitê "Codasyl Data Base Task Group". O segundo, denominado "modelo hierárquico" foi desenvolvido pela IBM para o SGBD IMS voltado para os main-frames IBM; não existe uma descrição formal do mesmo. Esses modelos foram suplantados pelo Modelo Relacional e hoje são de interesse apenas histórico. Nesta seção será feita apenas uma breve introdução aos mesmos sob o ponto de vista do MER.

O modelo de redes pode ser visto como o MER restrito a entidades e a relacionamentos binários 1 : N entre CEs. Existem dois tipos de objetos nesse modelo: os "tipos de registros lógicos" e "links". Os primeiros são conjuntos de registros e são semelhantes a um CE (o termo "registro" pode ser visto como um registro numa linguagem como Cobol ou Pascal ou uma estrutura em C, ou simplesmente como uma entidade). Quando participando de relacionamentos 1 : N eles podem ser do tipo "owner" ou do tipo "membro". Um "link" corresponde a um relacionamento 1 : N e é representado por um arco saindo de um registro tipo "owner" para registros tipo "membro".

Por exemplo, o relacionamento 1 : N, "Lotações" entre "Departamentos" e "Funcionários", visto na figura 2.4, seria representado no modelo de redes pelo diagrama da figura 3.1, onde o símbolo oval representa um "tipo de registro lógico".

Relacionamentos N : N no modelo de redes seriam representados através de um "tipo de registro lógico" auxiliar. Por exemplo, o relacionamento N : N "Alocações" de Funcionários em Projetos, seria representado na figura 3.2 onde "Em-Projetos" funciona como um tipo de registro ponte entre Funcionários e Projetos.

Figura 3.1: Exemplo de diagrama no modelo de redes

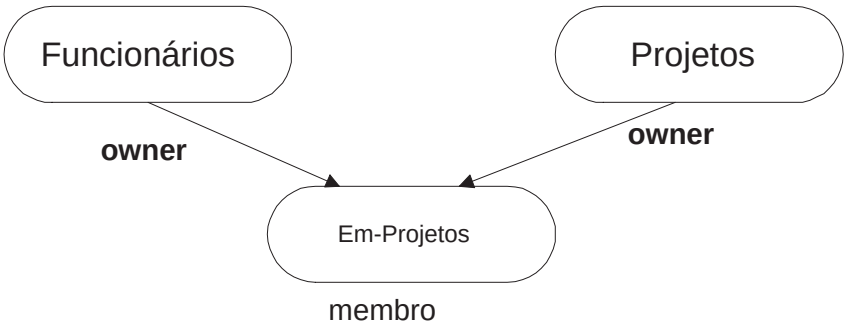


Figura 3.2: Representação de relacionamento N : N no modelo de redes

O modelo hierárquico organiza os dados como uma coleção de hierarquias do tipo árvore (isto é, uma floresta de árvores). Ele pode ser visto como uma restrição do modelo de redes: um registro tipo "owner" é chamado no modelo hierárquico de "pai" e um registro tipo "membro" é chamado de " filho" na estrutura em árvore. O relacionamento entre pai e filhos é 1 : N como antes, mas a hierarquia em árvore impede que um registro filho tenha mais que um pai como no caso do tipo de registro auxiliar "Em-Projetos" da representação do relacionamento "Alocações" visto acima. Nesse caso objetos denominados "cópias virtuais" dos registros Funcionários e Projetos, precisariam ser criados para representar esse relacionamento através de duas hierarquias. Por ser de interesse puramente histórico, vamos deixar o tópico de lado e passar diretamente para o projeto lógico através do Modelo Relacional.

3.2 O Modelo Relacional

O **Modelo Relacional (MR)** introduzido em [Codd70] utiliza um único objeto básico para representar dados: a "tabela" ou "relação". Uma tabela no MR é composta por uma coleção não ordenada de linhas onde cada linha corresponde a uma entidade e a coleção de linhas corresponde a um Conjunto de Entidades do MER (ou a um relacionamento, como veremos

posteriormente). [1] Cada linha é composta por uma lista de valores que correspondem aos valores dos atributos da entidade; o número de valores em cada linha é constante e corresponde ao número de atributos da entidade correspondente. Numa tabela o número de linhas é variável com o tempo e representa o número corrente de entidades.

Cada coluna da tabela corresponde a um atributo do CE. Se dermos um nome para a tabela e um nome para cada coluna teremos uma correspondência direta entre o esquema de uma tabela e o esquema de um CE.

No Modelo Relacional uma tabela é chamada de "relação" e uma linha de "tupla". Sob o ponto de vista matemático uma relação é um conjunto não ordenado de tuplas e cada tupla é um conjunto não ordenado de n pares $(A_1, v_1), (A_2, v_2), \ldots, (A_n, v_n)$ onde os A´s são os nomes dos atributos e os v´s são os valores correspondentes.

Para caracterizarmos completamente uma relação precisamos especificar quais são os valores possíveis para cada atributo, isto é, qual o tipo de dados de cada atributo. A esse tipo de dados damos o nome de "domínio" e ele especifica um conjunto de valores atômicos (isto é, indivisíveis) do mesmo tipo. A cada atributo A_i corresponde, portanto um domínio D_i (não necessariamente todos diferentes).

Outra forma de definir matematicamente uma relação é como sendo *" um subconjunto do produto cartesiano* $D_1 \times D2 \times \ldots \times D_n$". (Se Você não tem formação matemática, não se preocupe em entender esta definição; ela não será usada no que se segue).

Uma Base de Dados Relacional é um conjunto de tabelas ou relações onde cada tabela está associada a uma lista de domínios.

O "esquema" de uma relação é comumente denotado pelo nome da relação seguido pelos nomes dos atributos entre parênteses, exatamente como denotamos textualmente um CE. Por exemplo:

Funcionários(numf, RG, CPF, nome, endereço, salário)

Embora uma tabela possa ser visualizada como uma seqüência ordenada de linhas (vistas de cima para baixo) e cada linha como uma seqüência ordenada de colunas (vistas da esquerda para a direita), esta ordenação é uma mera visualização prática e não corresponde à organização interna ou lógica da tabela. Se rotularmos cada valor em uma linha com o nome do atributo correspondente conforme visto acima, a ordem desses valores se torna

[1] O Modelo Relacional foi criado seis anos antes do MER e, portanto, é independente do mesmo na sua definição e nos seus conceitos básicos. Do ponto de vista didático, porém, fica mais simples introduzir esses conceitos via analogia com o MER, além de que ela simplifica a compreensão do mapeamento do MER para o MR no projeto "top down" de uma BD.

irrelevante. No entanto, por economia de notação, é conveniente ordenar os atributos de acordo com o esquema da tabela, e que ficariam implícitos ao se escrever o conteúdo de uma tabela ou de uma linha da tabela. É comum também, ao se escrever uma instância de uma tabela, rotular as colunas com os nomes dos atributos.

Exemplo: instância da tabela Funcionários (esquema dado acima), contendo duas linhas:

numf	RG	CPF	nome	ender.	sal.
1234	9876543	537865443-67	Ana Paula A.	Rua da Lapa 133	9200
4321	1231234	987654321-99	Pedro Silva	Rua do Cabral 321	920

As seguintes analogias com conceitos tradicionais envolvendo tabelas e arquivos são úteis:

- Relação $\approx$ Tabela $\approx$ Arquivo
- Tupla $\approx$ Linha $\approx$ Registro
- Atributo$\approx$ Coluna $\approx$ Campo
- Domínio $\approx$ Tipo de dados da coluna $\approx$ Tipo de dados do campo

Assim como no MER todo CE possui um atributo determinante, no MR toda relação deve ter pelo menos uma "chave" que identifica de forma única cada linha da relação.

Uma *chave* é um atributo (ou lista de atributos) da relação, satisfazendo: (i) identifica de forma única cada linha da relação, isto é, dadas duas linhas quaisquer da relação o valor da chave nunca é o mesmo para as duas linhas, (ii) caso a chave possua mais de um atributo, ela deve ser "minimal", isto é, se retirarmos qualquer atributo da chave ela deixa de identificar univocamente as linhas da relação.

Uma relação pode ter mais de uma chave. Nesse caso uma delas deve ser escolhida como "Chave Primária". As outras chaves são denominadas *chaves alternativas*. Exemplo: a relação Funcionários, vista acima, tem 3 chaves, numf, RG e CPF. A chave numf foi escolhida como Chave Primária e por isto aparece sublinhada. RG e CPF são chaves alternativas.

É importante distinguir o "esquema" de uma relação de sua "instância". Esta é o conjunto de tuplas presentes na relação num determinado instante e varia com o tempo. O esquema é, normalmente, imutável e reflete o "projeto" ou especificação da relação. Por serem nomes similares, é importante também não confundir "relacionamento" (entre entidades do MER) com "relação" (que é uma tabela do MR).

De forma semelhante, definimos o esquema de uma BD Relacional como sendo o conjunto dos esquemas de todas as relações que fazem parte da BD, e uma instância da BD é o conteúdo instantâneo de todas as tabelas (relações) da BD.

3.2.1 Valores Nulos

O mundo real é imperfeito e desafia as mais precisas definições matemáticas. Uma das características indesejáveis do mundo real e que aflige as implementações de Bases de Dados é a ausência de informações completas sobre os dados.

Quando se preenchem as tabelas de uma BD relacional, é comum faltarem informações sobre atributos de uma ou mais tuplas. Poder-se-ia adotar a solução draconiana de que tuplas com informações incompletas seriam impedidas de inclusão na BD, mas em geral isto é inaceitável. Em muitos casos a informação não estava disponível no momento de criação da tupla. Em alguns casos a informação simplesmente não existe porque se perdeu no tempo ou é desconhecida. Em outros casos, para uma determinada entidade o valor daquele atributo simplesmente não se aplica ou não faz sentido.

Essas situações foram acomodadas criando-se um valor especial e genérico denominado "Vazio" ou "Nulo" (do inglês "Null Value") que é compatível com qualquer tipo de dados. Isto, claramente, é uma solução paliativa, pois vários significados distintos poderiam ser adotados para este valor como os acima descritos (temporariamente vazio, inexistente, desconhecido, não se aplica), e pior ainda, operações com esse valor em geral não fazem sentido (se for um valor numérico como peso, quantidade, por exemplo, ou se for um valor simbólico como uma cadeia de caracteres que queremos comparar com outra). Posteriormente veremos como essas operações são tratadas, mas de modo geral, as soluções são insatisfatórias. O termo "Nulo" é particularmente infeliz, pois dá a conotação (errônea) de "zero" para um atributo numérico, ou de cadeia vazia, no caso de um atributo tipo cadeia de caracteres.

Curiosamente, depois de muita discussão na literatura científica sobre o assunto sem nenhum consenso, os acadêmicos simplesmente desistiram do assunto por considerarem-no pouco interessante.

3.2.2 Restrições de Integridade

O MR permite expressar diretamente algumas restrições de integridade ou "regras de consistência" que dizem respeito a valores de atributos permissíveis em relações.

A primeira regra, dita "Regra de Integridade Existencial", é muito simples e diz que a Chave Primária de uma tupla não pode tomar o valor Nulo. A

razão é óbvia pois viola diretamente a definição de chave.

A segunda regra diz respeito ao inter-relacionamento entre 2 relações, tipicamente entre uma tabela informalmente dita "mestre" e outra dita "detalhe" ou "auxiliar". Um exemplo esclarece esses termos informais:

Temos uma tabela de Funcionários e queremos armazenar dados sobre os dependentes de cada funcionário. Dependentes são um exemplo típico de atributo multivalorado, pois um funcionário pode ter 0, 1 ou n dependentes. Tais atributos não são suportados pelo Modelo Relacional e a solução trivial é criar uma tabela auxiliar, chamada digamos, " Dependentes", contendo, para cada funcionário que possui dependentes, uma linha para cada dependente, identificando o funcionário e o dependente. Para identificar o funcionário é natural usar um atributo do mesmo tipo do atributo da Chave Primária da tabela Funcionários, digamos numf e com o mesmo valor. Mais precisamente, se existe uma linha na tabela Dependentes onde Dependentes.numf = x então x deve ser o número de um funcionário real, isto é, deve existir na tabela Funcionários uma linha com Funcionários.numf = x. Esta regra de consistência é chamada de "Regra de Integridade Referencial" e é genérica, isto é, vale para qualquer Base de Dados Relacional.

3.2.3 Restrições sobre atributos

As seguintes restrições podem ser colocadas sobre um atributo:

- o atributo não pode tomar o valor "Nulo"
- o atributo pode tomar o valor "Nulo"
- o atributo deve satisfazer restrições de domínio, por exemplo ser ≥ 10 e < 100

Integridade referencial pode ser mantida de diversas formas:

- inserções e atualizações numa tabela contendo Chave Estrangeira (como Dependentes) que violam a Integridade Referencial são abortadas;
- remoção de uma tupla em tabela mestre (por exemplo Funcionários) e que seja referenciada por alguma Chave Estrangeira é abortada;
- remoção de uma tupla em tabela mestre (por exemplo Funcionários) e que seja referenciada por alguma Chave Estrangeira é executada, porém todas as tuplas que a referenciam em tabelas auxiliares são removidas ("remoção em cascata"). Por exemplo se um funcionário é removido, então todos os seus dependentes são removidos da tabela Dependentes; outra opção seria mudar os valores das Chaves Estrangeiras afetadas da tabela Dependentes para "Nulo".

Todos os recursos acima estão disponíveis nas versões mais recentes da linguagem SQL.

3.2.4 Mapeamento do MER para o Modelo Relacional

O próximo passo no projeto "top down" de uma BD é fazer o mapeamento do Projeto Conceitual obtido via diagramas ER para relações do MR.

O domínio dos requisitos necessários para um mapeamento adequado do Projeto Conceitual para o MR requer familiarização com os conceitos de dependência funcional e de normalização, que serão detalhados no capítulo 5. Algumas regras de mapeamento a serem vistas e colocadas informalmente, são plenamente justificadas através desses conceitos. Por esta razão, recomendamos ao leitor retornar ao capítulo corrente após o estudo do capítulo 5.

O mapeamento de CEs é direto: a cada CE corresponde uma relação com o mesmo nome e os mesmos atributos do CE; os tipos dos atributos (domínios) devem ser especificados nesta fase. A Chave Determinante do CE passa a ser a Chave Primária da relação. Chaves Estrangeiras devem também ser especificadas nessa fase assim como outras restrições sobre atributos conforme visto na seção acima. Posteriormente veremos que a linguagem SQL permite definir diretamente no esquema da relação todas essas restrições.

O mapeamento de relacionamentos para relações pode ser feito de duas formas conforme detalharemos a seguir. Relacionamentos N : 1 podem ser embutidos nas relações que representam o lado N do relacionamento (Exercício: por que do lado N?) como um novo atributo que será uma Chave Estrangeira na relação que corresponde ao CE do lado N. Por exemplo, o relacionamento "Lotações", visto na figura 2.7, especifica o departamento em que cada funcionário está lotado. Pode ser expresso colocando o número do departamento como um novo atributo de Funcionários, referenciando a Chave Primária de Departamentos. Atributos do relacionamento, por exemplo, "data de lotação" também poderiam ser inseridos como novos atributos de Funcionários. O novo esquema de funcionários ficaria então:

Funcionários(numf, RG, CPF, nome, endereço, salário, num_depto)

Outra possibilidade que seria adequada quando há vários atributos do relacionamento, é criar uma tabela auxiliar de forma análoga à que será vista na representação dos relacionamentos N : N.

Considere agora o relacionamento 1 : 1 entre Correntistas e Contas_Correntes, na figura 3.3. Neste caso poderíamos embuti-lo em qualquer um dos lados, mas é natural colocá-lo dentro de Contas_Correntes, por ser este um CE "fraco" isto é, sua existência é dependente de Correntistas:

Figura 3.3: Relacionamento 1 : 1

Correntistas(num_correntista, nome, end, ...)

Contas_Correntes(num_conta, num_correntista, limite, ...),

onde *num_correntista* é uma Chave Estrangeira com a restrição not null, referenciando a Chave Primária de Correntistas.

Auto-relacionamentos N : 1 também podem ser representados via Chave Estrangeira na própria tabela mestre. Por exemplo, o auto-relacionamento Gerenciamentos entre Funcionários pode ser expresso acrescentando à tabela de Funcionários um atributo contendo o número do gerente a quem o funcionário é subordinado. Este atributo é uma Chave Estrangeira que deve conter um número válido de funcionário. Este é um exemplo de Chave Estrangeira que referência a Chave Primária da própria relação.

Nos exemplos que se seguem, Chaves Estrangeiras serão indicadas no texto e não nos esquemas das relações, unicamente para simplificar a apresentação. Todos os esquemas de relações apresentados poderiam incluir definições das Chaves Estrangeiras apropriadas, usando uma notação similar à da linguagem SQL e que será vista no capítulo 6. Da mesma forma, a participação total de Funcionários no relacionamento "Lotações", pode ser imposta colocando adicionalmente a restrição *not null* no atributo *num_depto*. Por exemplo, a relação Funcionários poderia ter o seu esquema especificado assim:

Funcionários(numf, RG, CPF, nome, endereço, salário, num_depto
num_depto, not null, referencia Departamentos.num_depto)

Relacionamentos N : N devem ser representados via tabela auxiliar. Por exemplo, o relacionamento Fornecimentos, entre Materiais e Fornecedores, visto no capítulo 2 e reproduzido na figura 3.4, seria representado por uma tabela cujo esquema é:

Fornecimentos(num_mat, num_fornecedor, prazo, preço, quantidade)

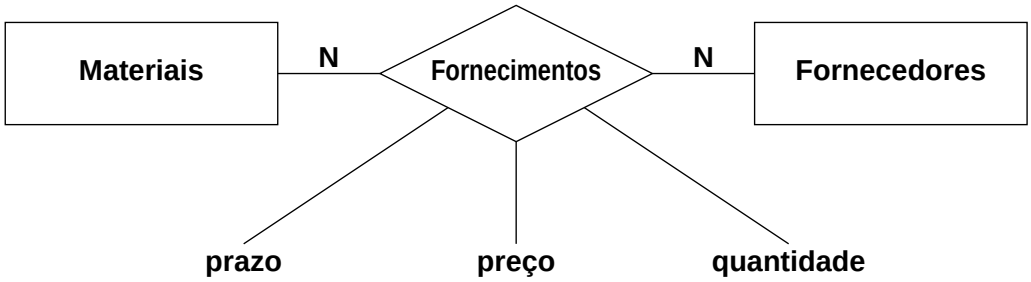


Figura 3.4: Relacionamento com Atributos

Observe que a Chave Primária dessa tabela é o par (num_mat, num_fornecedor) e que num_mat é Chave Estrangeira referenciando a Chave Primária de Materiais, assim como num_fornecedor é Chave Estrangeira referenciando a Chave Primária de Fornecedores.

Outro exemplo: o auto-relacionamento N : N Composições, na figura 3.5,

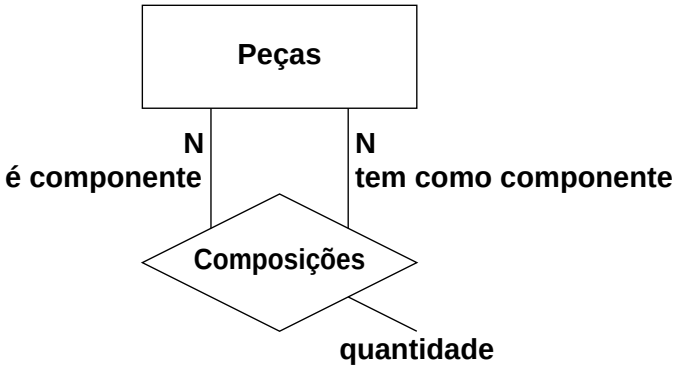


Figura 3.5: Exemplo de Auto-relacionamento

seria representado pela tabela:

Composições(num_peça,num_componente, quantidade)

Mapeamento de atributos multivalorados

Atributos multivalorados, mesmo quando não explicitamente expressos no diagrama do MER como um CE fraco, também são representados por uma tabela auxiliar. Por exemplo, se um funcionário pode ter mais de um local de trabalho, uma tabela pequena poderia ser criada contendo apenas os locais extra de trabalho dos funcionários. Idem para telefones.

Se um CE possui mais de um atributo multivalorado, em princípio deve-se criar uma tabela auxiliar para cada atributo multivalorado. As razões para isto ficarão claras quando estudarmos a 4^a Forma Normal no capítulo sobre normalização.

Relacionamentos triplos também serão representados por tabelas auxiliares. Por exemplo, o relacionamento triplo PAD, visto na figura 2.12, entre Professores, Alunos e Disciplinas, de classe 1 : N : N, seria representado pela relação:

PAD (num_aluno, num_disciplina, num_prof)

Note que a Chave Primária é o par de atributos (num_aluno, num_disciplina) pois, dado um aluno e uma disciplina só há um professor. Este ponto ficará claro quando estudarmos normalização de relações. Lá veremos que este mapeamento tem o inconveniente de gerar uma relação violando a 2^a Forma Normal.

O relacionamento triplo N : N : N, entre Materiais, Requisições e Pedidos, visto na figura 2.13, seria representado pelo esquema:

MRP (num_mat, num_req, num_pedido, quant_requisitada, quant_pedida)

Observe que agora os três primeiros atributos constituem a Chave Primária (como deveria ser, devido à classe N : N : N do relacionamento) e que os atributos do relacionamento passam a ser atributos da relação criada: simples e direto!.

Vamos agora fazer o mapeamento da agregação vista no exemplo da figura 2.14 do capítulo anterior:

Materiais(num_mat,descrição,unidade, . . . , etc)

Requisições (num_req, emitente, data, local)

Pedidos(num_pedido, num_comprador, num_fornecedor, data)

Itens_de_requisições(num_mat, num_req, quan_requisitada)

Itens_de_pedidos(num_pedido, num_mat, num_req, quant_pedida)

As Chaves Primárias das relações acima devem ficar claras ao analisarmos as classe dos relacionamentos, assim como as Chaves Estrangeiras presentes nas relações *Itens_de_requisições* e *Itens_de_pedidos*.

O mapeamento de especializações é também simples. Se lembrarmos o

exemplo de especialização de Funcionários na figura 2.15 do último capítulo, vimos que Secretárias, Técnicos, Engenheiros, Gerentes, etc, são especializações do CE Funcionários. Para cada uma delas criamos uma tabela auxiliar com o nome correspondente, tendo como Chave Primária um atributo, *numf*, que é ao mesmo tempo Chave Estrangeira referenciando a Chave Primária *numf* de Funcionários. Por exemplo:

Funcionários(numf, RG, CPF, nome, endereço, salário)

Engenheiros(numf, especializ, anos-formado, ...
numf referencia Funcionários.numf)

3.2.5 Mapeamento do Projeto Conceitual da BD "Torneios de Tênis" para o MR

Finalmente, vamos fazer o mapeamento do Projeto Conceitual desenvolvido para a BD "Torneios de Tenis" para o MR. A passagem do diagrama ER da figura 2.16 para esquemas de relações é praticamente direta:

Jogadores(numj, nome, pnome, pais, ano_n, ano_p, cid_n, cid_res, tit_s, tit_d, vits, derrs, natp)

Lista_torneios(numt, nomet, pais, cat, quadra, nparts)

Torneios(numt, ano, numj, premio)

Observe que o relacionamento 1 : N : N, "Vencs_torneios" foi representado pela relação Torneios (para ter um nome menor) e que o CE "Anos_realização" não foi mapeado por ser desnecessário (apenas o seu atributo determinante, ano, foi incluído na relação Torneios)

3.2.6 Exercícios

1. Traduza para o Modelo Relacional todos os exercícios dados no capítulo anterior, especificando o esquema de cada uma das relações (nome da relação, atributos), correspondendo aos CEs e relacionamentos do Projeto Conceitual. Para cada relação especifique a Chave Primária e, quando apropriado, defina Chaves Estrangeiras, indicando as Chaves Primárias que elas referenciam (de forma gráfica ou textual).

Capítulo 4

Linguagens de Manipulação de Dados Relacionais

O Modelo Relacional teve um impacto profundo na comunidade de BD não só pela sua simplicidade, mas também pela elegância das linguagens de manipulação de dados conjuntamente propostas pelo seu criador.

Duas linguagens foram propostas por Codd, Álgebra Relacional e Cálculo Relacional. Embora com sintaxes bem diferentes elas têm o mesmo poder de expressão, isto é, são capazes de expressar as mesmas (complexas) consultas sobre a BD e tiveram profunda influência sobre as linguagens práticas posteriormente desenvolvidas para o MR.

As linguagens de manipulação de dados concentram-se no principal aspecto sob o ponto de vista do usuário, que é formular consultas ("queries") relativamente complexas sobre os dados de uma BD Relacional.

4.1 Álgebra Relacional (AR)

Como o nome indica, a Álgebra Relacional (AR) é uma álgebra de expressões envolvendo relações. A partir de uma ou mais relações da BD tomadas como operandos, outras relações podem ser progressivamente construídas através de operadores especiais; conforme veremos, a relação final construída, é, na verdade, o "resultado" da consulta sobre a BD.

Uma consulta é, portanto, uma expressão da Álgebra Relacional.

Alguns operadores básicos foram propostos por Codd. Se lembrarmos que relações são conjuntos matemáticos (de tuplas) esses operadores essencialmente manipulam conjuntos e estão em 4 categorias:

1. operadores convencionais sobre conjuntos: união, subtração, interseção e produto cartesiano.

2. operadores que removem partes de uma relação: um operador que elimina linhas ("seleção") e um operador que elimina colunas, ou seja, atributos ("projeção");

3. operadores que combinam linhas de duas relações de acordo com condições booleanas diversas (essas operações selecionam subconjuntos do produto cartesiano das duas relações).

4. um operador especial ρ permite renomear o esquema de uma relação, essencialmente para permitir expressar de forma não ambígua operações envolvendo auto-relacionamentos. Exemplo:
$\rho_{D_1(f1,n1,p1)}(Dependentes(numf, nomed, parentesco))$
renomeia a relação Dependentes e seus atributos para $D_1(f1, n1, p1)$

Notação: usaremos letras maiúsculas do final do alfabeto para denotar relações, R, S, T, etc e letras maiúsculas do início do alfabeto para denotar atributos A, B, C, etc; a letra grega Θ será usada para denotar um dos seis operadores de comparação $=, \neq, <, >, \leq, \geq$. As operações de união, interseção e diferença têm o significado convencional. Elas só fazem sentido, porém, para relações do mesmo tipo, isto é, cujos atributos sejam os mesmos. Nesse caso se diz que as relações "são compatíveis quanto à união".

$R \bigcup S$: é uma relação que contém as tuplas que estão em R, em S ou em ambas; se uma tupla está presente em R e em S ela só aparece uma vez na união.

$R \bigcap S$: é uma relação que contém as tuplas que estão em ambas R e S.

$R - S$: contém as tuplas que estão em R porém não estão em S. Observe que $R - S$ é diferente de $S - R$.

Vamos apresentar a seguir exemplos para essas 3 operações. Por simplicidade e economia, vamos utilizar relações com atributos numéricos apenas. Considere as seguintes instâncias das relações *R(A,B) e S(A,B)*:

R		S	
A	B	A	B
1	2	3	4
3	4	5	6
		5	8

Então:

$$R \cup S = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 5 & 8 \end{pmatrix}, \quad R \cap S = \begin{pmatrix} 3 & 4 \end{pmatrix}, \quad R - S = \begin{pmatrix} 1 & 2 \end{pmatrix}$$

Projeção: $\Pi_{A_i,A_j,\ldots A_k}(R)$: constrói uma nova relação a partir de R contendo apenas os atributos $A_i, A_j, \ldots A_k$. Observe que tuplas duplicadas devem ser eliminadas ao se fazer a projeção. Exemplo:

$$\Pi_A(S) = \begin{pmatrix} 3 \\ 5 \end{pmatrix}$$

Seleção: seja uma expressão booleana C do tipo $R.A\ \Theta\ c$, onde c é uma constante do tipo de A que queremos comparar com o atributo A de uma linha de R. Então,

$\sigma_C(R)$ é uma relação que inclui unicamente as linhas de R para as quais C é verdadeiro.Exemplo:

$$\sigma_{B \geq 6}(S) = \begin{pmatrix} 5 & 6 \\ 5 & 8 \end{pmatrix}$$

Expressões booleanas mais complexas envolvendo "ands" e "ors" de operandos do tipo $R.A\ \Theta\ c$, podem ser avaliadas através de operações de união e interseção, mas, para simplificar a notação vamos admitir em C expressões booleanas arbitrariamente complexas.

Se quisermos, podemos compor quaisquer operações, por exemplo:

$$\Pi_B(\sigma_{B \geq 6}(S)\ ) = \begin{pmatrix} 6 \\ 8 \end{pmatrix}$$

Produto Cartesiano: $R \times S$: é uma relação obtida "concatenando-se" cada linha de R com cada linha de S, isto é, os atributos do produto cartesiano são os atributos de R seguidos pelos atributos de S (se houver coincidência nos nomes de alguns atributos de R e S, uma forma conveniente de distingui-los deve ser usada, como por exemplo R.C e S.C). Se R possui n atributos e k linhas e S possui m atributos e l linhas então $R \times S$ possui n + m atributos e $k \times l$ linhas. Exemplo:

$$R \times S = \begin{pmatrix} R.A & R.B & S.A & S.B \\ 1 & 2 & 3 & 4 \\ 1 & 2 & 5 & 6 \\ 1 & 2 & 5 & 8 \\ 3 & 4 & 3 & 4 \\ 3 & 4 & 5 & 6 \\ 3 & 4 & 5 & 8 \end{pmatrix}$$

Apesar de simples, o Produto Cartesiano é extremamente importante no

que se segue. Se ficou alguma dúvida, releia o exemplo e faça outros. As próximas operações usam subconjuntos do Produto Cartesiano e caso este não seja plenamente compreendido, parecerão mágica!

Junção Θ : nesta operação comparamos duas colunas do mesmo tipo de R e S, por exemplo $R.A\ \Theta\ S.B$. Se a comparação der verdadeira, selecionamos a linha correspondente do produto cartesiano $R \times S$. A junção Θ é denotada por: $\begin{array}{c} R \mid\times\mid S \\ R.A\ \Theta\ S.B \end{array}$

A junção Θ de duas relações é, portanto, um subconjunto do seu produto cartesiano e pode ser expressa pela identidade:

$$\begin{array}{rcl} R \mid\times\mid S & = & \sigma_{(R.A\ \Theta\ S.B)}(R \times\ S) \\ R.A\ \Theta\ S.B & & \end{array}$$

Exemplo: vamos utilizar outra relação para exemplificar a junção Θ . Seja a relação T com esquema *T(A, B, C)* e uma instância de T abaixo:

T		
B	C	D
2	2	6
4	7	8
3	2	1

Então,

$$\begin{array}{r} R \mid\times\mid T \\ R.B\ \geq T.D \end{array}$$

terá o valor:
$$\begin{pmatrix} A & R.B & T.B & C & D \\ 1 & 2 & 3 & 2 & 1 \\ 3 & 4 & 3 & 2 & 1 \end{pmatrix}$$

Junção natural: quando Θ é o operador de igualdade o resultado do produto cartesiano terá duas colunas idênticas, R.A e S.B. Nesse caso é conveniente projetar fora uma dessas colunas e o resultado final se chama de "junção natural" de R com S . Se as colunas sobre as quais fizermos a junção tiverem o mesmo nome em R e S , então se pode simplificar a notação escrevendo apenas:

$R \mid\times\mid S$

para indicar esta operação, ficando implícitas as colunas sobre as quais é feita

a comparação. Exemplo: a junção das tabelas R e T acima, sobre as colunas R.B e T.B seria expressa por:

$$R \mid\times\mid T = \begin{pmatrix} A & B & C & D \\ 1 & 2 & 2 & 6 \\ 3 & 4 & 7 & 8 \end{pmatrix}$$

A junção natural é uma das operações mais comuns em consultas envolvendo duas ou mais tabelas relacionadas (por exemplo Funcionários e Dependentes).

Dentre os operadores vistos acima apenas estes são essenciais: união, diferença, produto cartesiano, projeção, seleção e renomeação. Observe que a interseção pode ser obtida a partir da conhecida fórmula para conjuntos:

$R \bigcap S = S - (S - R)$

No entanto, os operadores de junção são tão úteis para expressar consultas práticas que eles serão usados com freqüência nos exemplos que se seguem.

O leitor atento deverá estar se perguntando, "que consultas realmente práticas" se podem fazer com operadores tão estranhos?

Os exemplos a seguir devem dirimir essas dúvidas.

4.1.1 Exemplos de consultas usando Álgebra Relacional

Considere as Relações F e D que representam os CEs Funcionários e Dependentes, respectivamente, com o seguinte esquema simplificado: *F(numf, nomef) e D(numf, nomed, par)*, onde: numf: número do funcionário, nomef: nome do funcionário, nomed: nome do dependente e par: parentesco, que pode ser um dentre: 'filho', 'filha', 'esposa/o', etc. Observe que em D, numf é uma Chave Estrangeira que referencia a Chave Primária numf de F. Abaixo é mostrada uma possível instância dessas relações:

F		**D**		
numf	nomef	numf	nomed	par
01	F1	01	Alice	filha
02	F2	02	Alice	esposa
03	F3	02	Clara	filha
04	F4	03	Jose	filho

Vamos obter agora a junção natural $F \mid\times\mid D$ (ela vai usar a coluna *numf* para junção):

numf	nomef	nomed	par
01	F1	Alice	filha
02	F2	Alice	esposa
02	F2	Clara	filha
03	F3	Jose	filho

Observe que a junção natural apresenta para cada funcionário que possui dependentes, os dados do funcionário e dos seus dependentes. Como o funcionário 04 não possui dependentes, ele não aparece no resultado da junção.

Vamos agora obter expressões da álgebra relacional que respondem às seguintes consultas:

1. Quais os nomes e parentescos de todos os dependentes?

   $\Pi_{nomed,par}(D)$

2. Quais funcionários(números de) possuem dependentes filhas?

   $\Pi_{numf}(\sigma_{par='filha'}(D))$

3. Quais funcionários não possuem dependentes?

   $\Pi_{numf}(F) - \Pi_{numf}(D)$

4. Dê os nomes dos funcionários que possuem algum dependente.

   $\Pi_{nomef}(F \mid\times\mid D)$

5. Dê o nome de cada funcionário que possui uma dependente chamada Alice.

   $\Pi_{nomef}(F \mid\times\mid (\sigma_{nomed='Alice'}(D))$

6. Quais funcionários possuem mais de um dependente?

   $\rho_{D_1(numf,nomed1,par1)}(D), \quad \rho_{D_2(numf,nomed2,par2)}(D)$

   $\Pi_{numf}(\sigma_{nomed1 \neq nomed2}(D_1 \mid\times\mid D2)$

   Justificação: a junção $D_1 \mid\times\mid D2$ na coluna *numf* vai emparelhar, para cada funcionário, cada dependente consigo mesmo e com os outros dependentes do mesmo funcionário. O resultado dessa junção está na tabela abaixo:

numf	nomed1	par1	nomed2	par2
01	Alice	filha	Alice	filha
02	Alice	esposa	Alice	esposa
02	Alice	esposa	Clara	filha
02	Clara	filha	Alice	esposa
02	Clara	filha	Clara	filha
03	Jose	filho	Jose	filho

Observe que obtivemos 4 linhas para o funcionário 02 e não 2 linhas, pois estamos fazendo o produto cartesiano de Dependentes consigo mesmo. Ao fazer a seleção sobre a junção, usando como condição nomes distintos de dependentes, eliminamos os emparelhamentos com um só dependente, ficando aqueles com dois ou mais dependentes. O resultado dessa seleção é a tabela abaixo:

numf	nomed1	par1	nomed2	par2
02	Alice	esposa	Clara	filha
02	Clara	filha	Alice	esposa

pois, apenas o funcionário 02 possui mais de um dependente. A projeção apresenta então os números dos funcionários desejados, no caso o funcionário 02.

7. Quais funcionários possuem exatamente um dependente?

   $\Pi_{numf}(D) - \Pi_{numf}(\sigma_{nomed1 \neq nomed2}(D_1 \mid \times \mid D2)$

   Justificação: usamos diretamente o resultado anterior: subtraímos do conjunto de funcionários que possuem dependentes, aqueles que possuem **mais de um** dependente: o que sobra são os que possuem "exatamente um dependente". A técnica usada para responder essas duas consultas será usada com freqüência em outros problemas.

8. Quais funcionários não têm Alice como dependente (i.é., nenhuma dependente chamada Alice)

   $\Pi_{numf}(D) - \Pi_{numf}(\sigma_{nomed='Alice'}(D))$

   Observe que seria incorreta a solução (justifique porque):

   $\Pi_{numf}(\sigma_{nomed \neq 'Alice'}(D))$

9. Para cada funcionário que tem uma dependente chamada Alice, dê o número do funcionário e o nome dos outros dependentes, se houverem.

   Utilize D_1 e D_2 conforme definidos em 6. acima:

$\Pi_{numf,nomed2}(\sigma_{\ nomed2 \neq\ 'Alice'}(\sigma_{\ nomed1='Alice'}(D_1) \mid \times \mid D_2))$

10. Dê os nomes dos funcionários que possuem exatamente um dependente.

    Utilizamos diretamente a solução do problema 7. acima para obter os nomes através da operação de junção:

    $\Pi_{\ nomef}(F \mid \times \mid(\Pi_{\ numf}(D) - \Pi_{\ numf}(\sigma_{\ nomed1 \neq\ nomed2}(D_1 \mid \times \mid D_2))))$

4.1.2 Um exemplo mais elaborado: composições de materiais

É dada a relação *Materiais(num-material, nome, tipo)* e a relação
Contem(num-material, num-mat-contido)
que representa um auto-relacionamento (parcial) N : M entre Materiais, onde uma tupla (k, j) de uma instância de *Contem* significa que o material de número k contém o material de número j.

Queremos obter expressões da álgebra relacional para as seguintes consultas:

1. dê o número e nome dos materiais do tipo "parafuso".

2. dê o número dos materiais "simples", i.é., que não possuem componentes.

3. dê o número, nome e tipo dos materiais que possuem um ou mais componentes.

4. dê os números dos materiais que não são componentes de nenhum outro material.

5. quais materiais simples são "obsoletos", i.é., não são componentes de nenhum outro material?

6. dê os números dos materiais que possuem mais de um componente.

7. dê os números dos materiais que possuem exatamente um componente.

8. dê os números dos materiais que possuem pelo menos um componente e também são componentes de outro(s) material(s).

Soluções:

1. $\Pi_{\ num-material,nome}(\sigma_{\ tipo='parafuso'}(Materiais))$

2. Subtraia do conjunto de todos as materiais aqueles que possuem componentes:

   $\Pi_{num-material}(Materiais) - \Pi_{num-material}(Contem)$

3. $Materiais \ |X| \ \Pi_{num-material}(Contem)$

4. Subtraia do conjunto de todos as materiais, os materiais que são componentes de outras materiais:

   $\Pi_{num-material}(Materiais) - \Pi_{num-mat-contido}(Contem)$

5. Subtraia dos materiais simples (ex. 2) os materiais que são componentes:

   $(\Pi_{num-material}(Materiais) - \Pi_{num-material}(Contem)) - \Pi_{num-mat-contido}(Contem)$

6. $\rho_{C1(m1,c1)}(Contem), \quad \rho_{C2(m1,c2)}(Contem)$

   $\Pi_{m1}\sigma_{c1 \neq c2}(C1 \mid \times \mid C2)$

   Justificação: a junção sobre $C1.m1 = C2.m1$ emparelha, para cada material m1, seus componentes consigo mesmos e somente aqueles materiais com mais de um componente serão recuperados pela operação de seleção que se segue.

7. Subtraia dos materiais que possuem um ou mais componentes aquelas que possuem dois ou mais componentes:

   $\Pi_{num-material}(Contem) - resultado ex 6$

8. $\Pi_{num-material}(Contem) \bigcap \Pi_{num-mat-contido}(Contem)$

   ou, alternativamente,

   $\rho_{C1(m1,c1)}(Contem), \quad \rho_{C2(m2,c2)}(Contem)$, então:

   $$\Pi_{m1}(C1 \mid \times \mid C2)$$
   $$m1 = c2$$

Se Você quiser verificar manualmente os resultados das expressões acima, as instâncias a seguir das tabelas Materiais e Contem podem ser úteis:

Materiais			**Contem**	
num-material	nome	tipo	num-material	num-mat-contido
1	M1	mont	1	2
2	P2	submont	2	3
3	M3	parafuso	2	4
4	M4	parafuso		
5	M5	porca		

4.1.3 * Operador divisão da álgebra relacional

Vamos introduzir este importante conceito através de um exemplo: considere as três relações a seguir: Fornecedores, *F(f)*, Materiais, *M(m)*, e Fornecimentos de materiais, *FM(f, m)* (esta última representando o relacionamento N x M entre Fornecedores e Materiais).

Desejamos obter o "conjunto dos fornecedores que fornecem todos os materiais".

Esta consulta aparece em muitos contextos e vamos denominá-la de: *a consulta clássica do Modelo Relacional*. Ela não é trivialmente expressa seja em Álgebra Relacional, seja no Cálculo Relacional de Tuplas, ou na própria linguagem SQL, conforme veremos posteriormente.

Para obter uma expressão da Álgebra Relacional que expresse o resultado desta consulta, vamos definir as seguintes expressões intermediárias:

Suponha que: $F = \Pi_f(FM)$
$F \times M$ = conjunto de todos os possíveis pares (f, m)
$F \times M - FM$ = conjunto de pares (f, m) que não pertencem a FM,
isto é, contêm os fornecedores que *não fornecem algum material*, ou seja:
$\Pi_f(F \times M - FM)$ são os fornecedores que não fornecem algum material.
Logo,
$F - \Pi_f(F \times M - FM)$
é a solução desejada.

A expressão acima gera um conjunto de tuplas f que, concatenadas com cada tupla m de M, produzem tuplas (f, m) pertencentes a FM. Este conjunto foi denominado por Codd de *divisão da relação FM por M*, e denotado por *FM/M*, ou seja:

$$FM/M = \Pi_f(FM) - \Pi_f(\Pi_f(FM) \times M - FM)$$

Vamos agora dar uma definição geral para o operador de divisão. No que se segue suponha que X, Y, sejam denominações de conjuntos de atributos de uma dada relação (ao contrário de A, B, C, etc que usamos para denominar um atributo apenas).

Seja R uma relação com esquema *R(X,Y)* e S uma relação com esquema *S(Y)*, isto é, os atributos de S são um subconjunto próprio dos atributos de R.
Def: O quociente de R e S, denotado por R/S, é uma relação com atributos X cujas tuplas t satisfazem: para todas as tuplas s de S a tupla ts formada pela concatenação de t com s, pertence a R.

O problema: obter uma expressão da Álgebra Relacional que seja equivalente a R/S, utilizando os operadores vistos anteriormente.

A expressão:

$\Pi_X(R) \times S$
nos dá todas as tuplas (x, y) possíveis, e a expressão:
$\Pi_X(R) \times S - R$
nos dá todas as tuplas (x, y) que não pertencem a R, e, portanto
$\Pi_X(\Pi_X(R) \times S - R)$
nos dá um conjunto de tuplas t tal que, t concatenada com cada tupla s de S, nem todas as tuplas ts obtidas pertencem a R. Logo,
$\Pi_X(R) - \Pi_X(\Pi_X(R) \times S - R)$
é a solução desejada.

Apesar de podermos expressar na Álgebra Relacional consultas bastante complexas como a consulta acima, certas consultas não podem ser expressas: consultas envolvendo contagem ou ordenação de tuplas não podem ser formuladas na Álgebra Relacional como, por exemplo, a consulta: "qual funcionário possui o maior número de dependentes?". Veremos posteriormente que extensões colocadas na linguagem SQL permitem expressar facilmente tais consultas. Extensões análogas também foram feitas à Álgebra Relacional, mas não julgamos importante incluí-las neste texto.

4.2 Cálculo Relacional de Tuplas (CRT)

A segunda linguagem proposta por Codd, é bem mais próxima da linguagem SQL do que a Álgebra Relacional e pode ser vista como uma forma compacta de expressar consultas em SQL. Ela usa o importante conceito de variável tupla, que é usado de forma implícita em SQL, e os quantificadores existencial $\exists$ e universal $\forall$ para os quais estamos apresentando uma notação mais simples do que a comumente usada nos livros texto sobre o assunto.

Notação: Relações serão denominadas por letras maiúsculas do final do alfabeto, R, S, T, etc.
Variáveis tuplas serão denominadas por letras minúsculas do final do alfabeto, r, s, t, etc. Uma variável tupla t pode tomar o valor de qualquer uma das tuplas de uma relação R (ou em alguns casos, de mais de uma relação, quando elas forem "compatíveis quanto à união". Diz-se, nesse caso, que t varia sobre R ou que o rango de t é R.
Atributos serão denotados por letras minúsculas do início do alfabeto, a, b, c, etc e dado um atributo a o seu valor para uma tupla t será denotado por t.a.

Uma "expressão do cálculo relacional de tuplas" tem a forma geral:
$\{t \mid F(t)\}$
onde se deve ler: "o conjunto de tuplas t tal que F(t) é verdadeiro".

O resultado de uma expressão do CRT é, portanto, uma relação contendo as tuplas t satisfazendo a condição ou "fórmula" F(t) que é uma expressão booleana construída de forma a ser detalhada a seguir. Uma expressão do CRT é também chamada de consulta do CRT.

Na prática é comum explicitar a relação R de onde provêm as tuplas t e uma projeção como resultado desejado da consulta, ou seja, o formato abaixo para uma expressão do CRT é comumente usado:
$\{t.a, t.b, \ldots, t.k \mid t \in R \; and \; F(t)\}$
F(t) é uma expressão booleana composta de átomos e operadores e definida recursivamente da seguinte forma:

(i) $s \in S$ é um átomo (cujo valor é verdadeiro se s pertence a S e falso caso contrário)

(ii) $s.a \; \theta \; t.b$ é um átomo (θ é o operador de comparação visto na álgebra relacional)

(iii) $s.a \; \theta \; c$ é um átomo (c é uma constante do mesmo domínio que a)

(iv) um átomo é uma fórmula.

(v) se F e G são fórmulas então $F \; and \; G$, $F \; or \; G$ e $not \; F$ são fórmulas (com o significado usual).

(vi) $\exists s \in S(F(s))$ é uma fórmula e deve ser lida como: "existe pelo menos uma tupla s pertencente a S tal que a fórmula F(s) é verdadeira". Exemplo:

$\exists s \in S(s.nome \; = \; 'Paulo')$

é verdadeiro se existe pelo menos uma tupla de S para a qual o atributo nome é igual a 'Paulo'.

(vii) $\forall s \in S(F(s))$ é uma fórmula e deve ser lida como "para toda tupla s pertencente a S a fórmula F(s) é verdadeira". Exemplo:

$\forall s \in S(s.salario > 1000)$

é verdadeiro se para todas as tuplas de S o atributo salário tem valor maior que 1000.

(viii) se F é uma fórmula então (F) também é uma fórmula.

(ix) ordem de precedência para avaliação de uma fórmula, de maior para menor:
θ , $\exists$ *e* $\forall$ *not*, *and*, *or*

A seguinte equivalência de expressões é bastante útil para passar de consultas usando o operador $\forall$ para usar $\exists$ e vice versa:
$\forall s \in S(F(s)) \equiv \ not\ \exists\ s \in S(notF(s))$
que quer dizer, em palavras: "para todo s pertencente a S tal que F(s) é verdadeiro", então "não existe s pertencente a S tal que F(s) seja falso", e vice-versa.

A notação apresentada permite traduzir de forma direta as operações básicas da álgebra relacional, conforme se pode ver abaixo:

1. União: $R \cup S$
   $\{t|\ t\ \in\ R\ or\ t\ \in\ S\}$
2. Diferença: $R - S$
   $\{t|\ t\ \in\ R\ and\ not\ t\ \in\ S\}$
3. Interseção: $R \cap S$
   $\{t \mid t \in\ R\ and\ t \in\ S\}$
4. Produto Cartesiano: $R\ \times\ S$
   $\{t, s \mid t\ \in\ R\ and\ s\ \in\ S\}$
5. Projeção de R nas colunas c e d: $\Pi_{c,d}(R)$
   $\{t.c, t.d \mid t \in\ R\}$
6. Seleção: seja F uma fórmula válida sobre colunas de R e a seleção $\sigma_F(R)$
   $\{t \mid t \in\ R\ and\ F\}$
   Exemplo: $\{t \mid t \in R\ and\ (t.nome\ =\ 'Paulo'\ or\ t.nome\ =\ 'Jose')\}$
7. Junção θ de R com S sobre as colunas a de R e b de S: $\begin{matrix} R \mid \times \mid S \\ a\ \theta\ b \end{matrix}$
   $\{t, s \mid t \in\ R\ and\ s \in\ S\ and\ t.a\ \theta\ s.b\}$

Observações sobre a notação:

- Fórmulas do CRT podem ser arbitrariamente aninhadas com operadores $\exists$ *e* $\forall$ e e uma definição do "escopo" das varáveis tuplas nelas usadas se faz necessária. Uma analogia com linguagens de programação com

estrutura de blocos é particularmente útil para este fim: $\forall s \in S(\ldots)$ e $\exists\ s \in S(\ldots)$ podem ser vistas como declarações de uma variável local s cujo escopo é o da fórmula contida no par de parênteses que se segue $(\ldots)$. Nesse caso se diz que s está "ligada" (em inglês, "bound") ao operador $\exists$ ou ao operador $\forall$. Já a variável t em $\{t \mid F(t)\}$ é dita "livre" e pode ser vista como uma variável global a F(t). As únicas variáveis livres ou globais de uma expressão do CRT são as que aparecem "declaradas" à esquerda do símbolo |.

- a notação $\forall s \in S(F(s))$ é uma simplificação não cosmética de:

  $(\forall s)(s \notin S\ or\ s \in S\ and\ (F(s))$
  garantindo que a expressão que usa $\forall$ seja "segura".[1] Nos parece que esta notação simplifica consideravelmente as consultas envolvendo o quantificador universal $\forall$ como veremos nos exemplos a seguir.

- a notação $\exists\ s\ \in\ S(F(s))$ é uma simplificação cosmética da notação $(\exists\ s)(s \in S\ and\ F(s))$

4.2.1 Exercícios resolvidos sobre CRT

Vamos apresentar a seguir expressões do CRT para resolver as consultas sobre Funcionários (*F(numf, nomef)*) e seus Dependentes (*D(numf, nomed, par)*), vistas na introdução à Álgebra Relacional.

1. Quais os nomes e parentescos de todos os dependentes?

   $\{t.nomed, t.par |\ t\ \in\ D\}$

2. Quais os números dos funcionários que possuem filhas?

   $\{t.numf\ \mid t\ \in\ D\ and\ t.par\ =\ 'filha'\}$

3. Dê os nomes dos funcionários que possuem algum dependente.

   $\{t.nomef \mid t\ \in\ F\ and\ \exists s \in D(t.numf\ =\ s.numf)\}$

   Este é um exemplo típico de junção natural expressa no CRT. A solução fica fácil de entender se imaginarmos t como uma variável global que "varre" todas as tuplas de F e para cada valor de t, num laço interno, a variável s "varre" as tuplas de D. Esta interpretação "algoritmica" é condizente com o escopo das variáveis tupla t e s, vistos na apresentação teórica sobre o CRT.

[1] Uma expressão do CRT se diz "segura" quando ela não gera relações com infinitas tuplas, o que não faria sentido [Ullman88].

4. Quais funcionários não possuem dependentes?

   $\{t.numf \mid t \in F \text{ and not } \exists s \in D(t.numf = s.numf)\}$

5. Dê o nome de cada funcionário que possui uma dependente chamada ´Alice´.

   $\{t.nomef \mid t \in F \text{ and } \exists s \in D(t.numf = s.numf \text{ and } s.nomed = 'Alice')\}$

6. Quais funcionários possuem mais de um dependente?

   $\{t.numf \mid t \in D \text{ and } \exists s \in D(t.numf = s.numf \text{ and } t.nomed \neq s.nomed)\}$

7. Quais funcionários possuem exatamente um dependente?

   $\{t.numf \mid t \in D \text{ and not } \exists s \in D(t.numf = s.numf \text{ and } t.nomed \neq s.nomed)\}$

8. Quais funcionários não têm Alice como dependente, isto é, nenhuma dependente chamada Alice?

   $\{t.numf \mid t \in D \text{ and not } \exists s \in D(t.numf = s.numf \text{ and } s.nomed = 'Alice')\}$

9. Para cada funcionário que tem uma dependente chamada Alice, dê o número do funcionário e o nome dos outros dependentes, se houverem.

   $\{t.numf, t.nomed \mid t \in D \text{ and } t.nomed \neq 'Alice' \text{ and } \exists s \in D(t.numf = s.numf \text{ and } s.nomed = 'Alice')\}$

10. Dê os nomes dos funcionários que possuem exatamente um dependente.

    $\{t.nomef \mid t \in F \text{ and } \exists u \in D(t.numf = u.numf \text{ and not } \exists s \in D(u.numf = s.numf \text{ and } u.nomed \neq s.nomed)\}$

4.3 Exercícios

1. Suponha que as relações R e S possuem n e m tuplas, respectivamente. Dependendo das instâncias de R e S dê o mínimo e o máximo de tuplas que cada uma das expressões abaixo pode ter. Justifique!

   (a) $R \bigcup S$

   (b) $R \mid \times \mid S$

   (c) $\sigma_C(R) \times S$ para alguma condição C

   (d) $\Pi_l(R) - S$, para alguma lista de atributos l

2. *Qual a dificuldade de escrever uma expressão da Álgebra Relacional para a consulta:"para cada funcionário dê o seu nome, e se houverem, os nomes de cada um dos seus dependentes"? Sugestão: utilize uma relação auxiliar *AUX(na)* que contem apenas uma tupla com uma constante como atributo, 'NA'.

   Este problema expõe uma limitação do operador de junção da Álgebra Relacional, que foi posteriormente resolvida na linguagem SQL através de uma operação especial de junção denominada "junção exterior".

3. Considere as relações *AD(a, d)* e *PD(d, p)*, representando os seguintes relacionamentos entre "Alunos", "Disciplinas" e "Professores" (*a, d* e *p* identificam univocamente alunos, disciplinas e professores, respectivamente):

   - Um aluno pode cursar várias disciplinas.
   - Uma disciplina é ministrada por um único professor.

   Escreva expressões da Álgebra Relacional para as seguintes consultas:

   (a) Dê uma lista das disciplinas ministradas pelo professor p= 'X'.

   (b) Dê uma lista dos alunos que cursam alguma disciplina ministrada pelo professor p= 'X'.

   (c) Dê uma lista dos alunos que não cursam (*nenhuma das*) disciplinas ministradas pelo professor p= 'X'.

   (d) *Dê uma lista dos alunos que não cursam *alguma das* disciplinas ministradas pelo professor p= 'X'.

   (e) *Dê uma lista dos alunos que cursam *todas as* disciplinas ministradas pelo professor p= 'X'.

   (f) *Dê uma lista dos alunos que cursam *mais de uma* disciplina ministrada pelo professor p= 'X'.

4. Seja D uma relação com esquema *D(p, f)* representando um relacionamento N : N entre pessoas da seguinte forma: uma tupla (k, l) de uma instância de D significa que a pessoa k possui como filho ou filha a pessoa l (diremos que k é "pai" de l e que l é "filho" de k, independentemente do sexo de k ou do sexo de l). Suporemos também que não existem valores nulos em D e que se uma pessoa l aparece na coluna f de D então necessariamente existe em D duas linhas (k, l) e (m, l) representando os pais de l (sem especificar o sexo), porém é possível uma pessoa k aparecer na coluna p mas não aparecer na coluna f (ou seja para cada pessoa presente na tabela D, ou existe informação sobre

ambos os pais ou nenhuma informação sobre os mesmos).
Obs.: no esquema dado, veja p e f como mnemônicos para *pai* e *filho*, respectivamente.

Obtenha expressões da Álgebra Relacional para cada consulta abaixo. Explique o significado de cada sub-expressão da sua solução.

(a) Dê uma lista de pessoas que são "avós".

(b) Dê uma lista de pessoas que são "netos".

(c) Dê uma lista de pessoas cujos pais estão presentes em D.

(d) Dê uma lista de pessoas cujos pais não estão presentes em D.

(e) Dê uma lista de todas as pessoas.

(f) Dê uma lista de pessoas que não têm filhos.

(g) *Dê uma lista de pares de pessoas, sem repetição, e que são (ou foram) cônjuges, i. é., são pais de um ou mais filhos em comum.

(h) *Dê uma lista de pessoas que são bisavós.

(i) *Dê uma lista de casais (cônjuges), sem repetição, que também são avós.

(j) *Dê uma lista de pares de pessoas que são irmãos ou meio-irmãos.

(k) **Dê uma lista de pares de pessoas que são primos em primeiro grau.

(l) **Dê uma lista de pares de pessoas que são irmãos de pai e mãe (ou seja, não são apenas meio-irmãos).

(m) **Dê uma lista de pares de pessoas (s, t) tal que s é sobrinho de t.

5. Obtenha expressões do CRT para as consultas do problema de composições de materiais visto na seção sobre Álgebra Relacional, envolvendo as relações *Materiais(num-material, nome, tipo)* e
Contem(num-material, num-mat-contido)

6. ** Seja uma relação FM com esquema *FM(f, m)* que representa um relacionamento N : N entre Fornecedores e Materiais, onde f identifica o fornecedor e m identifica o material; uma tupla (k, j) de uma instância de FM significa que o fornecedor k fornece o material j. Denomine:

   $F = \Pi_f(FM)$ = conjunto dos Fornecedores e,
   $M = \Pi_m(FM)$ = conjunto dos Materiais.

   Uma prática muito comum em empresas organizadas é ter uma "lista de fornecedores alternativos", definida das seguintes formas:

(a) Para cada fornecedor f, dê os fornecedores f' que fornecem, pelo menos, todos os materiais que f fornece, i.é., dê um conjunto de pares (f, f') com o significado acima.

(b) Para cada fornecedor f, dê os fornecedores f' que fornecem exatamente os materiais que f fornece, (isto é, f' não fornece nem mais nem menos materiais que f).

Observe que o problema (b) é um caso particular do problema (a), isto é, todo par (f, f') que satisfaz (b) também satisfaz (a) mas não vice-versa. Observe também que um fornecedor não é alternativo de si próprio.

O seu problema:

Escreva expressões da Álgebra Relacional que respondam à consulta (a) e à consulta (b) acima, apresentando uma prova informal e detalhada da sua solução.

7. *Considere a relação *FM(f, m)* do problema acima e a nova relação *DM(d, m)*, que representa um relacionamento N : N entre depósitos e materiais (i.é., quais materiais m estão armazenados em cada depósito d). Obtenha expressões do CRT para as consultas abaixo:

   (a) Quais fornecedores fornecem exatamente um material.

   (b) Quais fornecedores fornecem todos os materiais.

   (c) Quais fornecedores não fornecem nenhum material armazenado no depósito d=10.

   (d) Quais fornecedores fornecem todos os materiais armazenados no depósito d=10.

   (e) Quais fornecedores fornecem exatamente os materiais armazenados no depósito d=10 (isto é, nem mais, nem menos materiais).

8. *Obtenha expressões da Álgebra Relacional para as consultas (c), (d) e (e) do problema anterior.

Capítulo 6

A linguagem SQL

O Modelo Relacional desenvolvido por [Codd70] definiu as meta-linguagens Álgebra Relacional e Cálculo Relacional, que implementam os conceitos básicos do modelo. Elas tiveram grande influência no desenvolvimento subseqüente de protótipos do Modelo Relacional.

SQL ("Structured Query Language) é uma linguagem de Definição e de Manipulação de dados relacionais, desenvolvida nos laboratórios da IBM nos anos 70 e hoje padronizada pelos comitês ISO/ANSI.

6.1 Histórico

SQL teve sua origem na linguagem SEQUEL no IBM Research Center em 1973. Posteriormente uma extensão da mesma, SEQUEL/2, foi adotada num projeto ambicioso da IBM para implementação de um SGBD Relacional denominado Sistema R, nos laboratórios da IBM San José. Juntamente com o projeto INGRES da Universidade de Berkeley, Califórnia, o sistema R mostrou a viabilidade de implementação de SGBDs Relacionais práticos. O desenvolvimento de sistemas comerciais procedeu a partir de 1978 em grande velocidade na IBM, Oracle, Relational Technology (que comercializou o sistema INGRES) e outras empresas. Um dos primeiros sistemas maduros foi o sistema SQL/DS da IBM em 1983 para os main-frames IBM. No entanto, quem terminou dominando o mercado de SGBDs relacionais foi uma empresa nascente na época, a Oracle, que lançou seu SGBD relacional numa grande variedade de plataformas de hardware (IBM, Digital VAX e vários sistemas Unix). Atualmente existem também vários SGBDs relacionais confiáveis e de código aberto, como Postgres (derivado do sistema Ingres), Firebird (derivado do sistema Interbase), Mysql e SAP (derivado do sistema Adabas).

Em 1982 o comitê americano ANSI começou a padronização da linguagem SQL, baseada na implementação da IBM, surgindo em 1986 o padrão

SQL/86. No ano seguinte ele foi adotado pelo comitê europeu ISO. O trabalho de padronização continua até hoje: em 1989 surgiu o padrão SQL/89, incorporando as restrições de Integridade Existencial e Referencial. O comitê ISO/ANSI propôs uma extensão considerável à linguagem em 1992, ficando conhecida como SQL/92 (e também por SQL2). Este é o padrão seguido pela maioria dos SGBDS (incluindo correções e extensões posteriores a 92)[Date97]. SQL foi originalmente projetada para uso interativo ou direto: o usuário digita interativamente uma seqüência de comandos SQL e o resultado é apresentado na tela da sua estação. Recursos para invocar SQL a partir de linguagens de 3^a geração como C, Pascal, Cobol e PL/I foram acrescentados em SQL/92 (curiosamente, a maior parte desse esforço foi perdido, pois dessas linguagens, apenas C ainda é largamente utilizada).

No final de 1999 o comitê ISO/ANSI padronizou uma versão ainda mais ambiciosa da linguagem SQL, ocupando várias centenas de páginas, com o nome de SQL:1999 (previamente SQL3). Dentre as facilidades mais marcantes do SQL:1999 podemos citar a introdução de recursividade na linguagem SQL, de "triggers", e de orientação a objetos sob a forma de "tipos abstratos de dados" e "métodos". Uma nova versão do padrão SQL está prevista para 2004.

Qual a situação dos SGBDs relacionais comerciais? É bastante variável de um fornecedor para outro: vários recursos do SQL/92 não estão disponíveis e alguns recursos avançados do SQL:1999 como "procedimentos armazenados", "triggers", recursão e orientação a objetos estão sendo oferecidos (sob formas diferentes) por vários SGBDs comerciais.

No que se segue apresentaremos o que é essencialmente a versão interativa da linguagem SQL e que o padrão chama de "SQL direto". Esta não é a forma usual de escrever aplicações para BDs relacionais: elas são geralmente escritas através de "SQL embutido" (*embedded SQL*) numa linguagem de 3^a geração como Delphi ou VB e recentemente através de linguagens *script* como Php, Perl, Java, Python, para aplicações via Web. Este importante assunto será tratado no capítulo 8. Por outro lado, o SQL interativo ou direto é extremamente útil para o Administrador da Base de Dados ("Data Base Administrator" ou DBA) realizar testes da BD e de recursos do sistema e para tarefas administrativas como criação, alteração de tabelas (e de outros objetos da BD), atribuição de privilégios e direitos de acesso a usuários da BD, etc. O processador do SQL interativo vem disponível em todos os SGBDs, muitas vezes com interfaces gráficas, ao contrário das outras ferramentas de desenvolvimento que precisam ser adquiridas ou instaladas.

6.2 Características Básicas

Ao contrário de C ou Pascal, que são linguagens algorítmicas, SQL é essencialmente uma linguagem "não procedural" ou "não algoritmica", ou seja, a sintaxe da linguagem especifica "que resultados se deseja obter" e não uma "seqüência de instruções de como obter os resultados". Isto ficará mais claro quando detalharmos os seus recursos.

SQL possui recursos de uma "Linguagem de Definição de Dados" (DDL), que essencialmente permitem especificar os esquemas de tabelas (o padrão evita sistematicamente o termo relações!) e de objetos relacionados e recursos de uma "Linguagem de Manipulação de Dados" (DML) que permitem modificar e recuperar dados. Possui também comandos relacionados com a concessão e revogação de privilégios de acesso e manipulação de tabelas por parte de usuários do sistema. Suporte ao conceito de "transações atômicas" também fazem parte do padrão SQL/92.

6.2.1 O ambiente de um SGBD relacional

Embora não diretamente relacionados com a linguagem SQL, os conceitos a seguir, provenientes do padrão SQL92, são úteis para caracterizar o funcionamento de um sistema baseado num SGBD relacional. Eles estão presentes no ambiente operacional da maioria dos SGBDS, embora com nomenclatura distinta da do padrão, que por exemplo, não menciona os termos "Base de Dados" ou "usuário".

O termo *ambiente SQL* é usado com o significado de um ambiente computacional composto por:

- uma ou mais instâncias do (programa) SGBD representando o *agente Servidor SQL*, juntamente com os programas de comunicação necessários para que um ou mais *agentes clientes* possam se conectar localmente ou remotamente ao *agente Servidor SQL*. Por exemplo, o *processador SQL interativo* referido anteriormente, é um *processo cliente* através do qual um usuário numa máquina cliente em que o pacote *agente cliente* do SGBD tenha sido instalado, pode submeter consultas SQL a um *agente Servidor SQL* remotamente localizado. Este cenário é suportado pela maioria dos SGBDs modernos.

- uma ou mais Bases de Dados controladas pelo Servidor SQL,

- uma coleção de usuários cadastrados e de programas com acesso ao SGBD e às suas Bases de Dados.

Uma "Base de Dados SQL" é composta por uma hierarquia começando por um *catálogo* definido dentro do *ambiente SQL*, contendo por sua vez um

ou mais *SQL-schemas*. Um *SQL-schema* é uma coleção de *descritores* de objetos (tabelas, visões [1], restrições e outros objetos) criados por um usuário nessa Base de Dados. O nome *default* de um schema é o nome do usuário. Uma tabela digamos, *mytable*, criada pelo usuário *scott*, no catálogo *mydbase* poderia ser referenciada assim:

mydbase.scott.mytable

Dentro de um catálogo, um *SQL-schema* especial denominado *Information Schema* possui um conjunto de tabelas SQL que descrevem todos os objetos definidos em todos os *SQL-schemas* desse catálogo. Na verdade, os SGBDs usam diversos nomes para o *Information Schema* como *Catálogo do Sistema, Dicionário do Sistema, Tabelas do Sistema* , etc. Um catálogo descreve, portanto, todos os objetos de todos os usuários da Base de Dados associada ao catálogo. Um *ambiente SQL* pode ter várias Bases de Dados (ou seja, vários catálogos).

Para poder usar um *ambiente SQL* um usuário (previamente cadastrado pelo DBA) deve se "conectar" ao SGBD através do comando:

connect "nome-servidor" e/ou "nome-base-dados", [as nome-conexão], "nome-usuário", ["senha"]

fornecendo o nome do Servidor SQL (seguido opcionalmente do nome da Base de Dados), opcionalmente um nome para a conexão, o nome do usuário e opcionalmente a senha do usuário. [2]

O comando *connect* inicia uma *sessão SQL* para o usuário que o invocou, a qual define um *contexto* para as operações executadas pelo usuário na Base de Dados. Para encerrar uma sessão e liberar os recursos a ela associados o usuário deve executar o comando *disconnect*. O padrão SQL92 permite a um usuário criar várias sessões, invocando múltiplos comandos *connect*, mas apenas a última sessão fica "ativa", e as outras ficam "dormentes". Um comando especial *set connection nome-conexão* permite passar uma conexão dormente para ativa. No entanto, em todos os exemplos desse livro suporemos que foi criada uma única conexão a um Servidor e Base de Dados específicos e,

[1] um tipo especial de *tabela virtual* a ser descrita mais adiante.

[2] A sintaxe desse comando varia de um SGBD para outro. Veja o manual do seu SGBD preferido. Alguns SGBDs permitem especificar na primeira cadeia a localização remota do Servidor SQL, precedendo o seu nome com o endereço DNS da máquina que o hospeda, por exemplo, no caso de Firebird/Interbase : *myserver.ic.unicamp.br:mydatabase.gdb*. Já o SGBD Oracle embute a localização do Servidor remoto num arquivo de configuração do *agente cliente*, e o usuário especifica apenas o nome de um *alias* para esse Servidor no comando *connect*.

portanto, uma única sessão.

6.2.2 Tipos de dados

Como outras linguagens, SQL usa identificadores definidos pelo usuário, constantes e palavras chave (reservadas). Nomes definidos pelo usuário e palavras reservadas não são sensíveis a maiúsculas e minúsculas. Esta característica será usada com freqüência nos exemplos de forma que o leitor já fica alertado sobre a mesma. Um identificador começa por uma letra seguida em qualquer ordem de outras letras, dígitos e do caractere '_' (até um máximo de 128 caracteres).

Os seguintes tipos de dados são suportados pelo padrão:

- CHAR(n) cadeia de caracteres com exatamente n caracteres,
- VARCHAR(n) cadeia com até n caracteres,
- BIT(n) cadeia com exatamente n bits,
- VARBIT(n) cadeia com até n bits,
- INT inteiro com sinal (precisão definida pela implementação, usualmente 32 bits)
- SMALLINT inteiro com sinal (precisão definida pela implementação)
- NUMERIC(p,q) número decimal com p dígitos mais sinal e ponto decimal implícito com q dígitos a partir da direita.
- DECIMAL(p,q) semelhante a NUMERIC, mas a precisão pode ser maior que p dígitos.
- FLOAT número em ponto flutuante (precisão definida pela implementação)

Existem tipos de dados para suportar diversas formas de datas e horas, por exemplo:

DATE '1999-12-25' , TIME '18:45:00' , TIMESTAMP '1999-12-25 18:45:00'.
No final do capítulo 6 veremos exemplos de como utilizá-los.

Constantes (ou literais)

cadeia de caracteres: 'Rua do Cabral 1500' (sensível a maiúsculas/minúsculas),
numéricas: 4, -95.7, +364.5, +36.5E-5,
cadeia de bits: B'11001100', B'0101', X'0D0A', X'ffaa' (binários e hexadecimais)

operadores aritméticos

"+", "-", "*" e "/", com o significado usual

operadores de comparação

" > ", " >= ", " < ", " <= ", " = ", " <> ", *ou* " ! = ", com o significado usual

concatenador de cadeias de caracteres: ||

'Pedro Alvares' || 'Cabral'

6.2.3 Comandos para definição de dados

CREATE TABLE é o principal comando de definição de dados e permite definir o esquema de uma tabela. *ALTER TABLE* permite alterar o esquema de uma tabela e *DROP TABLE* permite eliminar uma tabela.

Vamos apresentar agora os esquemas completos para criação das tabelas da BD "Torneios de Tenis da ATP", seguindo o padrão SQL92, introduzindo ao mesmo tempo a sintaxe para definir Chaves Primárias, Chaves Estrangeiras, chaves alternativas e valores *default*. Ela ficará clara pelo conhecimento que já temos da BD, detalhados no capítulo 3. Valores *default* são constantes aplicadas automaticamente sobre o valor de uma coluna, quando o mesmo não é fornecido pelo comando de inserção da linha correspondente na tabela.

A seguir, os esquemas simplificados das tabelas, vistos no capítulo 3:

Jogadores(numj, nome, pnome, pais, ano_n, ano_p, cid_n, cid_res, tit_s, tit_d, vits, derrs, natp)

Lista_Torneios(numt, nomet, pais, cat, quadra, nparts)

Torneios(numt, ano, numj, premio)

Os comandos SQL para criar os esquemas dessas tabelas seriam:

```
Create Table Jogadores
(numj          numeric(4)          not null,
nome           varchar(16)  unique not null,
pnome          varchar(16)         not null,
pais           char(3)             not null,
ano_n          numeric(4)          not null,
ano_p          numeric(4)          not null,
cid_n          varchar(16)         not null,
cid_res        varchar(16)         not null,
tit_s          numeric(4),
tit_d          numeric(4)        default 0,
vits           numeric(4),
derrs          numeric(4),
natp           numeric(4)          unique,
primary key(numj))
```

```
Create Table Lista_torneios
(numt                    numeric(2),   not null,
nomet                    varchar(24)   not null,
pais                     char(3)       not null,
cat                      char(2)       not null,
quadra                   char(1)       not null,
nparts                   numeric(4),   not null,
primary key(numt))

Create Table Torneios
(numt                numeric(2)              not null,
ano                  numeric(4)              not null,
numj                 numeric(4)              not null,
premio               numeric(4)              not null,
primary key(numt,ano),
foreign key(numt)    references   Lista_torneios(numt),
foreign key(numj)    references      Jogadores(numj))
```

Por conveniência, e para simplificar certas consultas, suporemos que nomes de jogadores (na verdade sobrenomes) são únicos. Caso isto não ocorra, acrescentaremos ao nome as iniciais do jogador. Esta restrição foi colocada na coluna *nome* no esquema da tabela Jogadores, através das palavras reservadas *unique not null*, que permitem definir uma chave alternativa para uma coluna. Felizmente isto não foi necessário com as instâncias escolhidas das tabelas, mas o seria, por exemplo, se tivéssemos incluído dados das melhores tenistas da atualidade (que incluem as irmãs Williams). Observe que a coluna *natp* tem apenas a restrição *unique*. Isto significa que *natp* pode tomar valores Nulos, porém todos os valores não Nulos devem ser distintos, e isto é um dos requisitos dessa Base de Dados. [3]

Para remover uma tabela da BD usaríamos o comando:

Drop Table Lista_Torneios

Existem comandos correspondentes para criar, alterar e eliminar domínios: (*CREATE DOMAIN, ALTER DOMAIN e DROP DOMAIN*), mas nem todos os SGBDs os implementam.

Um exemplo do comando *create domain*:

```
create domain pais_default char(3) default 'EUA'
```

Poderíamos substituir no comando *create table jogadores* anterior, a linha

[3] Colunas definidas sem a restrição *not null*, como *tits_s*, podem tomar valores Nulos, mas isto é uma convenção que depende da configuração do SGBD e que pode ser mudada. Nesse caso deveríamos especificar *null* para essas colunas.

pais char(3), por:

pais pais_default,

O comando Create Table apenas registra no SGBD o esquema de uma tabela, inicialmente vazia. Vejamos agora alguns comandos SQL simples para inserir linhas numa tabela (este assunto será elaborado de forma mais completa posteriormente). O comando a seguir insere uma linha na tabela Jogadores. Observe que valores do tipo char, varchar, date, são delimitados por aspas simples; valores numéricos são colocados diretamente.

insert into jogadores

values(817, 'Courier', 'Jim', 'EUA', 70, 78, 'Sanford', 'Orlando',

23, 6, 506, 237, NULL)

Vejamos agora um exemplo para cada uma das tabelas Lista_torneios e Torneios:

insert into Lista_torneios

values(1, 'Australian Open', 'AUS', 'GS', 'D', 128)

insert into torneios

values(1, 1992, 817, 278)

A maioria dos SGBDs possui utilitários para popular tabelas a partir de arquivos texto, onde cada linha representa uma linha da tabela e onde as colunas são delimitadas por um caracter prefixado, por exemplo, ';'.

Outros comandos permitem remover linhas de uma tabela (comando *delete*) e modificar valores de uma ou mais colunas de uma tabela (comando *update*). Eles serão detalhados na seção 6.4.

Vamos apresentar a seguir instâncias das três tabelas. Elas serão úteis quando quisermos "visualizar" o resultado de algumas consultas. Elas contêm dados reais sobre jogadores e torneios da ATP, extraídos do portal *www.atptour.com* em Setembro de 2002. Lembramos ao leitor que as últimas 5 colunas da tabela Jogadores possuem dados cumulativos (com estatísticas de torneios e partidas de cada jogador), periodicamente atualizados pela ATP, e portanto os dados que aparecem no texto estarão desatualizados.

Observe que a tabela Jogadores foi dividida na apresentação textual em duas partes para facilitar a leitura e os exercicios. Na segunda parte, para maior clareza, repetimos os valores das duas primeiras colunas.

Jogadores

numj	nome	pnome	pais	ano_n	ano_p	cid_n	cid_res
817	Courier	Jim	EUA	70	78	Sanford	Orlando
29	Agassi	Andre	EUA	70	86	Las Vegas	Las Vegas
19	Edberg	Stefan	SWE	66	83	Vastervik	Londres
16	Bruguera	Sergi	ESP	71	88	Barcelna	Barcelona
812	Sampras	Pete	EUA	71	88	Washington	Los Angeles
102	Muster	Thomas	AUT	67	85	Leibnitz	Monte Carlo
1122	Becker	Boris	GER	67	84	Leimen	Monte Carlo
218	Kafelnikov	Yevgeny	RUS	74	92	Sochi	Sochi
126	Krajicek	Richard	HOL	71	89	Rotterdham	Monte Carlo
910	Kuerten	Gustavo	BRA	76	95	Florianopolis	Florianopolis
1228	Rafter	Patrick	AUS	72	91	Mount Isa	Bermuda
123	Korda	Petr	CHE	68	87	Praga	Monte Carlo
827	Moya	Carlos	ESP	76	95	Palma Maiorca	Genebra
127	Safin	Marat	RUS	80	97	Moscou	Monte Carlo
913	Ivanisevic	Goran	CRO	71	88	Split	Monte Carlo
324	Johansson	Thomas	SWE	75	94	Linkoping	Monte Carlo
625	Costa	Albert	ESP	75	93	Lerida	Barcelona
224	Hewitt	Lleyton	AUS	81	98	Adelaide	Adelaide
56	Sa	Andre	BRA	77	96	Belo Horizonte	Belo Horizonte
412	Meligeni	Fernando	BRA	71	90	Buenos Aires	São Paulo

Jogadores (continuação)

numj	nome	tit_s	tit_d	vits	derrs	natp
817	Courier	23	6	506	237	NULL
29	Agassi	53	1	731	227	470
19	Edberg	42	18	806	270	NULL
16	Bruguera	14	3	447	271	171
812	Sampras	64	2	762	222	871
102	Muster	44	1	622	271	NULL
1122	Becker	49	15	713	214	NULL
218	Kafelnikov	26	25	573	274	274
126	Krajicek	17	3	400	209	1271
910	Kuerten	17	8	281	136	971
1228	Rafter	11	10	358	191	1272
123	Korda	10	10	410	248	NULL
827	Moya	11	0	309	176	876
127	Safin	10	1	214	123	180
913	Ivanisevic	22	9	595	320	1971
324	Johansson	7	0	224	179	375
625	Costa	12	0	325	204	675
224	Hewitt	16	2	245	78	281
56	Sa	0	1	42	65	597
412	Meligeni	3	7	194	206	471

Lista_Torneios

numt	nomet	pais	cat	quadra	nparts
1	Australian Open	AUS	GS	D	128
2	Roland Garros	FRA	GS	S	128
3	Wimbledon	ING	GS	G	128
4	U.S. Open	EUA	GS	D	128
5	Indian Wells	EUA	MS	D	64
6	Miami	EUA	MS	D	96
7	Monte Carlo Open	MON	MS	S	64
8	Roma	ITA	MS	S	64
9	Hamburgo	GER	MS	S	64
10	Toronto	CAN	MS	D	64
11	Cincinati	EUA	MS	D	64
12	Madri	ESP	MS	T	64
13	Paris Open	FRA	MS	T	64
14	Tennis Masters Cup 2002	CHN	MS	D	8

As quatro primeiras linhas da tabela Torneios têm o formato:

Torneios

1	1992	817	278
2	1992	817	490
3	1992	29	501
4	1992	19	500
...	...	...	...

A tabela Torneios é pouco elucidativa. Vamos apresentá-la a seguir de forma não normalizada e compacta. Por economia de espaço, foram incluídos apenas os resultados de 1992 a 2002 dos quatro torneios do Grand Slam, com o nome do vencedor e o prêmio ganho pelo mesmo no torneio (em milhares de dólares). Para maior clareza, foi colocado o nome do torneio e não o seu número que é o que aparece na tabela real normalizada (embaixo do nome do torneio aparece o primeiro ano em que o torneio se realizou; não faz parte da tabela real).

	Torneios			
Ano	Aus Open 1905	premio KUS$	French Open 1925	premio KUS$
1992	Courier	278	Courier	490
1993	Courier	275	Bruguera	537
1994	Sampras	340	Bruguera	527
1995	Agassi	341	Muster	637
1996	Becker	410	Kafelnikov	640
1997	Sampras	457	Kuerten	644
1998	Korda	407	Moya	649
1999	Kafelnikov	459	Agassi	668
2000	Agassi	501	Kuerten	590
2001	Agassi	462	Kuerten	594
2002	Johansson	520	Costa	718

	Torneios	(cont)		
Ano	Wimbledon 1877	premio KUS$	US Open 1881	premio KUS$
1992	Agassi	501	Edberg	500
1993	Sampras	462	Sampras	535
1994	Sampras	504	Agassi	550
1995	Sampras	549	Sampras	575
1996	Krajicek	609	Sampras	600
1997	Sampras	702	Rafter	650
1998	Sampras	727	Rafter	700
1999	Sampras	727	Agassi	750
2000	Sampras	717	Safin	800
2001	Ivanisevic	706	Hewitt	850
2002	Hewitt	786	Sampras	900

Ao trabalhar os exercícios deste capítulo, o leitor não deve supor que a tabela real contém apenas esses torneios e resultados. A tabela real, conforme mostrado no esquema anterior, possui apenas 4 colunas: número do torneio, ano, número do vencedor e prêmio. Ficaria muito longo apresentá-la no texto.

6.2.4 Comandos para alterar a definição de tabelas

Após a criação do esquema de uma tabela via comando *Create Table*, pode haver necessidade de alterar o esquema da tabela, por exemplo, para acrescentar uma nova coluna, ou uma nova restrição sobre valores de coluna(s), modificar o tipo de dados de uma coluna, remover uma coluna ou restrição,

etc. Isto pode ser problemático se a tabela já está populada e nem todos os SGBDs suportam todas essas operações. O comando para esta finalidade se chama *Alter Table*. Vamos apresentar a sintaxe de algumas opções desse comando via exemplos.

Se quisermos acrescentar uma coluna à tabela Torneios com o número do jogador que foi vice-campeão, poderíamos fazer assim:

Alter Table Torneios add column num_vice numeric(4) default 0

Se quisermos adicionar uma restrição de integridade poderíamos fazer:

Alter Table Lista_Torneios add constraint c1 check(cat in ('GS', 'MS', 'IS'))

Se agora quisermos remover essa restrição:

Alter Table Lista_Torneios drop constraint c1

Para remover uma coluna:

Alter Table Lista_Torneios drop column num_vice

Para remover uma tabela da BD:

Drop Table Lista_Torneios

6.2.5 Um guia prático para nomes de tabelas e colunas

Da mesma forma que definimos nomes para Conjuntos de Entidades no plural, usaremos nomes para tabelas também no plural.

É conveniente dar para um atributo que é Chave Estrangeira o mesmo nome da Chave Primária referenciada. Como freqüentemente serão feitas consultas requerendo junções nessas colunas, fica simples identificá-las, conforme mostra a tabela Torneios, onde a coluna representando o vencedor recebeu o nome *numj* e a coluna representando o torneio recebeu o nome *numt*.

Sufixos devem ser usados de forma consistente, por exemplo *numt e nomet* se referem à tabela Lista_Torneios.

Evite colocar sufixos que tenham informação adicional sobre a tabela, como um índice para indicar o número de atributos ou o número de índices.

6.3 O comando *select*

Este é o comando para fazer consultas ("queries") à BD. Ele engloba quase todo o poder da linguagem SQL. Variantes do mesmo são usadas para operações de inserção, remoção e atualização de linhas numa tabela. Num único comando *select* podemos expressar todas as operações da Álgebra Relacional ou do Cálculo Relacional (e outras não suportadas por ambos!).

6.3.1 Características Básicas

O comando *select* consiste de várias cláusulas que devem ser colocadas numa ordem prefixada (ver exemplo a seguir). Em geral fica mais legível colocando uma cláusula por linha, embora isto não seja necessário: o formato é inteiramente livre. Vamos adotar a convenção de uma cláusula por linha na maioria dos exemplos. Considere o seguinte exemplo[4]:

q1:
select pnome, nome, ano_n, cid_res
from jogadores
where pais = 'BRA'
order by pnome

De forma simplificada, cada cláusula desse exemplo (exceto a cláusula *order by*) corresponde a uma operação da Álgebra Relacional. As duas primeiras cláusulas *select e from* são obrigatórias: se a cláusula *from* apresenta apenas o nome de uma tabela, esta é a tabela da qual queremos obter os dados (se apresenta o nome de duas ou mais tabelas, isto corresponde ao produto cartesiano dessas tabelas, o que será usado quando virmos a operação de junção em SQL).

A cláusula *select* possui como entrada uma lista de atributos. Esses atributos definem o esquema da relação resultado e correspondem a uma operação de projeção da Álgebra Relacional.

A cláusula *where*, no caso mais comum, como no exemplo acima, apresenta uma expressão booleana idêntica à que é usada na operação de seleção da Álgebra Relacional, e possui a mesma finalidade: selecionar as linhas da tabela (ou do produto cartesiano, conforme o caso), que satisfazem à condição dada.

A esta altura deve estar clara a finalidade dessa consulta:
selecione da tabela Jogadores todos os que são de nacionalidade brasileira, apresentando para cada um o prenome, sobrenome, ano de nascimento e cidade de residência, ordenando alfabeticamente as linhas do resultado pelo prenome. Observe que a coluna *nome* corresponde na verdade ao *sobrenome* do jogador e assim foi chamada por economia.

Da mesma forma, é conveniente imaginar (conceitualmente) a execução do comando *select* como uma seqüência de operações onde uma tabela intermediária é produzida por cada cláusula e usada como entrada da cláusula

[4]Na configuração padrão da maioria dos SGBDs, nomes de tabelas, de atributos, de comandos e palavras reservadas, não são sensíveis ao uso de letras maiúsculas ou minúsculas, exceto constantes literais entre aspas como a cadeia 'BRA' no exemplo. Por conveniência e para maior legibilidade, usaremos letras minúsculas na maioria dos exemplos.

seguinte, começando pela cláusula *from* que escolhe a tabela inicial, passando-a para a cláusula *where* que seleciona linhas dessa tabela e as passa para a cláusula *order by* que ordena essas linhas (em ordem crescente dos prenomes), passando finalmente as linhas ordenadas para a cláusula *select* que faz a projeção nas colunas desejadas.

Vejamos alguns desses passos utilizando a instância da tabela *Jogadores*:

1. Tabela intermediária após a cláusula *where*:

numj	nome	pnome	pais	ano_n	ano_p	cid_n	cid_res	etc
910	Kuerten	Gustavo	BRA	76	95	Florianopolis	Florianopolis	...
56	Sa	Andre	BRA	77	96	Belo Horizonte	Belo Horizonte	...
412	Meligeni	Fernando	BRA	71	90	Buenos Aires	São Paulo	...

2. Tabela intermediária após *order by*:

numj	nome	pnome	pais	ano_n	ano_p	cid_n	cid_res	etc
56	Sa	Andre	BRA	77	96	Belo Horizonte	Belo Horizonte	...
412	Meligeni	Fernando	BRA	71	90	Buenos Aires	São Paulo	...
910	Kuerten	Gustavo	BRA	76	95	Florianopolis	Florianopolis	...

3. Tabela final após projeção no prenome, nome, ano de nascimento e cidade de residência:

pnome	nome	ano_n	cid_res
Andre	Sa	77	Belo Horizonte
Fernando	Meligeni	71	São Paulo
Gustavo	Kuerten	76	Florianopolis

Se quiséssemos agora saber todos os atributos dos jogadores, poderíamos usar apenas o símbolo '*' no lugar da lista de atributos. Ele tem o significado: "todos os atributos na ordem do esquema da tabela". Por exemplo,

q2:
*select **
from jogadores,

nos apresenta uma cópia da instância atual da tabela *Jogadores*, incluindo em cada linha todos os atributos de um jogador.

Quando dissemos que a cláusula select "faz uma projeção nas colunas desejadas", cometemos um erro sutil, pois isso nem sempre é verdade. Aqui, pela primeira vez aparece uma diferença fundamental entre SQL e Álgebra Relacional: algumas expressões SQL não produzem conjuntos matemáticos

(sem elementos repetidos), e a cláusula *select* tem esse efeito indesejável. Por exemplo:

q3:
select ano_n
from jogadores

não é equivalente à expressão: $\Pi_{ano\_n}(jogadores)$,

isto é, não nos fornece um vetor com um elemento para cada possível ano de nascimento dos jogadores, mas um vetor com o ano de nascimento de **cada** jogador, repetindo aqueles de jogadores que nasceram no mesmo ano! Aparentemente isto foi deixado assim no padrão SQL por razões de eficiência. Para eliminarmos os anos repetidos temos que usar na cláusula select a palavra reservada *distinct*:

q4:
select distinct ano_n
from jogadores

Achamos infeliz a opção adotada pelo padrão. O resultado "default" deveria ser um conjunto matemático de acordo com o Modelo Relacional e, caso se desejasse as tuplas repetidas, uma palavra reservada como *all* poderia ser usada antes da lista de atributos, da mesma forma que faz o operador *union* para união de tabelas, conforme veremos adiante.

No exemplo abaixo distinct aplica-se a ambas as colunas:

q5:
select distinct cid_res, ano_p
from jogadores

Recursos da cláusula *where*

Diversos operadores podem ser usados na cláusula *where* para compor uma expressão booleana arbitrariamente complexa, avaliada para cada linha da tabela especificada na cláusula from, e que, se verdadeira, seleciona a linha:

6.3.2 Operadores de comparação

São os 6 operadores que vimos na Álgebra Relacional: =, <>, >, <, >= *e* <= Exemplo:

q6: "Obtenha o nome dos jogadores que se tornaram profissionais com menos de 22 anos".

select nome
from jogadores
where ano_p - ano_n < 22

Este exemplo ilustra o uso de uma expressão aritmética envolvendo valores numéricos de colunas de uma tabela. Elas podem usar os operadores aritméticos $+$, $-$, $*$, $/$, com o significado usual, e ser arbitrariamente aninhadas.

6.3.3 Conectores lógicos *and, or, not*

Usados na cláusula *where* para compor expressões booleanas, com o significado usual. Parênteses podem ser usados para impor a precedência desejada. Exemplo:

select nome
from jogadores
where (ano_p - ano_n < 22
and pais = 'BRA')
or cid_res = 'Florianopolis'

6.3.4 Operador *between*

A expressão "A between B and C" é equivalente a:
$(A >= B)$ *and* $(A <= C)$
Já "not (A between B and C)" equivale a:
A not between B and C
q7: "Obtenha o nome e ano de nascimento dos jogadores nascidos entre 1975 e 1981"

select nome, ano_n
from jogadores
where ano_n between 75 and 81

6.3.5 Operadores *in, not in*

Usados para verificar se um valor está contido (ou não) num conjunto de constantes:

q8:
select numj
from jogadores
where cid_res in ('Las Vegas', 'Monte Carlo', 'Los Angeles')

Obs: o conjunto definido entre parênteses acima só pode ter constantes.
A not in (10, 20, 30) tem o significado: not (A in (10,20, 30))
Exercício:
q9: "Dê o nome dos jogadores que não moram nem em Las Vegas nem em Los Angeles".

6.3.6 Operador *like*

O operador *like* permite fazer um casamento de padrões de cadeias de caracteres:

q10:
select nome
from jogadores
where nome like 'B%',

obtém os nomes dos jogadores cujo nome começa pela letra B.
O símbolo % significa: "qualquer cadeia com zero ou mais caracteres".

q11:
select nome
from jogadores
where nome like '%n' ,

obtém o nome de cada jogador que termina com a letra n.
O caractere '_' quando usado num padrão significa: "exatamente um caractere naquela posição":

q12:
select nome
from jogadores
where nome like '%e_' ,

obtém o nome de cada jogador em que o caractere *e* é a penúltima letra.
Se quisermos buscar uma cadeia contendo o caractere '_ ', temos que usar uma forma de evitar a interpretação acima. Isto pode ser feito através de um caractere definido como *escape*. Por exemplo:

where nome like '%=_%' escape '=',
procura os nomes que possuem o caractere '_'.
Exercícios:
Escreva uma consulta para cada um dos casos abaixo:
"Dê o nome de cada jogador cujo nome satisfaz:

1. q13: contem a seqüência de letras 'is',
2. q14: possui exatamente 5 caracteres,
3. q15: possui pelo menos 5 caracteres,
4. q16: tem a letra a como a segunda e penúltima letra.

6.3.7 Funções para manipular cadeias de caracteres

Há várias funções intrínsecas para este fim. Elas tomam como operando uma cadeia de caracteres que pode ser o valor de uma coluna da lista da *cláusula select*, ou operando da *cláusula where* ou uma constante literal. Nas especificações a seguir parâmetros opcionais estão incluídos entre colchetes '[ ]' e parâmetros alternativos serão separados por '|'.

- position (cadeia1 in cadeia2) - obtém a posição onde a cadeia1 começa na cadeia2 (retorna 0 se cadeia1 não encontrada). Exemplo:

  position ('W' in 'Hello World!') - retorna 7

- char_length(cadeia1) - obtém o número de caracteres da cadeia1

- bit_length(cadeia1) - obtém o comprimento de uma cadeia de bits. Exemplo:

  bit_length(B'01010101') - retorna 8

- upper(cadeia1) - retorna o argumento com letras maiúsculas. Exemplo:

  select upper(nome), upper(cid_n)
  from jogadores
  where numj = 910

- lower(cadeia1) - retorna o argumento com letras minúsculas

- substring(cadeia1, start [, comp]) - extrai uma subcadeia da cadeia1, começando na posição *start* e com comprimento *comp* caracteres. Se este for omitido, extrai até o fim de cadeia1. Exemplo:

  substring ('Hello World!', 7, 4) retorna: *'World'*

- trim([[both | leading | trailing]][char] [from] cadeia1 - remove todos os caracteres *char* do começo (*leading*), do final (*trailing*) ou de ambos (*both*) da cadeia cadeia1. Se o 1^o parâmetro for omitido, toma ambos (*both*); se *char* for omitido considera como branco (' '). Exemplo:

  trim (' foo ') - retorna 'foo'

- overlay(cadeia1, cadeia2, start [, comp]) - semelhante à função *substring*, porém substitui a subcadeia começando na posição *start* de *cadeia2* pela cadeia *cadeia1*. Exemplo:

  overlay('Campinas', 'Hello World!', 7, 4) - retorna 'Hello Campinas!'

- operação de concatenação: '||' - útil para concatenar valores alfanuméricos provenientes de 2 ou mais colunas da lista da *cláusula select*. Exemplo:

  select pnome || ' ' || nome
  from jogadores
  where numj = 910

  Esta consulta retorna a cadeia *'Gustavo Kuerten'*

As seguintes funções usam operandos numéricos:

- abs(valor-numérico). Exemplo: abs(-12) retorna 12
- mod(valor1, valor2). Exemplo: mod(8, 3) retorna 2

Função *cast* para conversão de tipos de dados

Esta função é útil quando precisamos comparar tipos de dados diferentes, por exemplo, uma cadeia de caracteres que representa um número e que queremos comparar com um literal numérico (ou usar como operando de uma expressão aritmética). Suponha, por exemplo, que a coluna *numt* fosse do tipo char(3) e contendo valores variando de '001' a '999'. Então poderíamos usar na cláusula *where*:

where cast (numt as int) > 100

6.3.8 Operação de junção

Vejamos agora como fazer a importante operação de junção de duas tabelas. A seguinte consulta pode ser expressa com esta operação:

q17: Para cada torneio, dê o seu nome, ano e o número do seu vencedor.

Observe que a tabela *Torneios* tem, do torneio, apenas o seu número; para obter o seu nome teríamos que "olhar" na tabela *Lista_torneios*, por exemplo, através de uma junção de *Torneios* com *Lista_torneios* na coluna *numt*:

select nomet, ano, numj
from torneios, lista_torneios
where torneios.numt = lista_torneios.numt

Observe que para evitarmos ambigüidade usamos a notação *torneios.numt* com o significado "a coluna numt da tabela torneios". Fazendo analogia com o Cálculo Relacional de Tuplas, nesse exemplo os nomes das tabelas *torneios e lista_torneios* estão sendo usados como variáveis tuplas. Esse recurso será usado com freqüência nos exemplos.

Podemos interpretar a execução desse comando da seguinte forma:

1. obtenha o produto cartesiano de Torneios com Lista_torneios;
2. para cada linha do produto cartesiano selecione aquelas em que *torneios.numt = lista_torneios.numt*
3. faça uma projeção do resultado nas colunas *nomet, ano e numj.*

Vamos agora visualizar os passos 2 e 3 desse comando com as instâncias das tabelas *Torneios e Lista_torneios*, lembrando que o passo 1 concatena cada linha da tabela *Torneios* com cada linha da tabela *Lista_torneios*:

2. Seleção das linhas do produto cartesiano onde *torneios.numt = lista_torneios.numt*

1	1992	817	278	1	Australian Open	AUS	GS	D	128
2	1992	817	490	2	Roland Garros	FRA	GS	S	128
3	1992	29	501	3	Wimbledon	ING	GS	G	128
4	1992	19	500	4	U.S. Open	EUA	GS	D	128
1	1993	817	275	1	Australian Open	AUS	GS	D	128
				- - -					
				- - -					

3. Projeção no nome do torneio, ano, número do vencedor:

nomet	ano	numj
Australian Open	1992	817
Australian Open	1993	817
	- - -	
	- - -	
Roland Garros	1992	817
Roland Garros	1993	16
	- - -	
	- - -	
Wimbledon	1992	29
Wimbledon	1993	812
	- - -	
	- - -	
U.S. Open	2001	224
U.S. Open	2002	812

Existem mais duas cláusulas para o comando *select*, que não apareceram nos exemplos acima por acrescentarem certa complexidade ao comando: *group by* e *having*. Se usadas, elas devem aparecer nessa ordem, após a cláusula *where*, mas antes da cláusula *order by*. Exemplos com as mesmas serão vistos mais adiante.

6.3.9 Sinônimos para nomes de tabelas (*alias*)

Assim como na Álgebra Relacional precisamos do operador de renomeação, e de variáveis tuplas no Cálculo Relacional, em SQL podemos criar sinônimos para o nome de uma tabela, com o mesmo objetivo. Isto é fundamental em consultas envolvendo uma junção de uma tabela consigo mesma (para obter o equivalente a um auto-relacionamento). Exemplo:

q18: "obter os nomes dos jogadores mais velhos que Kuerten".

Na solução abaixo, j1 e j2 são sinônimos da tabela Jogadores:

select j1.nome
from jogadores j1, jogadores j2
where j2.nome = 'Kuerten' and j1.ano_n < j2.ano_n

Às vezes precisamos fazer a junção de 3 tabelas para obter os dados desejados. O exemplo abaixo esclarece o ponto e introduz sinônimos (*alias*) para as tabelas a fim de minimizar o texto da consulta.

q19: "Para cada jogador que venceu algum torneio obtenha o seu nome, o nome do torneio e o ano da vitória no torneio".

select j.nome, lt.nomet, t.ano
from jogadores j, lista_torneios lt, torneios t
where j.numj = t.numj and t.numt = lt.numt

6.3.10 Problemas com operandos Nulos

Valores Nulos são problemáticos, não só pela falta de disponibilidade dos dados, mas, principalmente, quando aparecem como operandos de consultas SQL. A semântica dessas operações não é claramente definida no padrão e resultados inesperados podem ocorrer. No que se segue daremos apenas uma breve introdução aos problemas do que é talvez, uma das facetas mais obscuras da linguagem SQL.

Vamos denotar com o símbolo '?' o valor Nulo.

Qualquer expressão aritmética em que um dos operandos tem o valor Nulo tem como resultado Nulo.

Exemplo: $10 + ? = ?$

Comparações envolvendo valores Nulos são também problemáticas: se x tem o valor Nulo qual deve ser o resultado da comparação 'x = 3'? Não é *falso* nem *verdadeiro*, mas *desconhecido*! Isto nos leva a uma lógica booleana de três valores: 'falso', 'verdadeiro' e 'desconhecido'. Vamos denotar também por '?' o valor booleano *desconhecido* e β um valor qualquer não Nulo.

Expressão	Resultado
$\beta\ \theta\ ?$	?
$10\ >\ ?$	?
$?\ \theta\ ?$	?
$?\ =\ ?$	?

A existência de valores Nulos pode levar a resultados inesperados em certas consultas:

q20:
*select * from jogadores*
where natp = NULL

não retorna nada, embora haja vários jogadores cujo natp tem o valor Nulo.

q21:
*select * from jogadores*
where natp = natp

retorna apenas os jogadores cujo natp não é Nulo, em vez de retornar todos os jogadores, já que existem valores nulos para o atributo natp.

Para testar se um determinado valor é Nulo temos operadores especiais, *IS NULL* e *IS NOT NULL*:

q22:
*select * from jogadores*
where natp IS NULL,

nos dá todos os jogadores aposentados.

q23:
*select * from jogadores*
where natp IS NOT NULL,

nos dá todos os jogadores que participam de torneios oficiais da ATP.

A seguinte consulta deve ser formulada em SQL com cuidado:

q24: "dê o nome de **todos** os jogadores exceto daquele cujo natp é 1271".

A solução abaixo não dá o resultado esperado:

select nome
from jogadores
where natp <> 1271

Expressões booleanas envolvendo valores Nulos são também problemáticas. A seguinte tabela nos dá o resultado de expressões booleanas onde pelo menos um dos operandos tem o valor Nulo (*desconhecido*):

Expressão	Resultado
? *and* ?	?
? *or* ?	?
not ?	?
β *and* ?	? ou F(caso β seja Falso)
β *or* ?	? ou T(caso β seja True)

Observe que agora, na presença de Nulos, linhas de uma tabela não serão selecionadas pela cláusula *where* quando o resultado da expressão booleana for *falso* **ou** *desconhecido*.

Exercícios:

1. q25: Obtenha uma solução correta para a consulta: "dê o nome de **todos** os jogadores exceto daquele cujo natp é 1271".

2. q26: Qual o resultado do comando:

   select numj, natp

   from jogadores

   where natp >= natp

Consultas aninhadas (subconsultas)

O poder da linguagem SQL começa a aparecer quando na cláusula *where* usamos como operando de uma expressão booleana o resultado de um outro comando *select* (e que chamaremos de "consulta aninhada" ou subconsulta). Há várias formas como isto pode ser feito:

6.3.11 Subconsultas com o operador *in*

q27: "Obtenha o nome de cada jogador que venceu pelo menos um torneio."

select nome
from jogadores
where numj in
(select numj from torneios),

Obs: a consulta aninhada (entre parênteses) gera um conjunto de valores contendo o número de cada jogador que venceu torneios, e esse conjunto é usado pelo operador *in*. Poderíamos fazer a mesma consulta usando uma junção:

q28:
select distinct nome
from jogadores, torneios
where jogadores.numj = torneios.numj

Observe que, como um jogador pode ter vencido vários torneios, necessitamos de usar *distinct* na cláusula *select*, para evitar que o seu nome apareça mais de uma vez.

Suponha que queremos agora, "q29: obter os nomes dos jogadores que venceram uma ou mais vezes o torneio 1":

select nome
from jogadores
where numj in
(select numj from torneios
where numt = 1),

q30: O exemplo acima poderia também ser feito com uma junção (tente!).

Há casos, porém, em que não podemos usar uma consulta aninhada no lugar de uma junção:

q31: "Para cada jogador que venceu torneios, obtenha o seu nome, o número do torneio e o ano".

select nome, numt, ano
from jogadores, torneios
where jogadores.numj = torneios.numj

O contrário também é verdadeiro: há casos em que uma consulta aninhada não pode ser substituída por uma junção:

q32: "obtenha os nomes dos jogadores que não venceram torneios":

select nome
from jogadores
where numj not in
(select numj from torneios)

* O que resultaria se usarmos para essa consulta a seguinte junção de forma intuitiva:

q33:
select nome
from jogadores, torneios
where jogadores.numj <> torneios.numj

6.3.12 Subconsultas com operador de comparação

q34: "Obtenha o prenome e o nome do jogador que ganhou o torneio 2 em 1997".

select pnome, nome
from jogadores
where numj =

(select numj
from Torneios
where numt=2 and ano= 1997)

q35: "Obtenha o nome de cada jogador que é mais velho que Kuerten".

select nome
from jogadores
where ano_n <
(select ano_n
from jogadores
where nome = 'Kuerten')

Obs: como o operador de comparação compara dois valores escalares, é fundamental garantir que a subconsulta só retorne um valor. [5] A especificação da tabela *jogadores* garante isto, nesta consulta.

Considere agora a seguinte variante da consulta anterior:

q36: "Obtenha o(s) nome(s) do(s) jogador(es) mais velho(s) que todos os outros jogadores".

Deve ser óbvio que a solução abaixo é incorreta:

select nome
from jogadores
where ano_n <=
(select ano_n from jogadores)

6.3.13 Operadores *all e any* em subconsultas

O operador *all* resolve o problema da solução errônea da consulta anterior:

select nome
from jogadores
where ano_n <= all
(select ano_n from jogadores),

onde $v\ \theta\ all\ (c_1, c_2, \ldots, c_n)$ significa: $v\ \theta\ c_1\ and\ v\ \theta\ c_2\ and\ \ldots v\ \theta\ c_n$

Se agora quisermos obter q37: "o nome da cada jogador que não pertence ao grupo dos mais velhos" , utilizaremos o operador any:

[5] caso contrário o SGBD acusará um erro durante a execução da consulta

select nome
from jogadores
where ano_n > any
(select ano_n from jogadores)

any $(c_1, c_2, \ldots\ c_n)$ significa: "algum dentre" $c_1, c_2, \ldots\ c_n$, ou seja,
$v\ \theta$ *any* $(c_1, c_2, \ldots, c_n)$ significa: $v\ \theta\ c_1$ *or* $v\ \theta\ c_2$ *or* $\ldots v\ \theta\ c_n$

Observe que apenas o(s) jogador(es) mais velho(s) dá(dão) um resultado falso para a comparação acima.

6.3.14 Operadores *exists e not exists*

Esses operadores têm função semelhante ao operador $\exists$ do Cálculo Relacional. Eles são sempre usados na forma:

... where [not] exists
(subconsulta)

se o resultado da subconsulta é uma tabela[6] não vazia então *exists* retorna verdadeiro, senão retorna falso. Se usarmos *not exists*, então o contrário se passa: *not exists* retorna verdadeiro se e somente se o resultado da subconsulta é uma tabela vazia.

A seguinte consulta, antes resolvida com o operador in, também pode ser expressa com o operador *exists*, embora de forma menos clara:

q38: "obtenha o nome de cada jogador que venceu algum torneio".

select nome
from jogadores
where exists
*(select * from torneios*
where torneios.numj = jogadores.numj)

Este é o primeiro exemplo do importante conceito de "subconsulta correlacionada". Até agora uma subconsulta era inteiramente avaliada e o seu resultado passado para a consulta mais externa; neste caso, porém, para cada linha da tabela Jogadores a subconsulta é avaliada, utilizando o valor corrente de *jogadores.numj*. Por esta razão, subconsultas correlacionadas podem ser mais ineficientes que subconsultas comuns, e devem ser evitadas quando possível; os exemplos desta seção servem apenas para ilustrar

[6] consideramos que consultas retornando apenas uma linha ou um valor escalar retornam uma tabela degenerada

a semântica desses operadores e poderiam ser facilmente expressos sem os mesmos e através de subconsultas não correlacionadas. No entanto, em consultas mais complexas, o recurso é de inestimável valor, como veremos mais adiante.

Observe que, como só estamos interessados em saber se o resultado da subconsulta é ou não uma tabela vazia, é comum usar a cláusula: *select ** (embora não seja obrigatório, qualquer coluna da tabela mencionada na cláusula *from* poderia ser utilizada).

Outro exemplo:
q39: "obtenha o nome de todos os jogadores que não foram vencedores de torneio".

select nome
from jogadores
where not exists
*(select * from torneios*
where torneios.numj = jogadores.numj)

Exercícios:
Utilize o operador *exists* ou *not exists* para expressar as consultas abaixo:

1. q40: "Obtenha o nome de cada jogador que venceu o torneio de Wimbledon".

2. q41: "Obtenha o nome de cada jogador que não foi vencedor de torneios vencidos pelo jogador Sampras (numj = 812)".

6.3.15 Escopo de nomes em comandos *select*

Para maior clareza, nos dois últimos exemplos qualificamos o número do jogador com o nome da tabela correspondente. O escopo de nomes em SQL não nos obriga a essa qualificação: um atributo não qualificado numa cláusula *where* e que existe com o mesmo nome na cláusula *from* imediatamente anterior, é considerado como se referindo ao atributo dessa tabela. Em outras palavras, a cláusula *where* da subconsulta anterior poderia ser escrita, sem ambigüidade, assim:
where numj = jogadores.numj

Esse esquema de escopo de nomes é recursivo: podemos imaginar as cláusulas *select e from* como definindo um novo escopo de variáveis locais para tabelas e suas colunas, que é visível em todas as consultas aninhadas que se seguem, a menos que haja redefinição de alguma tabela em alguma

consulta mais aninhada. Essa visibilidade de nomes de tabelas e de suas colunas é semelhante à de variáveis em linguagens de programação com estrutura de blocos como Pascal.

6.3.16 Operadores união *(union)*, interseção *(intersect)* e diferença (*except).*

Esses operadores permitem fazer as operações indicadas com tabelas ou partes de tabelas "compatíveis quanto à união".

Exemplo:

Suponha que duas tabelas foram criadas na nossa BD, uma contendo os jogadores aposentados e outra contendo os jogadores ativos:

	Jogadores_Aposentados		Jogadores_Ativos
numj	nome	numj	nome
817	Courier	412	Meligeni
19	Edberg	29	Agassi
102	Muster	16	Bruguera
1122	Becker	812	Sampras
123	Korda	218	Kafelnikov
⋮	⋮	⋮	⋮

q42: "Dê o número e nome de todos os jogadores".

```
select numj, nome
from jogadores_aposentados
union
select numj, nome
from jogadores_ativos
```

Exemplos semelhantes poderiam usar o operador *intersect* ou o operador *except.* [7]

As seguintes restrições valem para esses operadores:

- Várias uniões são admissíveis encadeando comandos *select* via *union*; idem para *intersect* e *except*; qualquer combinação deles é admissível,

[7] Nem todos os SGBDs utilizam as mesmas palavras reservadas para esses operadores. Por exemplo, Oracle usa *minus* para diferença

- apenas o último *select* pode conter *order by*,
- linhas duplicadas são eliminadas, a menos que se use a palavra reservada *all* após o nome do operador.
- colunas correspondentes devem ser do mesmo tipo
- *distinct* não é permitido

A eliminação de duplicatas com esses operadores, ao contrário do que faz o comando *select* sem os mesmos, pode dar resultados inesperados. Por exemplo, a operação de união, aparentemente redundante no exemplo abaixo, produz menos linhas no resultado da consulta (no caso, 3) do que o comando select sem a união (produziria nesse exemplo, 13 linhas):

```
select numj, numt
from torneios
where numj = 812
union
select numj, numt
from torneios
where numj = 812
```

O mesmo efeito ocorreria nesse exemplo, se no lugar de união usássemos o operador de interseção (*intersect*).

6.3.17 Cláusula *order by*

A cláusula *order by* define critérios de ordenação de linhas a partir de uma ou mais colunas do resultado do comando *select*. É possível especificar múltiplas colunas para ordenação:

```
q43:
select numj, ano
from torneios
order by numj, ano
```

Nesse caso é feita uma ordenação lexicográfica: por jogador, e para cada jogador pelo ano do(s) torneio(s) que venceu.

É possível especificar em vez do nome, a posição na lista *select* da coluna desejada para ordenação. No exemplo anterior poderíamos escrever:

```
order by 1, 2
```

A ordenação é ascendente por *default*, mas pode ser descendente usando a palavra reservada *desc*

q44:
select nome, cid_res
from jogadores
order by cid_res asc, nome desc

A ordenação envolvendo Nulos depende da implementação do SGBD, como no exemplo:

q45:
select natp
from jogadores
order by natp

Nos SGBD SQL Server e Sybase Nulos aparecem antes de qualquer valor, seja para ordem ascendente ou descendente; no Oracle, aparecem em último lugar se ordem ascendente e em primeiro lugar se ordem descendente.

6.3.18 Funções que produzem valores agregados

SQL possui um conjunto de funções intrínsecas para obter estatísticas sobre os valores de uma dada coluna. Elas são chamadas de "funções agregado":
count(nome-coluna): obtém o número de valores na coluna especificada
min(nome-coluna): obtém o valor mínimo da coluna
max(nome-coluna): obtém o valor máximo da coluna
sum(nome-coluna): soma os valores da coluna
avg(nome-coluna): obtém a média aritmética dos valores da coluna
Observações:

- *count(*)* conta o número de linhas
- *sum* e *avg* só admitem colunas com valores numéricos
- *count, sum* e *avg* podem ter como argumento: *distinct nome-coluna*. Nesse caso somente valores distintos são considerados.
- valores Nulos na coluna especificada são desconsiderados; por esta razão *count(nome-coluna)* pode dar menor que *count(*)* se *nome-coluna* possui valores Nulos.

6.3.19 Funções agregado na cláusula *select*

Elas são aplicadas sobre todas as linhas selecionadas:

q46: "Quantos torneios existem?".

```
select count (*)
from lista_torneios
```

q47: "Qual foi o prêmio de valor máximo?".

```
select max(premio)
from torneios
```

q48: " Quantas cidades de residência distintas existem?".

```
select count (distinct cid_res)
from jogadores
```

q49: " Quantos jogadores vivem em Monte Carlo?".

```
select count (*)
from jogadores
where cid_res = 'Monte Carlo'
```

q50: "* Quantos prêmios são iguais ao mínimo prêmio?".

```
select count (premio)
from torneios
where premio =
(select min (premio)
from torneios)
```

q51: "Obtenha para cada jogador que venceu torneios, o valor do seu maior prêmio, o número do torneio e o ano em que foi ganho".

```
select numj, premio, numt,ano
from torneios t1
where premio =
( select max(premio )
from torneios t2
where t2.numj = t1.numj )
```

Exercícios:

1. Qual a diferença entre os comandos abaixo?

   q52:

   select avg (premio)

   from torneios

   e

   *select sum (premio) / count (*)*

   from torneios

2. O que há de errado no comando abaixo?

   q53:

   select count (*), numj

   from jogadores

3. q54: Qual o prêmio médio para jogadores que venceram o torneio 1 ?

4. q55: * Obtenha o valor médio de jogadores por cidade de residência, i. é, total de jogadores / total de cidades de residência

5. Obtenha as seguintes estatísticas de desempenho dos jogadores:

   q56: Qual a porcentagem de vitórias de cada jogador em relação ao número total de torneios disputados, isto é: *vits*100/(vits + derrs)* Apresente os jogadores em ordem decrescente dessa métrica.

   q57: Qual jogador obteve o melhor desempenho segundo a métrica acima?

   q58: Qual a porcentagem de vitórias em torneios de simples e duplas de cada jogador, em relação ao número total de torneios disputados? Apresente-os em ordem decrescente dessa medida.

   q59: Qual o jogador que obteve o melhor desempenho segundo essa métrica?

   q60: Quais jogadores tiveram mais derrotas que vitórias em torneios da ATP?

   q61: Qual jogador obteve a maior soma de títulos em torneios de simples e duplas?

   q62: * Obtenha os nomes dos jogadores cujas vitórias são maiores que o dobro das derrotas, em ordem decrescente dessa métrica.

5. intervalos day-time são compatíveis entre si

6. não há outras possíveis compatibilidades, tanto para atribuição de valores quanto para comparação. Por exemplo, intervalos year-month e day-time não podem ser comparados ou usados em atribuições e expressões aritméticas. A razão desta restrição é que meses possuem diferentes dias, causando ambigüidades na execução de operações: se somarmos 29 ou 30 dias a um intervalo *year-month* o mês deveria ser incrementado ou não?

As seguintes conversões podem ser necessárias antes de se efetuar uma atribuição:

- A atribuição de um intervalo *year* a um intervalo *year to month* implica na extensão do primeiro com um campo *month* contendo 0.

- A atribuição de um intervalo *month* a um intervalo *year to month* implica na extensão do primeiro com um campo *year* e o ajuste do valor do mês: por exemplo se o valor a ser atribuído é de 18 meses o ajuste será feito para "1 ano e 6 meses".

- A atribuição de um intervalo *year* a um intervalo *month* implicará na conversão esperada: por exemplo, 2 anos serão convertidos para 24 meses antes da atribuição.

- A atribuição de um intervalo *month* a um intervalo *year* implicará na conversão do primeiro para *year*. Se houver perda de informação um erro de execução (exceção) ocorrerá.

- Os seis operadores de comparação Θ podem ser usados e conversões análogas às feitas acima para atribuição de valores podem ser necessárias.

- A comparação de intervalos leva em conta o sinal e a magnitude.

- A comparação de valores *datetime* leva em conta a ordem cronológica dos mesmos.

6.8 * Usos avançados do comando *select*

Consultas mais complexas em SQL normalmente requerem construções mais elaboradas de conjuntos (isto é, de tabelas) envolvendo:

- o complemento de uma tabela ou a diferença de duas tabelas, [12]
- a construção de uma tabela S, tal que $S \supseteq T$ ou $S \subseteq T$, onde T é uma tabela com o mesmo esquema de S. [13]
- a construção de uma tabela S, tal que $S = T$.

q131: "Obtenha os jogadores que nunca venceram o torneio 1".
Por que a solução abaixo é incorreta?:

select numj
from torneios
where numt <> 1

Resposta: a coluna selecionada pode ter mais de uma linha por jogador, e se numa linha numt <> 1 e noutra linha numt = 1, ele será selecionado!
Solução (quase) correta:

q132:
select numj
from torneios
where numj not in
(select numj
from torneios
where numt = 1)

Observe que estamos fazendo a subtração de dois conjuntos: do conjunto dos jogadores que venceram torneios estamos subtraindo (via operador *not in*) o conjunto dos jogadores que venceram o torneio 1.
A solução acima não é totalmente correta porque não inclui os jogadores que nunca ganharam torneio algum. A consulta abaixo resolve o problema (apresentando o nome do jogador em vez do número):

q133:
select nome
from jogadores
where numj not in
(select numj

[12] Tais operações são comumente expressas em linguagem natural através de uma sentença contendo uma negação, conforme veremos nos exemplos a seguir.
[13] O termo tabela está sendo usado aqui no sentido de uma expressão cujo resultado é uma tabela gerada dinamicamente por um comando select ou por uma subconsulta. Tais expressões serão vistas de forma mais geral na próxima seção.

from jogos
where numt = 1)

q134: "Dê o número de cada torneio que o jogador 910 não venceu".

select numt
from lista_torneios
where numt not in
(select numt
from torneios
where numj = 910)

q135: "Dê o número de cada jogador que nunca venceu nem o torneio 1 nem o torneio 2".

select numj
from jogadores
where numj not in
(select numj
from torneios
where numt = 1 or numt=2)

Exercícios:

1. q136: Qual o problema se na consulta anterior usarmos a tabela torneios no lugar de jogadores no comando select externo?

2. q137: "Dê o nome de cada torneio que o jogador 910 não venceu".

q138: "Obtenha o número de cada jogador que venceu algum torneio não vencido pelo jogador 812".
Solução:

select numj from torneios
where numt not in
(select numt from torneios
where numj = 812))

Na consulta acima, se no lugar do número do jogador quisermos o seu nome, precisamos fazer ou duas consultas aninhadas ou uma junção e uma consulta aninhada:

q139:

select nome
from jogadores
where numj in
(select numj from torneios
where numt not in
(select numt from torneios
where numj = 812))

Observação: a consulta mais aninhada obtém o conjunto dos torneios que foram vencidos pelo jogador 812; a consulta imediatamente externa a esta, obtém os números dos jogadores que venceram torneios que não esses; a partir daí a consulta mais externa obtém os seus nomes.

Exercício: q140: expresse a consulta acima com uma junção seguida de uma consulta aninhada.

q141: "*Quais jogadores não venceram todos os torneios do 'Grand Slam'?"
Solução:

1. fixe um jogador *t1.numj,* e obtenha o conjunto T de torneios que ele venceu (subconsulta mais interna),

2. do conjunto de todos os torneios do Grand Slam subtraia o conjunto T (subconsulta mais externa), e se o resultado for uma tabela não vazia, ela contém os torneios que *t1.numj* não venceu e, portanto, o qualifica pela cláusula *where EXISTS*,

3. repita para todos os jogadores de t1 (comando select externo):

select distinct numj
from torneios t1
where EXISTS
(select numt
from lista_torneios
where cat = 'GS' and numt not in
(select numt
from torneios t2
where t2.numj = t1.numj))

Se quisermos agora saber:
q142: "Quais jogadores venceram todos os torneios do Grand Slam?",
a mudança na consulta acima deve ser óbvia.