

Banco de Dados I



ULBRA



Organizado por Universidade Luterana do Brasil

Banco de Dados I

Christiano Cadoná

Universidade Luterana do Brasil – ULBRA
Canoas, RS
2015

Conselho Editorial EAD

Andréa de Azevedo Eick
Ângela da Rocha Rolla
Astomiro Romais
Claudiane Ramos Furtado
Dóris Gedrat
Honor de Almeida Neto
Maria Cleidia Klein Oliveira
Maria Lizete Schneider
Luiz Carlos Specht Filho
Vinicius Martins Flores

Obra organizada pela Universidade Luterana do Brasil.
Informamos que é de inteira responsabilidade dos autores
a emissão de conceitos.

Nenhuma parte desta publicação poderá ser reproduzida
por qualquer meio ou forma sem prévia autorização da
ULBRA.

A violação dos direitos autorais é crime estabelecido na Lei
nº 9.610/98 e punido pelo Artigo 184 do Código Penal.

Dados Internacionais de Catalogação na Publicação – CIP

C125b Cadoná, Christiano.
Banco de dados I / Christiano Cadoná. – Canoas : Ed. ULBRA, 2015.
219 p. : il.

1. Banco de dados. 2. Modelo relacional. 3. Software. 4. Sistema de
informação. 5. Modelagem de dados. I. Título.

CDU 681.3.65

Setor de Processamento Técnico da Biblioteca Martinho Lutero – ULBRA/Canoas

ISBN: 978-85-68453-76-6

Dados técnicos do livro

Diagramação: Jonatan Souza

Revisão: Igor Campos Dutra

Introdução

Prezados(as) alunos(as)

Sejam bem-vindos(as) à disciplina de Banco de Dados I. Nessa disciplina, vocês entenderão a importância da utilização de banco de dados, além dos conceitos relacionados a este.

Também serão abordados modelos os quais são utilizados para representar os dados, tendo como foco o modelo Relacional, em que serão estudados conceitos sobre entidades, relacionamentos, atributos e chaves primárias e estrangeiras.

Durante a disciplina, você estará se capacitando a analisar e criar um banco de dados, passando por todas as etapas do processo de banco de dados.

Também terá conhecimento necessário para criar e manipular tabelas por meio de instruções SQL de inserção, alteração, remoção e seleção de dados. Tendo em vista que a seleção de dados possui uma série de especificações, esse material foi elaborado com exemplos práticos da utilização de critérios existentes do comando de seleção como, junções, produto cartesiano e subconsultas.

Assim, ao concluir a disciplina, você terá conhecimento necessário para criar um projeto de banco de dados e manipular as informações nele armazenado, podendo aplicar o conhecimento adquirido em diversos segmentos da área de computação, como programação, análise e gerente de infraestrutura.

Sumário

Capítulo 1 - Introdução a banco de Dados	5
Capítulo 2 - Entendendo os Modelos de Base de Dados.....	24
Capítulo 3 - Modelagem Entidade Relacionamento.....	43
Capítulo 4 - Exemplos de Implementação do Modelo Conceitual .	68
Capítulo 5 - Modelagem Lógica.....	89
Capítulo 6 - Modelagem Física, Normalização de Dados e Álgebra Relacional.....	113
Capítulo 7 - Introdução a SQL	135
Capítulo 8 - Operações CRUD.....	158
Capítulo 9 - Critérios de Seleção de Dados.....	178
Capítulo 10 - Consultas envolvendo mais de uma tabela	195

Christiano Cadoná*

Capítulo **1**

Introdução a banco de Dados

➔ **E**ste capítulo tem por objetivo contextualizar o aluno quanto à importância de banco de dados e os principais conceitos relacionados a esse tema.

Serão destacadas as vantagens e as desvantagens do uso de sistemas de banco de dados, além de entender o principal papel do administrador de banco de dados.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Importância da Informação no mundo moderno

O momento em que vivemos na história é definitivamente marcado pela era digital. Tudo ou quase tudo que possuímos, possui alguma funcionalidade digital. E o que mais impressiona, é que estamos tão atrelados a essa tecnologia, que não conseguimos mais nos separar dessa relação, criando uma dependência de tecnologia. Há muito que se debater, se isso é bom ou ruim para humanidade, contudo, uma coisa é certa, não vivemos mais sem comunicação, controles e funcionalidades automatizadas. Essa inserção de tecnologia trouxe uma série de benefícios à vida moderna. Muitos controles, principalmente controles operacionais, proporcionaram às organizações seu crescimento e principalmente a possibilidade de conhecer melhor seu negócio, produtos e seus clientes.

Estamos na era da informação, na qual capital social físico de uma empresa não é tão valioso quanto à qualidade das informações geradas nos sistemas de informação.

O Google e Facebook são bons exemplos de empresas muito valiosas, e, se analisadas, possuem como maior ativo os dados nelas armazenadas. Você já avaliou o quanto e quais informações somente essas duas empresas possuem relacionadas à população mundial? Somente o Brasil, possuía, no ano de 2013, mais de 47 milhões de usuários do Facebook, e o Gmail (produto da Google) possuía, em 2012, mais de 286 milhões de usuários no mundo todo. Essas empresas sabem exatamente, onde cada usuário mora, o que mais gostam, que tipo de informação recebe, quais são os amigos que

se relacionam, o local onde esteve, entre outras informações que cada usuário “autoriza” essas empresas a coletar dados no momento em que se cadastram para usufruir das funcionalidades oferecidas por estas. Nesse momento, torna-se simples imaginar o quanto dessas informações é estrategicamente importante para gestores de empresas. Sendo assim, justifica-se o valor que cada uma possui no mercado de negócios.

Agregado a isso, o avanço do uso de sistemas inteligentes (Sistemas de recomendação, sistemas de apoio à decisão, mineração de dados, OLAP etc.) como mecanismo de diferencial competitivo de uma empresa, potencializa ainda mais a necessidade de obtenção de um número cada vez maior de dados armazenados nas bases de dados.

Avaliando esse contexto, é possível entender o porquê de as empresas que mais faturam do mundo possuírem, como fator estratégico ou principal insumo de seus resultados, o uso de sua base de dados. Bancos, empresas de energia, empresas de Telecom, empresas de TI, entre outras, destacam-se das demais, e se observar, ambas necessitam de seus dados armazenados para sobreviver.

O governo brasileiro, após muitos anos, se deu conta da necessidade de informatizar suas autarquias de forma a possuir maiores controles e, por consequência, melhorar o atendimento e, principalmente, a arrecadação.

Se avaliar os contextos até aqui apresentados, é possível concluir que banco de dados está ligado diretamente com todas as ações do mundo moderno. Dessa forma, fica clara a dependência existente da tecnologia em todos os segmentos.

Não é possível imaginar o mundo em que vivemos sem os processos automatizados sejam eles aplicados a qualquer tipo de segmento. Vivemos cercados de informação e ficamos cada dia mais sedentos da disponibilidade desta.

Agora, imagine um colapso em um sistema, cujas informações são apresentadas de forma errada e ou simplesmente não são apresentas, devido a problemas no armazenamento, o quanto impactante seria essa experiência, tanto para o usuário final, quanto para a empresa e ou governo que é o detentor desses dados. Por exemplo, imagine, se por algum motivo, todos os *emails* da conta do Gmail e ou todas as contas do Facebook simplesmente sumissem. O impacto seria significativo. Agora, imagine algo muito pior como, por exemplo, erros nos registros de documentos de veículos e ou remoção de todas as informações cadastrais dos proprietários de veículos? Qual seria o tamanho do impacto? Realmente, não há como descrever o colapso causado por algum evento dessa magnitude. Falhas desse tipo, geralmente ocorrem no repositório de dados.

Tendo em vista todos os fatos apresentados, é inevitável entender como, e por que, é tão necessário o estudo dos repositórios de armazenamento desses sistemas. Entender os conceitos e melhores práticas de modelagem, armazenamento e manipulação de dados são competências indispensáveis de um profissional da área de computação.

Banco de dados

Banco de dados é uma coleção de dados inter-relacionados, representando informações sobre um domínio específico. Tendo como premissa essa definição, é possível concluir que, no momento que existir um grupo de informações que caracterizam (representam) algo, como, por exemplo, uma planilha eletrônica contendo informações estruturadas de produtos, estamos tratando de um banco de dados.

Em sua forma mais primitiva, o exemplo apresentado pode sim ser considerado como um banco de dados. É possível ainda agregar que, os dados contidos no banco de dados, devem ser armazenados de forma independente dos programas que façam uso destes, possibilitando que essas informações possam ser acessadas por múltiplas soluções computacionais. No caso de uma planilha, é possível que suas informações sejam acessadas por aplicativos diferentes do Excel, por exemplo, como uma solução desenvolvida em linguagem PHP, Java ou C#.

Sendo assim, podemos concluir que qualquer repositório de informação, que esteja organizado, representando algo, e suas informações possam ser acessadas, caracteriza a existência de um banco de dados.

Essa definição pode ser um tanto quanto genérica, pois um texto escrito em um jornal ou em um *site*, por exemplo, possui um conjunto de palavras que estão organizadas, caracterizando um banco de dados. Contudo o termo banco de dados é mais restrito e tem as seguintes particularidades (Elmasri, 2011):

- Um banco de dados representa algum aspecto do mundo real, às vezes chamado de minimundo ou universo de discurso.
- Um banco de dados é uma coleção logicamente coerente de dados com significado inerente. Uma variedade aleatória de dados não pode ser corretamente chamada de banco de dados.
- Um banco de dados é projetado, construído e populado com dados para uma finalidade específica. Ele possui um grupo definido de usuários e algumas aplicações previamente concebidas nas quais esses usuários estão interessados.

Dessa forma, somente será chamado de banco de dados, um conjunto de informações que possuir relação com algo que represente a realidade, e que possa sofrer alguma interação, como, por exemplo, ser utilizado como repositório de informações de um sistema comercial, ou conteúdo dinâmico de um *site*.

Sistema de Gerenciamento de Bancos de Dados (SGBD)

É muito comum confundir os conceitos de banco de dados e Sistema de Gerenciamento ou Gerenciador de Banco de Dados, porém são coisas muito distintas. O SGBD é um *software* ou conjunto de *software* (não o repositório de dados) com recursos específicos para facilitar a manipulação das

informações dos bancos de dados e o desenvolvimento de programas aplicativos.

"É um sistema cujo objetivo principal é gerenciar o acesso e a correta manutenção dos dados armazenados em um banco de dados."

Por exemplo, o ORACLE, Mysql e SQLServer, são sistemas gerenciadores de banco de dados, e possuem um conjunto de bancos de dados que são gerenciados por estes.

O conceito de SGBD determina que os dados do banco de dados nunca serão acessados por uma aplicação final e sim sempre serão acessados por meio do seu SGBD.

A Figura 1 apresenta um esquema que ilustra a forma típica de acesso realizada por aplicativos desenvolvidos por linguagens de programação atuais.

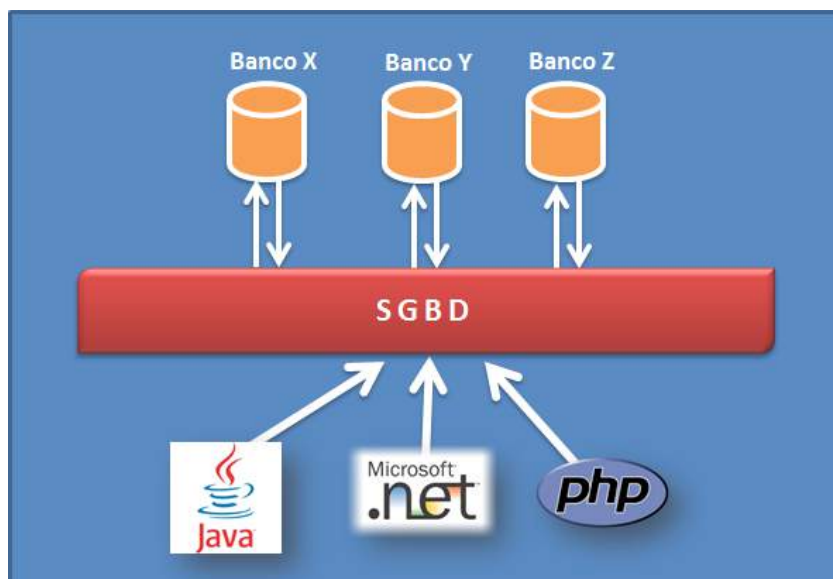


Figura 1 Esquema típico de conexão a dados utilizando SGBD

Perceba, no exemplo apresentado na Figura 1, que as linguagens de programação (Java, .NET e PHP) não realizam acesso direto ao banco de dados (Banco X, Y e Z), mas, sim, ao SGBD que, por sua vez, realiza operações com os dados de cada banco.

Sistema Bancos de Dados

O sistema de banco de dados não pode ser confundido com SGBD. Um sistema é algo muito maior, que possui um SGBD como parte do sistema. Pode ser definido como *“um sistema de manutenção de registros por computador”*. Ele é dividido em quatro componentes principais:

- Dados: representam a informação armazenada em cada banco de dados. Esses dados, dependendo do porte da empresa, podem ser armazenados fisicamente de forma centralizada (em aplicações de dispositivos móveis ou aplicações comerciais de pequeno porte) ou distribuídas (como é o caso de empresas como Google e a Receita Federal). Para quem acessa as informações, não percebe qualquer divisão de informações. Esses dados também podem ser compartilhados permitindo, por exemplo, que de forma simultânea seja possível visualizar dados cadastrados de um cliente em um terminal e, em outro terminal, o cadastro de outro cliente no mesmo repositório de informação (banco de dados).

- Hardware: um SGBD necessita de recurso de *hardware* para suportar suas funcionalidades. Geralmente, os recursos de *hardware* estão relacionados à capacidade de disco, memória principal, processador e dispositivos de entrada e saída.
- Software: são os conjuntos de arquivos existentes que realizam todo o controle sobre os dados armazenados. É nesse conjunto de aplicações que o SGBD está inserido no sistema. “O SGBD é, de longe, o componente mais importante de todo o sistema, mas não é o único”. Existem outros componentes como ferramentas de modelagem, ferramentas de monitoramento entre outras, que fazem parte do conjunto de *software* do sistema de banco de dados.
- Usuários: são personagens que interagem diretamente com os dados armazenados e ou com SGBD. Estes podem ser divididos em três grupos, sendo o primeiro o de programadores, que são responsáveis por elaborar aplicações utilizando o repositório de dados. O segundo, formado por usuários finais, que em geral são os clientes dos aplicativos desenvolvidos, e, por fim, os administradores de banco de dados (DBA), com responsabilidade direta sobre os dados e o sistema de banco de dados.

Existe uma série de objetivos de um sistema de banco de dados, contudo dois deles resumem todo o conjunto. O primeiro é o de isolar os usuários dos detalhes mais internos do banco de dados (abstração de dados). Para os usuários do banco, sejam

eles programadores, usuários finais e ou administradores, não é necessário saber qual é a rotina que realiza a gravação dos dados no disco físico, ou como é que ele recupera do banco de dados uma informação armazenada. O que importa é, quando encaminhado uma solicitação de armazenamento de um cadastro de cliente, por exemplo, o sistema realizar o procedimento, sem que ocorra inferência do usuário.

Já o segundo é o de prover independência de dados às aplicações (estrutura física de armazenamento e à estratégia de acesso). O sistema deve possibilitar que os dados possam ser armazenados de forma distribuída caso necessário, além de proporcionar mecanismos de segurança no acesso a esses dados, como o uso de permissão de acesso a um banco ou a um conjunto de tabelas.

Vantagens de um sistema de banco de dados

Além, é claro, dos objetivos que um sistema de banco de dados proporciona, existe uma série de vantagens em sua utilização:

- ➔ Rapidez na manipulação e no acesso à informação: como em geral os sistemas de banco de dados possuem uma linguagem de manipulação de registros universal, não se torna necessário criar rotinas que possibilitem a manipulação de registros, além de estes possuírem rotinas de otimização de código internas o que melhora de forma significativa a busca por informações.

- Redução do esforço humano (desenvolvimento e utilização): como para o usuário não interessa a rotina que o sistema de banco de dados utiliza para manipular as informações, não é necessário perder tempo construindo rotinas automatizadas, pois estas já existem. Dessa forma, o desenvolvedor apenas deve executar comandos de manipulação de dados, pois todos os controles já estão previamente codificados no sistema de banco de dados.
- Disponibilização da informação no tempo necessário: como já relatado, não é necessário construir rotinas de acesso, basta utilizar as já existentes, possibilitando que em pouco tempo seja possível elaborar um relatório apenas com comandos de busca a dados, por exemplo. A grande parte dos sistemas de banco de dados está configurada para apresentar o retorno de informações em tempo real.
- Controle integrado de informações distribuídas fisicamente: o sistema de banco de dados possui rotinas de integração em bases distribuídas fisicamente, possibilitando ao usuário uma única fonte de informação. A distribuição é realizada de forma automática. Imagine que, o Google, por exemplo, não possua apenas um servidor de dados, e sim muitos, e suas informações estão sim distribuídas fisicamente, inclusive entre países diferentes, entretanto para quem realiza uma pesquisa no *site*, a informação aparece para o usuário de forma única.
- Redução de redundância de informações: redundância faz referência a duplicidade de informações armazenadas. Pode ocorrer que existam aplicações distintas em que

cada uma cadastre a mesma informação de forma separada. Imagine, em uma escola, um sistema que cadastre alunos de uma chamada, e outro sistema que cadastre as parcelas de pagamento dos alunos. Ambos compartilham dados de alunos. Com o uso de sistemas de banco de dados, é possível elaborar a estrutura de forma que ambos os sistemas cadastrem em apenas um local os alunos, reduzindo a redundância de informações.

- ➔ Redução de inconsistência de informações: a inconsistência pode ser derivada da redundância de informação, pois sem o controle de redundância, conforme o exemplo citado, os dois sistemas cadastrariam informações distintas sobre alunos. Caso seja alterada uma das informações de um aluno em apenas um sistema como sua data de nascimento, o outro sistema não teria a informação atualizada. Nesse caso, dizemos que o banco está inconsistente, pois não sabemos qual das datas de nascimento de aluno é a correta. Existem outros tipos de inconsistência a inconsistência de dados, chaves e relações que serão abordadas nos próximos capítulos de forma mais detalhada. Mas, em ambas as inconsistências relacionadas, o sistema de banco de dados possui mecanismos de controle automatizados.
- ➔ Compartilhamento de dados: um dos grandes fatores dos sistemas de banco de dados terem evoluído, está associado ao fato de compartilhar um único repositório de dados com várias aplicações distintas.
- ➔ Aplicação automática de restrições de segurança: o sistema de banco de dados possibilita uma série de mecanismos de segurança ao acesso e manutenção dos da-

dos armazenados. O tipo de restrição mais comum de ser encontrado em sistema de banco de dados é o uso obrigatório de usuários e senhas de acesso à base. Porém, é possível impor, dependendo o banco de dados, restrições de acesso a tabelas e campos específicos. Isso dependerá de cada sistema de banco de dados.

- Redução de problemas de integridade: a integridade assegura que os dados armazenados estão corretos. Pode parecer estranho imaginar que é possível armazenar informações incorretas, contudo é mais comum do que se imagina. Por exemplo, existe uma série de modelagens de banco que permitem que seja armazenado um valor de salário ou idade negativa. Caso seja armazenada uma informação, em um desses campos, negativa, podemos concluir que os dados estão incorretos ou não íntegros. Um sistema de banco de dados possui regras para impor integridade em suas informações armazenadas.

Desvantagens de um Sistema de banco de dados

A escolha e implantação de um sistema de banco de dados proporciona uma série de vantagens já apresentas. Porém, existem algumas desvantagens inerentes à implantação do sistema. São elas:

- Sem dispositivos de controle adequados, a segurança pode ficar comprometida, como, por exemplo, no caso de acesso não autorizado a dados. Nunca esqueça que,

na base de dados, estão as informações mais importantes do negócio de uma empresa. É indispensável mantê-las em segurança.

- ➔ A integridade das informações pode ser comprometida se não houver mecanismos de controle quando ocorrer manipulações de dados de forma concorrente. Imagine um sistema em que duas pessoas acessam e alteram ao mesmo tempo dados de uma mesma informação como o produto “Abacaxi”. Dependendo da forma como for implementado o processo concorrente, um dos usuários sobrescreverá a informação que o outro acabou de alterar, tornando os dados não íntegros.
- ➔ A operação do sistema de banco de dados e o desenvolvimento de aplicações precisam ser feitos com muita precisão para evitar que informações não correspondam à realidade.
- ➔ A administração do sistema de banco de dados pode se tornar muito complexa em ambientes distribuídos, com grande volume de informações manipuladas por uma grande quantidade de usuários. Se não for planejada e documentada a estrutura de controle do sistema de banco de dados, a administração tende a se tornar complexa mesmo em empresas de médio porte.

Administrador de Banco de dados (DBA)

○ Administrador de Banco de dados – DBA (*Database Administrator*) possui um papel importante no sistema de banco de dados. É ele que toma as decisões estratégicas e de

normas e geralmente quem as implementa no SGBD. Segue algumas das atribuições de um DBA dentro do sistema de banco de dados:

- ➔ Definição e atualização do esquema dos bancos de dados (definem quais e como as informações serão mapeadas para o banco).
- ➔ Definição da estrutura de armazenamento e da estratégia (ou método) de acesso.
- ➔ Concessão de autorização para acesso a dados.
- ➔ Definição de controles de integridade.
- ➔ Definição de estratégias de cópia de segurança e recuperação.
- ➔ Monitoração de desempenho.
- ➔ Execução de rotinas de desempenho.
- ➔ Atualização da organização física.

Linguagem de definição de dados (DDL)

A linguagem de definição de dados DDL (*data-definition language*) é utilizada para definir a estrutura (esquema) de um banco de dados, principalmente nos níveis conceitual e de visões de usuário. A DDL define, por exemplo, quais serão os dados que são armazenados (tipo de dado: número, texto, data) e os índices de pesquisa. O conjunto de parâmetros da DDL é armazenado no que chamados de dicionário de dados. O dicionário de dados possui uma estrutura contendo todas as definições que o banco de dados possui.

Linguagem de manipulação de dados (DML)

A linguagem de manipulação de dados DML (*Data Manipulation Language*), permite aos usuários e aplicações, acessar ou manipular as informações contidas em um banco de dados. Manipular dados envolve incluir, alterar, remover e consultar dados no banco de dados.

Em geral, os bancos de dados relacionais (foco principal dessa disciplina), trabalham com a linguagem SQL (*Structured Query Language*), ou Linguagem de Consulta Estruturada. O Sistema de banco de dados interpreta comandos emitidos em SQL e realiza o processamento da operação.

É importante salientar que o SQL não é a única linguagem existente como DDL de um sistema de banco de dados. Geralmente, cada SGBD, possui comandos próprios que podem ser utilizados para operacionalizar rotinas com os dados armazenados.

Recapitulando

Como foi percebido neste capítulo, a informação armazenada nos bancos de dados é indispensável em nosso contexto de mundo moderno. Tanto usuários finais como as empresas estão dependentes das informações armazenadas nesses repositórios. Daí a importância do estudo em banco de dados.

Percebemos que banco de dados, em sua essência, nada mais é do que um repositório de dados organizados que representam algo de um domínio específico. Já os SGBDs compõem uma estrutura de aplicações que possibilitam o gerenciamento desses bancos de dados.

O sistema de banco de dados então é formado não apenas pelos dados e aplicativos armazenados e, sim, por um conjunto composto de dados, *hardware*, *software* e usuários desse sistema. Entre os usuários de banco de dados, destaca-se o DBA que é responsável pela correta definição, criação e manutenção do sistema de banco de dados.

Por fim, em um sistema de banco de dados, existem duas linguagens principais: a DDL que define o banco de dados e a DML que possibilita a interação de aplicações com os dados armazenados.

Atividades

- 1) Assinalar (V) para as assertivas Verdadeiras e (F) para as Falsas.
() Um exemplo de banco de dados é o Mysql.

- () Arquivos texto estruturados são exemplos de banco de dados.
- () Uma planilha do Excel com dados de clientes pode ser considerada como banco de dados.
- 2) Tendo como base o conteúdo apresentado no Capítulo 1, marque somente a alternativa que está errada.
- a) Sistema gerenciador de banco de dados possui um conjunto de aplicações para gerenciamento do banco de dados.
- b) DDL, é a linguagem de manipulação de registros.
- c) O sistema de banco de dados possibilita uma série de mecanismos de segurança ao acesso e manutenção dos dados armazenados.
- 3) Não é uma vantagem dos sistemas de banco de dados.
- a) Redução da inconsistência de informações.
- b) Compartilhar o mesmo banco de dados com mais de uma aplicação.
- c) Administração do banco de dados se torna simples em ambientes de distribuídos.
- 4) A partir dos estudos desenvolvidos nesse capítulo, marque (X) somente nas assertivas verdadeiras (múltipla escolha).
- a) Um DBA é responsável por executar rotinas de gerenciamento no sistema de banco de dados.

b) O dicionário de dados possui a concessão de autorização para acesso a dados.

c) Um sistema de banco de dados pode ser dividido em 4 componentes principais: Dados, Usuários, *Hardware* e *Software*.

5) É uma desvantagem do uso de sistemas de banco de dados.

a) Sem dispositivos de controle adequados, a segurança pode ficar comprometida.

b) Monitoração de desempenho.

c) A rapidez na manipulação e no acesso à informação.

Gabarito:

1) F, V, V; 2) b ; 3) c; 4) a, c; 5) a.

Christiano Cadoná*

Capítulo 2

Entendendo os Modelos de Base de Dados

➔ Este capítulo tem por objetivo apresentar os principais modelos de modelagem de banco de dados e trazer características mais específicas do modelo de dados relacional que é o foco de nosso estudo. Assim, o leitor terá uma visão das principais modelagens e um aprofundamento na modelagem relacional.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Modelo de Base de Dados

Um modelo de base de dados é a forma na qual é possível estruturar os dados. É o instrumento que permite uma representação do mundo real a partir de informações. No decorrer da história, foram definidos vários modelos de base de dados. A decisão de qual modelo é o ideal, depende do objetivo, da complexidade e do volume de dados esperados para um domínio desejado. Os modelos de dados diferenciam-se principalmente pela forma com que os dados são representados. Geralmente, possuem uma representação simples, normalmente gráfica, sendo que os mais difundidos são o modelo hierárquico, modelo em redes, modelo relacional e modelo orientado a objetos.

O modelo hierárquico foi desenvolvido na década de 1960, com o objetivo de gerenciar grandes quantidades de dados. Foi a primeira grande evolução em relação ao formato de armazenamento. No modelo hierárquico, as informações são estruturadas no formato de árvores hierárquicas, em que cada um dos nós (registros) contém uma coleção de atributos, e cada atributo contém apenas uma informação (ELSMARI e NAVATHE, 2011). Nesse modelo, cada nó pai pode possuir vários nós filhos e um nó filho somente pode possuir um único nó pai. Em banco de dados, chamamos isso de uma relação 1:N (um para muitos).

A Figura 2 demonstra um exemplo de modelagem hierárquica. Nela, percebemos que cada derivação de um nó pai é chamada de nível ou segmento. No exemplo, o “Componente C”, que está no nível 1, é pai nos nós “Montagem B” e “Mon-

tagem C", que estão no nível 2. Por sua vez, o nó "Componente C", é filho do nó "Montagem Final".

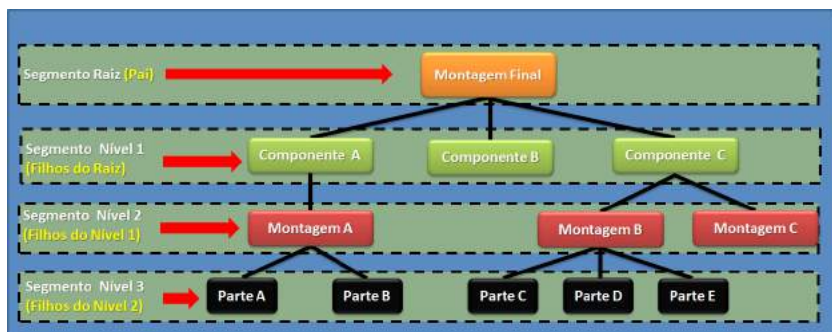


Figura 2 Modelo de dados Hierárquico (modificado de Peter Rob e Carlos Coronel, 2011)

O acesso aos dados armazenados no modelo hierárquico é feito por meio de ponteiros, ou seja, ela é iniciada a partir do topo da árvore (nó raiz) e da esquerda para a direita, até alcançar o nó que contém a informação pretendida.

O modelo hierárquico se tornou predominante na década de 1970, no entanto esse modelo possuía suas limitações como a complexidade nas consultas, a limitação das ligações entre os nós (um nó filho pode ter apenas um nó pai) e possibilidade de se trabalhar apenas com dados primitivos. Essas limitações desencadearam análises de alternativas para solução dos problemas por parte de profissionais de banco de dados, o que resultou em modelos alternativos, sendo o mais promissor o Modelo de Rede.

O modelo de rede surgiu como uma alternativa aos problemas apontados pelo modelo hierárquico. Sua principal mu-

dança foi a eliminação do conceito de hierarquia. Com isso, o modelo de redes permitiu que um nó, possuísse várias associações (N:N – muitos para muitos) caso necessário. Outra modificação é a implementação do conceito de conjunto. Um conjunto representa um relacionamento entre o seu proprietário e um membro do diagrama.

O exemplo apresentado por Peter Rob e Carlos Coronel (2011), representado na Figura 3, ilustra uma representação de uma organização de vendas comum. Nesse modelo, CLIENTE, REPCOMERCIAL, FATURA, FAT_LINHA, PRODUTO e PAGAMENTO representam os tipos de registro. Nesse exemplo, a FATURA é de propriedade tanto do REPCOMERCIAL, como do CLIENTE. Os conjuntos são identificados no próprio diagrama, como é o caso do conjunto de vendas, formado pelos tipos CLIENTE e FATURA.



Figura 3 Modelo de Rede (modificado de Peter Rob e Carlos Coronel, 2011)

Ao contrário do modelo hierárquico, em que qualquer acesso a dados passa pela raiz, no modelo em redes, o acesso aos dados pode ser realizado a partir de qualquer nó da rede. De forma prática, com esse modelo, é possível acessar os dados de pagamento sem que seja necessário acessar os dados de cliente, o que, no modelo hierárquico, era impossível (ELSMARI e NAVATHE, 2011).

O modelo relacional surgiu como alternativa do modelo hierárquico e de redes. Utilizada uma série de conceitos destes, porém, diferencia-se por possuir um aumento de independência de dados, além de prover um conjunto de funções apoiadas em álgebra relacional tanto para o armazenamento quanto para a recuperação de dados. O modelo relacional foi concebido pelo matemático britânico Edgar Frank Codd, enquanto pesquisador da IBM nos anos 70, como uma nova maneira de representação de dados a partir de uma visão relacional dos dados, o que permite a descrição dos mesmos de uma forma natural, sem a necessidade de estruturas adicionais, além de prover uma maior independência dos dados em relação aos programas. Sua estrutura simples de ser interpretada, e a forma de recuperação de dados, agradou tanto que desencadeou uma adaptação desse modelo para o modelo entidade relacionamento, o qual praticamente domina a grande parte das aplicações que atualmente conhecemos.

Sua estrutura é formada por tabelas, e relações entre colunas (campos) dessas tabelas. Cada linha com conteúdo é chamada de registro ou tupla e representa de forma única uma informação. Da mesma forma que o modelo de rede, o acesso

às informações no modelo relacional pode ser realizado por qualquer tabela, diferentemente do modelo hierárquico.

A Figura 4 apresenta um exemplo de conteúdo armazenado em uma estrutura relacional contendo uma tabela de fornecedor e uma tabela de produtos. Observe como o conceito de tabela se torna mais simples de ser assimilada, como também a implementação do conceito de relacionamento.

Fornecedor		
codFornecedor	NomeFornecedor	FoneFornecedor
50	Central	52-3333-4444
51	Cia Brasil	52-5555-6666
52	Comércio de alimentos	52-7777-8888

Produtos			
codProduto	NomeProduto	ValorProduto	codFornecedor
2	Pera	1,99	50
3	Abacaxi	2,99	52
4	Banana	1,99	50
5	Uva	3,45	50
6	Abacate	0,79	52

Figura 4 Modelo de relacional

Nesse exemplo, o produto “Uva”, possui como fornecedor o fornecedor de nome “Central”. Também é possível fazer outro tipo de leitura, como, por exemplo, dizer que o fornecedor “Comércio de alimentos”, possui 2 produtos associados a ele, o “Abacaxi” e o “Abacate”. Observe que isso ocorre tendo em vista que existe uma informação “codFornecedor” que gera

uma relação entre as duas tabelas. Sabendo que o modelo Entidade Relacionamento é muito utilizado no contexto atual pelas empresas desenvolvedoras e grandes corporações, no próximo tópico serão abordadas outras características significativas desse modelo.

Por fim, o modelo orientado a objetos veio como alternativa ao uso do modelo relacional. A finalidade de existir um banco de dados orientado a objetos, é implementar uma série de benefícios que a programação orientação a objetos possui em relação a programação estruturada, além de possibilitar o armazenamento e recuperação de dados da mesma forma como ocorre a construção de um objeto nessas linguagens, sem que ocorra o processo de mapeamento objeto relacional. Nesse modelo, é possível, por exemplo, lidar com construtores de tipo que permitem especificar objetos mais complexos contendo tuplas e arrays, encapsulamento de operações e mecanismos de herança e polimorfismo. Mesmo com todas essas características, os modelos de dados orientados ao objeto são implementados fisicamente por meio de tabelas, ou seja, da mesma forma que no modelo relacional, tendo a semântica da aplicação modelada e representada a partir de objetos, enquanto sua implementação física é feita de forma relacional (GALANTE *et al.*, 2007).

A Figura 5 apresenta um exemplo de uma modelagem orientada a objetos, contendo as classes “cliente” e “conta”. Perceba que o diagrama utilizado para modelagem é o mesmo que é utilizado para representar classes na UML (*Unified Modeling Language*). O que ocorre em geral é que, na modelagem, não se especifica os métodos no diagrama.

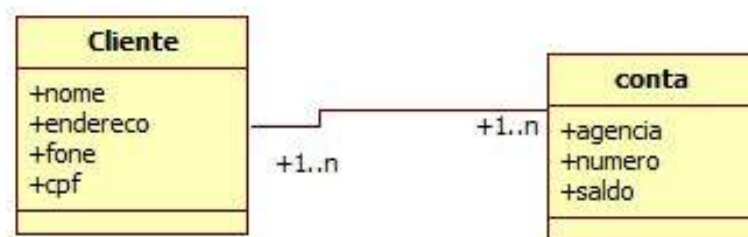


Figura 5 Modelo de relacional

Por mais que o modelo orientado a objetos tenha sido constituído posterior ao modelo relacional, pesquisas apontam que o modelo relacional será utilizado por muito tempo, tendo em vista a confiança e autonomia que este apresentou durante o decorrer dos tempos.

Outro fator que deve ser considerado é que, na prática, observamos que o paradigma de programação orientado a objetos é utilizado de forma maciça no desenvolvimento de aplicações, sejam elas para qualquer fim. Porém, em sua maioria, essas aplicações armazenam os dados em banco de dados relacionais e não em banco de dados orientados a objeto. A etapa de armazenamento e busca de dados dentro de uma aplicação que utiliza banco de dados relacional e é desenvolvida com lingual de programação orientada a objetos, chama-se “mapeamento objeto relacional”. Essa etapa então é responsável pela transformação dos dados armazenados em estrutura relacional para objetos dentro da linguagem de programação.

Modelo Relacional

Como já mencionado, o modelo relacional é um modelo lógico de dados, baseado em registros, que provê abstração de dados nos níveis conceituais e de visões do usuário. Nesse modelo, tabelas representam dados e relacionamentos entre dados. A grande maioria dos SGBD que atualmente conhecemos e são consagrados implementam esse modelo como estrutura, por exemplo, Oracle, SqlServer, Postgress, Mysql, Firebird, entre outros.

As principais características do modelo relacional são:

- Estrutura tabular: os dados são representados na forma de tabelas, nas quais cada linha representa um registro (ou tupla), e as colunas representam os atributos ou domínios.
- Álgebra Relacional: a manipulação dessas tabelas é feita por meio de operadores que permitem qualquer tipo de acesso a uma tabela ou a um conjunto de tabelas (seleção, união, junção – join etc.). Nos próximos capítulos, falaremos especificamente sobre álgebra relacional.
- Linguagem DDL: definindo, o esquema de dados do banco de dados.
- Linguagem DML: possibilidade de acesso e manipulação dos dados armazenados em alto nível por meio de uma linguagem de manipulação.
- Esquema Conceitual: representa o ponto de vista global da empresa sobre a organização dos dados. Nesse esquema, entra em cena o papel do administrador de dados, que identifica padrão de entidades, atributos e

relacionamentos, determinando restrições de integridade e segurança ao modelo de dados.

- Esquema Físico: o esquema físico está associado ao armazenamento de dados, acesso, local de armazenamento, tipos de dados, entre outras especificações.
- Ação direta do DBA: no modelo relacional, o DBA possui um papel indispensável. Geralmente, é ele que realiza o mapeamento do modelo lógico para o físico, define as estruturas de acesso, índices, modo de transmissão de dados entre outras atividades.

Como já estudados, existe uma série de sistemas gerenciadores de bancos de dados disponíveis no mercado. Em sua maioria, são modelados utilizando o conceito de modelagem relacional. Como forma de garantir que um sistema de banco de dados possua características de um banco de dados relacional, este deve possuir as seguintes características:

- O sistema de banco de dados deve gerenciar o banco de Dados exclusivamente a partir de capacidades relacionais.
- Toda a informação é apresentada sempre em nível lógico por valores em tabelas.
- Todo o dado tem a garantia de ser logicamente acessível. Se o dado existir e estiver armazenado, deve ser possível recuperá-lo.
- Os valores nulos são representados por ausência de valores. Caracteres especiais ou zero não representa ausência de valor. Geralmente, nesse caso, existe um símbolo

chamado “NULL” que pode ser utilizado para representar ausência de valores no respectivo campo do registro.

- Deve haver pelo menos uma linguagem que permita a definição dos dados, definição de visão, manipulação dos dados, restrições de integridade, autorização de acesso, limites de transação. Não existe uma regra, mas a grande parte dos SGBDs existentes possuem SQL como linguagem padrão de manipulação.
- Inserção, atualização e remoção de alto nível. Assim, não interessa como o SGBD irá manipular os dados de forma física (como ele fará para arquivar ou alterar um dado no banco). Em um banco de dados relacional, apenas devemos informar um comando de alteração e este deverá implementar internamente a funcionalidade.
- Independência de dados físicos onde, os programas devem permanecer inalterados quando ocorre mudança nos métodos de acesso ou nas representações de memória.
- Independência de dados lógicos em que programas devem permanecer inalterados quando ocorre mudança no banco de dados.
- Independência de integridade definida na linguagem DML e DDL.
- Linguagem de acesso não pode ferir princípios de integridade.

O modelo relacional é composto por tabelas ou relações, chaves e restrições de integridade. A seguir, serão apresentados de forma mais detalhada as partes do modelo relacional.

Tabelas ou Relações

Uma tabela é um conjunto não ordenado de registros (linhas ou tuplas). Cada linha é composta por uma série de valores de campo (valor de atributo), identificados, cada um, por nome de coluna (nome de atributo – domínio). A Figura 6 apresenta um exemplo de uma tabela com suas devidas partes.

O diagrama mostra uma tabela com o cabeçalho 'Produto' dividido em quatro colunas: 'Código', 'Nome', 'Valor' e 'Fornecedor'. Há cinco linhas de dados. As anotações incluem: 'Coluna - Atributo (domínio)' apontando para o cabeçalho; 'Nome da Coluna' apontando para o texto 'Fornecedor'; 'Linha - Tupla - Registro' apontando para a terceira linha; e 'Valor de Campo (valor de atributo)' apontando para o valor '0,79' na última linha.

Produto			
Código	Nome	Valor	Fornecedor
72	Pera	1,99	Central
3	Abacaxi	2,99	Comércio de alimentos
41	Banana	1,99	Central
25	Uva	3,45	Central
18	Abacate	0,79	Comércio de alimentos

Figura 6 Tabela no modelo relacional

As tabelas e relações possuem como principais características:

- Cardinalidade de uma tabela: número de linhas que faz relação com outra entidade. A cardinalidade somente irá existir se houver relações entre tabelas. Como o exemplo da Figura 5 apresenta apenas uma tabela, não é possível representar cardinalidade entre tabelas. A cardinalidade será melhor detalhada no próximo capítulo.
- Grau de uma tabela: número de colunas existente em uma tabela. No caso do exemplo da Figura 5, o grau

da tabela é 4, pois a tabela é formada por 4 colunas (código, nome, valor e fornecedor).

- ➔ Linhas de uma tabela não estão ordenadas: As informações de uma tabela não são armazenadas de forma ordenada. Perceba o exemplo da Figura 5, que não existe ordem por nenhuma coluna. Caso seja necessário listar um conjunto de dados de forma ordenada, o SGBD relacional poderá disponibilizar algum comando.
- ➔ Não é possível referenciar linhas de uma tabela por posição: da mesma forma que o conceito de ordenação, o banco relacional não possui acesso pelo número da linha armazenado. O SGBD poderá disponibilizar algum artifício (em geral comando) que realize uma consulta dessa forma.
- ➔ Valores de campo de uma tabela são atômicos: ou seja, não será possível armazenar mais do que um tipo de informação em cada campo. Um campo de valor de um registro específico, somente conterá um único valor.
- ➔ Dados podem ser recuperados por quaisquer critérios envolvendo os campos de uma ou mais linhas. A linguagem DDL, deve possibilitar que seja possível acessar qualquer campo contendo qualquer informação armazenada.

Chaves

São atributos utilizados para identificação de registros e ou relações entre entidades. Elas são classificadas em chaves primárias, candidatas ou chaves alternativas e chaves estrangeiras.

- ➔ Uma chave primária é composta por uma ou mais colunas de uma tabela cujos valores distinguem uma linha

das demais dentro de uma tabela. Dessa forma, existem algumas propriedades indispensáveis para atender essa premissa, como o valor contido em seu campo nunca se repetir, e este não permitir que sejam armazenados valores nulos. Uma chave primária pode ser simples ou composta. Como definição, **toda entidade deve possuir uma chave primária**. A Figura 7 apresenta um exemplo em que estão sendo definidas as chaves primárias das tabelas de produto e dependente. Na tabela de produto, a chave primária simples está sendo definida no atributo “código” que, por sua vez, nunca poderá se repetir. Na tabela de dependente, ocorre a definição de uma chave primária composta pelos atributos “Cod_emp” e “Num_dep”.

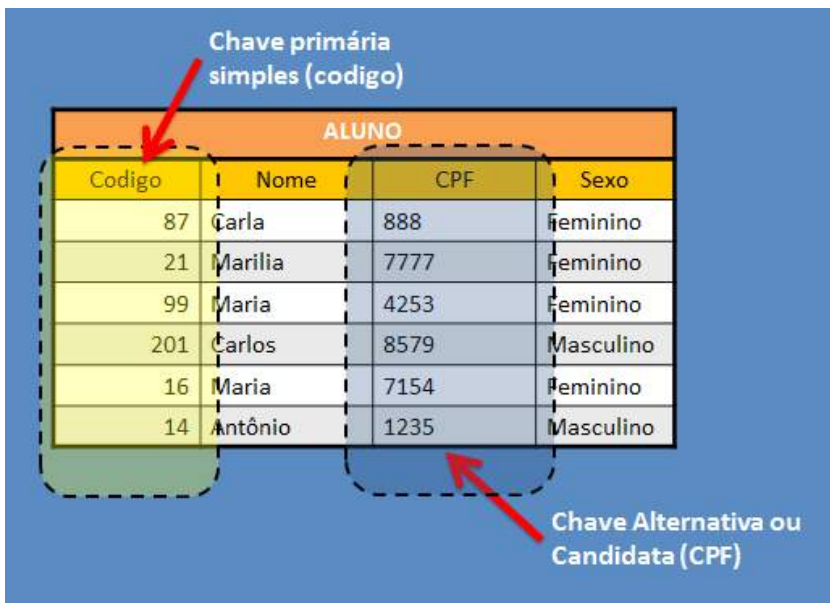
O diagrama mostra duas tabelas sobrepostas. A tabela 'Produto' está no topo e a 'Dependente' está abaixo dela. A tabela 'Produto' tem uma chave primária simples no campo 'Código', indicada por uma seta vermelha e um retângulo tracejado. A tabela 'Dependente' tem uma chave primária composta nos campos 'Cod_emp' e 'Num_dep', indicada por uma seta vermelha e um retângulo tracejado.

Produto			
Código	Nome	Valor	Fornecedor
72	Pera	1 99	Central
3	Abacaxi		
41	Banana		
25	Uva		
18	Abacate		

Dependente			
Cod_emp	Num_dep	Nome_dep	Tipo
77	01	Ana	Filho(a)
26	01	Carolina	Filho(a)
77	02	Beatriz	Esposo(a)
79	01	Camila	Filho(a)
79	02	Andreza	Esposo(a)
30	01	Fabiana	Esposo(a)

Figura 7 Exemplo de chaves primárias simples e compostas

- ☞ Chamamos de chave alternativa ou candidata colunas ou uma combinação de colunas que podem servir para distinguir uma linha das demais. De certa forma, toda chave primária é uma chave alternativa, porém não as definimos como tal, pois foram definidas como chaves primárias. Nesse caso, não existe necessidade que exista uma chave alternativa em uma entidade, se a chave primária já foi definida. O exemplo apresentado na Figura 8 ilustra a entidade aluno, contendo como chave primária “código” e como chave alternativa o campo “CPF”. Isso se deve pelo fato que o campo CPF representa de forma única um registro. Por exemplo, o CPF “8579” pertence ao aluno “Carlos” e somente ao aluno “Carlos”.



Codigo	Nome	CPF	Sexo
87	Carla	888	Feminino
21	Marília	7777	Feminino
99	Maria	4253	Feminino
201	Carlos	8579	Masculino
16	Maria	7154	Feminino
14	Antônio	1235	Masculino

Figura 8 Exemplo de chaves Alternativa ou Candidata (CPF)

- A Chave Estrangeira é uma coluna ou uma combinação de colunas, cujos valores aparecem necessariamente na chave primária de outra tabela ou da mesma tabela. É um mecanismo que permite a implementação de relacionamentos em um banco de dados relacional. No exemplo apresentado na Figura 9, a chave estrangeira existente na tabela de produto chamada, “Código_Fornecedor”, contém em seu conteúdo os valores existentes na chave primária “código” da tabela de fornecedor. Para garantir o conceito de integridade em um banco de dados relacional, necessariamente o valor contido nos atributos de chave estrangeira devem ter elementos contidos em chaves primárias associadas e ou serem nulos (NULL). Isso dependerá da forma como for modelada a integridade das informações.

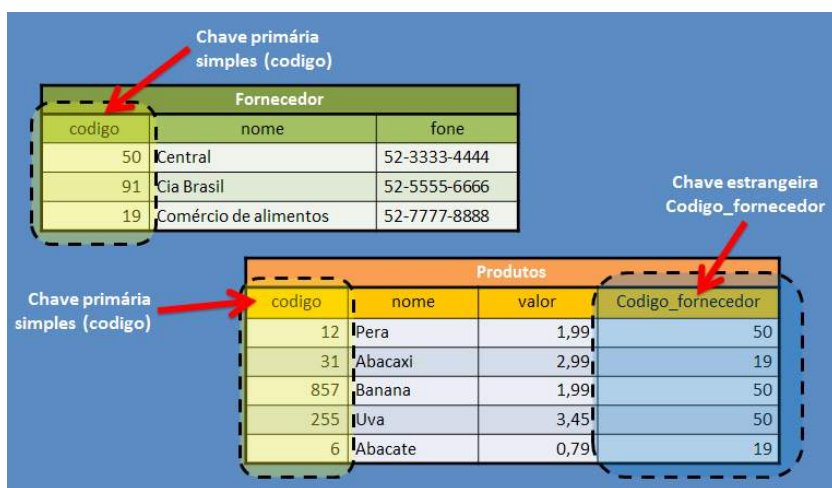


Figura 9 Exemplo de chaves estrangeiras (Codigo_fornecedor)

Restrições de Integridade

Restrições de Integridade são regras de consistência de dados que são garantidas pelo próprio SGBD. Um dos objetivos primordiais de um SGBD é a integridade de dados. Dizer que os dados de uma base de dados estão íntegros significa dizer que eles refletem corretamente a realidade e são consistentes entre si. A restrição de integridade é classificada em:

- **Integridade de Vazio:** campos de uma coluna podem ou não ser vazios (se a coluna é obrigatória ou opcional). Não esqueça que os campos que compõem a chave primária devem ser sempre diferentes de vazio.
- **Integridade Referencial:** os valores dos campos que aparecem em uma chave estrangeira devem aparecer na chave primária da tabela referenciada ou devem ter valor igual à NULL.
- **Integridade de Chave:** os valores da chave primária e alternativa devem ser únicos.
- **Integridade de Domínio:** o valor de um campo deve obedecer à definição de valores admitidos para a coluna. Por exemplo, se for definido que no campo de idade, somente deverá ser armazenado números inteiros maiores ou iguais a zero, não deverá ser possível, por exemplo, que seja armazenada uma idade negativa.

Recapitulando

Após o término deste capítulo, é possível ter uma visão dos principais modelos de dados (hierárquico, redes, relacional e objeto), assim como entender algumas características de cada modelo.

Constatamos que o modelo relacional definido na década de 1970, ainda é muito utilizado e terá um bom espaço por ser um modelo consagrado por sua confiança e autonomia no decorrer do tempo.

Por fim, foram aprofundadas algumas características e a estrutura do modelo relacional em que foram ilustrados exemplos dos conceitos de tabelas, chaves e integridade.

Atividades

Tendo como base as afirmações abaixo relacionadas, marque (V) para as assertivas Verdadeiras e (F) para as Falsas.

() O modelo hierárquico é uma evolução do modelo em rede, pois resolveu o problema de ordem de acesso a dados. No modelo Hierárquico, os dados são acessados de sua raiz para os níveis mais inferiores, utilizando o conceito de árvore.

() O modelo orientado a objetos utiliza conceitos do paradigma de orientação a objeto, porém em sua maioria, fisicamente armazena em estrutura tabular.

() O modelo relacional é o modelo mais utilizado, tendo em vista que ele aplica o conceito de todos os demais modelos.

() No modelo relacional, os dados são armazenados em formato de tabelas, as quais são constituídas por colunas e tuplas. Essa tabela deve possuir uma coluna que identifique uma tupla de forma única. Para isso, existe o conceito de chave primária.

() Toda entidade deve possuir pelo menos uma chave estrangeira.

Gabarito:

F - V - F - V - F

Christiano Cadoná*

Capítulo 3

Modelagem Entidade Relacionamento

➔ Neste capítulo, serão apresentados os conceitos relevantes quanto a modelagem Entidade Relacionamento (E-R), além de apresentar o processo de implementação dessa modelagem, apresentando as suas partes e componentes. Dessa forma, o aluno terá conhecimento necessário para modelar situações práticas de contextos de armazenamentos, além de entender mais a fundo as características do modelo relacional já apresentado anteriormente.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Modelo Entidade Relacionamento

O modelo Entidade Relacionamento (E-R) foi proposto por Peter Chen em 1976 e teve como base a teoria relacional criada por Codd (1970). Ele utiliza praticamente todos os conceitos do modelo relacional, além de propor uma visão mais conceitual do mundo real. O principal objetivo era levar aos projetistas ou analistas a possibilidade de ter uma única visão de uma realidade sem redundância e bem resumida. O modelo ER propõe que o mundo real é formado por um conjunto de objetos que são denominados de entidades e relacionamentos entre essas entidades. O modelo ER é um dos modelos com maior capacidade semântica (refere-se à tentativa de representar o significado dos dados). Seu autor enfatizava que:

“O mundo está cheio de objetos (entidades), que possuem características próprias (atributos) e que se relacionam entre si (relacionamentos).” (Peter Chen)

É importante destacar que, em geral, modelamos utilizando o modelo Entidade Relacionamento, porém criamos o banco e manipulamos as informações utilizando banco de dados relacional (Oracle, SqlServer, Mysql,...). Essa distinção deve ficar clara, para continuar o entendimento sobre o modelo ER. O que ocorre é que em certa etapa do processo de criação de um banco de dados, convertemos o modelo de forma que este seja representado conforme o modelo relacional de cada banco de dados.

Processo de Implementação de um Banco de dados ER

O processo de elaboração de Banco de dados foi elaborado como forma de definir etapas as quais podem ser seguidas que resultarão na criação de forma física de toda estrutura de armazenamento. A Figura 10 apresenta um esquema que demonstra o processo de banco de dados, no qual podem ser destacadas as etapas de análise de requisitos, modelagem conceitual, mapeamento lógico e projeto físico.

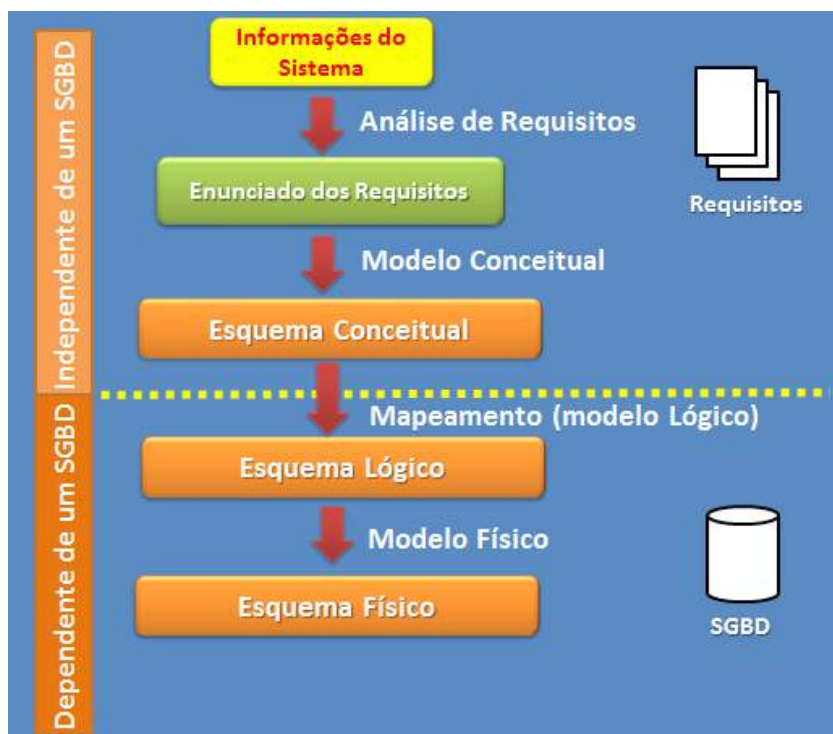


Figura 10 Exemplo de processo de implementação de banco ER

O processo inicia a etapa de análise de requisitos que faz parte do processo de desenvolvimento de um sistema. A etapa de análise consiste em identificar os requisitos e funcionalidades que um sistema deva possuir. São selecionadas todas as características e particularidades que estão relacionadas à modelagem de banco.

Somente após conhecer as especificações necessárias para o desenvolvimento do sistema, será possível avançar de forma a não implicar futuras modificações na estrutura de armazenamento e ou alterações que possam ser implementadas de forma simples com menos risco. Como já apresentado na disciplina de engenharia de *software*, um dos custos que mais impacta o desenvolvimento de um sistema, está atrelado em sua possível manutenção. Geralmente, essas alterações estão associadas a modificações da estrutura de armazenamento de dados, e isso pode resultar em possível inconsistência de informação, o que, segundo conceito de modelagem ER, viola regras de modelagem.

A próxima etapa é a modelagem conceitual que relata a descrição do conteúdo da base de dados, e não das estruturas de dados usadas para representá-lo. Seu enfoque é a compreensão e descrição da “realidade”, das propriedades relevantes da informação além de desconsiderar todo e qualquer aspecto de implementação. Essa é, sem dúvida, a maior contribuição de Peter Chen para comunidade de banco de dados, pois a partir dessa modelagem, descrevemos a realidade sem nos preocuparmos em qual banco de dados iremos implantar, possibilitando uma visão macro da estrutura de armazenamento. Como resultado da implementação do modelo conceitual, te-

remos um digrama que representa o esquema conceitual da estrutura de armazenamento. Este capítulo tratará em mais detalhes dessa modelagem no próximo tópico.

Assim que o modelo conceitual estiver definido, este é convertido para o modelo lógico o qual descreve as estruturas que representam a base de dados. Sua modelagem dependerá do tipo de SGBD alvo (relacional, rede, hierárquico, OO). Nessa etapa, possui como principal característica a definição no diagrama, das chaves primárias e estrangeiras, o que define de forma física as relações entre as entidades. Perceba que, a partir dessa etapa, é necessário conhecer o SGDB que se pretende trabalhar. Como resultado dessa etapa, será gerado um diagrama que representa a estrutura lógica do banco de dados.

Por fim, e não menos importante, o modelo lógico será transformado no modelo físico, em que cada entidade, e seus atributos serão representados de forma que o banco de dados possa ser construído. Nessa modelagem, ocorre a escolha ou refinamento das estruturas de armazenamento e métodos de acesso. O modelo físico leva em consideração o produto de SGBD específico. Como resultado, será gerada uma descrição do esquema na DDL do SGBD. É comum nessa etapa que no modelo físico já esteja implementado a partir de uma DML, códigos que executem a criação do banco de dados, como linhas de comando em SQL que criam tabelas e estabeleçam relações entre estas.

Esquema Conceitual

Como já apresentado, o esquema conceitual tem como objetivo desenvolver um diagrama que represente a realidade, sem se preocupar em qual SGBD será implementado. Seu enfoque está na compreensão e descrição da informação, na seleção das propriedades relevantes da informação e na descrição das restrições sobre esses dados. Esse esquema possui como característica o diálogo com usuários finais do sistema, de forma que possa ser modificado para atender os requisitos, sem se preocupar com aspectos de implementação.

O seu formalismo gráfico possui um padrão que possibilita a sua interpretação de forma universal. Assim, fica mais fácil de aprender, compreender e associar com um requisito de sistema.

Seu formalismo gráfico possui como primitivas básicas:

- ➡ Entidade
- ➡ Relacionamento
- ➡ Atributo
- ➡ Cardinalidade

Entidade

Uma entidade descreve características comuns a todos os elementos que deseja representar. Ela representa algo concreto como, por exemplo, informações de uma pessoa, dados de um veículo ou pode ser algo abstrato, como departamento de

uma empresa ou informações de um curso. As entidades são os primeiros elementos a ser identificados, principalmente pelo fato de estarem explícitos ou evidentes. Uma entidade possui um nome (que define a entidade), atributo (características da entidade) e uma identificação (atributo que identifica de forma única uma informação armazenada. Em geral, esse campo é a chave primária da entidade). O Exemplo 1 apresenta uma especificação de uma entidade chamada produto.

Exemplo 1:

Entidade Produto: Descrevem características comuns a todos os produtos que possam existir em um sistema de informação em um dado momento, como Código, nome do produto, seu estoque e valor de venda.

- **Nome:** Produto.
- **Atributos:** Código, nome do produto, seu estoque e valor de venda.
- **Identificação:** Código.

Sua representação gráfica é um retângulo contendo em seu centro o nome da entidade que está representando. Por convenção, o nome de uma entidade deve ficar sempre no singular. A Figura 11 apresenta um exemplo de duas entidades “Produto” e “Fabricante”.



Figura 11 Exemplo de definição de entidades

Atributo

Característica, propriedade ou qualidade de uma ocorrência de entidade ou de relacionamento. Atributos são usados na definição de conjuntos de entidades e de conjuntos de relacionamento. Atributos também são conhecidos como propriedades, ou campos de uma entidade. No Exemplo 1, já apresentado, os atributos da entidade produto são: código, nome, estoque e valor de venda.

A Figura 12 apresenta sua representação gráfica de acordo com o modelo proposto por Peter Chen. Nesse modelo, cada entidade possui um conjunto de atributos. Como premissa, toda entidade necessita ter pelo menos um atributo para existir. Observe que o atributo que identifica um elemento específico possui como representação um círculo fechado, enquanto os demais atributos são representados por círculos abertos.

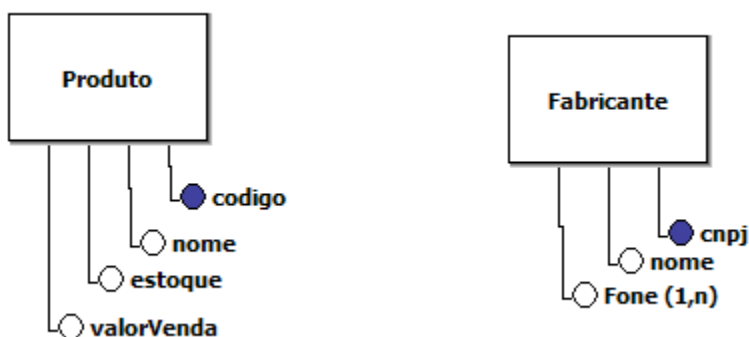


Figura 12 Exemplo de atributos em entidades

Um atributo pode ser simples (monovalorado) ou repetitivo (multivalorado). No caso de ser simples, este possui um único valor associado a cada ocorrência de uma entidade ou relacionamento. Por exemplo, o atributo “valor Venda”, da entidade “Produto” da Figura 12, possui apenas um único valor de venda para cada produto armazenado. Já se este for repetitivo ou multivalorado, um atributo poderá possuir mais do que uma informação armazenada no mesmo atributo, como é o caso do atributo “Fone” da entidade “Fabricante”, que poderá armazenar mais do que um telefone para um mesmo fabricante.

Outra característica importante de um atributo é se este é obrigatório ou opcional. Dizemos que o atributo é obrigatório, quanto é necessário que exista uma informação armazenada no atributo para que seja possível armazenar um registro. Por exemplo, se for definido que o campo “nome” da entidade “Produto” é obrigatório, só será possível armazenar um registro na tabela de “Produto”, se for informado algo no campo “nome”. Caso ocorra uma tentativa de armazenamento sem que seja informada uma informação nesse campo, o SGBD não permitirá o armazenamento e retornará um erro. Porém se o campo for definido como opcional, o SGBD permitirá o armazenamento mesmo que não informada nada no atributo.

Um valor de um atributo corresponde ao conteúdo existente em um registro em um dado momento (dado armazenado na base de dado). Por exemplo, o conteúdo armazenado no campo “nome” da tabela “produto” do de um registro específico é “Abacaxi”.

Cada atributo possui um domínio, que caracteriza o que pode ser armazenado na coluna específica, como exemplo, tipo de informação (uma data, uma hora, um número inteiro, um número real, um texto,...). Nesse domínio, é possível especificar uma regra de armazenamento como permitir somente números inteiros no intervalo de 10 a 200.

Relacionamento

Segundo Peter Chen, 1976, um relacionamento “*é uma associação entre 2 ou mais entidades*”. Dessa forma, quanto existir uma associação entre entidades distintas, dizemos que existe um relacionamento entre essas entidades. A Figura 13 apresenta um exemplo da implementação de um relacionamento. As entidades Professor e Curso possuem um conjunto de elementos que estão associados na relação chamada “**Atuação Docente**”.

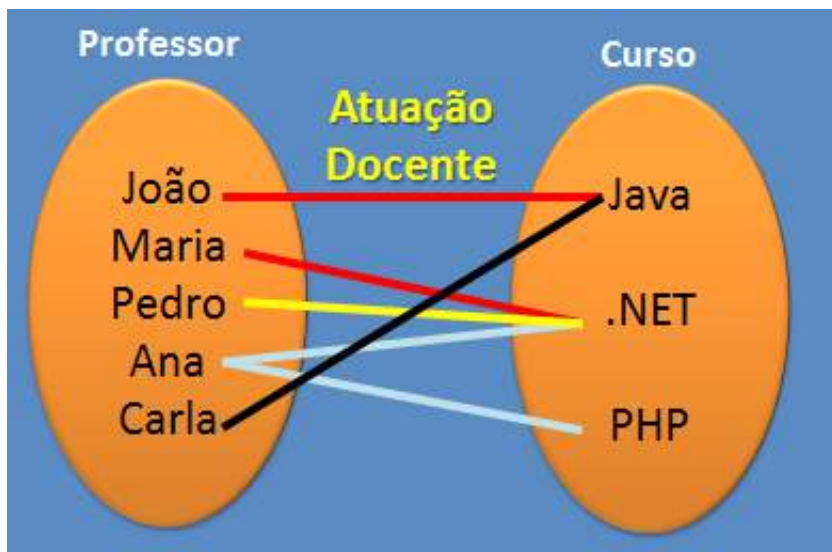


Figura 13 Exemplo de conjuntos de dados relacionados

Cada ocorrência da entidade participante desempenha um papel no relacionamento. Na Figura 13, é possível atribuir um papel para o professor e um papel para o curso. Exemplo:

- ➔ “Ana **ministrou** o curso de PHP”, ou “o curso de PHP **foi ministrado** pela Ana”.
- ➔ “Carla **é professora** de...” ou “Java **possui como professor...**”.

Dessa forma, um conjunto de relacionamentos possui:

- ➔ Nome: **ATUAÇÃO-DOCENTE**
- ➔ Conjuntos de entidades participantes: **PROFESSOR, CURSO**

- Para cada conjunto de entidades (ex.: **PROFESSOR**)
 - cardinalidade (máxima): **N**
 - cardinalidade (mínima): **0**
 - papel desempenhado (opcional): **ministra**
 - atributos (opcional): **ano/sem**

Existe uma série de notações para modelagem de dados, contudo utilizaremos, nesse documento, o modelo proposto por Peter Chen. Assim, a Figura 14 demonstra o modelo ER gerado de uma relação entre as entidades Professor e Curso, denominada “Atuacao_Docente”.

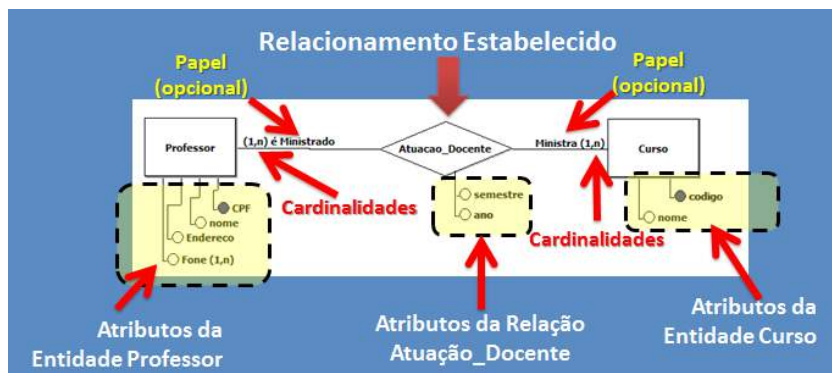


Figura 14 Exemplo de relacionamento

A entidade professor possui como atributos o CPF, nome, endereço e fone, sendo o atributo CPF selecionado como o que identifica os registros ou atributo que será definido como de chave primária. A entidade Curso possui dois atributos, o código e nome. Da mesma forma que o atributo CPF, o atri-

buto código está definido como atributo de chave primária. O relacionamento *Atuação_Docente* também possui 2 atributos, o semestre e o ano. Observe que, nesse modelo, não existe necessidade de especificar uma chave primária em um relacionamento, como é o caso dos atributos CPF e Código das demais entidades. Também é importante destacar que não existe obrigatoriedade da existência de atributos no relacionamento como é o caso da obrigatoriedade da existência desse tipo de atributo em entidades. Os atributos semestre e ano foram criados para ilustrar a possibilidade da existência de atributos em uma relação.

O exemplo, também aborda o conceito dos papéis (“ministra” e “é ministrado”) os quais são exercidos por qualquer uma das entidades. Observe que, nesse caso, os papéis são colocados no lado oposto da entidade que o origina. Exemplo: o papel do professor é “ministra” e está ao lado do curso, possibilitando que seja possível realizar uma leitura da relação da seguinte forma. “Um professor ministra Curso”. Já o papel de curso (“é ministrado”) está ao lado da entidade professor. Nesse caso, lê-se “um curso é ministrado por professor”. Os papéis servem para realizar uma leitura da relação existente entre as entidades e não são obrigados a ser apresentados no diagrama. Contudo, sugere-se que sejam colocados para melhor compreensão dos conceitos envolvidos, principalmente quando o processo de conhecimento de banco de dados está na etapa inicial.

Outra característica importante no modelo apresentado é a existência das cardinalidades de uma relação, representadas no Exemplo 14 pelos valores (1,n).

Cardinalidade

As cardinalidade identificam dentro de uma relação quantas ocorrências de uma entidade participam no mínimo e no máximo do relacionamento. São divididas em:

- ➔ **Cardinalidade Mínima**: define se o relacionamento entre duas entidades é obrigatório ou não. Para isso, o diagrama define os valores 0 (zero) para não obrigatório e 1 como obrigatório.
- ➔ **Cardinalidade Máxima**: define a quantidade máxima de ocorrências da entidade que pode participar do relacionamento. Nesse caso, a cardinalidade máxima sempre será maior do que zero. A cardinalidade máxima define as restrições de integridade, adicionando semântica (significado) ao diagrama. O modelo proposto por Peter Chen define 1 como uma ocorrência máxima ou N para representar mais de uma ocorrência de relação.

A Figura 15 demonstra de forma prática a aplicação do relacionamento emprego, gerado entre a relação professor e escola. A cardinalidade mínima sempre é informada primeiro, seguida da cardinalidade máxima. No exemplo (0,n), definimos o elemento 0 (zero) como cardinalidade mínima e n como cardinalidade máxima.

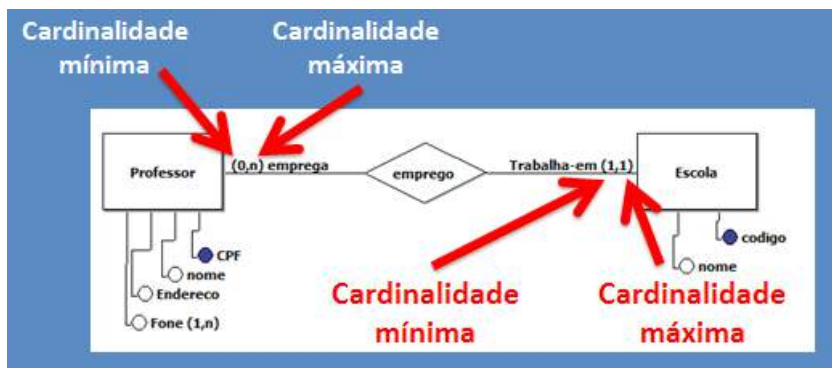


Figura 15 Exemplo de Cardinalidade Mínima e Máxima

Nesse exemplo, um professor trabalha em pelo menos uma universidade (cardinalidade mínima) e no máximo em uma escola (cardinalidade máxima), já uma escola emprega nenhum (cardinalidade mínima) ou vários professores (cardinalidade máxima).

Tipos de Cardinalidade de acordo com o Relacionamento

Como já apresentado, as cardinalidades definem a quantidade de ocorrências de uma entidade que estão presentes na relação com outra entidade. Também constatamos que existem dois tipos de cardinalidade, a mínima e a máxima. A grande parte dos modelos de dados apenas leva em consideração a cardinalidade máxima, pois essa identifica o tipo de cardinalidade de acordo com o relacionamento. Existem apenas 3 tipos:

- 1:1 (um para um)
- 1:N (um para muitos)
- N:N (muitos para muitos)

A Figura 15, possui uma relação 1:N, já a Figura 14 possui uma relação N:N. Observe que, para definir esses tipos de cardinalidade ou também conhecido como tipos de relação, foram utilizadas apenas as cardinalidades máximas de cada um dos exemplos apresentados nas figuras 14 e 15.

Cardinalidade Um para Um (1:1)

Dada duas entidades definidas como entidades A e B, dizemos que existe uma cardinalidade 1:1, quando uma entidade em A está associada no máximo a uma entidade em B e uma entidade em B está associada no máximo a uma entidade em A. A Figura 16 apresenta o conjunto de elementos de duas entidades A e B distintas com cardinalidade 1:1. Observe que cada elemento de cada uma das entidades possui apenas uma relação com um elemento da outra entidade.

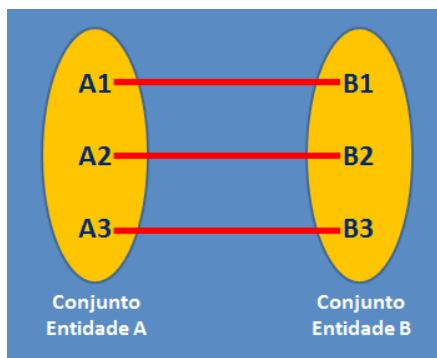


Figura 16 Exemplo um conjunto de dados de Cardinalidade 1:1

A Figura 17 apresenta um exemplo de uma relação 1:1 entre as entidades Homem e Mulher, chamada casamento monogâmico.



Figura 17 Exemplo de Cardinalidade 1:1 – Casamento Monogâmico

Nesse exemplo, cada homem pode participar (desempenhar o papel de “esposo”) em no máximo uma ocorrência de CASAMENTO, e cada mulher pode participar (desempenhar o papel de “esposa”) em no máximo uma ocorrência de CASAMENTO.

Cardinalidade Um para Muitos (1:N)

Dada duas entidades definidas como entidades A e B, dizemos que existe uma cardinalidade 1:N, quando uma entidade em A está associada a qualquer número de entidades em B, enquanto uma entidade em B está associada no máximo a uma entidade em A. A Figura 18 apresenta um exemplo de dois conjunto de dados A e B com relação 1:N

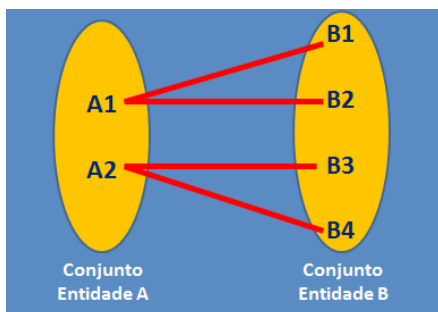


Figura 18 Exemplo de um conjunto de dados de Cardinalidade 1:N

Os diagramas apresentados na Figura 19 descrevem exemplos do uso de relações 1:N. Observe que o exemplo “a” da Figura 19 é praticamente idêntica ao apresentado na Figura 17, porém, ao alterar a cardinalidade, o sentido do diagrama modificou-se completamente.

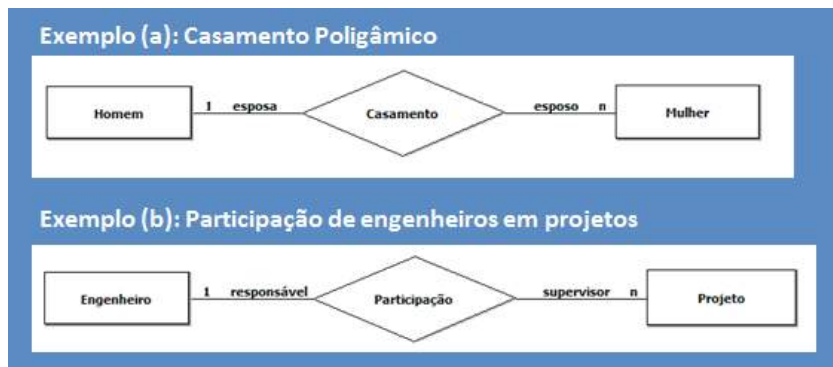


Figura 19 Exemplo de Cardinalidade 1:N

No exemplo “a”, casamento Poligâmico, cada homem pode participar (desempenhar o papel de “esposo”) em várias

ocorrências de CASAMENTO, já cada mulher pode participar (desempenhar o papel de “esposa”) em no máximo uma ocorrência de CASAMENTO.

No exemplo “b”, que representa a participação de engenheiros em um projeto, cada engenheiro pode participar em uma ou mais ocorrências de projeto, já cada projeto pode possuir como responsável no máximo uma ocorrência de PARTICIPAÇÃO de um engenheiro.

Cardinalidade Muitos para Muitos (N:N)

Dada duas entidades definidas como entidades A e B, dizemos que existe uma cardinalidade N:N, quando uma entidade em A está associada a qualquer número de entidades em B, e uma entidade em B está associada a qualquer número de entidades em A. A Figura 20 apresenta um exemplo de dois conjunto de dados A e B, em que seus elementos podem se relacionar com quaisquer elementos da outra entidade e vice-versa.

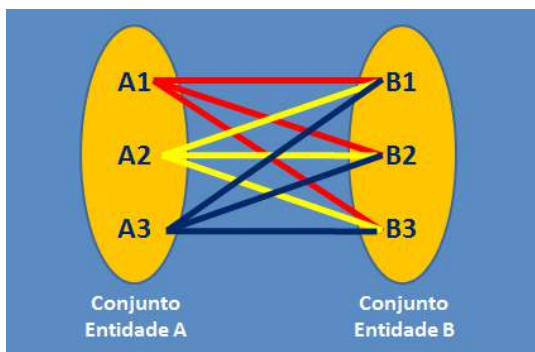


Figura 20 Exemplo de um conjunto de dados de Cardinalidade N:N

Os diagramas apresentados na Figura 21 descrevem de forma prática como ocorre a definição de cardinalidade N:N.

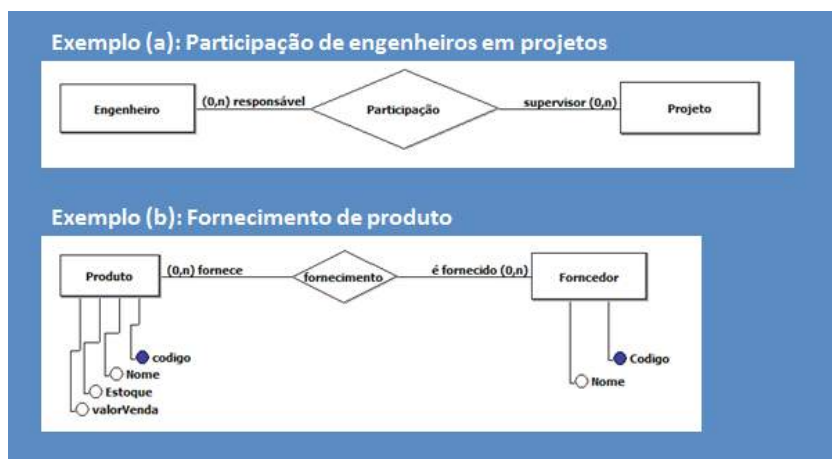


Figura 21 Exemplo de Cardinalidade N:N

No exemplo “a” da Figura 21, que representa a participação de engenheiros em um projeto, cada engenheiro pode participar em uma ou mais ocorrências de projeto, e cada projeto também pode possuir como responsável no máximo uma ou várias ocorrências de PARTICIPAÇÃO de um engenheiro. No exemplo “b”, de fornecimento de produto, um produto pode ser fornecido por vários fornecedores, já um fornecedor pode fornecer mais do que um produto.

É importante ressaltar que foram apresentados exemplos que possuem as mesmas entidades e relacionamentos, entretanto o que mudou foi apenas o tipo de relação entre eles. Pode parecer sem muita importância essa alteração, porém essa definição implicará elaborar o banco de dados com estruturas

completamente diferentes de acordo com a definição do tipo. A definição correta implicará também elaborar uma interface correta para cada tipo de relação. A escola equivocada trará um grande retrabalho para a equipe de desenvolvimento. Dessa forma, fica o questionamento: “como saber o correto tipo de relação a ser aplicado?”. A resposta é simples, ela está na etapa de levantamento de requisitos que foi elaborada junto ao cliente, antes de dar início ao desenvolvimento do projeto de banco. Uma análise mal elaborada causará a aplicação de forma errada do tipo de relação e, com isso, o desenvolvimento equivocado. Certamente, ao apresentar parte do projeto ao cliente, será identificado o erro, porém o processo já gerou custos, e atrasos em sua entrega. Sendo assim, sempre analise cada relação de forma a implementar o tipo de relação que atenda as necessidades do cliente. A mesma relação poderá possuir tipos diferentes de acordo com a implementação que se deseja realizar. A correta implantação dependerá de cada caso. Isso pode ser evidenciado no exemplo “b” da Figura 19 e no exemplo “a” da Figura 21, em que o que muda é apenas a cardinalidade da relação. Ambas estão corretas, mas sua aplicação dependerá de cada cliente. Nesse caso, é necessário saber se o cliente realmente deseja que um projeto possua como responsável apenas um engenheiro ou mais de um. Somente de posse dessa informação será possível aplicar o tipo correto que atenderá a necessidade do cliente final.

Generalização/Especialização

Muitas vezes, quando analisamos um conjunto de entidades, percebemos conteúdos iguais, que geralmente implicam criar atributos iguais em entidades diferentes. Por exemplo, imagine aluno e professor, ambos possuem nome, CPF, data de nascimento entre outros campos semelhantes. Normalmente, criamos entidades distintas para ambos e replicamos os atributos, adicionando outras características específicas para cada entidade (carteira de trabalho, por exemplo, na entidade professor e nome dos pais na entidade aluno). Em alguns casos, existe alternativa de representação. Ao invés de criar várias entidades, cria-se uma única entidade agrupando atributos comuns e outras entidades contendo somente os atributos não compartilhados, em uma relação de generalização ou especialização. Esse é o conceito de generalização e especialização aplicado a banco de dados. Por exemplo, vamos imaginar que seja necessário armazenar informações cadastrais de motoristas e engenheiros, ambos empregados de uma empresa. Dependendo o cenário, ambos podem possuir nome, endereço, bairro e CPF. Porém, um motorista deve possuir o número de sua carteira de habilitação e categoria. Já um engenheiro deve possuir o seu registro junto ao CREA (Conselho de Regulamentação de Engenheiros e Arquitetos).

Nesse caso, ao invés de criarmos duas entidades contendo informações específicas de cada entidade (engenheiro e motorista) é possível utilizar o conceito de generalização/especialização. A Figura 22 apresenta como seria a implementação desse conceito.

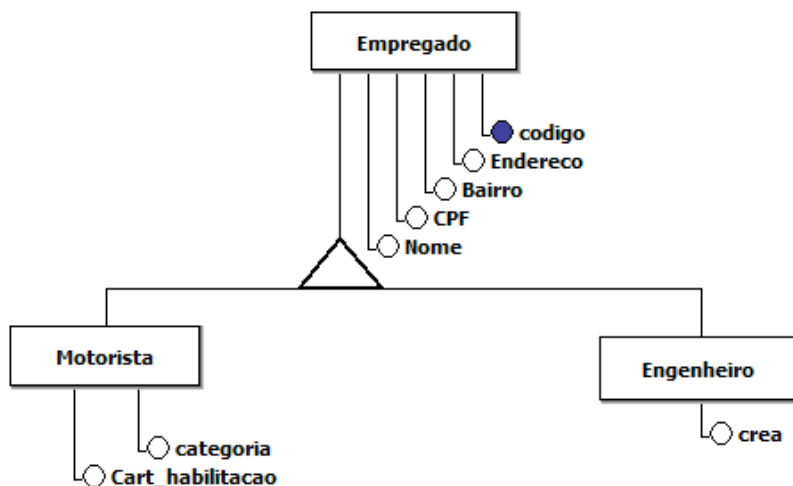


Figura 22 Exemplo de Generalização/Especialização

Na solução proposta, foi criada uma nova entidade chamada de empregado, que contém todas as informações comuns entre os engenheiros e motoristas. Em seguida, foram definidas as entidades restantes contendo apenas informações peculiares a estas. Pela representação, foi adicionado um triângulo representando a ligação entre a entidade emprego e as entidades motorista e engenheiro, permitindo realizar uma leitura dizendo que além dos campos categoria e car_habilitacao existentes em motorista, este também possui como campos todos os campos da entidade empregado. O mesmo ocorre com a entidade engenheiro.

Recapitulando

Ao concluir este capítulo, o leitor percebeu que o processo de implementação de um banco de dados ER, possui etapas importantes que iniciam no levantamento de requisitos e é concluído na implementação de forma física do banco de dados.

Também foi estudada a estrutura proposta pelo modelo de Peter Chen, em que foram apresentadas as definições de entidades, atributos, relações entre entidades e cardinalidades.

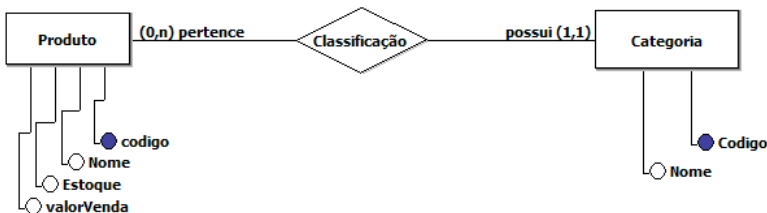
Segundo estudos, a definição correta do tipo de relação (1:1, 1:N e N:N) é indispensável para atingir os objetivos do projeto elaborado de desenvolvimento da aplicação, minimizando a ocorrência de aumento de custos e atraso no desenvolvimento.

Atividades

Tendo como base as afirmações abaixo relacionadas, marque (V) para as assertivas Verdadeiras e (F) para as Falsas.

- 1 - () Na etapa do esquema conceitual que compreende a modelagem conceitual do processo de implementação de um banco de dados, não é necessário saber qual o banco de dados que será implementado, pois a visão da informação não implica qual banco será implementado.
- 2 - () Um atributo, como característica, pode conter dados Monovalorados ou dados repetitivos além de permitir a obrigatoriedade ou não da existência de informação em seu conteúdo.

- 3 - () Os tipos de Cardinalidade estão associados à cardinalidade mínima de uma relação.
- 4 - () No modelo conceitual, um relacionamento possui:
- Nome
 - Entidades que participam do relacionamento
 - Cardinalidades mínimas e absolutas
 - Papel desempenhado por qualquer entidade
- 5 - Marque a alternativa que identifica o tipo de relação utilizada no exemplo que segue:



- a) 1:1 d) 0:N
- b) 1:N e) 0:1
- c) N:N

Gabarito:

1) V; 2) V; 3) F (o correto é associação pela cardinalidade máxima); 4) F (o erro está em que não existe cardinalidade e absoluta); 5) B.

Exemplos de Implementação do Modelo Conceitual

➡ Neste capítulo, serão retomados alguns conceitos do modelo conceitual e apresentado algumas novas especificidades desse modelo, por meio de exemplos de construção desse diagrama. Dessa forma, o aluno estará diante de exemplos práticos que subsidiarão a elaboração desses diagramas contribuindo na correta elaboração de um projeto de banco de dados.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Modelo Conceitual

Como já abordado no capítulo anterior, o modelo conceitual, apresenta uma visão de armazenamento, independente do SGBD que será implementado. A partir desse modelo, é possível identificar todas as entidades e relações existentes entre essas entidades, além dos atributos necessários de cada entidade de um determinado domínio.

Partindo desse pressuposto, é importante retomar que o modelo dependerá exclusivamente do que a etapa de levantamento de requisitos trouxe de informação para a elaboração do diagrama. Dessa forma, os exemplos que seguem serão fruto exclusivo dos domínios que estão sendo apresentados em cada exemplo. É importante que seja destacado que o diagrama poderia ser diferente dependendo da descrição do exemplo.

O diagrama conceitual é a base para o desenvolvimento do modelo lógico e, por consequência, do modelo físico. Assim, sua correta elaboração implicará a correta elaboração dos demais modelos, já que, pelo processo, um depende do outro.

Exemplo passo a passo de criação do Diagrama Conceitual

Tendo em vista, a necessidade da elaboração correta dos diagramas conceituais, esse tópico abordará um exemplo de uma descrição que terá como consequência a criação de um diagrama conceitual. Não existe uma ordem exata de desenvol-

vimento do diagrama, a única premissa é que exista um domínio estabelecido contendo as necessidades a serem implementadas. Contudo, será adotado neste documento um processo de elaboração, no qual, inicialmente, a partir de informações do domínio, serão identificadas as entidades, seus atributos, a relação entre as entidades, determinando o tipo de relação (1:1, 1:N ou N:N) e, por fim, determinar os atributos identificadores. É importante que seja especificado que não está sendo considerado e aplicado nesses primeiros exemplos o conceito de normalização de dados, tema esse que será abordado nos próximos capítulos. Dessa forma, caso exista algum aluno que já conheça esse conceito, perceberá essa característica.

Assim, o Exemplo 2 apresenta uma descrição de um domínio hipotético, no qual servirá de base para elaboração do diagrama conceitual.

Exemplo 2: “Petshop”

Uma Petshop deseja obter alguns controles sobre os animais vendidos em seu estabelecimento e atendimentos realizados a estes. Segundo levantamento, é necessário armazenar o nome do animal, raça, data de nascimento e peso. Uma venda possui um cliente, um único animal, uma data e um valor que faz referência ao valor de venda do animal. Um animal somente pertencerá a uma única venda. É necessário armazenar alguns dados sobre o cliente como seu nome, CPF, endereço completo e telefone. Esse cliente poderá efetuar várias compras de animais. Um animal possui uma raça, já uma raça é de vários animais. A clínica também gostaria de registrar os atendimentos realizados para cada animal. Cada atendimento possui um animal, um veterinário, uma data de atendimento e um código de atendimento. Um veterinário poderá atender nenhum ou vários animais. Um animal poderá ser atendido várias vezes, porém um atendimento possui apenas um animal. Um veterinário possui nome, registro no órgão responsável e fone de contato. Um veterinário deverá conter um nome, um código de registro e um telefone de contato.

Como sugestão, vamos inicialmente identificar as entidades que estão presentes na descrição. Uma entidade possui características específicas sobre algo. Observe que as informações sobre os dados das entidades estão misturadas ao texto, por isso, o correto seria realizar mais de uma leitura. Ao terminar a leitura, é possível identificar em princípio 5 entidades: Animal, Cliente, Veterinário, Atendimento e Venda, pois, em cada uma dessas entidades, o texto relata informações para serem armazenadas. Nesse ponto, já é possível desenhar as entidades e identificar os atributos que cada uma das entidades faz parte. A Figura 23 apresenta um esboço da elaboração dessa etapa. Na realização da primeira leitura do texto, cria-se uma falsa

impressão de que existem outros atributos em algumas entidades, como, por exemplo, no trecho “... Uma venda possui um cliente, um único animal, uma data, e um valor que faz referência ao valor venda do animal...”. Nesse caso, a tendência é que o cliente e o animal sejam atributos da entidade Venda, porém tanto o animal e o cliente são entidades, pois, no texto, ambos possuem suas características estabelecidas. Na verdade, o que será criado é uma relação entre essas entidades, que refletirá no mesmo efeito de forma prática. Sendo assim, não se preocupe, inicialmente, em colocar esses elementos como atributos e, ao perceber que o domínio descreve outras informações, gere uma entidade. Lembre-se de que futuramente será gerada uma relação.

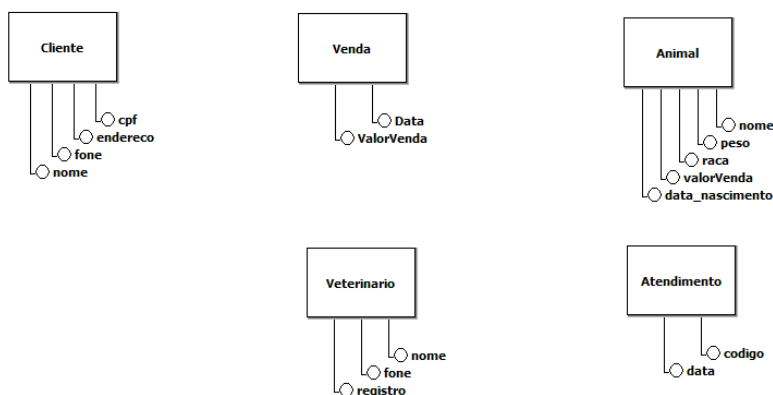


Figura 23 Exemplo parcial 1 – Petshop

O próximo passo é a definição das relações existentes entre cada uma das entidades. Como já abordado, atributos que inicialmente faziam parte da entidade e no decorrer do texto

se transformaram em outra entidade, proporcionarão uma relação entre as entidades que originalmente faziam parte com a entidade criada. É o caso dos atributos cliente e animal, que davam a impressão de ser atributos da entidade venda. Nesse caso, será criada uma relação entre as entidades cliente e venda e outra entre as entidades animal e venda. O mesmo ocorre na entidade atendimento, em que o texto informava que o atendimento possui animal e veterinário e ambos se transformaram em entidades. Nesse caso, deve haver duas novas relações, sendo a primeira entre as entidades animal e a entidade atendimento e a outra entre as entidades “veterinário e atendimento”. A Figura 24 apresenta uma nova versão do diagrama, agora com os respectivos relacionamentos.

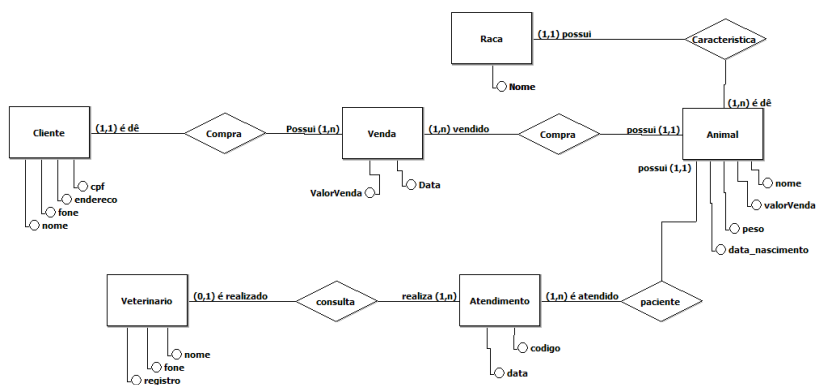


Figura 24 Exemplo parcial 2 – Petshop

Outro ponto a ser destacado no diagrama da Figura 24 é a criação da entidade Raça, que na Figura 23 era atributo da entidade animal. Isso se deu pelo fato de que o texto induz a criação da entidade. Segundo o texto “... Um animal possui

uma raça, já uma raça é de vários animais...". Perceba que o texto relata um tipo de relação 1:N. Nesse caso, devemos criar uma nova entidade e determinar um relação ligando a entidade que originou (nesse caso, a entidade animal) e a nova entidade (nesse caso, a entidade Raça).

Os demais tipos de relação também foram determinados de acordo com o texto. Segundo o exemplo, temos 5 relações com os respectivos tipos de relação:

- **Relação compra** entre as entidades Cliente e Venda: a relação ficou definida como 1:N, pois o texto informa "... Uma venda possui um cliente...", o que determina que a cardinalidade máxima do lado do cliente será 1, e "... Esse cliente poderá efetuar várias compras de animais...", que determina a cardinalidade N ao lado da venda. A Figura 25 ilustra a definição do tipo de relação 1:N, representando por meio do texto extraído as cardinalidades máximas da relação.

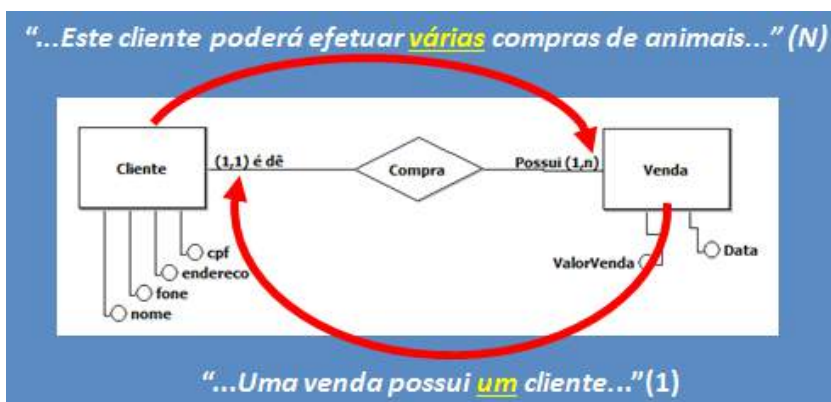


Figura 25 Relação compra entre entidades cliente e venda

- **Relação compra** entre as entidades Animal e Venda: a relação ficou definida como 1:1, pois o texto informa “... Uma venda possui... um único animal...”, o que determina que a cardinalidade máxima do lado do animal será 1, e “... Um animal somente pertencerá a uma única venda...”, que determina a cardinalidade 1 ao lado da venda. A Figura 26 descreve, por meio do texto, como foi definida a relação 1:1.

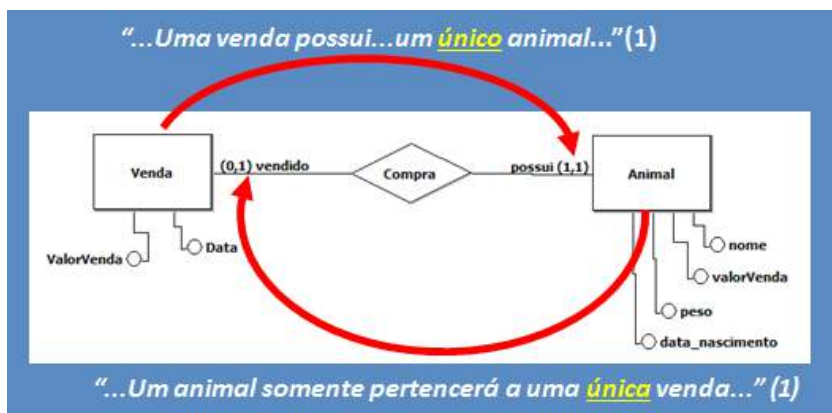


Figura 26 Relação 1:1 entre as entidades Venda e Animal

- **Relação Característica** entre as entidades Animal e Raça: como já mencionado anteriormente, o texto induziu a criação de uma nova entidade, pois definiu uma relação 1:N. Sendo assim, foi criada uma nova entidade e definida uma relação que aqui foi denominada de característica. A Figura 27 apresenta a definição do tipo de relação 1:N, juntamente com o trecho do texto que originou a relação.

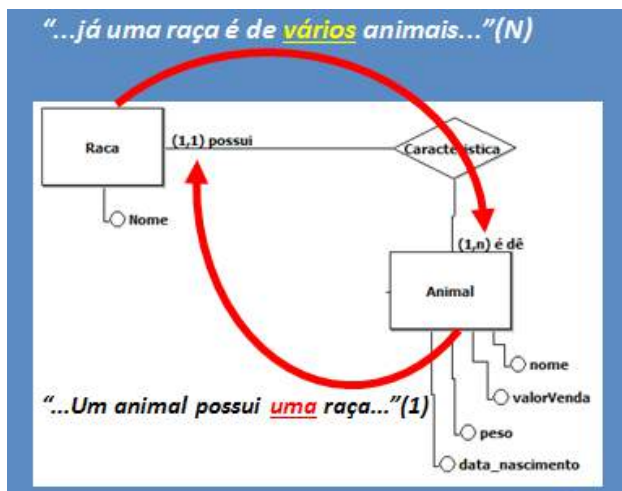


Figura 27 Relação Característica (1:N)

➔ **Relação Consulta** entre as entidades "Veterinário e Atendimento": como é possível observar na Figura 28, a relação denominada consulta possui um tipo de relação 1:N, em que um veterinário realiza N atendimentos e um atendimento é realizado por 1 veterinário. Da mesma forma que os exemplos anteriores, essa definição estava especificada no texto.



Figura 28 Relação consulta (1:N)

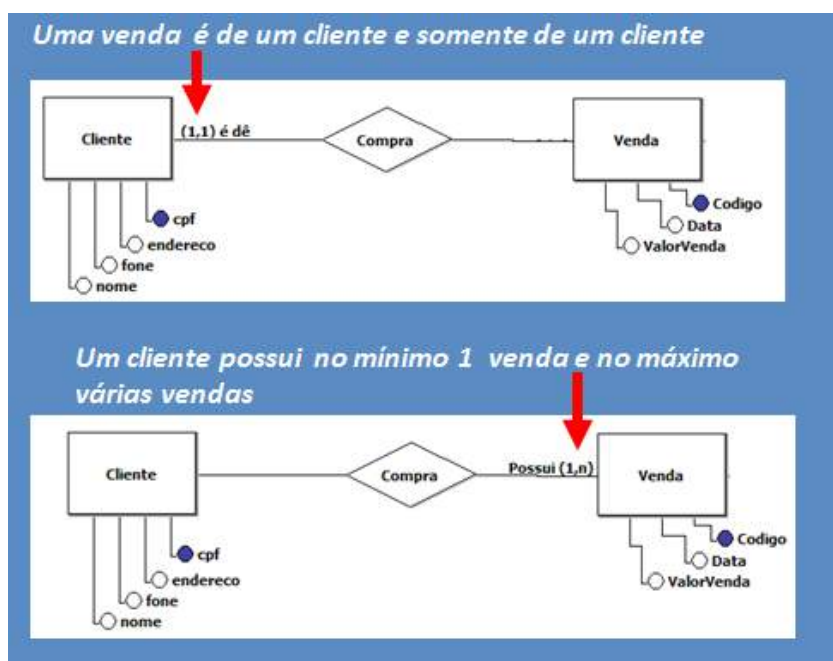
- **Relação Paciente** entre as entidades Animal e Atendimento: da mesma forma que os demais tipos de relação, o tipo determinado para a relação paciente teve origem no texto apresentado no Exemplo 2. Segundo o mesmo, a relação paciente possui o tipo 1:N, em que um animal pode ser atendido em vários (N) atendimentos e um atendimento possui apenas um animal. A Figura 29 ilustra, de acordo com o trecho do Exemplo 2, a definição do tipo 1:N



Figura 29 Relação Paciente (1:N)

A Figura 24 também apresentou além dos tipos de relação às cardinalidades máximas e mínimas definidas. A cardinalidade máxima está explícita no texto, tanto que foram criados os tipos de relação. Já a cardinalidade mínima até aparece em alguns trechos do texto, mas não o suficiente para elaboração completa do diagrama apresentado na Figura 24. Essa característica não é exclusividade somente no nosso Exemplo 2. Em sua maioria, as especificações determinam somente as relações e não a obrigatoriedade de todas essas relações. Isso se deve pelo fato de que, quando se realiza o levantamento de requisitos, não se desenvolve um diagrama conceitual, até porque isso somente será realizado posteriormente ao levantamento de dados. Nesses casos, a solução é contar com a expertise ou a lógica do analista de banco de dados, que deverá sempre validar seu diagrama junto ao cliente em caso de dúvida.

O exemplo apresentado na Figura 29.a demonstra aplicação das duas cardinalidades na relação de compra e a forma de como deve ser lida a relação compra. Segundo o texto, uma venda possui um cliente. Se possui um cliente, entende-se que é obrigado que exista um cliente para que a venda exista, e, nesse caso, a cardinalidade mínima foi definida como 1. O texto relata que o cliente possui várias compras, mas não relata nada se este pode não possuir compra alguma. Nesse caso, foi optado em imaginar que um cliente deva possuir pelo menos uma venda. Porém, observe que, nesse caso, é necessário consultar o nosso cliente final para ter certeza dessa informação.



29.a Definição de cardinalidades mínimas

A última etapa é a definição dos atributos identificadores que futuramente serão determinados como chaves primárias. Devemos selecionar um atributo de cada entidade como atributo identificador. Caso não seja possível, pois o conteúdo do atributo não garanta integridade de chave, devemos selecionar mais de um atributo, criando uma chave composta, e ou criar um novo atributo que gere a identificação. O diagrama apresentado na Figura 30 está finalizado e contém em cada uma das entidades um atributo que o identifique.

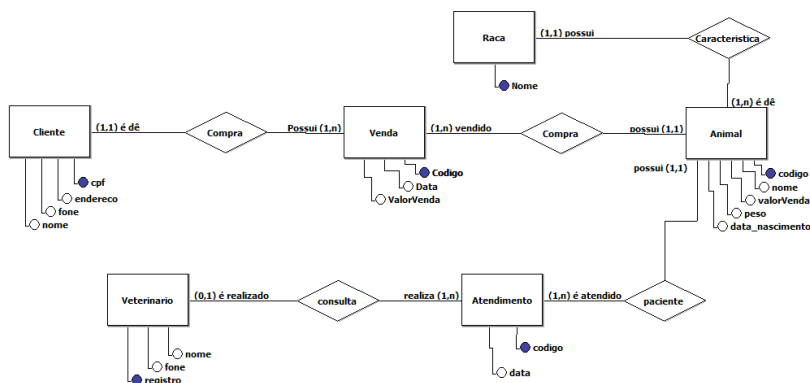


Figura 30 Exemplo final do Diagrama Conceitual da Petshop

Observe que foi criado um atributo novo nas entidades Venda, Animal e Atendimento chamado de código, o qual foi utilizado como atributo identificador da entidade. Geralmente, a solução quando não existe um identificador entre os atributos é a criação de um código. Constate também que a entidade raça só possui um atributo denominado nome, e este foi utilizado como atributo identificador, pois, pela lógica, não será cadastrada raças com o mesmo nome. Alguns analistas,

nesse caso, preferem criar um novo atributo (código) e colocá-lo como atributo identificador.

O Exemplo 3 apresenta outra descrição como forma de exemplificar a criação do modelo conceitual. O domínio do Exemplo 3 é um hotel que necessita controlar seu processo de hospedagem.

Exemplo 3: “Hotel”

Um hotel deseja informatizar seu processo de hospedagem. Sabe-se que é necessário armazenar informações de seus clientes, hospedagens por ele realizadas, e produtos consumidos na hospedagem. O cliente possui código, nome, endereço, cidade, estado, CPF, RG e data de nascimento. Uma hospedagem possui um cliente, já um cliente pode se hospedar nenhuma ou várias vezes. Uma hospedagem possui também uma data de entrada e uma data de saída. Uma hospedagem possui um quarto associado a ela; já um quarto pode estar relacionado a várias hospedagens. Um quarto é de uma categoria, já uma categoria, pode estar relacionada com vários quartos. Um quarto possui nome, andar, categoria e número de camas.

Tendo como base o Exemplo 3, podemos seguir o mesmo processo realizado no Exemplo 2, no qual, inicialmente, identificamos as entidades. Em seguida, os atributos e relações e, por fim, definir quais serão os atributos identificadores. A Figura 31 apresenta o diagrama conceitual gerado na descrição apresentada no Exemplo 3.

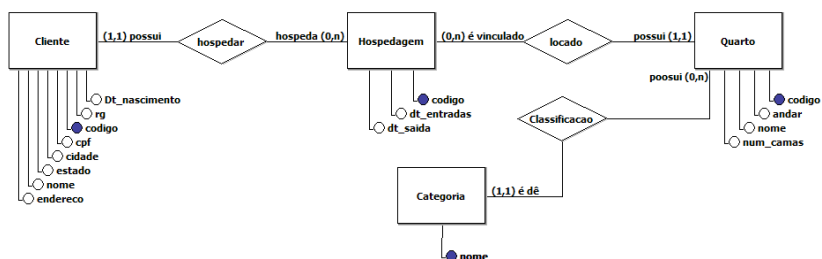


Figura 31 Diagrama Conceitual parcial do Hotel

Aparentemente, o diagrama possui todas as etapas definidas, entidade, atributos, relacionamento, cardinalidades e atributos identificadores. Em experiências já constatadas com alunos, ao apresentar o texto do Exemplo 3, estes, em sua maioria, desenvolvem um diagrama muito semelhante ao da Figura 31. Ocorre que, se prestarmos atenção, faltou um item importantíssimo no diagrama que passou despercebido. A segunda frase do Exemplo 3 relata “... Sabe-se que é necessário armazenar informações de seus clientes, hospedagens por eles realizadas, e produtos consumidos na hospedagem...”. Como o restante do texto não fala nada sobre produtos consumidos na hospedagem, a tendência é desenvolver um diagrama idêntico ao diagrama da Figura 31. Alguns até identificam essa informação e adicionam um atributo à entidade hospedagem chamado produtos. Estaria até correto, porém a única informação que temos de “produtos consumidos” está no plural, o que leva a dizer que uma hospedagem possui mais de um produto. Da mesma forma que no atributo Raça da entidade Animal, descrito no Exemplo 2, a solução correta seria criar uma nova entidade, relacionando-a com a entidade hospedagem.

A Figura 32 apresenta uma forma de solução do Exemplo 3 em que foi criada uma nova entidade chamada produto relacionada com a entidade hospedagem.

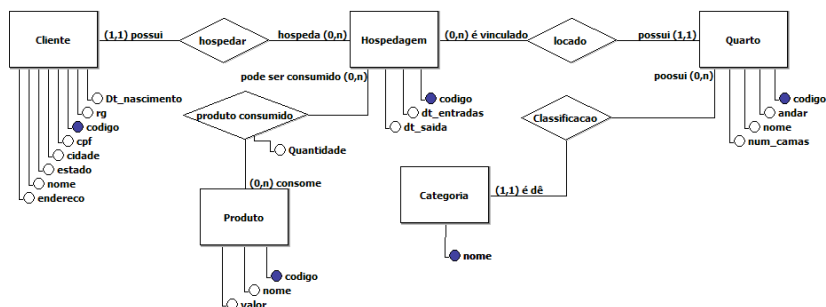


Figura 32 Diagrama Conceitual do Hotel – Solução 1

Nessa solução, foram adicionados atributos que não são mencionados no texto, como valor do produto na entidade produto e o atributo quantidade na relação “produto consumido”. Isso ocorreu pela experiência do administrador de banco que ao identificar que se trata de controlar o consumo de produtos, imaginou que fosse necessário saber quanto de cada produto foi consumido e o valor que cada produto custou. Nesse exemplo, o atributo quantidade ficou associado à relação, pois quantidade faz referência a quantidade consumida e não ao produto. Nesse exemplo, um produto possui características próprias de produtos. A relação criada é lida da seguinte maneira. Um produto pode ser consumido em nenhuma ou em várias hospedagens. Exemplo: é possível vender refrigerante para várias hospedagens. Já uma hospedagem pode possuir nenhum ou vários produtos consumidos (exem-

plo: em uma hospedagem é possível vender um refrigerante e um salgadinho).

Outra solução é não armazenar os dados de produtos de forma separada do consumo realizado. Nesse caso, o diagrama conceitual deverá ser modificado. A Figura 33 apresenta essa outra solução para os produtos consumidos.

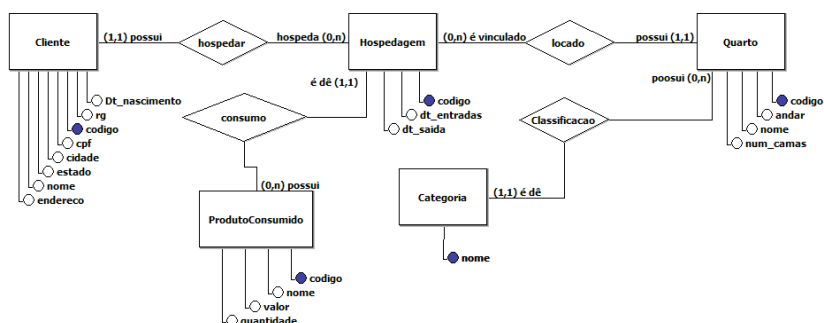


Figura 33 Diagrama Conceitual do Hotel – Solução 2

Nessa segunda solução, todos os atributos foram colocados na entidade denominada “Produtos Consumidos”, além do tipo de relação ter sido alterada. A primeira solução apresentada na Figura 32 possuía relação N:N, já a solução da Figura 33 possui relação 1:N. Na prática, isso dependerá do que o cliente final deseja implementar, pois, se a implementação for a solução 1, este terá uma listagem de produtos e selecionará o produto que foi consumido. Já na solução 2, ele não terá uma listagem de produto e sim deverá cadastrar sempre os produtos que foram consumidos. Novamente, não existe solução correta até que essa informação seja validada junto ao cliente. Esse tipo de situação novamente ocorre por problemas

na etapa de levantamento de requisitos, pois é nessa etapa que se obtém informações suficientes para solução desse tipo de problema, o que na prática não ocorreu no Exemplo 3.

Sendo assim, a implementação do diagrama conceitual, dependerá sempre do correto levantamento de requisitos. Outro fator importante a ser considerado é que, em geral, não existe uma única solução de desenvolvimento do diagrama conceitual. Isso justifica o porquê existem tantos sistemas desenvolvidos com o mesmo objetivo. Cada um possui uma estrutura diferente. Avaliar se um diagrama está correto ou não dependerá exclusivamente da existência de requisitos que possam subsidiar o avaliador.

Recapitulando

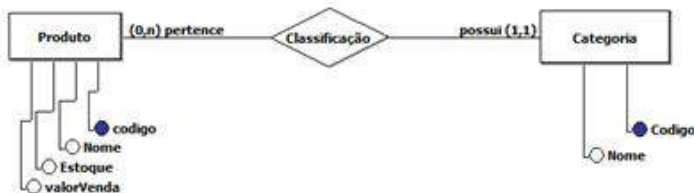
Ao concluir esse capítulo, é possível realizar uma série de constatações práticas sobre o modelo conceitual, tendo em vista os exemplos de Petshop e Hotel trabalhados.

Tendo como base as descrições dos domínios, é possível compreender como ocorre a aplicação dos tipos de relação e a implantação das cardinalidades máximas e mínimas. Também foi apresentada a solução de criação de novos atributos para atender a necessidade de definição de atributos identificadores.

Segundo exemplos, foi possível constatar a necessidade da correta definição dos dados extraída no levantamento de requisito, como forma de elaborar um diagrama conceitual sem que ocorram dúvidas que possam ocasionar atraso ou incerteza sobre a solução de banco que está sendo desenvolvida.

Atividades

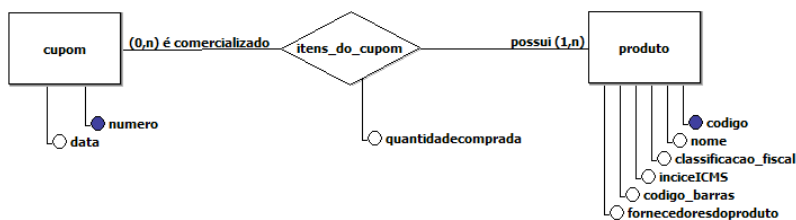
- 1) Marque a alternativa que determina a correta leitura das cardinalidades apresentadas no diagrama que segue:



- a) Um produto possui uma categoria e uma categoria pertence a um ou vários produtos.
- b) Um produto pertence a uma categoria e uma categoria possui um ou vários produtos.
- c) Um produto pertence a nenhum ou a várias categorias, já uma categoria possui no mínimo um e no máximo um produto.
- d) Um produto possui nenhuma ou a várias categorias, já uma categoria possui no mínimo um e no máximo um produto.
- e) Um produto possui uma categoria e uma categoria pertence a nenhum ou vários produtos.

2) Tendo como base o trecho que determina a criação de um modelo conceitual e o modelo gerado, marque (V) para as assertivas Verdadeiras e (F) para as Falsas que refletem as considerações referentes ao diagrama elaborado com base no domínio apresentado no texto:

“Um supermercado necessita informatizar sua venda de produtos por meio do cupom fiscal. Sabe-se que um cupom fiscal é composto de um número, uma data e produtos que o compõe. Cada produto possui um código, nome, uma classificação fiscal, um índice de ICMS, e código de barras, e os fornecedores que fornecem os produtos ao supermercado. Para cada produto pertencente ao cupom, é necessário armazenar a quantidade comprada do item”.



Tendo como base nas afirmações abaixo relacionadas, marque (V) para as assertivas Verdadeiras e (F) para as Falsas.

- ☐ O atributo `quantidadecomprada` deveria estar na entidade `cupom` e não na relação.
- ☐ O atributo `fornecedoresdoproduto` deveria ser uma entidade possuindo relação com a entidade `produto`.

c) () O tipo de relação desse diagrama deveria ser 1:N e não N:N.

d) () O nome de produto não poderia ser identificador, pois pode ocorrer de existir produtos cadastrados com mesmo nome.

Gabarito:

1) e; 2) a) F b) V c) F d) V

Christiano Cadoná*

Capítulo 5

Modelagem Lógica

➔ **D**ando prosseguimento ao processo de banco de dados, este capítulo abordará a etapa de Modelagem lógica. Será estudada a transformação do modelo conceitual para o modelo lógico relacional, possibilitando ao aluno compreender de forma prática os conceitos de chave primária e chave estrangeira, além de boas práticas na especificação das nomenclaturas do diagrama.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Modelo Lógico

O modelo lógico descreve as estruturas que representam a base de dados. Sua representação é semelhante ao modelo conceitual, porém possui suas particularidades. Diferentemente do modelo conceitual, para modelar um diagrama lógico, dependemos saber qual tipo de sistema gerenciador de banco de dados será implementado (relacional, rede, hierárquico, OO). Como a proposta deste livro está embasada no modelo relacional, este será utilizado como base no decorrer deste material, até porque, como já mencionado anteriormente, o modelo relacional predomina no cenário de desenvolvimento que estamos atualmente inseridos. Sendo assim, o modelo lógico selecionado para representação dos exemplos é o modelo relacional.

Relembrando, o modelo lógico possui, como principal característica, a definição no diagrama das chaves primárias e estrangeiras, o que define de forma física as relações entre as entidades. Da mesma forma que o modelo conceitual, existem inúmeras representações deste diagrama, e geralmente os fabricantes dos SGBD criam interfaces para modelagem seguindo um modelo pré-estabelecido. O importante a ser definido aqui, é que, independentemente do modelo, o resultado de forma prática será o mesmo, o que muda é sua representação. De forma geral, ousar definir que, se alguém detém conhecimento sobre um determinado modelo, certamente entenderá outro modelo sem nenhuma dificuldade.

O modelo Entidade Relacionamento proposto por Peter Chen e implementado como modelo conceitual, é um modelo

semântico que fala de objetos (entidades e relacionamentos) que serão representados no banco de dados, já o modelo relacional, representa o modelo de dado propriamente dito que será definido no SGBD.

Enquanto o modelo ER descreve um banco de dados de forma independente de implementação o modelo relacional descreve um banco de dados em nível de SGBD relacional.

Mapeamento modelo ER para Modelo Relacional

Após estabelecido que o modelo lógico a ser implementado é o modelo relacional, partimos para transformação do modelo ER (conceitual), para o modelo Relacional (lógico).

Para isso, existe uma série de transformações proposta que é simples de ser implementada. A primeira delas é implementação das entidades. Cada conjunto de entidades é implementado por meio de uma tabela. Os atributos das entidades ER são implementados por meio de atributos no modelo relacional. O atributo identificador fornece a chave primária da relação.

A Figura 34 apresenta um exemplo da transformação da entidade Aluno na tabela aluno no modelo relacional. No exemplo, é possível visualizar a definição do modelo relacional de duas formas. A primeira, graficamente, e a segunda de forma textual. Ambas a formas de representação estão corretas, mas a mais utilizada é a gráfica, por possuir como informação adicional o tipo de dado que será definido para cada um dos

atributos, além de ser mais compreensível em situações de um conjunto contendo um número significativo de entidades.

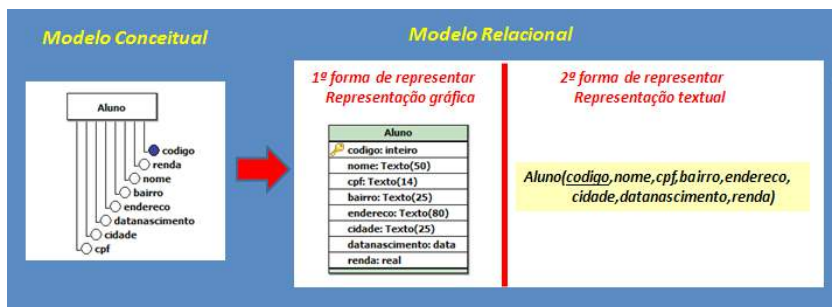


Figura 34 Transformação de entidade em tabela

Perceba que, no modelo relacional, o atributo de chave primária é identificado. No modelo gráfico, este ganhou um símbolo de uma chave, já no modelo textual, o atributo ficou sublinhado.

Como já abordado, o modelo gráfico já apresenta o conceito de tipo de dado associado a cada um dos atributos, diferentemente do modelo textual. Essa informação não é obrigatória nesse momento do processo de banco de dados, porém quase todas as notações já definem na elaboração do diagrama a definição do tipo de dado de cada atributo. Isso se deve pelo fato de já existir uma definição de qual será o banco de dados que será implementado o banco, e com isso já ter conhecimento dos tipos de dados suportados por este. No exemplo apresentado na Figura 34, apenas foi definido os tipos número, texto e data, como forma de representar as informações. Geralmente, o tipo de dados texto, definidos a

quantidade máxima de caracteres que seria necessária para armazenamento. Já se tratando de números, é importante dividir os campos que devem armazenar números inteiros dos campos de devem armazenar números reais. Alguns campos que normalmente são representados por números, podem ser armazenados como texto, como é o caso do campo CPF que é um número, e, nesse exemplo, foi definido com do tipo texto. Isso se deve, pois o analisa deseja armazenar possivelmente os símbolos juntamente com os números do CPF, o que não é possível se este for definido como número. Em alguns casos, existem banco de dados que possuem um recurso chamado formatação, que permite o armazenamento desse tipo de dado, e sua visualização respeita uma determinada formatação a qual apresenta os dados como se fosse um texto. Contudo, esse recurso não é implementado em todos os SGBD. Dessa forma, estamos definindo neste material que, se houver a necessidade de armazenamento de informações numéricas contendo caracteres especiais como CEP ou telefone, estes campos serão armazenados como texto.

Outra transformação relevante do modelo relacional em relação ao modelo ER é a definição das relações. Nesse caso, a transformação dependerá diretamente do tipo de relação definido no diagrama conceitual (1:1, 1:N ou N:N).

Transformação de Relação do tipo 1:1

○ mapeamento de uma relação 1:1 dependerá diretamente da cardinalidade mínima aplicada em cada relação. Caso

exista **obrigatoriedade em ambas cardinalidades mínimas** de uma relação, a solução é a junção das duas entidades participantes mais os atributos que existirem na relação em uma única tabela. O exemplo apresentado na Figura 35 ilustra a aplicação da transformação de uma relação do tipo 1:1 com as duas cardinalidades mínimas obrigatórias. Observe que as entidades empregado e equipamento se transformaram em uma única tabela chamada “EmpregadoEquipamento”, contendo todos os atributos das entidades mais o atributo “dataLocacao” que estava na relação “Alocado”. Como as duas entidades possuíam um identificador único no modelo conceitual, na transformação não ocorre a necessidade de existir uma chave primária composta, por mais que não esteja errado ocorrer. Nesse caso, foi elegido um dos atributos como chave primária e implementado no diagrama.

Outra consideração é a modificação dos nomes dos atributos. Observe que, nesse exemplo, o nome dos atributos foi alterado como forma de identificar melhor o que cada um irá armazenar, até porque, vamos imaginar que o campo identificado do equipamento possuísse o nome de código ao invés do número, ambos campos teriam o mesmo nome, e isso não poderia ser implementado na transformação.

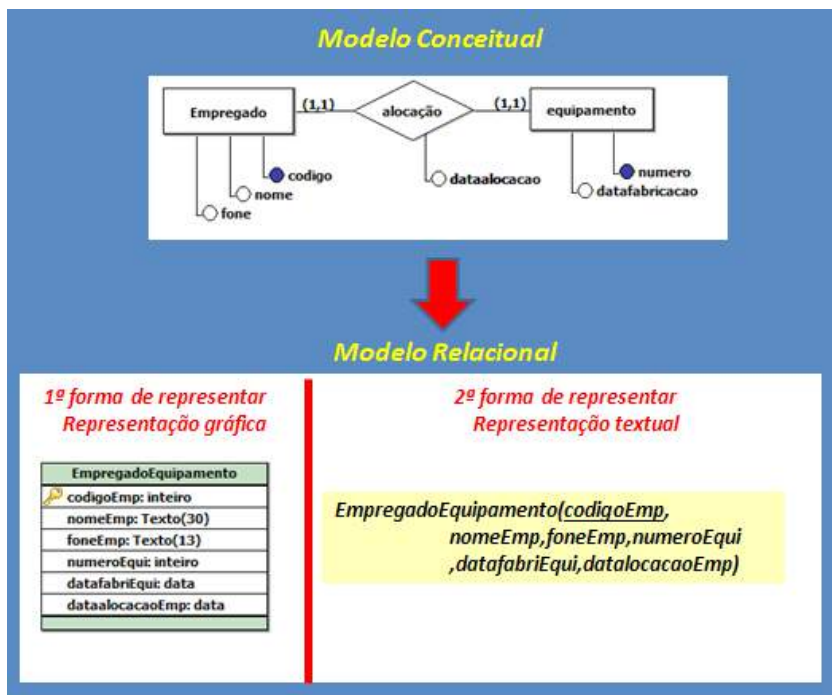


Figura 35 Transformação relação 1:1 com cardinalidades obrigatórias

A prática de determinar no próprio nome do atributo uma identificação é comum, além de possibilitar maior clareza sobre o que será armazenado em cada campo. Uma prática não usual, é a determinação de nomes de atributos a partir de identificadores como, por exemplo, atributos com nomes "A1", "A2", "A3", onde A1 representa o código, A2 representa o nome e A3 representa o telefone. Além de desenvolver um diagrama com pouco ou nenhum sentido, ocasiona ao desenvolvedor, que irá utilizar esse banco de dados, confusão e atraso no desenvolvimento, pois estes, além de se basear no diagrama desenvolvido, terão que consultar em uma tabela o que significa cada um dos atributos.

Caso exista uma relação 1:1 em que apenas um dos lados é opcional, ou seja, a cardinalidade mínima de um dos lados é 0 (zero) e a outra é 1 (um), a transformação é um pouco diferente. Nesse caso, serão geradas duas tabelas e a tabela que contém o lado opcional (0-zero), conterá um novo atributo que será um atributo de chave estrangeira referenciando a chave primária da entidade de lado obrigatório.

O exemplo apresentado na Figura 36 ilustra a transformação para o modelo relacional de uma relação 1:1 com as entidades correntista e cartão magnético, em que um dos lados é opcional, ou seja, um correntista pode não ter um cartão magnético, já um cartão magnético sempre será de propriedade de um correntista.

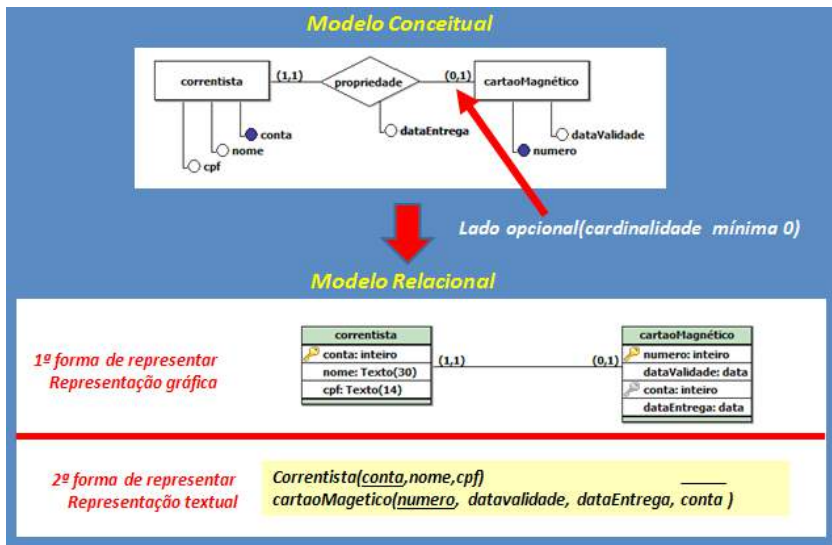


Figura 36 Transformação relação 1:1 com uma cardinalidade opcional

Observe que, nesse exemplo, ambas as entidades se transformaram em tabelas e a tabela “cartaoMagnetico”, possui 2 novos atributos, sendo o primeiro o atributo que estava na relação e o segundo o atributo conta, que é uma chave estrangeira referenciando a tabela de correntista. Na primeira forma de representar (representação gráfica), o campo conta possui como símbolo uma chave com outra cor, já na segunda forma de representar (textual), o atributo conta possui um traço em cima de seu nome, representando ser uma chave estrangeira.

No caso da existência de uma relação 1:1 em que ambos os lados são opcionais, ou seja, a cardinalidade mínima dos dois lados é 0 (zero), a transformação possui uma nova característica. Nesse caso, serão geradas além das duas tabelas que representam as entidades, mais uma tabela, que armazenará a relação propriamente dita. Essa nova tabela será composta pelos atributos que nela existirem, além dos atributos chave estrangeira que farão referência às outras duas entidades envolvidas na relação.

A Figura 37 apresenta uma relação entre as entidades pessoa e cargo do tipo 1:1, em que ambos os lados são opcionais, ou seja, uma pessoa pode estar alocada para, no máximo, um cargo e um cargo pode possuir, no máximo, uma pessoa alocada. Porém, ambos podem não possuir nenhuma alocação.

Segundo exemplo, a relação que se chamava alocação foi transformada em uma tabela contendo os atributos que já possuía (data e regime) além de mais outros dois atributos codpes e codcargo, que são chaves estrangeiras das entidades

pessoa e cargo. Por definição, sabemos que toda entidade ou tabela necessariamente precisa de uma chave primária que a identifique. Nesse caso, optamos por criar uma chave primária composta pelas duas chaves estrangeiras. Dessa forma, os atributos codpes e codcargo, são chaves primárias e estrangeiras ao mesmo tempo. Uma alternativa seria a criação de um novo atributo que possa ser utilizado como chave primária como, por exemplo, código.

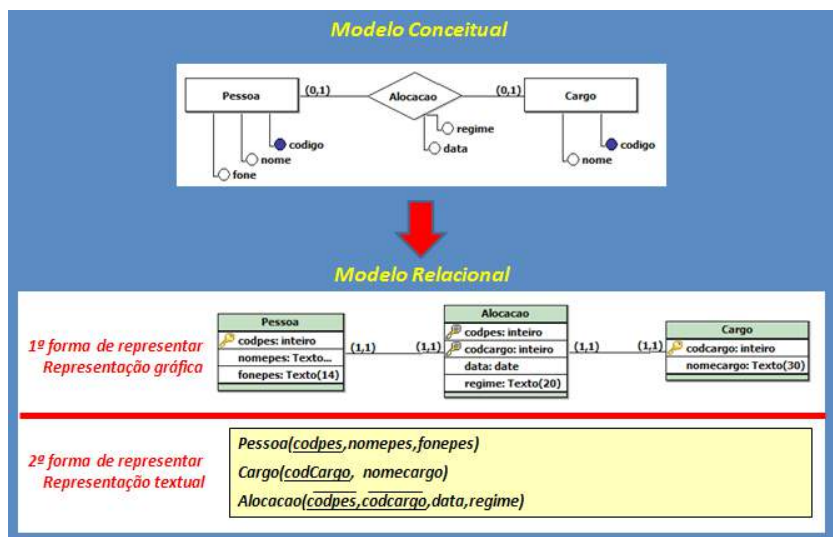


Figura 37 Transformação relação 1:1 com ambas cardinalidades mínimas opcionais

Perceba que as relações que foram criadas entre as entidades pessoa e entidade cargo continuam sendo 1:1, porém a cardinalidade mínima ficou definida como 1 e não mais 0 (zero) como o modelo conceitual. Dessa forma, é criada uma

obrigatoriedade de, ao informar um novo registro na tabela de “Alocacao”, ser obrigado a informar um cargo e uma pessoa.

Transformação de Relação do tipo 1:N

O mapeamento de uma relação 1:N é bem mais simples do que o tipo 1:1. No mapeamento 1:N, transformamos as entidades em tabelas e a entidade que possui o lado N conterá um novo atributo que será a chave estrangeira referenciando a chave primária da outra entidade. Todos os atributos que pertencerem à relação também serão encaminhados para a entidade que possui seu lado N.

O exemplo apresentado na Figura 38 ilustra a transformação do modelo conceitual para o modelo lógico relacional das entidades departamento e empregado contendo uma relação 1:N. Observe que, na tabela empregado, que possui o lado N, foi adicionado um novo atributo chamado coddep que foi definido como chave estrangeira da tabela de empregado. Outro fator relevante que ainda não havia sido comentado até então faz referência ao tipo de dado da chave estrangeira. Todo atributo de chave estrangeira deverá obrigatoriamente possuir o mesmo tipo de dado da chave primária que esteja referenciando.

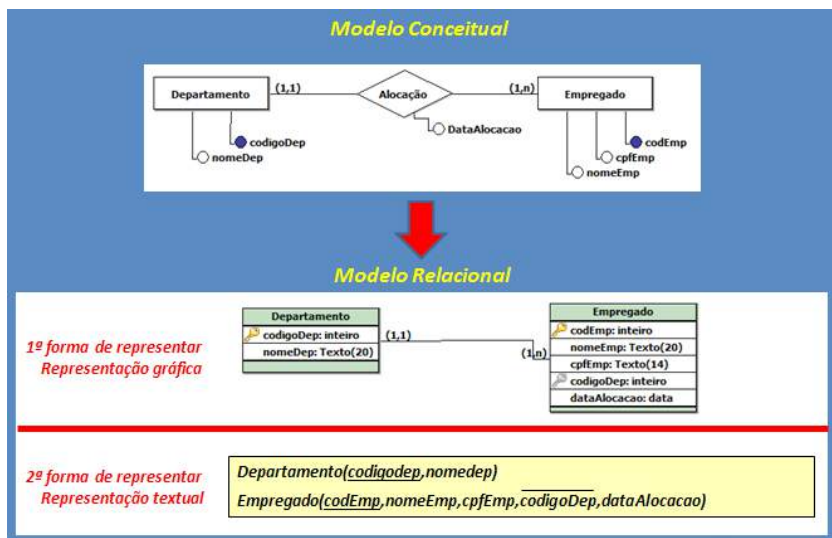


Figura 38 Transformação de modelo relação 1:N

O atributo “dataAlocacao” que estava na relação também foi encaminhado para a entidade que possuía o lado N, nesse caso, a entidade empregado.

A cardinalidade mínima que antes influenciava na criação das entidades aqui somente servirá para determinar se o atributo de chave estrangeira poderá armazenar valores nulos ou não. Sendo assim, essa somente deverá ser testada no lado N da relação 1:N. No exemplo da Figura 38, o lado N possui cardinalidade mínima 1, o que gera a obrigatoriedade de existir um departamento em um empregado. A definição dessa obrigatoriedade será aplicada na etapa de modelagem física, em que será construído o banco de dados.

Transformação de Relação do tipo N:N

Uma relação N:N, indica a existência de múltiplos registros entre entidades. Dessa forma, o mapeamento de uma relação N:N gerará impreterivelmente uma nova tabela no lugar da relação, unindo de forma física as outras duas tabelas geradas pela relação. Nesta, todos os atributos que faziam parte da relação farão parte na nova tabela, além da criação dos atributos que serão as chaves estrangeiras, que farão referência às outras tabelas que faziam parte da relação.

A Figura 39 apresenta a transformação da relação do tipo N:N entre as entidades engenheiro e projeto. Nesse exemplo, um engenheiro pode ser alocado a um ou a vários projetos, e um projeto possui um ou vários engenheiros vinculados.

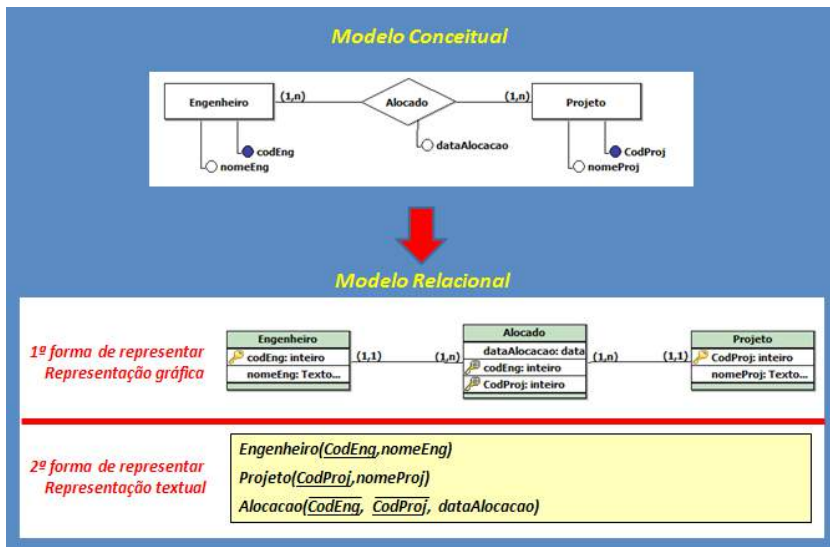


Figura 39 Transformação de modelo relação N:N

Quando transformamos um tipo de relação N:N, as cardinalidades máximas e mínimas sempre ficarão ao lado da relação e o outro lado receberá a cardinalidade mínima e máxima 1 (1,1). Isso pode ser percebido no Exemplo 39, em que as cardinalidades que estavam em engenheiro e projeto passaram a fazer parte na tabela “Alocacao”, enquanto ao lado das tabelas “Engenheiro” e “Projeto”, foram adicionadas as cardinalidades (1,1).

Perceba que a tabela “Alocado”, foi criada no lugar da relação, e essa possui o atributo “dataAlocacao”, que fazia parte da relação. Também foram criadas as chaves estrangeiras “CodEng” que faz referência à tabela de engenheiro e “CodProj” que faz referência à tabela de projeto. Nesse exemplo, foi definido que a chave primária da tabela de “Alocado” é composta pelos atributos de chave estrangeira. Com isso, garantimos que exista um identificador único e que não seja possível um mesmo engenheiro estar alocado para um mesmo projeto mais de uma vez.

Contudo, a criação de uma chave primária composta nem sempre é a melhor alternativa, e, em alguns casos, não pode ser implementada sob risco de inviabilizar o sistema desenvolvido. Observe o exemplo apresentado na Figura 40 que contém uma relação N:N entre as entidades “CupomFiscal” e “Produto”, no qual a relação “ItensVendidos”, foi transformada em uma nova tabela contendo como chave primária um novo atributo “codigoltem”.

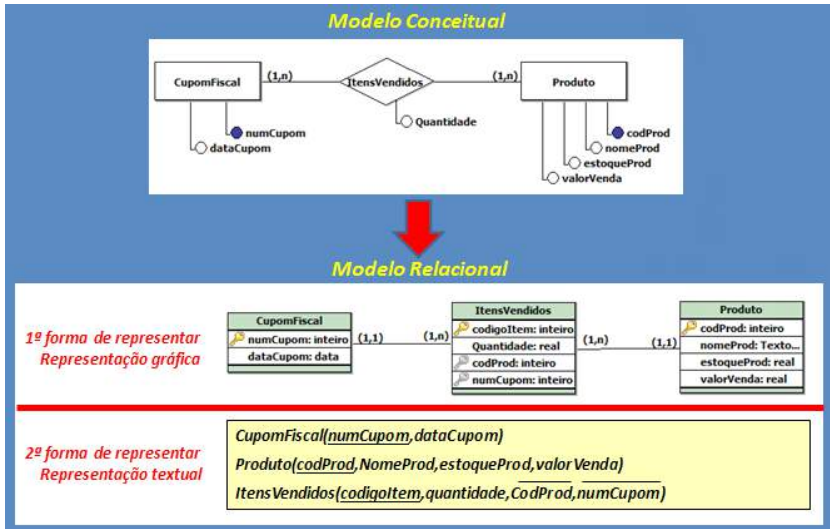


Figura 40 Transformação de modelo relação N:N (criando novo atributo primário)

Nesse caso, os atributos de chave estrangeira existentes na tabela “ItensVendidos” são “codProd” referenciando a tabela de “Produto” e “numCupom” referenciando a tabela de “CupomFiscal”. Estes não podem ser definidos como chave primária composta, pois, na prática, se determinarmos isso, não será possível existir em um mesmo cupom fiscal mais do que um lançamento de produto, o que é comum que ocorra em estabelecimentos comerciais (permitir mais de um registro do mesmo produto em uma venda). Imagine, como exemplo, um carrinho de compras de um supermercado contendo muitos itens. É normal que ocorra registros de mesmos itens na mesma compra em momentos diferentes, como passar um sabonete logo no início da compra e outro no final. O operador deve poder passar novamente um item idêntico ao já passado

anteriormente. Perceba que, se tivéssemos definido que a chave primária seria composta, isso não seria permitido, mas a alternativa apresentada na Figura 40 permite que isso ocorra sem problemas. Assim, a melhor forma de definição dependerá da aplicação a ser implementada. Em algumas situações, definiremos as chaves estrangeiras como chaves primárias compostas, e em outras definiremos outro atributo como chave primária.

Transformação de uma autorrelacionamento.

A transformação do modelo conceitual para o modelo lógico relacional de um autorrelacionamento dependerá do tipo de relação existente no autorrelacionamento. Se o tipo de relação aplicado no autorrelacionamento for 1:1 ou 1:N, deverá ser criado um novo atributo na entidade contendo uma chave estrangeira que será associada ao atributo de chave primária da mesma entidade. O exemplo apresentado na Figura 41 apresenta uma associação 1:N de um autorrelacionamento, em que é possível identificar que, no modelo relacional implementado, foi criado um campo chamado “codsup” que é uma chave estrangeira vinculada à chave primária “codEmp” da mesma tabela “empregado”.

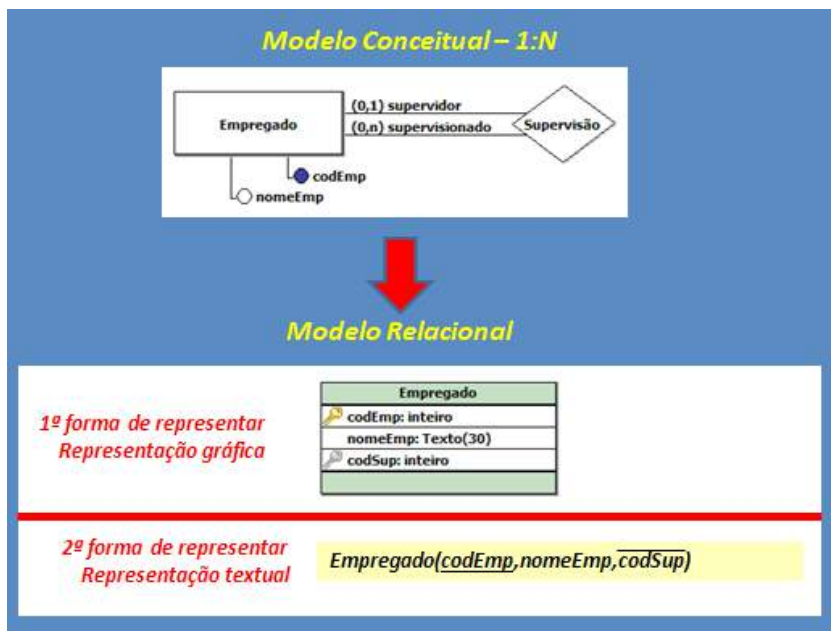


Figura 41 Transformação de um autorrelacionamento 1:N

Se o tipo de relação de um autorrelacionamento for N:N, ocorre a necessidade de criação de uma nova entidade no lugar na relação. Esta conterá dois campos definidos como chaves estrangeiras que farão referência ambos à mesma chave primária da tabela que contém a autorrelação. A Figura 42 apresenta o mesmo exemplo de entidade, porém com o tipo de relação N:N, em que um empregado pode ser supervisor de nenhum ou vários empregados, e um empregado poderá ser supervisionado por nenhum ou vários supervisores.

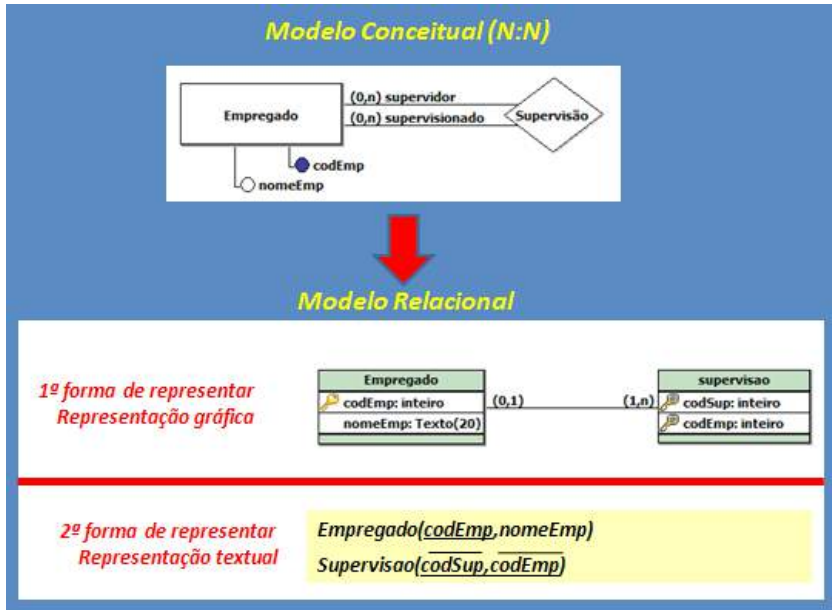


Figura 42 Transformação de um autorrelacionamento N:N

Nesse exemplo, a relação “supervisão” se transformou em uma tabela contendo dois campos (“codEmp e codSup”), sendo estes chaves primárias compostas e chaves estrangeiras referenciando a entidade “empregado”.

Transformação de Generalização/ Especialização.

Como já abordado, o conceito de generalização e especialização é utilizado no modelo conceitual para representar informações comuns de um conjunto de entidades (generalização),

e, em seguida, possuem uma divisão criando outras entidades contendo especificidades dessas entidades criadas (especialização). A transformação do modelo conceitual no modelo lógico relacional, nesse caso, pode ocorrer de três formas distintas, ambas corretas. A aplicação da melhor forma dependerá da forma como será implementado o sistema.

Usaremos como base a Figura 43, já apresentada anteriormente, como exemplo de generalização/especialização, para representar as três formas de transformação para o modelo lógico relacional.

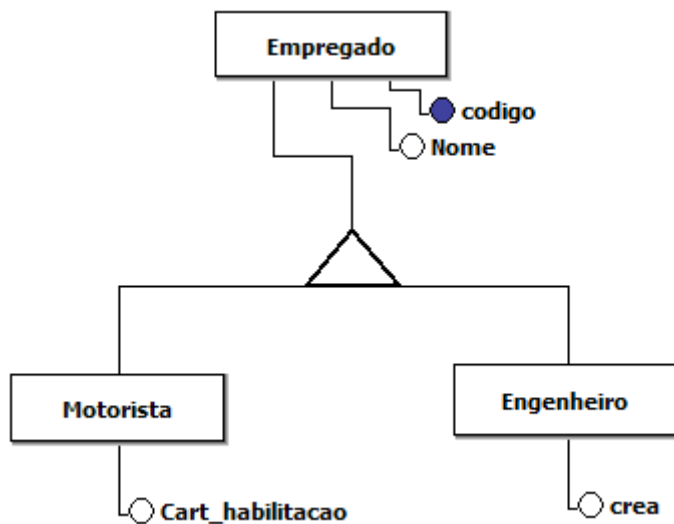


Figura 43 Modelo Conceitual Generalização/Especialização

A primeira alternativa de transformação é a implementação de diferentes tabelas com uma tabela para dados

comuns. Essa solução é muito semelhante ao conceito de classes e heranças de orientação a objetos. A Figura 44.a apresenta cada uma das entidades implementadas como tabelas, e nas entidades especializadas “Motorista” e “Engenheiro” existe um campo de chave estrangeira ligando-se à tabela de “Empregado”. Esses atributos de relação também serviram como atributos de chave primária.

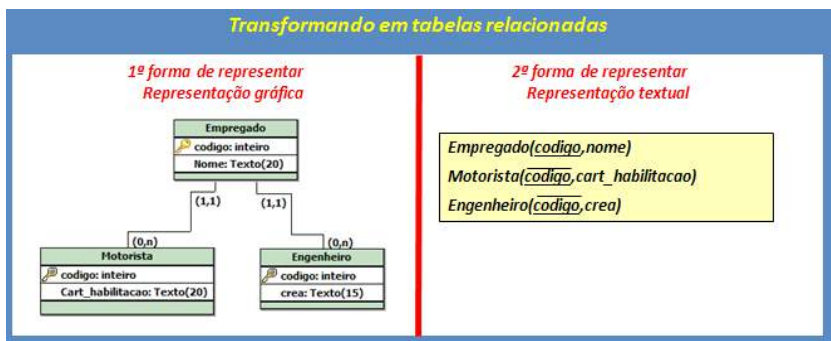


Figura 44.a Alternativa 1 de transformação generalização/especialização

A segunda alternativa de transformação é a implementação de uma única tabela contendo todos os campos das entidades participantes da generalização/especialização. Nessa solução, em geral, é criado um novo atributo como forma de identificar o tipo de especialização desejada. No exemplo da Figura 44.b, identificamos a implementação do modelo lógico relacional de generalização/especialização no qual é percebido a criação do atributo “tipo_emp” que servirá como identificador de tipo de empregado que se deseja armazenar. Esse tipo de atributo é conhecido como atributo *flag*, pois possui como funcionalidade “setar” uma determinada informação.

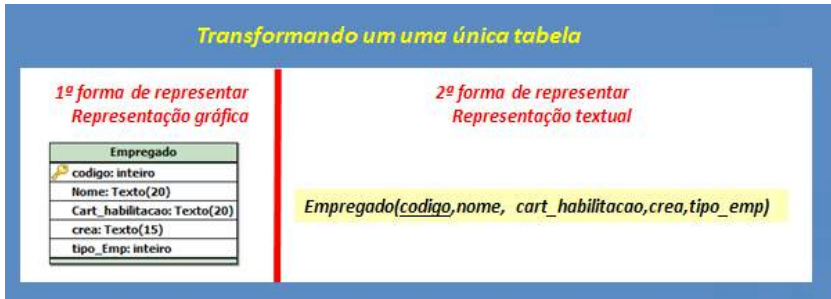


Figura 44.b Alternativa 2 de transformação generalização/especialização

A terceira alternativa de transformação é a implementação das entidades especializadas em tabelas contendo todos os atributos que estas possuírem, mais os atributos que são ori-gi-nários da entidade generalizada. As entidades resultantes não serão relacionadas e seus dados ficarão independentes. A Fi-gura 45 apresenta o resultado utilizando a terceira alternativa.



Figura 45 Alternativa 3 de transformação generalização/especialização

Recapitulando

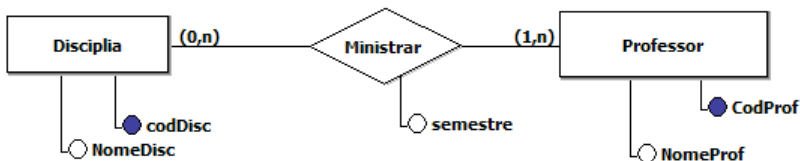
Ao concluir este capítulo, o aluno percebeu que o modelo lógico possui como principal característica o conceito de relacionamento a partir de campos de chave primária e estrangeira.

O modelo lógico relacional possui uma estrutura bem mais simples do que o modelo conceitual proposto por Peter Chen, tanto que o resultado da transformação dos modelos conceituais para o modelo lógico relacional resulta em um conjunto de tabelas, atributos e relações definidas pelos atributos de uma tabela.

A partir dos exemplos apresentados, foi possível identificar como ocorre a transformação do modelo conceitual para o modelo lógico, onde foram abordados a transformação das entidades em tabelas, dos tipos de relação e da generalização/especialização.

Atividades

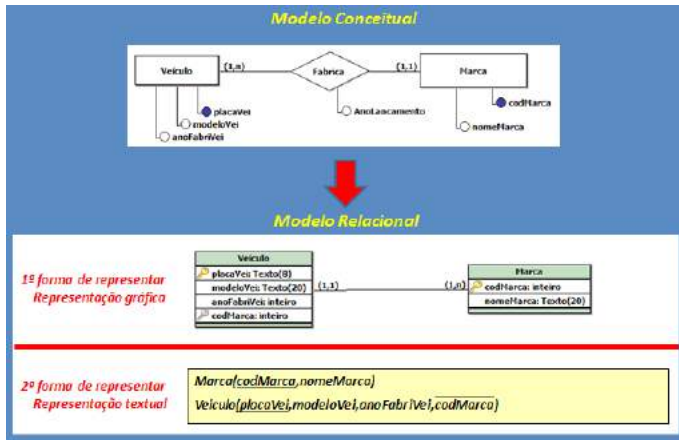
- 1) Tendo como base o diagrama conceitual, marque a alternativa correta que representa a transformação para o modelo conceitual.



- () Serão criadas duas tabelas, disciplina e professor, e o atributo semestre será armazenado na tabela professor.

- () Deverão ser criadas três tabelas, sendo duas para cada entidade e uma terceira para a relação ministrar que conterá, além do atributo semestre, mais outros dois atributos de chave estrangeira, referenciando as entidades disciplina e professor.
- () Serão criadas duas tabelas, disciplina e professor, e o atributo semestre será armazenado na tabela disciplina.
- () Será criada apenas uma única tabela contendo todos os dados das entidades disciplina e professor além do atributo semestre existente na relação ministrar.
- () Deverão ser criadas três tabelas, sendo duas para cada entidade e uma terceira para a relação ministrar que conterá dois atributos de chave estrangeira, referenciando as entidades disciplina e professor. Por ser uma relação, o atributo semestre deverá ser armazenado no lado obrigatório que, nesse caso, faz referência à tabela professor.

- 2) Tendo como base o diagrama conceitual apresentado e o diagrama lógico relacional proposto para este, marque (V) para as assertivas Verdadeiras e (F) para as Falsas:



- () A transformação do tipo de relação está correta, pois criou duas tabelas, colocando no lado N um atributo de chave estrangeira referenciando a outra entidade.
- () As cardinalidades existentes na primeira forma de representar estão invertidas.
- () O atributo existente na relação não foi implementado. O correto seria colocá-lo em uma tabela.
- () Pela tabela de veículo possuir um atributo de chave estrangeira, este deverá também fazer parte da chave primária de veículo.

Gabarito:

1) B; 2) V - V - V - F

Christiano Cadoná*

Capítulo 6

Modelagem Física, Normalização de Dados e Álgebra Relacional

➡ **D**ando prosseguimento ao processo de banco de dados, este capítulo inicia com a etapa de Modelagem lógica. Será estudada a transformação do modelo conceitual para o modelo lógico relacional, possibilitando ao aluno compreender de forma prática os conceitos de chave primária e chave estrangeira, além de boas práticas na

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

especificação das nomenclaturas do diagrama.

Em seguida, serão bordados conceitos de normalização de dados e álgebra relacional, apresentando exemplos práticos de sua aplicabilidade. Dessa forma, este estudo possibilitará ao aluno concluir o processo de banco de dados e ter uma visão de álgebra relacional a qual é a base de diversas linguagens DML como o SQL.

Modelo Físico

A modelagem física é a última etapa do processo de banco de dados. Nessa etapa, o modelo lógico será transformado em uma estrutura em que cada tabela e seus atributos serão representados de forma que o banco de dados possa ser construído. O resultado dessa aplicação é uma descrição do esquema DDL do SGDB, ou seja, teremos uma definição de todos os tipos de dados, quais serão os índices aplicados, métodos de acesso, entre outros. A partir desse modelo, é possível utilizar uma linguagem DML como, por exemplo, SQL para criar um banco de dados contendo todas as informações que o modelo lógico gerou. Impreterivelmente, nessa etapa, o banco de dados já deve ter sido definido.

Não existe nenhum diagrama para representar o modelo físico, sua representação dependerá da metodologia de trabalho que cada empresa desenvolvedora utiliza. O que ocorre geralmente nessa etapa, e o registro de criação a partir de uma linguagem DML das tabelas, como é o caso da utilização do comando `CREATE TABLE` e `ALTER TABLE` da linguagem SQL.

Como forma de representar o processo como um todo, o Exemplo 4 apresenta uma descrição de um domínio que serve como base para elaboração de um processo de banco de dados que se inicia na modelagem conceitual e é concluído no modelo físico.

Exemplo 4: “Matérias primas de Sorvete”

Uma sorveteria necessita informatizar o processo de registro de matérias primas necessárias para produção de seus sorvetes. Sabe-se que um produto possui um código, um código de barras, um nome comercial, e um valor de custo. Cada produto possui uma ou várias matérias primas, já uma matéria prima pode ser de nenhum ou vários produtos. Uma matéria prima possui um código, um nome e um valor de compra. Para cada matéria prima adicionada a produto é necessário armazenar uma quantidade necessária da matéria prima para compor o mesmo.

A Figura 46 apresenta uma alternativa de solução para o Exemplo 4 passando pela modelagem conceitual, sendo transformado para o modelo lógico relacional e concluído no modelo físico com instruções SQL de criação de tabelas e suas respectivas relações.

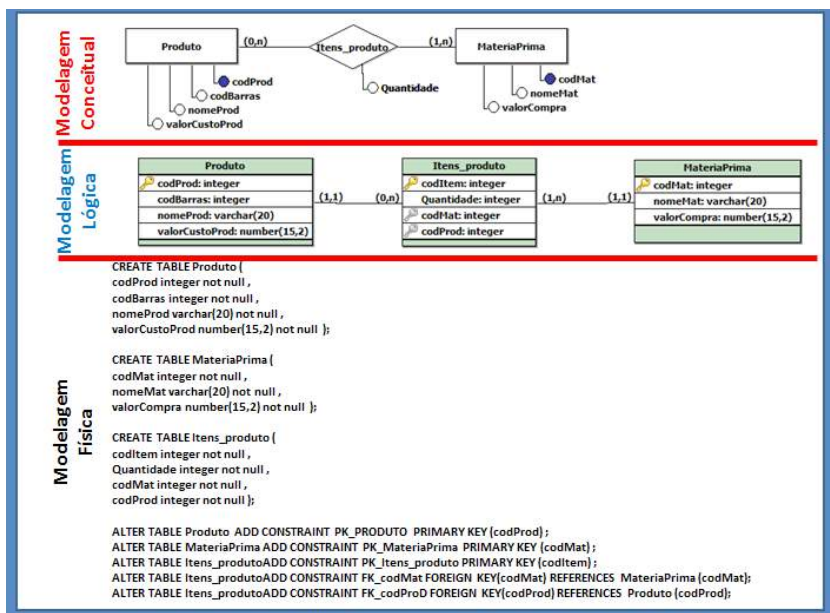


Figura 46 Etapas do processo de Banco de dados

O código gerado no modelo físico será detalhado no decorrer dos próximos capítulos, onde será abordada a linguagem SQL. O importante a ser destacado no exemplo de modelagem física, está relacionado ao tipo de dado. Perceba que o mesmo tipo de dado definido no modelo lógico foi aplicado no modelo físico, além, é claro, no mesmo nome dos atributos e entidades.

Normalização de Dados

O principal objetivo de normalizar tabelas está no fato de evitar os problemas causados por falhas no projeto do banco de dados, bem como eliminar a “mistura de assuntos”, evitando as correspondentes repetições desnecessárias de dados.

Um fato a ser sempre observado em um projeto de banco de dados baseado no modelo relacional é o de não misturar assuntos em uma tabela. Um exemplo disso seria, em uma tabela de clientes, colocar apenas os dados de clientes. Informações como seus pedidos, contas a receber, devem estar em outras tabelas. Não devemos misturar os assuntos, pois, como consequência, teríamos repetições desnecessárias, bem como uma possível inconsistência de informação.

O Processo de normalização aplica uma série de regras sobre as tabelas de um banco de dados, para verificar se estão bem projetadas. Embora exista 5 formas normais (regras), na prática, usamos apenas “**3 formas normais**”.

É comum, após a aplicação das regras de normalização, algumas tabelas acabarem sendo divididas em duas ou mais tabelas, o que, no final, gera um número maior de tabelas do que originalmente existe. Nesse processo, observamos a simplificação dos atributos colaborando significativamente com a estabilidade do modelo de dados, reduzindo-se consideravelmente as necessidades de manutenção.

Primeira forma normal

Uma tabela está na primeira forma normal quando seus atributos não possuem grupos de repetição. Considere a estrutura da tabela representada na Figura 47 e o exemplo de dados armazenados nesta.

produto					
codProd: integer	nomeProd: varchar(20)	categoria: varchar(20)	estoque: integer	fornecedor: varchar(20)	valor: number(15,2)

Produto(codProd,nomeProd,categoria,estoque,fornecedor,valor)

codprod	NomeProd	categoria	estoque	fornecedor	valor
50	Abacaxi	Fruta	200	Fornecedor X	2,99
40	Pão Frances	Padaria	180	Fornecedor Y	9,50
79	Chuleta porco	Açougue	52	Fornecedor X	11,80
21	Uva	Fruta	88	Fornecedor X	4,99
585	Banana	Futa	350	Fornecedor Y	3,50
25	picanha	Açougue	10	Fornecedor Y	21,50

Erro de Digitação

Figura 47 Problemas na primeira forma normal

Uma tabela com essa estrutura apresenta diversos problemas, por exemplo, se um fornecedor ou categoria possuir mais de um produto, será necessário digitar diversas vezes, tantas quanto forem os produtos. Isso forma um grupo de repetições. Além do mais, pode ser que, por erro de digitação, o nome da categoria ou fornecedor não seja digitado exatamente igual todas as vezes, o que poder acarretar problemas na hora de fazer pesquisas ou emitir relatórios. No exemplo de tabela populada da Figura 47, é possível evidenciar os dois problemas apresentados.

Esse problema ocorre porque “misturamos assuntos” em uma mesma tabela. Colocamos as informações de categoria e fornecedor na tabela de produto. A resposta para esse problema é simples: criamos uma tabela de fornecedor e uma tabela de categoria relacionando-as com a tabela de produto, utilizando um relacionamento para muitos, ou seja, um fornecedor poderá ter vários produtos; uma categoria pertence a vários produtos. A Figura 48 apresenta a solução para o problema de primeira forma normal.

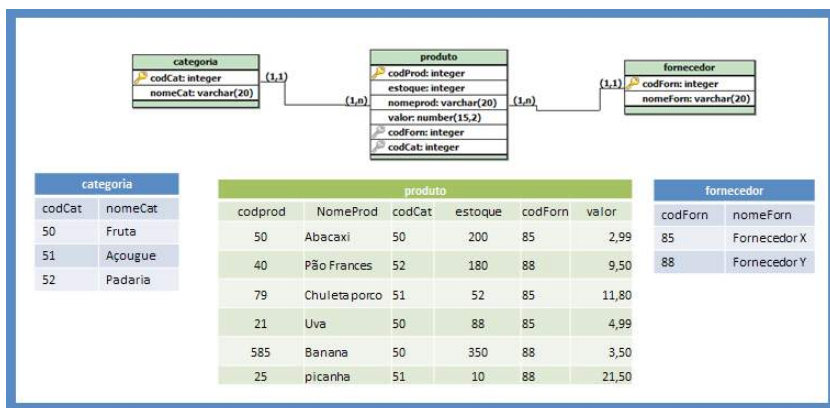


Figura 48 Aplicação da primeira forma normal

Conforme a solução apresenta, a tabela de produtos possuirá somente os dados de chave estrangeira das demais tabelas de relação.

Segunda forma normal

A segunda forma normal ocorre quando a chave primária é composta por mais de um campo (chave composta). Nesse

caso, devemos observar se todos os campos que não fazem parte da chave dependem de todos os campos que compõem a chave. Se algum campo depender somente de parte da chave composta, então esse campo deve pertencer à outra tabela. Observe a Figura 49.

numMatricula	codCurso	avaliacao	nomeCurso
1111	5	8,9	PHP
5555	4	7,5	Java
5555	3	6,5	Banco
2222	5	9,5	PHP
5555	5	10,0	PHP
1111	3	8,2	Banco

Figura 49 Erro na Segunda Forma normal

A chave primária composta é formada pela combinação dos campos “numMatricula” e “codCurso”. O campo “avaliacao” depende tanto do “codCurso” quanto do “numMatricula”, porém o campo “nomeCurso” depende apenas do “codCurso”, ou seja, dado o código do curso, é possível localizar a respectiva descrição, independentemente do número da matrícula. Com isso, temos um campo que não faz parte da chave primária (“nomeCurso”) e depende apenas de um dos

campos que compõem a chave primária composta. Dizemos que essa tabela não está na segunda forma normal.

A resolução desse problema também é simples. Dividimos a tabela que não está na segunda forma normal em duas tabelas, colocando o atributo que depende somente de um atributo da chave primária composta em outra tabela que contém como chave primária o atributo de dependência. A Figura 50 ilustra a solução da segunda forma normal.

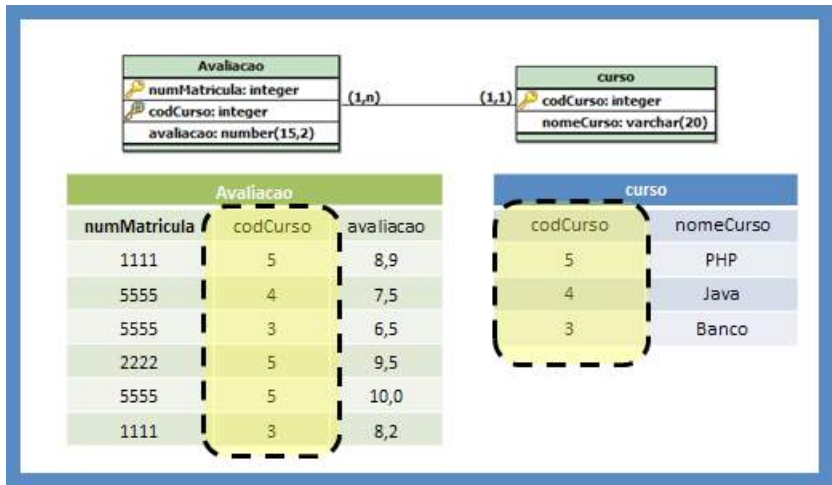


Figura 50 Solução para erro na segunda forma normal

Terceira forma normal

Na definição dos campos de uma entidade, podem ocorrer casos em que um campo não seja dependente diretamente de uma chave primária ou de parte dela, mas, sim, dependente de outro campo da tabela. Nesse caso, dizemos que a tabela

não está na terceira forma normal. Observe o exemplo da Figura 51.

funcionario	
numMatricula: integer	
nomeFunc: varchar(20)	
codCargo: integer	
nomeCargo: varchar(20)	

Funcionario(numMatricula,nomefunc,codCargo,nomeCargo)

numMatricula	nomeFunc	codCargo	nomeCargo
789	Ana	5	Diretora
012	Caroline	7	Gerente
325	Beatriz	5	Diretora
678	Ângela	5	Diretora
901	Andreza	9	Administradora
234	Bruna	5	Diretora




Figura 51 Erro na Terceira Forma Normal

Observe que o campo “nomeCargo” depende apenas do campo “codCargo”, o qual não faz parte da chave primária. Por isso, dizemos que essa tabela não está na terceira forma normal. A solução desse problema é simples, basta dividir a tabela em duas, colocando o campo dependente (nomeCargo) juntamente com o campo que o mesmo depende (codCargo) em uma outra tabela. A tabela original terá apenas uma chave estrangeira relacionada com a tabela criada em uma relação 1:N. A Figura 52 ilustra a solução para o problema apresentado.

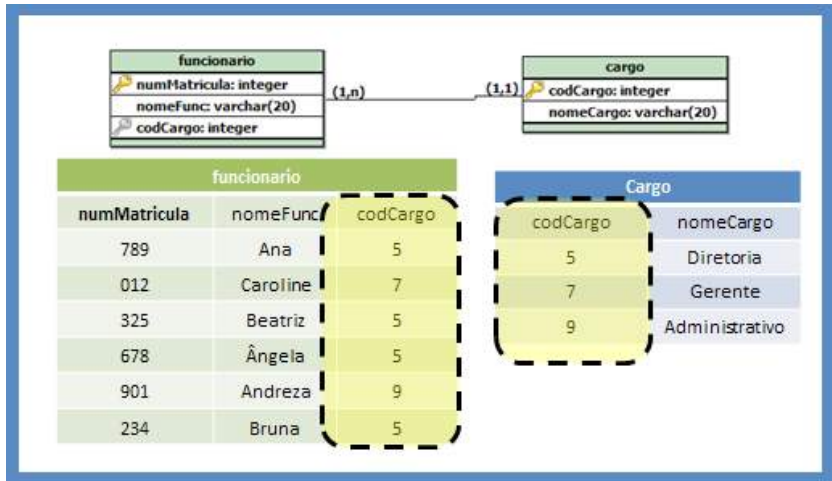


Figura 52 Solução da Terceira Forma Normal

Álgebra Relacional

A partir no desenvolvimento do modelo relacional, que definiu tabelas e relação entre tabelas na década de 1970, surgiu a necessidade de extrair informações armazenadas nesse modelo de dados. Nessa mesma época, foram estudadas diversas metodologias, porém a que mais obteve resultado significativo foi a álgebra relacional. A Álgebra Relacional é uma linguagem de consulta procedimental que possui operações que produzem, como resultado, uma relação contendo, como resposta, a pesquisa definida. Essa linguagem é a base do funcionamento de praticamente todas as instruções da linguagem SQL, que atualmente é a linguagem mais utilizada para manipulação de dados em um banco de dados relacional.

Ela possui uma série de operadores, mas os mais significativos são 6:

- ➔ Seleção
- ➔ Projeção
- ➔ Produto Cartesiano
- ➔ União
- ➔ Diferença
- ➔ Junção

Como forma de realizar exemplos relacionados aos cinco principais operadores, a Figura 53 apresenta duas tabelas populadas, que servirão como base para exemplificar cada um dos operadores.

Agencia		Contas		
nomeAge	cidadeAge	nomeAge	numConta	nomeCli
Assis Brasil	São Paulo	Ipiranga	777	Ana
Farrapos	Caxias	Farrapos	888	Maria
Ipiranga	São Paulo	Ipiranga	999	Carla
		Farrapos	111	Carolina
		Ipiranga	222	Bruna

Figura 53 Tabelas base para exemplificar operadores relacionais

Seleção

Esse operador tem como função selecionar tuplas que satisfazem uma dada condição (predicado). Sua aplicação produz um subconjunto horizontal de uma relação. Sua notação é:

$$\sigma_{\langle \text{predicado} \rangle} (\langle \text{relação} \rangle)$$

Onde o $\langle \text{predicado} \rangle$ permite o uso de operadores relacionais. A Figura 54 apresenta um exemplo da aplicação da operação de **seleção** para selecionar as tuplas da relação **contas** em que o nome da agência seja “Farrapos”

$\sigma_{\text{nomeAge} = \text{“Farrapos”}} (\text{Contas})$		
Contas		
nomeAge	numConta	nomeCli
Farrapos	888	Maria
Farrapos	111	Carolina

Figura 54 Exemplo da aplicação de Seleção

Projeção

O operador de projeção seleciona atributos de interesse produzindo um subconjunto vertical de uma determinada relação. Sua notação é:

$$\pi_{\langle \text{lista_atributos} \rangle} (\langle \text{relação} \rangle)$$

No exemplo apresentado na Figura 55, são listados apenas os campos de “numConta” e “nomeCli” da relação Contas.

$\pi_{\text{numConta, nomeCli}} (\text{Contas})$	
Contas	
numConta	nomeCli
777	Ana
888	Maria
999	Carla
111	Carolina
222	Bruna

Figura 55 Exemplo da aplicação de Projeção

Produto Cartesiano

O produto cartesiano compreende a combinação de tuplas para obtenção de dados presentes em duas ou mais relações. O produto cartesiano literalmente executa uma multiplicação dos registros envolvidos nas relações. Sua notação é:

(<Relação 1> X <Relação 2>)

Sua resposta é um conjunto contendo todas as colunas que envolvem as relações. A Figura 56 apresenta o produto cartesiano entre as tabelas Contas e Agência.

(Contas X Agencia)				
nomeAge	numConta	nomeCli	nomeAge	cidadeAge
Ipiranga	777	Ana	Assis Brasil	São Paulo
Farrapos	888	Maria	Assis Brasil	São Paulo
Ipiranga	999	Carla	Assis Brasil	São Paulo
Farrapos	111	Carolina	Assis Brasil	São Paulo
Ipiranga	222	Bruna	Assis Brasil	São Paulo
Ipiranga	777	Ana	Farrapos	Caxias
Farrapos	888	Maria	Farrapos	Caxias
Ipiranga	999	Carla	Farrapos	Caxias
Farrapos	111	Carolina	Farrapos	Caxias
Ipiranga	222	Bruna	Farrapos	Caxias
Ipiranga	777	Ana	Ipiranga	São Paulo
Farrapos	888	Maria	Ipiranga	São Paulo
Ipiranga	999	Carla	Ipiranga	São Paulo
Farrapos	111	Carolina	Ipiranga	São Paulo
Ipiranga	222	Bruna	Ipiranga	São Paulo

Figura 56 Exemplo da aplicação do Produto Cartesiano

Observe que, para cada registro da tabela de Contas, foi gerada uma relação com todos os registros da tabela Agência. Dessa forma, se existir, como é o caso do exemplo da Figura 53, cinco registros na tabela de Contas e três registros na tabela de Agência, terá um total de quinze registros como resposta ($5 \times 3 = 15$).

Sob o ponto de vista de produto cartesiano, está correta a informação apresentada na Figura 56, porém, sob o ponto de vista do conteúdo apresentado, somente 5 linhas das 15 possuem a informação correta. Sendo assim, podemos utilizar uma seleção contendo um predicado que relaciona as tabelas contidas no produto cartesiano. A Figura 57 apresenta a aplicação de uma regra sobre o produto cartesiano.

$\sigma_{\text{contas.nomeAge} = \text{agencia.NomeAge}} (\text{Contas X Agencia})$				
nomeAge	numConta	nomeCli	nomeAge	cidadeAge
Ipiranga	777	Ana	Ipiranga	São Paulo
Farrapos	888	Maria	Farrapos	Caxias
Ipiranga	999	Carla	Ipiranga	São Paulo
Farrapos	111	Carolina	Farrapos	Caxias
Ipiranga	222	Bruna	Ipiranga	São Paulo

Figura 57 Exemplo da aplicação do Produto Cartesiano com regra de predicado

A Figura 58 apresenta um refino ainda maior. Nesse exemplo, estamos apresentando apenas o nome do cliente e o nome da agência que está relacionada a conta do cliente.

$\pi_{\text{nomeCli}, \text{cidadeAge}} (\sigma_{\text{contas.nomeAge} = \text{agencia.NomeAge}} (\text{Contas X Agencia}))$	
nomeCli	cidadeAge
Ana	São Paulo
Maria	Caxias
Carla	São Paulo
Carolina	Caxias
Bruna	São Paulo

Figura 58 Exemplo da aplicação do Produto Cartesiano

União

O operador união une tuplas de duas relações que sejam compatíveis. Nesse caso, que possuam, o mesmo tipo de dado como resposta. A notação de união é:

$$\langle \text{relação1} \rangle \cup \langle \text{relação2} \rangle$$

O exemplo apresentado na Figura 59 ilustra a implementação da união de uma consulta de todos os clientes e o nome de todas as agências. A informação é apresentada em uma única coluna, pois as informações de nome de cliente e nome da agência são compatíveis (tipo de dados igual, nesse caso, texto).

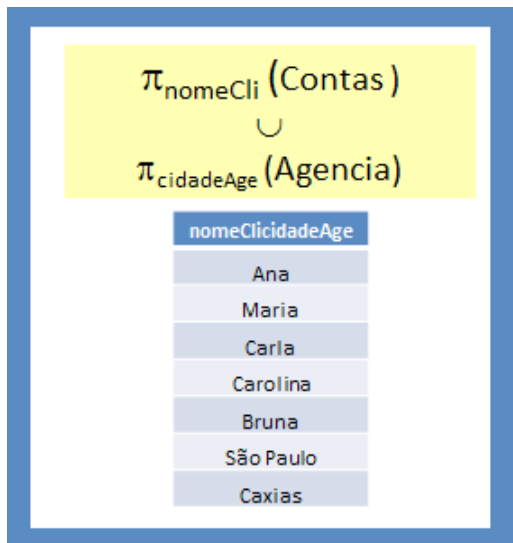


Figura 59 Exemplo da aplicação de União

Diferença

O operador diferença retorna as tuplas de uma relação 1 cujos valores dos atributos não estão presentes em uma relação 2. Sua notação é:

$$\langle \text{relação1} \rangle - \langle \text{relação2} \rangle$$

A Figura 60 apresenta um exemplo da aplicação do operador diferente mostrando como resposta o nome das agências da tabela de Agência que não são relacionadas na tabela de Contas.

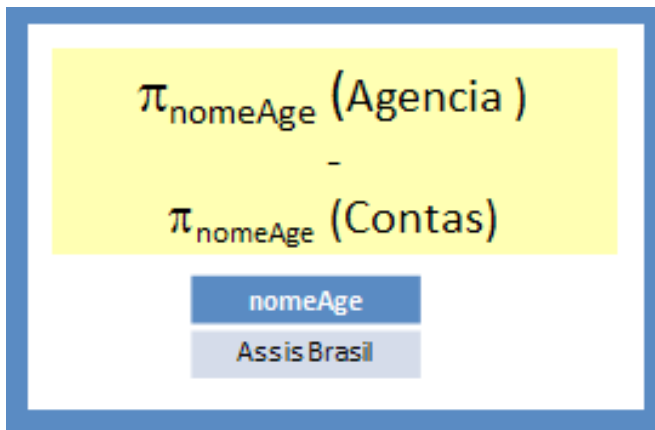


Figura 60 Exemplo da aplicação de União

Junção

A junção é a combinação dos operadores de produto cartesiano e seleção (retorna as tuplas de um produto cartesiano que satisfazem uma dada condição, onde a condição é um predicado). Sua notação é:

$$\langle \text{relação1} \rangle [X]_c \langle \text{relação2} \rangle$$

O exemplo apresentado na Figura 61 lista os dados das tabelas de Contas e Agência, que atendam à condição de o nome da agência da tabela de Contas ser o mesmo que o campo de nome de agência da tabela de Agência.

Contas [X] <small>contas.nomeAge = agencia.NomeAge</small> Agencia				
nomeAge	numConta	nomeCli	nomeAge	cidadeAge
Ipiranga	777	Ana	Ipiranga	São Paulo
Farrapos	888	Maria	Farrapos	Caxias
Ipiranga	999	Carla	Ipiranga	São Paulo
Farrapos	111	Carolina	Farrapos	Caxias
Ipiranga	222	Bruna	Ipiranga	São Paulo

Figura 61 Exemplo da aplicação de Junção

A junção é muito semelhante ao produto cartesiano, a diferença é que a junção necessita de uma relação, enquanto o produto cartesiano não.

Recapitulando

Ao término deste capítulo, o aluno compreendeu a última etapa do processo de banco de dados a qual define de forma física o banco. Também entendeu o quanto a normalização de dados é importante para garantir a integridade dos dados armazenados.

Para isto, é necessário aplicar, caso necessário, as três formas normais em cada entidade, o que poderá resultar na criação de outras entidades relacionando-as com as entidades de origem.

Por fim, foi feito um estudo sobre álgebra relacional que é utilizada como base por muitas linguagens DML relacionais como SQL. Nesse estudo, foram abordados os operadores de seleção, projeção, produto cartesiano, união diferença e junção.

Atividades

- 1) Assinalar (V) para as assertivas Verdadeiras e (F) para as Falsas.
 - a) () No modelo físico, definimos a estrutura de banco de dados tendo como base o modelo conceitual.
 - b) () Somente poderá ocorrer a aplicação da segunda forma normal se a tabela possuir uma chave primária composta e um outro atributo que dependa apenas de um campo da chave primária composta.
 - c) () Em alguns casos, quando aplicado a normalização, poderá ocorrer uma diminuição do número de tabelas de um diagrama relacional.

d) () O produto cartesiano retornará as tuplas de uma relação 1 cujos valores aparecem na relação 2.

- 2) Tendo como base o conjunto de elementos apresentado na Figura que segue, marque a alternativa que representa a aplicação de álgebra relacional para a consulta: “liste o nome da cidade de todas as agências que possuam relação com contas”.

Agencia	
nomeAge	cidadeAge
Assis Brasil	São Paulo
Farrapos	Caxias
Ipiranga	São Paulo

Contas		
nomeAge	numConta	nomeCli
Ipiranga	777	Ana
Farrapos	888	Maria
Ipiranga	999	Carla
Farrapos	111	Carolina
Ipiranga	222	Bruna

- a) σ contas.nomeAge = agencia.NomeAge (Contas X Agencia)
- b) π contas.nomeAge = agencia.NomeAge (Contas X Agencia)
- c) $\pi_{cidadeAge}$ (Contas [X] $_{contas.nomeAge = agencia.NomeAge}$ Agencia)
- d) $\sigma_{cidadeAge}$ (Contas [X] $_{contas.nomeAge = agencia.NomeAge}$ Agencia)
- e) π contas.nomeAge = agencia.NomeAge (Contas – Agencia)

Gabarito:

- 1) a) F (na verdade, o modelo base do modelo físico é o modelo lógico)
 b) V
 c) F (nunca após a aplicação de normalização teremos uma redução na quantidade de entidades)
 d) F (um produto cartesiano combina os elementos de duas relações)
- 2) C

Introdução a SQL

➔ **N**este capítulo, serão abordadas algumas considerações sobre o modelo físico, as quais são importantes para a correta aplicação de comandos SQL. Também serão abordados conceitos sobre a linguagem de consulta estruturada SQL, além dos principais comandos para criação de manutenção de tabelas.

Ao término deste capítulo, o aluno terá conhecimento necessário para construir tabelas e relações entre tabelas utilizando a linguagem SQL.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Considerações sobre Modelo Físico

Como já constatado, o processo de criação de banco de dados passa pelas etapas de modelagem conceitual, transformação do modelo conceitual para o modelo lógico e, por fim, a implementação para o modelo físico.

Muitos profissionais não seguem esse processo, ignoram a etapa de modelagem conceitual e partem diretamente para o modelo lógico. Essa ação não inviabiliza o desenvolvimento de um sistema, porém é indicada para profissionais que tenham experiência significativa na implementação de banco de dados, pois muitas premissas podem não ser identificadas e posteriormente tornar o processo de implementação mais demorado, ou até mesmo invalidado. As principais ferramentas disponibilizadas pelos fabricantes dos SGBD implementam apenas diagramas lógicos e não conceituais. O principal fator para que isso ocorra está relacionado ao fato destas já aplicarem os tipos de dados suportados pelo seu banco de dados e não de seu concorrente.

Inquestionavelmente, iniciar um projeto no modelo físico é extremamente desaconselhável, pois o desenvolvedor terá uma grande probabilidade de desenvolver algo errado ou perder um tempo significativo analisando o banco para geração de um relatório por exemplo. E esse cenário piora na medida em que exista mais de um desenvolvedor ou um desenvolvedor tenha que ser substituído no meio do projeto. Inevitavelmente, o custo do projeto será elevado. Um bom diagrama é o projeto (documento) indispensável de um administrador de banco de dados.

Somente de posse do diagrama (planta do nosso sistema de banco de dados), partimos para o processo codificação do banco de dados, atividade esta que faz parte do modelo físico.

A primeira tarefa a ser executada na transformação do modelo lógico para o modelo físico é a de identificar os tipos de dados que o SGBD no qual selecionamos como o banco de dados de nosso projeto suporta. Neste material, iremos trabalhar com o SGBD Oracle, que é considerado um dos melhores SGBD suportando muitas das funcionalidades que iremos trabalhar na linguagem SQL. Independentemente do banco de dados, a grande parte dos conceitos que serão apresentados utilizando o banco de dados Oracle podem ser aplicados nos demais SGBD como MySQL, SQLServer, BD2, Firebird, entre outros.

O Oracle possui uma série de tipos de dados em seu dicionário de dados. Como o objetivo deste material é tratar de forma mais generalizada possível as atividades por ela proposta, serão utilizados os tipos de dados mais comuns. A Tabela 1 apresenta os tipos que servirão de base para nossas atividades.

Tipo de dado	Descrição
integer	Armazena números inteiros. Exemplo: idade integer
number(precisão,escala)	Armazena números reais. Nesse tipo de dados, é possível que seja definido a precisão e a escala do número. Exemplo: salario number(15,2), onde a precisão são 15 dígitos representativos antes da vírgula e 2 usados como escala (números representativos após a vírgula).
Date	Armazena datas. O formato do armazenamento do tipo de dados data depende da configuração de cada instalação Oracle. Por padrão, quando o Oracle é instalado, ele utiliza o mesmo formato configurado de tipo de data do seu computador. Nos exemplos em que trataremos neste material, o tipo data terá o formato "dia/mês/ano". Exemplo: datanascimento date Obs.: cada banco de dados representa o tipo data de forma diferente, por exemplo, o Mysql possui como formato padrão de data "ano-mês-dia".

Tipo de dado	Descrição
<code>varchar(tamanho)</code> e <code>varchar2(tamanho)</code>	Armazena dados alfanuméricos, onde tamanho define a quantidade máxima de caracteres que pode ser armazenada no campo especificado por esse tipo de dados. O tipo <code>varchar2</code> tem como principal característica armazenar apenas a quantidade específica de bytes que seu conteúdo necessita para ser armazenado. Por exemplo, quando definimos que o máximo a ser armazenado é 30 (<code>nome varchar2(30)</code>), o máximo que o atributo <code>nome</code> poderá armazenar são 30 caracteres, porém, se for informado no campo <code>nome</code> "Ana", somente serão alocados 3 bytes, diferentemente do tipo de dado <code>varchar</code> , que ao definir seu tamanho máximo como 30 caracteres (<code>varchar(30)</code>), e informar no campo <code>nome</code> "Ana", o SGBD, aloca 30 bytes mesmo este possuindo apenas 3. Exemplo: <code>nome varchar(30)</code> <code>endereco varchar2(40)</code>

Tabela 1 Tipos de dados que serão utilizados

SQL (Structured Query Language)

O SQL que em português significa linguagem de consulta estruturada, surgiu em 1974, foi desenvolvida nos laboratórios da IBM e teve como base um artigo de 1970 escrito por Edgar Frank Codd. No decorrer do tempo, muitas linguagens foram criadas, mas o SQL tornou-se a mais popular. Em 1986, o *American National Standard Institute* (ANSI) definiu um padrão

para o SQL, o que possibilitou que essa linguagem pudes-se ser aplicada dentro de aplicativos independentemente do SGBD que se esteja conectado.

A linguagem SQL implementa quatro operações básicas de manipulação de registros em Banco de Dados: criação, leitura, alteração e exclusão de dados, convencionalmente conhecidos como CRUD ("Create, Read, Update e Delete"). O SQL também implementa comandos de criação de banco, tabelas e alteração na estrutura dessas tabelas.

Neste capítulo, será abordado a criação e manipulação de tabelas e relações utilizando a sintaxe do banco de dados Oracle. A sintaxe desses comandos pode variar de um banco para o outro, mas o objetivo de cada comando é o mesmo.

Tendo como base o modelo lógico apresentado na Figura 62, observamos a existência de duas entidades, funcionário e departamento, relacionadas a partir do campo de código de departamento (codigoDep).

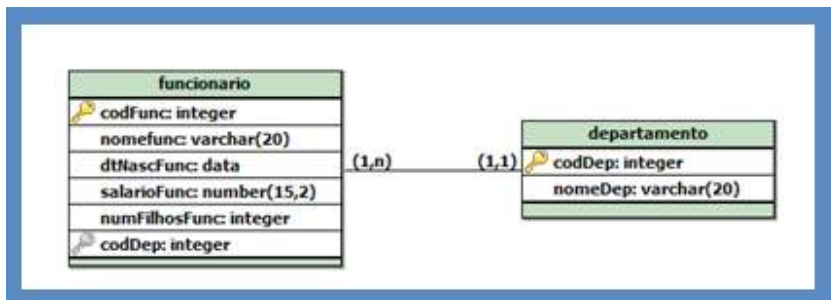


Figura 62 Exemplo base de tabelas relacionadas

Cada uma das tabelas deve ser criada com suas respectivas chaves primárias, além da definição da relação existente. Como forma de exemplificação, será tratada cada uma das atividades de forma separada, e no final deste capítulo, serão apresentados exemplos de criação de entidades, já especificando todas as relações existentes em um único comando SQL.

Comando para criar tabela

O comando **Create Table** deve ser utilizado quando é necessário criar uma nova tabela. Sua sintaxe é composta por:

```
Create table <nome_da_tabela>(  
    <atribulo 1> <tipo do atributo1> <Definições do atributo1>,  
    <atribulo 2> <tipo do atributo2>,<Definições do atributo2>,  
    : :  
    <atribulo n> <tipo do atributo n>);
```

Onde:

- **Nome da tabela:** representa o nome dado à entidade;
- **Atributo:** identifica o nome do atributo a ser criado;
- **Tipo de atributo:** define qual será o tipo de informação a ser armazenada em cada um dos atributos;
- **Definições do atributo:** são parâmetros que podem ser atribuídos a cada atributo, como, por exemplo, se este é obrigatório ou não (NOT NULL), de se este armazena um valor único em seu registro (UNIQUE), valores pa-

drões (DEFAULT) ou até mesmo restrições como chaves primárias, estrangeiras, além de validações (CHECK).

Com base na sintaxe definida, segue a implementação das entidades existentes na Figura 62 (funcionário e departamento) utilizando a sintaxe SQL nos exemplos 5 e 6:

Exemplo 5: "Criação da tabela departamento"

```
create table departamento (  
  codDep integer not null,  
  nomeDep varchar2(20) UNIQUE  
);
```

Exemplo 6: "Criação da tabela funcionario"

```
create table funcionario (  
  codFunc integer not null,  
  nomeFunc varchar2(20) not null,  
  dtNascFunc date,  
  salarioFunc number(15,2) default 0,  
  numFilhosfunc integer default 0,  
  codDep integer not null  
);
```

Obs.: Sempre que for criado um atributo de chave estrangeira, o seu tipo deverá ser o mesmo da chave primária que ele referencia. Nesse caso, observe o atributo chamado **codDep** da tabela **funcionário**. Esse atributo é do tipo **integer**, pois o campo **codDep** da tabela **departamento** é do tipo **integer**.

É importante destacar que os atributos que serão definidos como chaves primárias devem ser definidos como NOT NULL, ou seja, são atributos obrigatórios, não permitindo que seja informado nesse atributo um valor vazio. Nessa transformação, além das chaves primárias, o campo de nome de funcionário também foi definido como um campo obrigatório. O atributo **coddep** da tabela de funcionário será um atributo de chave estrangeira. Isso é reflexo da obrigatoriedade identificada no diagrama da Figura 62, em que a cardinalidade mínima é 1. É importante ressaltar que um atributo de chave estrangeira não possui como característica a obrigatoriedade em sua implementação. Isso depende da cardinalidade mínima da relação existente com a entidade na qual ele referencia.

Também pode ser identificado a existência da definição de atributo DEFAULT como 0(zero) nos atributos de **salarioFunc** e **numFilhosFunc** da tabela de funcionário. Nesse caso, se for inserido ou alterado um registro na tabela de funcionário e o conteúdo desses campos seja vazio, o SGDB definirá estes como 0 (zero), ou seja definiu um valor padrão para os dados caso não seja informado uma informação.

Adicionando mais um atributo à tabela existente

Todas as vezes que é necessário alterar alguma informação da estrutura de uma tabela, deve ser executado o comando **ALTER TABLE**. Caso ocorra a necessidade de adicionar mais um atributo à tabela, utilizamos a cláusula **ADD** junto ao comando **ALTER TABLE**. A sintaxe do comando para adicionar um novo atributo a tabela é:

```
Alter Table <nome_da_tabela>  
  add <nome da nova coluna> <tipo do novo atributo> <Definições do atributo>
```

Por exemplo, se quiséssemos adicionar mais um campo à tabela de funcionário chamado de *email* do tipo `varchar2(50)` o comando seria:

Exemplo 7: "Adicionando um novo atributo a uma tabela já existente"

```
alter table funcionario  
  add email varchar2(50);
```

Alterando o nome de um atributo já existente

A sintaxe do comando para alterar o nome de um atributo já existente é:

```
Alter Table <nome_da_tabela>  
rename column <nome do atributo antigo> TO <novo nome do atributo>;
```

Por exemplo, vamos imaginar que o atributo *email* tenha sido criado com o nome errado e o correto seria *emailFunc*. Para isso, executaríamos o Exemplo 8 que segue.

Exemplo 8: "Alterando o nome de um atributo de uma tabela"

```
alter table funcionario  
rename column email to emailFunc;
```

Alterando o tipo de dado de um atributo já existente

Para alterar um tipo de dado de um atributo já existente, utiliza-se a seguinte sintaxe:

```
Alter Table <nome_da_tabela>  
modify <nome do atributo> <novo tipo de dado>;
```

Como forma de exemplificar uma alteração de tipo de dado, vamos imaginar que o tamanho do campo emailFunc deva ser alterado para varchar2(100) ao invés de 50. O Exemplo 9 apresenta a solução para essa alteração.

Exemplo 9: “Alterando o tipo de dado de um atributo já criado”

```
alter table funcionario  
  modify emailFunc varchar2(100);
```

Obs.: No **caso de alteração** de tipo de dado, **o banco** de dados **somente irá permitir a alteração** do tipo de um atributo, **se o novo atributo suportar os dados do antigo atributo**. Por exemplo, caso seja necessária a alteração de dados do campo numFilhosFunc que é integer para number(10,2), o banco irá aceitar o comando. Porém, se executado uma linha de comando que tente alterar o campo salariofunc do tipo number(15,2) para o tipo date, o banco não conseguirá executar esse comando. Isso também pode ser identificado em atributos do tipo VARCHAR, quando é necessário alterar o tamanho do campo. Caso queira aumentar o tamanho de um campo, não haverá problema algum, mas, se for necessário diminuir o tamanho do campo e exista um conteúdo armazenado no campo que possua uma quantidade inferior de dados que o novo tamanho definido pelo VARCHAR, o banco não permitirá a alteração.

Removendo um atributo de uma tabela

Caso queira remover um atributo de uma tabela, basta executar a sintaxe que segue:

```
Alter Table <nome_da_tabela>  
DROP column <nome do atributo>;
```

O exemplo que segue apresenta a sintaxe para remoção do atributo emailFunc.

Exemplo: “Removendo um atributo de uma tabela”

```
alter table funcionario  
drop column emailFunc;
```

Criando Chave Primária

Por definição, toda tabela possui um atributo que o identifique de maneira única entre os demais registros do banco. Esse atributo deve ser definido como o atributo chave, ou como definimos em banco de dados, chave primária de uma entidade. Assim, toda entidade deve possuir uma chave primária. A chave primária pode ser definida na criação da tabela a partir do comando CREATE TABLE, ou após a tabela já ter sido cria-

da por meio de ALTER TABLE. A sintaxe para criação de uma chave primária depois de criada uma tabela é:

```
Alter Table <nome_da_tabela>  
<nome_da_constraint> PRIMARY KEY(<atributo_chave_primária>);
```

Onde:

- **nome_da_constraint**: é um identificador definido pelo usuário e serve para ter maior controle sobre as operações de banco de dados. Nesse caso, para definir o identificador no banco de dados que faz referência à chave primária. A partir dessa constraint, por exemplo, é possível remover uma chave primária de uma tabela.
- **atributo_chave_primária**: é o atributo da entidade que será definido como chave primária. Em caso de chave primária composta, devem ser informados os atributos separados por vírgula.

No exemplo da Figura 62, existem 2 atributos de chave primária, o primeiro chamado codDep da tabela de departamento e o segundo codFunc da tabela de funcionário. Os Exemplos 10 e 11 demonstram a definição das chaves primárias utilizando o comando ALTER TABLE.

Exemplo 10: "Criando chaves primárias através de ALTER TABLE"

```
alter table departamento add constraint departamento_pk primary  
key(codDep);
```

```
alter table funcionario add constraint tbfuncionario_pk primary  
key(codfunc);
```

*Obs.: Lembre-se de que os atributos que são definidos como chaves primárias não devem suportar conteúdos vazios. Dessa forma, devem ser definidos como **NOT NULL**. Isso pode ser observado nos Exemplos 5 e 6.*

Os Exemplos 11 a 14 apresentam alternativas de definição de chave primária no comando CREATE TABLE.

Exemplo 11: “Criando chave primária por meio de CREATE TABLE”

```
create table cidade(  
    codcid integer not null,  
    nomecid varchar(30) not null,  
    uf varchar(2),  
    primary key(codcid)  
);
```

Exemplo 12: “Criando chave primária por meio de CREATE TABLE”

```
create table cidade(  
    codcid integer not null,  
    nomecid varchar(30) not null,  
    uf varchar(2),  
    constraint cidade_pk primary key(codcid)  
);
```

Exemplo 13: “Criando chave primária por meio de CREATE TABLE”

```
create table cidade(  
    codcid integer not null,  
    nomecid varchar(30) not null,  
    uf varchar(2)  
);  
  
alter table cidade add constraint cidade_pk primary key (codcid);
```

Exemplo 14: “Criando chave primária por meio de CREATE TABLE”

```
create table cidade(  
    codcid integer not null constraint cidade_pk primary key,  
    nomecid varchar(30) not null,  
    uf varchar(2)  
);
```

Observe que, no Exemplo 11, a constraint não foi definida, porém a chave primária foi criada. Na criação de chaves primárias dentro do comando CREATE TABLE, não existe obrigatoriedade da definição do nome da constraint, porém as boas práticas sugerem o uso de um nome.

Criando Chave Estrangeira

As chaves estrangeiras são atributos criados para definir de forma física uma relação entre duas entidades. O conteúdo de

uma chave estrangeira deve ser identificado (existir) no atributo da chave primária que o atributo referencia. Da mesma forma que a criação da chave primária, a chave estrangeira pode ser definida tanto na criação da tabela, ou posterior à criação da mesma. A sintaxe para criação de uma chave estrangeira em uma tabela já criada utilizando ALTER TABLE é:

```
alter table <nome_da_tabela> ADD CONSTRAINT  
<nome_da_constraint> FOREIGN KEY(<atributo_chave_estrangeira>  
REFERENCES <tabela_referenciada>(<atributo_chave_primária_referenciado>);
```

Onde:

- nome_da_constraint: é um identificador que deve ser definido pelo usuário como forma de identificar a constraint, nesse caso, a chave estrangeira;
- atributo_chave_estrangeira: é o atributo criado como chave estrangeira da relação;
- tabela_referenciada>(<atributo_chave_primária_referenciado>): deve ser informada a tabela e o atributo de chave primária na qual a chave estrangeira fará referência.

No exemplo apresentado na Figura 62, existe o atributo codDep da tabela de funcionário fazendo referência ao atributo codDep da tabela de Departamento. O Exemplo 15 apresenta a solução a partir do comando ALTER TABLE para criação da chave estrangeira.

Exemplo 15: "Criando Chave estrangeira em uma tabela já existente"

```
alter table funcionario add constraint funcionario_codDepFK  
foreign key(codDep) references departamento(codDep);
```

Obs.: A definição de uma chave estrangeira somente ocorrerá se a chave primária da tabela em que a chave estrangeira fizer referência já tenha sido criada.

O Exemplo 16 apresenta a definição da chave estrangeira na criação da tabela de funcionário. Observe que, nesse exemplo, ambas as chaves estão sendo definidas na criação da tabela. Nesse caso, o comando somente será executado se a tabela departamento já tenha sido criada, juntamente com a sua chave primária codDep.

Exemplo 16: "Criando Chave estrangeira no comando CREATE TABLE"

```
create table funcionario (  
    codFunc integer not null,  
    nomeFunc varchar2(20) not null,  
    dtNascFunc date,  
    salarioFunc number(15,2) default 0,  
    numFilhosfunc integer default 0,  
    codDep integer not null,  
    constraint tbfuncionario_pk primary key(codfunc),  
    constraint funcionario_codDepFK foreign key(codDep)  
references departamento(codDep)  
)
```

Comandos úteis para trabalhar com estrutura de tabelas

Como forma de auxiliar o usuário a encontrar informações sobre a estrutura das tabelas, a Tabela 2 apresenta alguns comandos úteis ao administrador de banco de dados relacionados à estrutura de tabelas.

Descrição	Comando
Lista as Meta-informações sobre uma tabela	DESCRIBE nome_da_tabela ou DESC nome_da_tabela
Listar as tabelas criadas por um usuário	SELECT * FROM USER_TABLES
Listar as restrições criadas por um usuário	SELECT * FROM USER_CONSTRAINTS
Apagar uma constraint	ALTER TABLE nome_da_tabela DROP CONSTRAINT nome_da_constraint
Apagar uma tabela	DROP TABLE nome_da_tabela

Tabela 2 Comandos úteis para o administrador de banco

Recapitulando

Ao concluir este capítulo, foi possível compreender que iniciar um projeto já codificando banco de dados não é a melhor alternativa de desenvolvimento de banco, e poderá ocasionar um significativo aumento de custo do projeto.

Também foi abordada a necessidade de conhecer os tipos de dados de um banco de dados selecionado para um determinado projeto. O tipo de dados é indispensável para correta implementação do modelo físico.

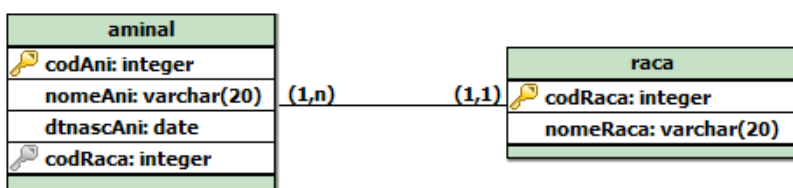
Com base em alguns comandos da linguagem SQL, foi possível conhecer como é possível criar uma tabela, alterar o nome e tipo de atributo, além de entender a forma de como se adiciona e remove atributos de uma tabela. Utilizando os comandos SQL, foi possível definir as chaves primárias e estrangeiras, determinando uma relação física entre as tabelas criadas. Por fim, foi apresentado uma relação de alguns comandos úteis para a manipulação da estrutura de tabelas.

Atividades

- 1 - Assinalar (V) para as assertivas Verdadeiras e (F) para as Falsas.
- a) () Para a correta implementação do modelo físico, é necessário que exista um diagrama lógico definido.
 - b) () O SQL obteve um padrão pela ISO em 1986, onde, a partir daí, se tornou popular e utilizado até hoje.

- c) () É possível criar uma chave primária ou estrangeira no próprio comando CREATE TABLE.
- d) () Só é possível criar uma chave estrangeira se a chave primária a qual a chave estrangeira estiver referenciando já estiver sido criada.

- 2) Tendo como base o diagrama lógico que segue, e sabendo que a tabela de raça juntamente com a sua chave primária já esteja definida, marque a alternativa que não poderá ser executada.



- a) CREATE TABLE aminal (
codAni integer not null,
nomeAni varchar(20) not null,
dtnascAni date not null
);
alter table aminal add codRaca integer not null;

- b) CREATE TABLE aminal (

```
codAni integer not null,  
nomeAni varchar(5) not null,  
dtnascAni date not null,  
codRaca integer not null  
);  
  
alter table aminal modify codRaca varchar(20) not null;
```

```
c) CREATE TABLE aminal (  
    codAni integer not null,  
    nomeAni varchar(20) not null,  
    dtnascAni date not null,  
    codRaca integer not null  
)
```

```
d) CREATE TABLE aminal (  
    codAni integer not null PRIMARY KEY,  
    nomeAni varchar(20) not null,  
    dtnascAni date not null,  
    codRaca integer not null,
```

```
constraint codRacaFK foreign key(codRaca) references  
raca(codRaca)  
)
```

e) `alter table raca drop nomeRaca varchar(20);`

Gabarito:

1) V – F – V – V; 2) e (comando mal aplicado, falta cláusula column e não é necessário colocar o tipo de dado do atributo a ser removido)

Christiano Cadoná*

Capítulo 8

Operações CRUD

➔ **E**ste capítulo abordará, por meio de exemplos, a utilização dos comandos de seleção, inserção, alteração e remoção de dados os quais compõem as quatro operações básicas de banco de dados pelas aplicações desenvolvidas.

Após a conclusão deste capítulo, o aluno terá condições de operacionalizar as tabelas de um sistema de banco de dados.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Operações com dados de uma tabela

Quem desenvolve aplicativos geralmente encontra muitos artigos de acesso e manipulação de registros relacionados com o termo CRUD.

CRUD, é um termo utilizado (convencionado) para designar as quatro operações básicas em Banco de Dados, criação, leitura, alteração e exclusão de dados ("Create, Read, Update e Delete"). A linguagem SQL possui as quatro operações com os comandos INSERT, SELECT, UPDATE e DELETE.

A definição do padrão em SQL proporcionou que esses operadores possam ser aplicados independentemente do SGBD selecionado. Dessa forma, a sintaxe que será trabalhada nos comandos de inserção, seleção, alteração e exclusão é a mesma para qualquer banco de dados. A única variação está relacionada à utilização de funções na operação de seleção, que depende de cada SGBD. Em geral todos os SGBD possuem funções para realização de uma determinada ação, porém o nome ou os parâmetros destas podem variar de acordo com o banco de dados. Como já acordado no capítulo anterior, será utilizado o SGBD ORACLE e, por consequência, a sintaxe de suas funções.

Como forma de apresentar conceitos e exemplos relacionados aos comandos que serão apresentados no decorrer deste capítulo, será utilizado o diagrama lógico da Figura 62 já trabalhado no capítulo anterior.

Inserindo dados em uma tabela

A operação de inserção é responsável por popular às tabelas. Sua sintaxe é:

```
insert into <nome_da_tabela>(<atributo_1>,<atributo_2>,...,<atributo_n>)  
values (<valor_do_atributo_1>,<valor_do_atributo_2>,...,<valor_do_atributo_n>);
```

Os Exemplos 17 e 18 apresentam a execução do comando INSERT na tabela de departamento e na tabela de funcionário.

Exemplo 17: "Inserindo um novo registro na tabela de departamento"

```
insert into departamento(coddep,nomedep) values (1,  
'Comercial');
```

Exemplo 18: "Inserindo um novo registro na tabela funcionario"

```
insert into funcionario(codfunc, nomefunc, dtnascfunc,  
salariofunc, numfilhosfunc, coddep) values(50, 'Ana  
Maria', '20/03/1995', 1234.32, 2, 1)
```

*Obs.: Para cada tipo de dado, existe uma representação de valor. Por exemplo, dados do tipo **integer** devem ser informados somente por meio de números. É o caso do valor dado ao `codDep(1)`, `codFunc(50)` e do campo `numFilhosDep(2)`. Tipo de dados **VARCHAR**, deve ser informado colocando seu conteúdo dentro de apóstrofo ('), como é o caso dos campos `nomeDep('Comercial')` e `nomeFunc('Ana Maria')`. Já atributos do tipo **DATE** deve ser informado entre apóstrofo (') com formatação de acordo como o seu SGBD está configurado. No Exemplo 17, percebemos que a data está formatada em 'dia/mês/ano', e por isso, no campo `dtNascFunc`, o valor foi informado como ('20/03/1995'). Os dados do tipo **number** devem possuir, como separador decimal, um ponto ao invés da vírgula que estamos acostumados. No Exemplo 17, campo `salarioFunc` possui como valor 1234.32 (um mil duzentos e trinta e quatro reais com trinta e dois centavos).*

Observe também que o valor contido no campo `codDep` do Exemplo 17, já estava cadastrado na tabela de departamento (Exemplo 16). Se o banco de dados estivesse vazio e os únicos comandos utilizados fossem os 16 e 17, não seria possível cadastrar um novo funcionário com um código de departamento diferente de 1, pois somente existe esse registro cadastrado na tabela e departamento.

A ordem em que os atributos aparecem no comando **INSERT** influencia na ordem como os valores serão informados. O Exemplo 19 apresenta a inserção de um novo departamento com a ordem de seus campos alterada.

Exemplo 19: “Inserindo um novo registro com atributos fora da ordem original”

```
insert into departamento(nomedep ,coddep) values  
('Financeiro',2);
```

Vamos imaginar que as tabelas departamento e funcionário já tenham sido criadas, e o SQL utilizado para criação da tabela de funcionário foi o do Exemplo 16 apresentado no capítulo anterior. No Exemplo 16, existem atributos não obrigatórios e outros obrigatórios. Partindo dessa premissa, o Exemplo 20 apresenta uma situação de inserção na tabela de funcionário interessante. Nesta, não está sendo informado o campo de salarioFunc nem o campo numFilhosFunc. Isso é possível, pois, na definição da criação da tabela (CREATE TABLE), os campos não eram obrigatórios (não havia cláusula NOT NULL). Dessa forma, dizemos que acabamos de cadastrar um funcionário sem informar o seu salário e número de filhos.

Exemplo 20: “Inserindo novo registro sem alguns atributos da tabela.”

```
insert into funcionario(codfunc, nomefunc, dtnascfunc, coddep)  
values(52, 'Carla', '17/05/1990', 1);
```

O comando INSERT também possibilita que seja realizado a inserção de dados, tendo como base dados armazenados em outras tabelas. O exemplo que segue realiza a inserção de dados na tabela de departamento buscando dados em uma tabela de categoria. Nesse caso, vamos imaginar que a tabela de categoria existisse e que essa estivesse populada. Vamos ima-

ginar que a consulta realizada pelo comando SELECT na tabela de categoria retornasse 10 registros, ambos os 10 seriam inseridos na tabela de departamento. Nesse caso, torna-se importante destacar que os tipos de campo devem coincidir.

Exemplo 21: “Exemplo de INSERT com SELECT”

```
insert into departamento(coddep,nomedep)(
    select codcategoria, nomecategoria from categoria
    where codcategoria>3)
```

Selecionando dados de uma tabela

O comando de seleção de dados é o comando mais utilizado do SQL. Este possui uma série de parâmetros em sua sintaxe. Como forma de exemplificar, nesse tópico somente serão destacados os parâmetros que serão utilizados neste capítulo. Aqui, somente será utilizada a consulta sobre uma única tabela. Seleção de dados com mais de uma tabela envolvida será abordado no próximo capítulo deste material.

Sintaxe básica do comando SELECT:

```
Select <atributos_de_retorno> from <nome_da_tabela>
[where <condições> ]
[order by <atributo>]
```

Onde cláusulas entre conchetes [] somente são utilizadas caso ocorra necessidade.

Como forma de exemplificar a utilização do comando de seleção, a Figura 63 apresenta uma tabela populada com um conjunto de informações de clientes quaisquer, a qual será utilizada como base para execução do comando de seleção.

tbcliente		tbcliente				
pkCodCli: integer		pkcodCli	nomeCli	endCli	dataNascCli	salarioCli
nomeCli: varchar(25)						SexoCli
endCli: varchar(40)		1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43 F
dataNascCli: date		2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12 M
salarioCli: number(15,2)		3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89 M
sexoCli: varchar(1)		4	Pelé Rocha	Rua sei lá 222	25/07/1978	245,21 M
		5	Faustão Domingos	AV do Carmo 171	13/05/1988	433,65 M
		6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30 F
		7	Caetano Velado	Rua Condor 50	04/07/1990	294,21 M
		8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66 M

Figura 63 Exemplo base para exercícios de seleção

Quando necessitamos mostrar todos os atributos de repostas, utilizamos o "*" (asterisco) para representar o retorno de todos os atributos. A Figura 64 apresenta um exemplo e o resultado da consulta realizada utilizando o asterisco para retorno dos campos. Nesse exemplo, estamos listando todos os dados de todos registros da tabela de Cliente.

```
select * from tbcliente
```

tbcliente					
pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43	F
2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12	M
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M
4	Pelé Rocha	Rua sei lá 222	25/07/1978	245,21	M
5	Faustão Domingos	AV do Carmo 171	13/05/1988	433,65	M
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F
7	Caetano Velado	Rua Condor 50	04/07/1990	294,21	M
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

Figura 64 Exemplo de seleção utilizando *

Porém, quando necessitamos que a resposta seja apenas atributos específicos, adicionamos entre as cláusulas select e from o nome dos atributos. A Figura 65 apresenta 3 formas diferentes de listar o somente o nome e salário de todos os clientes:

- a) `select nomecli,salariocli from tbcliente;`
- b) `select tbcliente.nomecli,tbcliente.salariocli from tbcliente;`
- c) `select cl.nomecli,cl.salariocli from tbcliente cl;`

nomeCli	salarioCli
Maria Antonieta	1250,43
Bozo da Silva	734,12
Fofão Machado	2354,89
Pelé Rocha	245,21
Faustão Domingos	433,65
Xuxa Antibes	799,30
Caetano Velado	294,21
Beltrano Carlos	666,66

Figura 65 Exemplos selecionando apenas algumas colunas de uma tabela

Ambas as soluções para pesquisa apresentadas na Figura 65 retornam a mesma resposta. No exemplo “a”, apenas foram colocados os campos nomeCli e o campo salarioCli. No exemplo “b”, foi adicionando antes do nome do campo o nome da tabela que possui o campo, nesse caso, “tbcliente”. Isso é necessário quando a consulta envolve mais de uma tabela, pois, assim, identificamos de qual tabela queremos as respostas. Já o exemplo “c” possui a mesma finalidade do que o exemplo “b”, a única variação é que, nesse exemplo, ao invés de colocarmos o nome da tabela antes de cada atributo de resposta, definimos um apelido para a tabela que deve ser colocado após o nome da mesma. Esse apelido poderá ser utilizado como se fosse a tabela referenciada. No exemplo, o apelido da tabela “tbcliente” ficou definido como “cl”.

Nos exemplos das Figuras 65, o nome das colunas de retorno são os mesmos especificados em sua pesquisa, nesse caso, “nomeCli” e “salarioCli”. Caso ocorra a necessidade de alterar apenas o nome destas colunas no momento de apresentar a resposta, basta adicionar, após o nome da coluna, a cláusula “AS” seguida no nome da nova coluna. A Figura 66 apresenta um exemplo de implementação da cláusula “AS”.




```
select c.nomecli as nome ,c.salariocli as salario, c.sexoCli as "Sexo do cliente" from tbcliente c;
```

nome	salario	Sexo do cliente
Maria Antonieta	1250,43	F
Bozo da Silva	734,12	M
Fofão Machado	2354,89	M
Pelé Rocha	245,21	M
Faustão Domingos	433,65	M
Xuxa Antibes	799,30	F
Caetano Velado	294,21	M
Beltrano Carlos	666,66	M

Figura 66 Definindo nome às colunas de retorno

Observe que, no exemplo apresentado na Figura 66, caso seja necessário colocar uma palavra composta, como é o caso de “Sexo do cliente”, esta deverá aparecer entre aspas duplas.

Além de retornar os dados de uma coluna na sintaxe do SQL, é possível criar uma nova coluna de resposta utilizando ou não dados de colunas já existentes. O exemplo da Figura 67 lista o nome do cliente, seu salário e seu salário acrescido de R\$ 150.34 de todos os clientes cadastrados. O SQL suporta as quatro operações matemáticas(soma, subtração, divisão e multiplicação) as quais podem ser aplicadas a qualquer atributo numérico. O uso de parênteses para realizar expressões também é permitido.



```
select c.nomecli,c.salariocli,(c.salariocli+150.34) as novoAtributo from tbcliente c;
```

nomecli	salariocli	novoAtributo
Maria Antonieta	1250,43	1400,77
Bozo da Silva	734,12	884,46
Fofão Machado	2354,89	2505,23
Pelé Rocha	245,21	395,55
Faustão Domingos	433,65	583,99
Xuxa Antibes	799,30	949,64
Caetano Velado	294,21	444,55
Beltrano Carlos	666,66	817,00

Figura 67 Criando nova coluna de reposta

Outro recurso interessante do SQL é a possibilidade de ordenar a resposta por uma determinada coluna. Para isso, utiliza-se a cláusula ORDER BY seguida do nome das colunas que se deseja ordenar. Caso seja necessário ordenar em ordem decrescente, basta utilizar, após o nome do campo ordenado, a cláusula DESC. Os Exemplos 22 a 24 mostram formas diferentes de realizar a operação de ordenação.

Exemplo 22: “Lista todos os dados de todos os clientes em ordem de nome de cliente”

```
select * from tbcliente c order by nomecli;
```

Exemplo 23: “Lista todos os dados de todos os clientes em ordem de nome de código de cidade e, em seguida, em ordem de salário”

```
select * from tbcliente c order by c.sexoCli,c.salariocli;
```

Exemplo 24: “Lista todos os dados de todos os clientes em ordem DECRESCENTE de nome de cliente”

```
select * from tbcliente c order by c.nomecli DESC;
```

Um dos principais usos do comando de seleção está relacionado à busca por uma informação específica. Para isso, utiliza-se a cláusula WHERE seguida de uma expressão condicional contendo um operador relacional. O SQL suporta diversos operadores relacionais. A Tabela 3 apresenta os operadores relacionais do SQL.

Operador	Descrição	Exemplo	Resultado
=	Igual a	358.50=358.50	Verdadeiro
>	Maior do que	584>352	Verdadeiro
<	Menor do que	584>352	Falso
>=	Maior ou igual	125>=854	Falso
<=	Menor ou igual	125<=854	Verdadeiro
<> ou !=	Deferente de	125<>854	Verdadeiro

Tabela 3 Operadores Relacionais

A Figura 68 apresenta dois exemplos da aplicação dos operadores relacionais nos tipos de dados numéricos e de data. Observe que, da mesma forma que no comando INSERT,

cada tipo de dado deve possuir uma forma de representá-lo em uma condição.

a) Lista todos os dados do cliente de código igual á 7
select * from tbcliente c where c.pkcodcli=7

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
7	Caetano Velado	Rua Condor 50	04/07/1990	294,21	M

b) Lista todos os dados dos clientes que possuam o salário maior ou igual a 799,30
select * from tbcliente c where c.salariocli>=799.30

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43	F
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F

c) Lista todos os dados dos clientes que nasceram após 21/03/2000
select * from tbcliente c where c.datanasccli>'21/03/2000'

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43	F
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

Figura 68 Exemplos de operadores relacionais

Antes de selecionar um atributo do tipo texto, é necessário saber se esse banco de dados é sensitive, ou seja, diferencia letras maiúsculas das minúsculas. Essa informação é indispensável, pois uma pesquisa pode ser prejudicada caso não saibamos dessa informação. Por exemplo, tendo como base que o cliente “Bozo da Silva” esteja armazenado no banco de dados, e, em uma consulta SQL, foi informado como parâmetro “bozo da silva”, a consulta não retornaria nada, pois as palavras não coincidem, tendo em vista que o SGBD diferencia letras maiúsculas e minúsculas. Nesse caso, podemos utilizar a função SQL UPPER que possibilita a conversão do conteúdo de seu parâmetro todo para maiúsculo.

Os dois exemplos apresentados na Figura 69, ilustram a aplicação da função upper.

a) select * from tbcliente c where <u>upper(c.nomecli)</u>='BOZO DA SILVA';					
pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12	M

b) select * from tbcliente c where <u>upper(c.nomecli)</u>=upper('BoZo Da silVa');					
pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12	M

Figura 69 Exemplos da função UPPER

Para dados do tipo texto, o SQL também possui um operador lógico chamado LIKE que retorna verdadeiro se uma cadeia de caracteres específica corresponde a um padrão. Um padrão pode incluir caracteres normais ou curinga representado pelo carácter “%” (porcento). Esse operador possibilita consultas como, por exemplo, listar todos os clientes que iniciam com a letra “b”. A Figura 70 ilustra 3 exemplos de busca utilizando o operador LIKE.

a) Lista todos os dados dos clientes que iniciam com a letra “b”

select * from tbcliente c where upper(c.nomecli) LIKE upper('b%')

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12	M
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

b) Lista todos os dados dos clientes cujo nome termine com “os”.

select * from tbcliente c where upper(c.nomecli) LIKE upper('%os')

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
5	Faustão Domingos	AV do Carmo 171	13/05/1988	433,65	M
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

c) Lista todos os dados dos clientes que possuam em seu nome “be”

select * from tbcliente c where upper(c.nomecli) LIKE upper('%be%');

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

Figura 70 Exemplos do operador LIKE

O comando de seleção também possui uma série de outras operações como a utilização de operadores lógicos, funções de agrupamento, além da utilização de mais de uma tabela na busca de dados. Esses itens serão apresentados no próximo capítulo deste documento.

Alterando dados de uma tabela

É normal que ocorra a necessidade de alterar o conteúdo já armazenado de um registro. Para realizar esse procedimento, utilizamos o comando UPDATE. Sua sintaxe é:

```
Update <nome_da_tabela> set
<atributo_1>=<novo valor do atributo1>,
<atributo_2>=<novo valor do atributo2>,
:
<atributo_n>=<novo valor do atributo_n>,
[ where <condições>]
```

O Exemplo 23 apresenta o comando UPDATE para alterar o nome do cliente para “Maria Joaquina”, a data de nascimento para “25/08/1998”, e seu salário acrescido de mais R\$ 200,00, o cliente de código igual a 1. Observe que, no caso da alteração de salário, a expressão utilizou o antigo salário acrescido do valor 200.

Exemplo 25: “Altera os dados do cliente de código 1”

```
update tblcliente set
  nomeCli='Maria Joaquina',
  datanascCli='25/03/1998',
  salarioCli =salarioCli+200
where pkcodcli=1;
```

A cláusula WHERE é opcional ao comando UPDATE, mas a ocorrência de sua utilização é significativa. Se o usuário no Exemplo 25 não tivesse adicionado uma condição, **TODOS** os registros da tabela de cliente seriam alterados com as informações determinadas pelo Exemplo 25, o que traria, sem dúvida, um grande problema, pois todos os clientes se chamariam “Maria Joaquina” e teriam sua data de nascimento definida para “25/03/1998”, além de seu salário ser acrescido com mais R\$ 200,00.

Mas o uso de UPDATE sem a cláusula WHERE possui suas vantagens. Vamos imaginar que ocorra a necessidade de au-

mentar 15% o valor de todos os salários dos clientes. Nesse caso, não haveria necessidade do uso da cláusula WHERE, pois todos os registros deveriam sofrer a mesma alteração. O Exemplo 26 apresenta a solução desse problema.

Exemplo 26: “Altera o salário para mais 15% de TODOS os clientes”

```
update tbcliente set  
    salarioCli =salarioCli+( salarioCli *10/100)
```

Todas as condições aplicadas no comando de seleção podem ser aplicadas como condição no comando de alteração de dados. Outra consideração importante sobre o comando de alteração é que só é possível de forma direta selecionar uma única tabela para realizar a alteração de dados.

Removendo dados de uma tabela

Da mesma forma como ocorre o cadastro e alteração de dados, é comum que o usuário deseje remover um determinado registro. Para realizar esse procedimento, utilizamos o comando DELETE. Sua sintaxe é:

```
Delete from <nome_da_tabela>  
[ where <condições>]
```

O Exemplo 27 apresenta um exemplo da aplicação do comando DELETE, no qual estão sendo removidos todos os registros de cliente do sexo masculino.

Exemplo 27: “Apaga todos os clientes do sexo masculino”
Delete from tbcliente **where** sexoCli='M'

Da mesma forma que no comando UPDATE, a cláusula where é opcional no comando DELETE. Agora, imagine o estrago que seria a execução do comando delete sem a cláusula where. Na execução sem where, todos os dados da tabela informada serão removidos e a tabela ficará vazia. Sendo assim, tome muito cuidado na execução desse comando, e só execute um DELETE sem where se possuir certeza dessa alteração.

Recapitulando

Ao concluir este capítulo, o aluno percebeu o quanto é valioso a compreensão das quatro operações básicas de banco de dados, criação, leitura, alteração e exclusão de dados. Percebeu que cada tipo de dado possui um tratamento diferenciado quando selecionado para uma operação, e compreendeu que o tipo de dado “data” é o tipo de dado que mais varia entre os SGBD.

Compreendeu as formas de retornar somente as colunas que deseja visualizar além de criar colunas como resposta não existentes no banco. Evidenciou com é possível definir nome de uma coluna na execução de SQL e obter a resposta ordenada. Estudou como é possível, por meio dos operadores relacionais, criar uma regra de busca específica. Estudou sobre o conceito de sensitive de dados do tipo texto e compreendeu como se obtém uma consulta de campos do tipo texto, independentemente da forma como os dados estão armazenados, podendo, com estes, realizar consultas com o operador LIKE.

Percebeu o quando a cláusula where é importante na execução dos comandos de alteração e remoção de dados, podendo, sem aplicação correta desta, comprometer definitivamente o conteúdo de um banco de dados.

Atividades

Assinale (V) para as assertivas Verdadeiras e (F) para as Falsas, tendo como base a Figura que segue:

tbcliente		tbcliente				
pkcodCli integer		pkcodCli	nomeCli	endCli	dataNascCli	salarioCli
nomeCli varchar(25)						SexoCli
endCli varchar(40)		1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43 F
dataNascCli date		2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12 M
salarioCli number(15,2)		3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89 M
sexoCli varchar(1)		4	Pelé Rocha	Rua sei lá 222	25/07/1978	245,21 M
		5	Faustão Domingos	AV do Carmo 171	13/05/1988	433,65 M
		6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30 F
		7	Caetano Velado	Rua Condor 50	04/07/1990	294,21 M
		8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66 M

- 1) () O comando abaixo relacionado lista o nome do cliente, a data de nascimento e o sexo de todos os clientes do sexo masculino?

select a.nomecli as nome,a.datanasccli, a.sexocli as sexo from tb-cliente a where a.sexocli='M';

- 2) () O comando abaixo relacionado altera o nome do cliente para "ana", todos os clientes que possuem em seu nome a letra "a"

update tbcliente set nomecli='Ana' where upper(nomecli) like upper('%a')

- 3) () O comando abaixo relacionado insere um novo registro na tabela de cliente

insert into tbcliente(pkcodCli, nomeCli, endCli, salarioCli, datanascCli, SexoCli) values (9, 'José', 'Rua solidão 501', '10/12/1998', 2530.52, 'M');

- 4) () A execução do comando que segue, apresenta o resultado correto?

```
select c.nomecli,c.salariocli, c.sexocli from tbcliente where  
c.salariocli>700 order by c.sexocli,c.nomecli
```

Resultado

NOMECLI	SALARIOCLI	SEXOCLI
Maria Antonieta	1250,43	F
Xuxa Antibes	799,3	F
Bozo da Silva	734,12	M
Fofão Machado	2354,89	M

- 5) () O comando abaixo relacionado lista o nome dos clientes que nasceram após o ano de 2000 em ordem decrescente de nome de cliente

```
select tbcliente.nomecli from tbcliente where tbcliente.datanasccli>='01/01/2000' order by tbcliente.nomecli desc
```

Gabarito:

1 – V

2 – F (faltou colocar um curinga na cláusula like; o correto seria like upper('%a%'))

3 – F (A ordem dos campos está errada; primeiro, deveria ser informado o salário ao invés da data de nascimento; a operação correta seria: insert into tbcliente(pkcodCli, nomeCli, endCli, salarioCli, datanascCli, SexoCli) values (9, 'José', 'Rua solidão 501', 2530.52, '10/12/1998', 'M');)

4 – V

5 – V

Christiano Cadoná*

Capítulo 9

Critérios de Seleção de Dados

➔ Neste capítulo, serão estudadas questões relacionadas à seleção de dados como a utilização dos operadores lógicos, funções de agrupamento, além de estudar subconsultas utilizando o operador IN.

Ao término deste capítulo, o aluno terá conhecimento necessário para aplicar as técnicas trabalhadas e resolver problemas reais de busca de informação em tabelas.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Outras características do comando de seleção simples

Como já abordado no capítulo anterior, a operação de seleção é, sem dúvida, a mais utilizada entre as quatro operações básicas. Além das funcionalidades já trabalhadas, o comando de seleção possui outras funcionalidades indispensáveis de se conhecer. No capítulo anterior, foi abordado parte da sintaxe do comando SELECT. Segue uma nova versão da sintaxe de seleção (mas não a versão final), em que é possível identificar algumas novidades em sua sintaxe.

```
Select [distinct] <atributos_de_retorno> from <nome_da_tabela>  
[where <condições> ]  
[group by <atributo>]  
[having <atributo>]  
[order by <atributo>]
```

Como forma de demonstrar a partir de exemplos os novos parâmetros, será utilizada a Figura 63 que apresenta uma tabela de cliente com exemplos de dados armazenados. Esses dados servirão como base dos exemplos que serão apresentados neste capítulo.

O SQL disponibiliza operadores lógicos utilizados nos critérios de busca. Os operadores lógicos mais conhecidos no SQL podem ser observados na Tabela 4.

Operador	Descrição
AND	Retorna verdadeiro se duas expressões booleanas retornarem verdadeiro
OR	Retorna verdadeiro se qualquer expressão booleana retornar verdadeiro
NOT	Inverte o valor de qualquer operador booleano
IN	Retorna verdadeiro se o operador for igual a um elemento contido na lista

Tabela 4 Operadores lógicos

A Figura 71 apresenta três exemplos da utilização dos operadores lógicos AND e OR. Estes são os operadores mais conhecidos e podem ser utilizados individualmente ou em conjunto. Com eles, é possível criar mais de um critério de busca, aumentando as possibilidades de pesquisa.

a) Lista todos os dados dos clientes com salário superior a R\$ 800,00 do sexo feminino

```
select * from tbcliente c where c.salariocli>800 and c.sexocli='F';
```

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43	F

b) Lista todos os dados dos clientes de código 3 ou 7

```
select * from tbcliente c where c.pkcodcli=3 or c.pkcodcli=7
```

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M
7	Caetano Velado	Rua Condor 50	04/07/1990	294,21	M

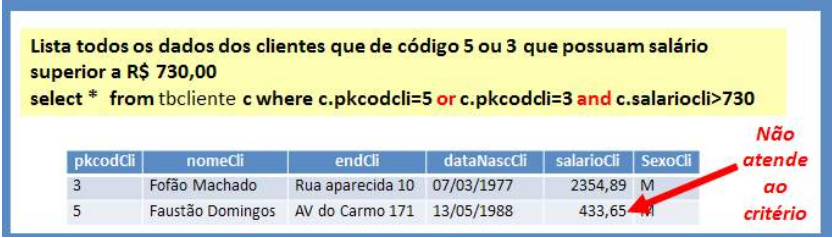
c) Lista todos os dados dos clientes que de código 5 ou 3 que possuam salário superior a R\$ 730,00

```
select * from tbcliente c where (c.pkcodcli=5 or c.pkcodcli=3) and c.salariocli>730
```

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M

Figura 71 Exemplos dos operadores lógicos AND e OR

O exemplo “a” apresenta a utilização do operador AND, já o exemplo “b” utiliza o operador OR. No exemplo “c”, ocorre a utilização de ambos os operadores como regra de busca. Observe, nesse exemplo, que, para correta implementação, foi necessário a utilização de parênteses. Isso foi necessário devido à ordem de precedência dos operadores lógicos. O operador AND é executado sempre com precedência em relação ao operador OR. Perceba, no exemplo apresentado na Figura 72, que, ao retirar os parênteses, a resposta será outra e não atende ao que o enunciado solicita. Nesse exemplo, o SGBD interpreta que devem ser listados os clientes cujo código seja 3 com salário superior a 730 ou que seu código seja 5.



Lista todos os dados dos clientes que de código 5 ou 3 que possuam salário superior a R\$ 730,00

```
select * from tbcliente c where c.pkcodcli=5 or c.pkcodcli=3 and c.salariocli>730
```

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M
5	Faustão Domingos	AV do Carmo 171	13/05/1988	433,65	M

Não atende ao critério

Figura 72 Exemplos de falta de aplicação de parênteses

Os exemplos apresentados na Figura 73 ilustram a aplicação do operador NOT e IN. No exemplo “a”, a regra é negar a expressão “c.sexocli='F'”, assim será retornado tudo que não for possuir essa sentença.

a) Liste os dados de todos os clientes que não possuem o sexo masculino
select * from tbcliente c where not c.sexocli='F';

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43	F
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F

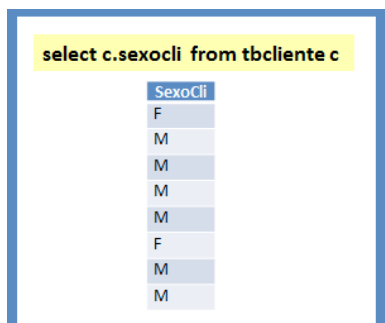
b) Lista todos os dados dos clientes de código 3 ou 4 ou 6 ou 7 ou 8
select * from tbcliente c where c.pkcodcli IN (3,4,6,7,8,23,29)

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M
4	Pelé Rocha	Rua sei lá 222	25/07/1978	245,21	M
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F
7	Caetano Velado	Rua Condor 50	04/07/1990	294,21	M
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

Figura 73 Exemplos de dos operadores NOT e IN

Já no exemplo “b”, perceba que foi criada uma lista de valores que serão comparados com o campo “pkcodcli”. Se algum dos argumentos existir, será listado como resposta.

Dando prosseguimento, vamos imaginar que necessitássemos elencar uma lista contendo todos os sexos das pessoas cadastradas. A partir do conhecimento até aqui apresentado, simplesmente mandaríamos selecionar a coluna “sexoCli” de todos os clientes. A resposta para essa execução está na Figura 74.

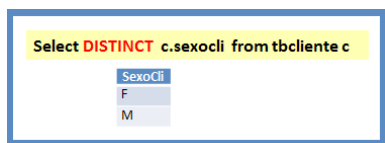


```
select c.sexocli from tbcliente c
```

SexoCli
F
M
M
M
M
F
M
M

Figura 74 Selecionando todos os sexos cadastrados

Ao avaliar a resposta, percebemos que existe o sexo “M” e “F”, porém estes aparecem repetitivos. Na verdade, uma linha para cada registro da tabela. Caso tivéssemos 10.000 clientes, seria listado um conjunto contendo 10.000 “M” e “F”. Contudo o objetivo aqui era apresentar somente duas linhas como resposta. Caso isso seja necessário, podemos utilizar a cláusula **DISTINCT** antes dos campos que precisamos retornar. Essa cláusula não permitirá apresentar resultados repetitivos. O Exemplo 75 apresenta a mesma consulta, porém agora com a cláusula **DISTINCT** definida. Observe que o retorno de informações é bem mais preciso.



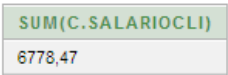
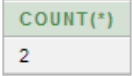

```
Select DISTINCT c.sexocli from tbcliente c
```

SexoCli
F
M

Figura 75 Exemplo do uso de distinct

Uma das grandes funcionalidades do uso de SQL está relacionada à possibilidade de execução de funções. Uma função

serve para executar uma determinada ação sobre um atributo ou conjunto de atributos. Cada SGBD, possui um conjunto de funções específicas em seu dicionário de dados. Em exemplos anteriores, já trabalhamos com a função UPPER. Existe um grupo de funções chamado funções de agregação que são muito utilizadas em implementações. A Tabela 5 apresenta as funções de agregação mais conhecidas.

Função	Descrição	Exemplo
• SUM(ATRIBUTOS)	Realiza uma soma do conteúdo dos atributos informados como parâmetro	<p>a) Liste a soma de todos os salários dos clientes</p> <pre>select sum(c.salariocli) from tbcliente c;</pre> 
• COUNT(ATRIBUTOS)	Retorna a quantidade de registros encontrados na ocorrência do select executado	<p>b) Liste a quantidade de clientes de sexo igual a "F"</p> <pre>select count(*) from tbcliente c where c.sexocli= 'F';</pre> 
• AVG(ATRIBUTOS)	Retorna a média aritmética do conteúdo existente nos atributos informados	<p>c) Liste a média de salário dos funcionários do sexo "F"</p> <pre>select avg(c.salariocli) from tbcliente c where c.sexocli= 'F';</pre> 



<ul style="list-style-type: none"> • MAX(ATRIBUTOS) 	Retorna o maior valor contido na coluna informada como parâmetro	<p>d) Liste a maior data de nascimento cadastrada na tabela de cliente</p> <pre>select max(c.datanasccli) from tbcliente c;</pre> 
<ul style="list-style-type: none"> • MIN(ATRIBUTOS) 	Retorna o menor valor contido na coluna informada como parâmetro	<p>e) Liste a menor data de nascimento cadastrada na tabela de cliente</p> <pre>select min(c.datanasccli) from tbcliente c;</pre> 

Tabela 5 Funções de agregação

Quando ocorre a necessidade de listar uma função de agregação juntamente com um atributo, é necessário agrupar sempre o atributo que não faz parte da função, para que a função de agregação funcione. Por exemplo, vamos imaginar que seja necessário listar o sexo dos clientes, a quantidade de clientes e a soma dos salários dos clientes do respectivo sexo. O campo sexoCli que não faz parte das funções de agregação count() e sum(), deverá ser agrupado. Para isso, utiliza-se a cláusula GROUP BY no final do código SQL antes do comando de ordenação de dados (caso exista).

A Figura 76 apresenta a implementação do exemplo citado utilizando a cláusula GROUP BY no campo “sexoCli”.

No exemplo, também foram definidos nomes para as colunas criadas por meio das funções de agregação.

```
select c.sexocli, count(*) as quantidade, sum(c.salariocli) as soma from  
tbcliente c group by c.sexocli
```

SexoCli	quantidade	Soma
M	6	4728,74
F	2	2049,73

Figura 76 Exemplo do uso GROUP BY

É comum também que o resultado de um SQL esteja condicionado a uma condição relacionada ao resultado de uma função. Caso isso ocorra, é necessário a execução da cláusula HAVING. Por exemplo, vamos imaginar que seja necessário listar em tela o sexo dos clientes, e a quantidade de clientes do respectivo sexo cuja quantidade seja superior a 4. Perceba que a quantidade de clientes faz parte de uma condição. Funções de agregação não podem ser utilizadas nas condições de uma cláusula WHERE por serem funções especiais. Nesse caso, após a aplicação de uma possível condição where utilizamos a cláusula HAVING.

A Figura 77 apresenta a solução para o problema apontado, em que é possível perceber a aplicação da cláusula HAVING sobre uma função de agregação.

```
select c.sexocli, count(*) as quantidade from tbcliente c
having count(*)>4
group by c.sexocli
```

SexoCli	quantidade
M	6

Figura 77 Exemplo do uso HAVING

Subconsulta

Esse termo é utilizado para designar a necessidade de realizar uma consulta dentro de outra consulta para atender a um determinado critério. Imagine que seja necessário listar todos os dados do cliente que possui o maior código, ou que seja necessário listar os dados do cliente mais velho. Utilizando os conceitos até aqui apresentados, não é possível executar esse tipo de consulta.

Para solucionar esse problema, utilizamos o operador IN. O operador IN realiza uma condição comparando um atributo ou valor a uma lista de valores. Ocorre que o SQL suporta que a lista existente no operador IN seja um resultado de outra consulta SELECT. Dessa forma, dizemos que utilizamos uma subconsulta para resolver o problema.

A Figura 78 apresenta dois exemplos da implementação de subconsultas, em que o primeiro soluciona o problema de listar os dados do cliente do sexo masculino que possui o

maior salário. O segundo, listando os dados do cliente mais velho que recebe abaixo de R\$ 1000,00.

a) select * from tbcliente c where c.salariocli in(select max(e.salariocli) from tbcliente e where e.sexocli='M')					
pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M

b) select * from tbcliente c where c.datanasccli in(select min(e.datanasccli) from tbcliente e where e.salariocli<1000)					
pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F

Figura 78 Exemplo do subconsulta em SELECT

O SQL irá primeiramente executar a consulta existente dentro do operador IN e criará como resposta, uma lista de opções que serão comparadas com a regra definida no operador IN. No exemplo “a”, a consulta interna retornou o salário R\$ 2354,89. Esse valor será a resposta a ser comparada com a primeira parte do select executado. É importante que seja destacado que a consulta existente no operador IN não poderá retornar duas colunas como resposta. Caso isso ocorra, o SGBD retornará um erro, não permitindo a pesquisa. O tipo de dado de retorno do operador IN deverá ser do mesmo tipo o qual está sendo comparado com o operador IN. No exemplo “b”, o tipo de retorno será uma data. Nesse caso, obrigatoriamente o campo de comparação do operador IN deve ser um campo do tipo data.

O operador IN é muito utilizado com operações UPDATE e DELETE, principalmente se a solução de uma alteração ou de uma exclusão depender de informações contidas em outras tabelas que não seja a tabela que sofrerá alteração ou exclusão. Isso porque esses dois comandos só podem possuir uma tabela como referência, ou seja, em um comando de alteração ou exclusão, somente dados de uma tabela poderão sofrer modificações. Observe a Figura 79 que possui um conjunto de duas tabelas (funcionário e cargo) relacionadas e populadas. O campo de relação entre as duas tabelas é “codCargo”.

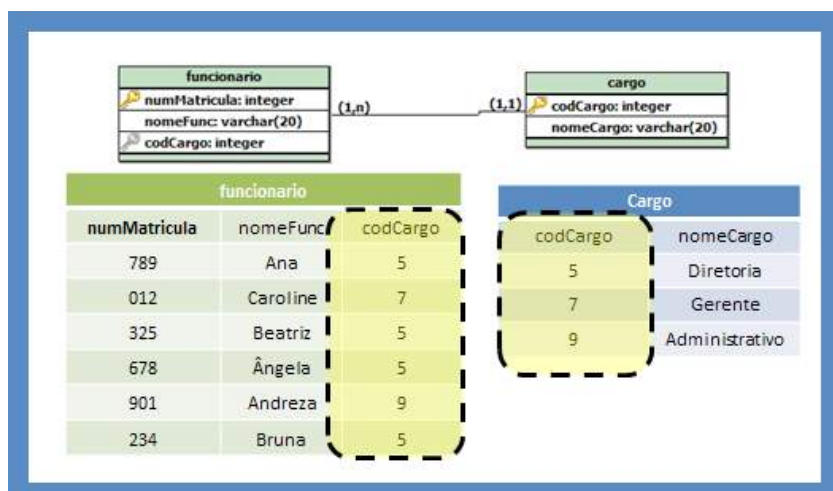


Figura 79 Exemplo de duas tabelas relacionadas e populadas

Agora, vamos imaginar que, com base do exemplo da Figura 79, ocorra a necessidade de alterar o nome de todos os funcionários para “Fulana” que possuam o cargo “Diretoria”. Observe que, nesse caso, a alteração deve ser feita na tabela “funcionário”, porém o critério de alteração está na tabela

de “cargo”. Como já mencionado, o comando UPDATE não suporta informar mais do que uma tabela, mas suporta a utilização do operador IN.

Assim, podemos executar a rotina de UPDATE selecionando uma tabela, e criar um critério de busca utilizando o operador IN buscando informações de outra tabela. O Exemplo 28 apresenta a solução para esse problema.

Exemplo 28: “UPDATE com subconsulta”

```
Update funcionario f set  
  f.nomeFunc='Fulana'  
where f.codCargo in(select d.codCargo from  
                    Cargo d where upper(d.  
                    nomecargo)=upper("Diretoria"))
```

Observe que, no Exemplo 28, foi utilizado os campos de relação entre as tabelas de funcionário e cargo. A consulta do operador IN retorna o código do cargo da tabela de cargo o qual é comparado com o código do cargo da tabela “funcionario”.

O mesmo ocorre quando necessitamos remover um registro o qual possui, como critério de remoção, dados de outra tabela. O comando Delete suporta apenas uma única tabela, mas suporta o operador IN como critério. Por exemplo, vamos imaginar que seja necessário remover todos os funcionários cujo cargo possua a letra “s” em seu nome. Nessa situação, utilizamos o comando Delete removendo dados da tabela de funcionário aplicando o operador IN que buscará dados da tabela de cargo. A consulta existente na cláusula IN, deverá retornar o código dos cargos que possuam a letra “s” em seu

nome. Assim, será retornado uma lista de códigos que serão utilizados para remoção dos dados da tabela de funcionário. O Exemplo 29 apresenta a solução para esse problema.

Exemplo 29: “DELETE com subconsulta”

```
Delete from funcionario f
      where f.codCargo in(select d.codCargo from Cargo
                           d where upper(nomecargo) like
                           upper('%s%'))
```

O Exemplo 30 apresenta uma linha de comando que remove o funcionário que possui o maior código. Perceba que, para isso, não é necessário uma outra tabela, mas, para isso, é necessário o uso de funções de agregação as quais não são suportadas pelos operadores DELETE e UPDATE. Nesse caso, a saída também foi a utilização do operador IN, retornando o maior código de funcionário como referência para realização da operação.

Exemplo 30: “DELETE com subconsulta a mesma tabela”

```
Delete from funcionario f
      where f.numMatricula in(select max(g.numMatricula) from
                              funcionario g)
```


Recapitulando

Ao concluir este capítulo, foi possível evidenciar diversas situações relacionadas a critérios de pesquisa. Foi constatada a importância dos operadores lógicos, os quais possibilitaram, além de outras funcionalidades, a criação de mais de uma regra de pesquisa.

Também foi destacada a ordem de execução dos comandos AND e OR, o qual, por meio de exemplos, foi possível concluir a necessidade, em alguns casos, do uso de parênteses como forma de execução correta dos critérios definidos.

As funções de agregação trouxeram maiores possibilidades de resultados, mas, em caso de utilizá-las juntamente com um outro atributo da tabela pesquisa, haverá necessidade do uso da cláusula de agrupamento GROUP BY. Essas funções de agregação também podem possuir condições, e, em caso de utilizá-las, é necessário colocá-la na cláusula HAVING do comando SQL, pois a condição WHERE não suporta funções de agregação.

Este capítulo também tratou do uso da cláusula IN como alternativa para realizar subconsultas as quais podem ser aplicadas em comandos de seleção, alteração e exclusão de dados. Dessa forma, muitas rotinas que aparentemente não eram possíveis de ser executadas podem ser utilizadas.

Atividades

Assinale (V) para as assertivas Verdadeiras e (F) para as Falsas, tendo como base a Figura que segue, contendo os dados armazenados na tabela de cliente.

tbcliente					
pkcodCli integer					
nomeCli varchar(25)					
endCli varchar(40)					
dataNascCli date					
salarioCli number(15,2)					
sexoCli varchar(1)					

pkcodCli	nomeCli	endCli	dataNascCli	salarioCli	SexoCli
1	Maria Antonieta	Rua sei lá n 43	20/08/2000	1250,43	F
2	Bozo da Silva	Av do Carmo 20	14/06/1990	734,12	M
3	Fofão Machado	Rua aparecida 10	07/03/1977	2354,89	M
4	Pelé Rocha	Rua sei lá 222	25/07/1978	245,21	M
5	Faustão Domingos	AV do Carmo 171	13/05/1988	433,65	M
6	Xuxa Antibes	Rua Voando 33	21/03/1950	799,30	F
7	Caetano Velado	Rua Condor 50	04/07/1990	294,21	M
8	Beltrano Carlos	Av Brasil 501	06/05/2001	666,66	M

- 1) () O comando abaixo relacionado lista todos os dados dos clientes do sexo "F" cujo endereço possua o número "3".

select * from tbcliente h where h.sexoCli='F' or upper(endcli) like upper('%3%')

- 2) () O comando abaixo relacionado lista os dados de todos os clientes que não tenham código 4, 6 ou 1.

select * from tbcliente x where not x.pkcodcli in(4,6,1)

- 3) () O comando abaixo relacionado lista a soma e a média de salário de todos os clientes que possuem salário abaixo de R\$ 1000,00 e nasceram antes de 20/10/1999.

select sum(t.salariocli) as soma, avg(t.salariocli) as media from tbcliente t where t.salariocli < 1000 and t.datanasccli < '20/10/1999'

- 4) () O comando que segue lista o sexo dos clientes e a quantidade de clientes do respectivo sexo, cuja média salarial esteja superior a R\$ 800,00, dos clientes cujo nome termine com a letra "o".

```
select j.sexocli, count(*) as quant from tbcliente j
```

```
where upper(j.nomecli) like upper('%o')
```

```
having avg(salariocli) > 800
```

```
group by j.sexocli
```

- 5) () () O comando abaixo relacionado aumenta mais R\$ 230,00 ao salário do cliente que possui o maior e o menor salário.

```
update tbcliente w set
```

```
w.salariocli = salariocli + 230
```

```
where w.salariocli in (select max(u.salariocli) from tbcliente u)
```

```
or w.salariocli in (select min(p.salariocli) from tbcliente p)
```

Gabarito:

- 1 – F (o operador correto, nesse exemplo, deveria ter sido AND)
- 2 – V
- 3 – V
- 4 – V
- 5 – V

Consultas envolvendo mais de uma tabela

➔ **N**este capítulo, será estudado e exemplificado a utilização de mais de uma tabela para execução de consultas. Serão implementados conceitos de produto cartesiano, junções, consultas envolvendo outra consulta e visões de dados.

Ao concluir este capítulo, o aluno terá o conhecimento necessário para implementar consultas complexas utilizando mais do que uma tabela como referência.

*Especialista em Desenvolvimento de Software para Web. Professor coordenador de atividades dos cursos superiores de tecnologia em Redes de Computadores e Análise e Desenvolvimento de Sistemas

Consulta envolvendo mais de uma tabela

Nos capítulos anteriores, foi estudada uma série especificações relacionadas à consulta de dados. Contudo, todas faziam referência à utilização de uma única tabela. De forma prática, um diagrama relacional dificilmente terá apenas uma ou duas tabelas, principalmente se neste forem aplicadas as técnicas de normalização de dados. Assim, para obter uma listagem que envolva mais de uma tabela, é necessário citar as tabelas e apresentar as relações existentes entre essas tabelas.

Como forma de exemplificar os exemplos que serão trabalhados neste capítulo, a Figura 80 apresenta um diagrama relacional envolvendo as tabelas de categoria, produto e fornecedor.

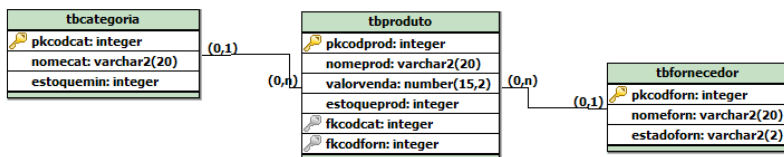


Figura 80 Diagrama ER base para exemplificar os tópicos deste capítulo

Perceba que as cardinalidades mínimas das relações não são obrigatórias, o que implica dizer que poderá haver produtos que não possuam categorias ou que não possuam fornecedores.

A Figura 81 apresenta um conjunto de tabelas populadas relacionadas ao diagrama apresentado na Figura 80. Nesta,

pode ser identificada a existência de dois produtos que não possuem fornecedores vinculados a este, ou seja, a chave estrangeira `fkcodforn` da tabela de produto está vazia (nula).

tbcategoria			tbproduto					
PKCODCAT	NOMECAT	ESTOQUEMIN	PKCODPROD	NOMEPROD	VALORVENDA	ESTOQUEPROD	FKCODCAT	FKCODFORN
1	Bebida	12	102	Ervilha Maravilhosa	1,34	20	2	53
2	Enlatado	18	103	Pão Francês	3,88	130	3	
3	Padaria	0	104	Refrigerante Gela	3	3	1	57
			105	Cachaca Velho Novo	7,22	5	1	50
			106	Margarina de Soja	4,77	50	3	50
			107	Cerveja quero mais	1,9	18	1	57
			108	Milho entalado	4,34	11	2	58
			109	Bolo de Banana	4,73	3	3	50
			110	Requeijão Cremoso	2,99	6	3	58
			111	Destilado Forte	89,91	6	1	50
			112	Vinho Sou Uva	15,7	21	1	53
			113	Pepino em Conserva	4,99	56	2	53
			114	Palmito da Floresta	7,55	7	2	57
			115	Bolo de fuba	5,7	8	3	
			116	Refrigerante Gasosa	2,99	32	1	53

tbfornecedor		
PKCODFORN	NOMEFORN	ESTADOFORN
50	Fornecedor X	SP
53	Fornecedor Y	RS
54	Fornecedor W	SP
57	Fornecedor K	RS
58	Fornecedor R	RJ

Figura 81 Tabelas populadas para exemplificação

É comum em diagramas com mais de uma tabela, como é o caso do diagrama ER da Figura 80, apresentar respostas em que seja necessário envolver atributos de mais de uma tabela. Nesse caso, é necessário que ocorra um relacionamento entre as entidades. A aplicação de relacionamento entre entidades pode ocorrer de formas diferentes. A maneira mais trivial de relacionar mais de uma entidade é informar o nome das tabelas entre a cláusula `FROM` e `WHERE` de um `select`. Segue uma nova atualização da sintaxe do comando `select`:

Por exemplo, caso seja necessário listar o nome do produto (**nomeprod**), o valor de venda do produto (**valorvenda**) e o nome da categoria (**nomecat**), basta listar os atributos que

deseja mostrar como retorno e acrescentar as tabelas de categoria (**tbcategoria**) e produto (**tbproduto**) após a cláusula FROM. O Exemplo 31 ilustra a aplicação de mais de uma tabela na mesma consulta.

Exemplo 31: “Listando dados de mais de uma tabela”

```
select p.nomeprod,p.valorvenda,c.nomecat from tbproduto p,  
tbcategoria c
```

Contudo, ao executar código do Exemplo 31 o usuário perceberá que a resposta não está plenamente correta. Isso se deve ao fato de que, ao colocar mais de uma tabela após a cláusula FROM do select, o SGBD realiza um produto cartesiano entre as entidades. Tendo como base que a tabela de categoria possua 3 registros e a tabela de produto possua 15 registros, o SQL resultante do exemplo 30 trará 45 registros (3×15), pois, para cada registro de produto, o SGBD adiciona uma categoria. Isso se deve ao fato do SQL utilizar o conceito de álgebra relacional, já abordado nos capítulos anteriores.

Da mesma forma que na álgebra relacional, a solução do produto cartesiano é trazer apenas os registros corretos, é identificar a relação existente entre as tabelas envolvidas. Para isso, basta criar um critério na cláusula WHERE do SELECT relacionando a chave estrangeira de categoria existente na tabela de produto (**fkcodcat**), e a chave primária da tabela de categoria (**pkcodcat**).

Assim, sempre que necessário relacionar mais de uma tabela em uma consulta SQL, deve também ser informado na

cláusula WHERE, a(s) condição(ões) que fazem referência ao relacionamento entre as entidades envolvidas. A Figura 82 apresenta a implementação correta do Exemplo 30 e o resultado a pesquisa realizada.



```
select p.nomeprod,p.valorvenda,c.nomecat from tbproduto p, tbcategoria c
where p.fkcodcat=c.pkcodcat
```

NOMEPROD	VALORVENDA	NOME CAT
Ervilha Maravilhosa	1,34	Enlatado
Pao Frances	3,88	Padaria
Refrigerante Gela	3	Bebida
Cachaca Velho Novo	7,22	Bebida
Margarina de Soja	4,77	Padaria
Cerveja quero mais	1,9	Bebida
Milho entalado	4,34	Enlatado
Bolo de Banana	4,73	Padaria
Requeijao Cremoso	2,99	Padaria
Destilado Forte	89,91	Bebida
Vinho Sou Uva	15,7	Bebida
Pepino em Conserva	4,99	Enlatado
Palmito da Floresta	7,55	Enlatado
Bolo de fuba	5,7	Padaria
Refrigerante Gasosa	2,99	Bebida

Figura 82 Exemplo da utilização de produto cartesiano

O mesmo ocorre quando existe a mais do que duas tabelas envolvidas na consulta. Por exemplo, se necessitássemos listar o nome do produto, o nome da categoria e o nome do fornecedor que o forneceu, cujos produtos possuam mais do que 20 unidades em estoque, seria necessário informar todas as tabelas envolvidas e os relacionamentos existentes entre as tabelas envolvidas. A Figura 83 apresenta a forma como deve ser implementado o comando de seleção e o critério de busca, juntamente com o conteúdo de resposta do comando.


```
select p.nomeprod,p.valorvenda,c.nomecat, f.nomeforn  
from tbproduto p, tbcategoria c, tbfornecedor f  
where p.fkcodcat=c.pkcodcat and p.fkcodforn=f.pkcodforn  
and p.estoqueprod>20
```

NOMEPROD	VALORVENDA	NOMECAT	NOMEFORN
Margarina de Soja	4,77	Padaria	Fornecedor X
Vinho Sou Uva	15,7	Bebida	Fornecedor Y
Pepino em Conserva	4,99	Enlatado	Fornecedor Y
Refrigerante Gasosa	2,99	Bebida	Fornecedor Y

Figura 83 Exemplo da utilização de produto cartesiano com 3 tabelas e uma condição

Utilizando JOIN

A técnica de utilização do produto cartesiano soluciona o problema de utilizar mais de uma tabela como resposta de uma pesquisa, porém não é a única alternativa. Uma alternativa é a utilização da cláusula JOIN no comando SELECT. Essa cláusula só funciona nos comandos de seleção e possui algumas características particulares que possibilitam resultados de busca bem interessantes. Dessa forma, segue novamente uma atualização da sintaxe do comando de seleção contendo JOIN.

```
Select [distinct] <atributos_de_retorno> from <tabela1>
[inner join tabela2 on <condições>]
[left join tabela2 on <condições>]
[right join tabela2 on <condições>]
[full join tabela2 on <condições>]
[where <condições> ]
[group by <atributo>]
[having <atributo>]
[order by <atributo>]
```

Em geral, a aplicação de JOIN é mais otimizada do que a utilização de tabelas após a cláusula FROM de um SELECT. Contudo, os SGBD mais modernos otimizaram a execução de consultas, não apresentando diferença de performance entre a utilização de ambos os métodos, com exceção da execução de consultas complexas.

A cláusula JOIN possui algumas variações (inner, left, right, full) trazendo resultados diferentes de acordo com sua utilização.

Para utilizar JOIN, basta informar uma tabela após a cláusula FROM e, em seguida, utilizar o JOIN informado a outra tabela. Após a aplicação da cláusula ON presente na sintaxe do JOIN, deve ser informada a relação existente entre as tabelas já especificadas. A Figura 84 apresenta a mesma consulta utilizada no exemplo da Figura 83, que lista o nome do produto, sua categoria e fornecedor, em que seu estoque seja superior a 20.

```
select p.nomeprod,p.valorvenda,c.nomecat, f.nomeforn from tbproduto p
inner join tbcategoria c on p.fkcodcat=c.pkcodcat
inner join tbfornecedor f on p.fkcodforn=f.pkcodforn
where p.estoqueprod>20
```

NOMEPROD	VALORVENDA	NOMECAT	NOMEFORN
Margarina de Soja	4,77	Padaria	Fornecedor X
Vinho Sou Uva	15,7	Bebida	Fornecedor Y
Pepino em Conserva	4,99	Enlatado	Fornecedor Y
Refrigerante Gasosa	2,99	Bebida	Fornecedor Y

Figura 84 Exemplo da utilização de inner join

Observe que as restrições que antes estavam sendo aplicadas (exemplos das Figuras 82 e 83) na cláusula WHERE agora fazem parte do INNER JOIN. A principal diferença está no momento em que for executada uma cláusula INNER, definir além do critério de relação entre as tabelas, outra(s) condição(es) que possam fazer parte de um critério de busca. Bem aplicada, consultas com essas características podem retornar os dados de forma mais otimizada (rápida).

Observe os exemplos “a” e “b” da Figura 85. Ambos retornam a mesma resposta, porém internamente o SGBD trata a chamada de consulta de forma diferente. Ambos os exemplos listam o nome do produto, o nome da categoria e o nome do fornecedor que forneceu o produto, de todos os produtos cuja categoria inicie com a letra “P”.

```
a) select p.nomeprod,c.nomecat,f.nomeforn from tbproduto p
inner join tbcategoria c on p.fkcodcat=c.pkcodcat
inner join tbfornecedor f on p.fkcodforn=f.pkcodforn
where upper(c.nomecat) like ('P%')
```

```
b) select p.nomeprod,c.nomecat,f.nomeforn from tbproduto p
inner join tbcategoria c on p.fkcodcat=c.pkcodcat and upper(c.nomecat) like ('P%')
inner join tbfornecedor f on p.fkcodforn=f.pkcodforn
```

Resposta das consultas realizadas

NOMEPROD	NOMECAT	NOMEFORN
Margarina de Soja	Padaria	Fornecedor X
Bolo de Banana	Padaria	Fornecedor X
Requeijao Cremoso	Padaria	Fornecedor R

Figura 85 Exemplo da utilização de INNER JOIN com mais um critério

A principal diferença do ponto de vista de otimização, está relacionada ao fato que, quando o SGBD realiza o produto cartesiano entre o produto e a categoria (primeira relação na cláusula INNER JOIN), ele fará somente com os registros cuja categoria inicie com a letra “P” no caso do exemplo “b”, enquanto o exemplo “a” só fará o filtro após realizar os demais produtos cartesianos existentes, nesse caso, a segunda linha do INNER JOIN (entre produto e fornecedor). Muitos SGBD inclusive detectam a possibilidade de otimizar e internamente alteram a linha SQL de forma que a consulta possua melhor performance. Sendo assim, se você desenvolver uma consulta e esta demorar para a ser processada, implemente alternativas como o exemplo “b” da Figura 85, assim, poderá ocorrer melhorias no resultado da mesma.

Cabe salientar que a velocidade na recuperação das informações é importante para qualquer porte do sistema. Para

atingi-la, existe uma série de mecanismos, que iniciam na modelagem do banco e são concluídos com o método de pesquisa e criação de índices.

Vamos avaliar a Figura 86, que possui duas consultas distintas, em que, na primeira ("a"), estamos listando o nome dos produtos e suas respectivas categorias, e na segunda estamos listando o nome dos produtos e seus respectivos fornecedores ("b"). Perceba que o número de retorno de registros é diferente de uma e da outra pesquisa realizada.

a) select p.nomeprod,c.nomecat from tbproduto p inner join tbcategoria c on c.pkcodcat=p.fkcodcat		b) select p.nomeprod,f.nomeforn from tbproduto p inner join tbfornecedor f on f.pkcodforn=p.fkcodforn	
NOMEPROD	NOMECAT	NOMEPROD	NOMEFORN
Ervilha Maravilhosa	Enlatado	Ervilha Maravilhosa	Fornecedor Y
Pao Frances	Padaria	Refrigerante Gela	Fornecedor K
Refrigerante Gela	Bebida	Cachaca Velho Novo	Fornecedor X
Cachaca Velho Novo	Bebida	Margarina de Soja	Fornecedor X
Margarina de Soja	Padaria	Cerveja quero mais	Fornecedor K
Cerveja quero mais	Bebida	Milho entalado	Fornecedor R
Milho entalado	Enlatado	Bolo de Banana	Fornecedor X
Bolo de Banana	Padaria	Requeijao Cremoso	Fornecedor R
Requeijao Cremoso	Padaria	Destilado Forte	Fornecedor X
Destilado Forte	Bebida	Vinho Sou Uva	Fornecedor Y
Vinho Sou Uva	Bebida	Pepino em Conserva	Fornecedor Y
Pepino em Conserva	Enlatado	Palmito da Floresta	Fornecedor K
Palmito da Floresta	Enlatado	Refrigerante Gasosa	Fornecedor Y
Bolo de fuba	Padaria		
Refrigerante Gasosa	Bebida		

Figura 86 Exemplo de seleção com INNER JOIN

Na consulta "a", são listados 15 registros, exatamente os 15 produtos cadastrados que foram definidos na Figura 81. Já no exemplo "b" são listados apenas 13 registros. Estão faltando os registros de "Pão Frances" e "Bolo de fubá". Isso se deve ao fato dos produtos "Pão Frances" de código "103" e o produto "Bolo de fubá" de código "115", não possuírem for-

necedores relacionados (conteúdo no campo `fkcodforn` nulo). Quando executamos um `INNER JOIN` definindo uma relação em que o conteúdo de um atributo deve ser igual ao conteúdo de outro atributo de outra entidade, criando uma regra de obrigatoriedade.

Se ocorrer a necessidade de, por exemplo, listar o nome do produto (*nomeprod*) e o nome do fornecedor vinculado ao produto (*nomeforn*) em que obrigatoriamente seja necessário listar o nome do produto independentemente de que exista ou não um fornecedor vinculado a ele, pode ser utilizado a cláusula **LEFT JOIN** ou **RIGHT JOIN**. A escolha de um ou outro depende do tipo de informação que é necessário ser apresentado. A cláusula `LEFT JOIN` é utilizada quando existe a necessidade de obrigatoriedade de visualização dos dados da tabela que se encontra à esquerda da cláusula `LEFT JOIN`. Já a cláusula `RIGHT JOIN` é utilizada quando ocorre a necessidade de visualizar, de forma obrigatória, informações da tabela que está à direita da cláusula `RIGHT JOIN`. A Figura 87 apresenta o resultado da pesquisa utilizando a cláusula `LEFT JOIN`.

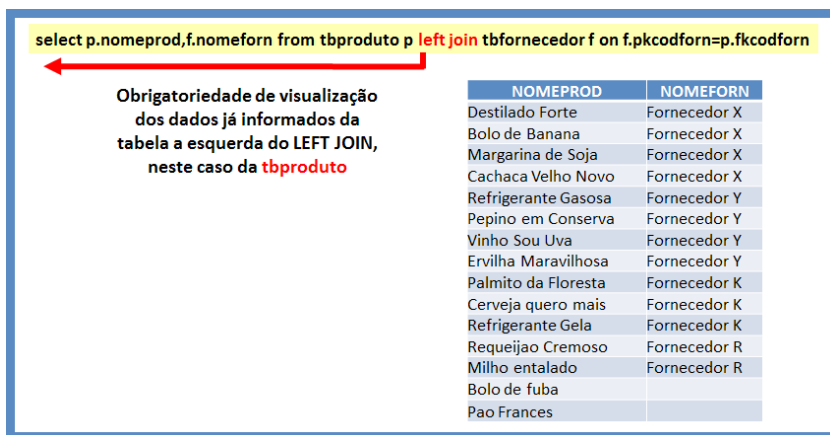


Figura 87 Exemplo de seleção com LEFT JOIN

Nesse caso, a consulta retornou todos os produtos, independentemente se o produto possui ou não fornecedor. Nos registros os quais este não possui fornecedor (chave estrangeira nula), o nome do fornecedor fica em branco.

Caso tivéssemos utilizado a cláusula RIGTH JOIN para essa mesma consulta, o resultado não seria o mesmo. O exemplo apresentado na Figura 88 ilustra o mesmo exemplo apenas alternado para a cláusula RIGTH JOIN.

Observe que a resposta ficou completamente diferente do exemplo da Figura 87. Além de não apresentar os dois produtos que não possuíam a chave estrangeira, retornou, como resposta, o fornecedor que não possuía nenhum produto relacionado. Isso se deve à definição da cláusula. No exemplo da Figura 88, o SQL sempre retornará os dados de todos os fornecedores, pois a tabela está à direita da cláusula RIGHT

JOIN, além de listar os produtos que possuem relação com fornecedor.



Figura 88 Exemplo de seleção com RIGHT JOIN

Existe também a possibilidade de utilizar a cláusula **FULL JOIN**, que possui como característica listar todos os elementos das tabelas envolvidas e, caso exista relação, mostrará o elementos relacionados. A Figura 89 apresenta a implementação do mesmo exemplo agora utilizando a cláusula FULL JOIN.


```
select p.nomeprod,f.nomeforn from tbproduto p full join tbfornecedor f on f.pkcodforn=p.fkcodforn
```

Listra todos os dados das tabelas **tbproduto** e **tbfornecedor** e em caso de existir relação apresenta os elementos relacionados.

NOMEPROD	NOMEFORN
Destilado Forte	Fornecedor X
Bolo de Banana	Fornecedor X
Margarina de Soja	Fornecedor X
Cachaca Velho Novo	Fornecedor X
Refrigerante Gasosa	Fornecedor Y
Pepino em Conserva	Fornecedor Y
Vinho Sou Uva	Fornecedor Y
Ervilha Maravilhosa	Fornecedor Y
Palmito da Floresta	Fornecedor K
Cerveja quero mais	Fornecedor K
Refrigerante Gela	Fornecedor K
Requeijao Cremoso	Fornecedor R
Milho entalado	Fornecedor R
Bolo de fuba	
Pao Frances	
	Fornecedor W

Figura 89 Exemplo de seleção com FULL JOIN

Utilização de consultas como base para realização de outras consultas

A grande maioria dos SGBD dão suporte à utilização do resultado de uma consulta no lugar de uma tabela física para execução de uma outra consulta. A implementação desse tipo de SQL é muito comum, no processo de otimização de consultas complexas e ou facilitar encontrar dados em tabelas relacionadas. A utilização dessa técnica é simples, basta colocar entre parênteses o código SQL desejado no lugar do nome da tabela de uma cláusula SQL convencional. Nesse caso, é necessário que essa consulta interna possua um alias (apelido). O alias definido, poderá ser setado como se fosse uma tabela no restante da consulta, e os atributos de retorno da consulta, serão os atributos de resposta dessa suposta tabela.

O exemplo apresentado na Figura 90 apresenta uma lista-gem contendo todos os dados da tabela produto, que possua relação com outra consulta denominada “cat”, a qual lista o código e nome de categoria de todas as categorias cujo nome da categoria inicia com a letra “B”.

```
select * from tbproduto p, (select c.pkcodcat as codigo, c.nomecat as categoria from tbcategoria c
where upper(c.nomecat) like 'B%') cat where p.fkcodcat=cat.codigo
```

PKCODPROD	NOMEPROD	VALORVENDA	ESTOQUEPROD	FKCODCAT	FKCODFORN	CODIGO	CATEGORIA
104	Refrigerante Gela	3	3	1	57	1	Bebida
105	Cachaca Velho Novo	7,22	5	1	50	1	Bebida
107	Cerveja quero mais	1,9	18	1	57	1	Bebida
111	Destilado Forte	89,91	6	1	50	1	Bebida
112	Vinho Sou Uva	15,7	21	1	53	1	Bebida
116	Refrigerante Gasosa	2,99	32	1	53	1	Bebida

Figura 90 Exemplo 1 de consulta como base de novas consultas

Observe que a consulta de código e nome de categoria que inicia com a letra “B” é executada dentro da cláusula FROM do SELECT, e seu resultado é atribuído para o alias “CAT”. O SQL trata “cat” como se fosse uma tabela contendo valores nela armazenada.

Nesse caso, o SGBD executa primeiramente a consulta destacada retornando apenas os registros que iniciam com a letra “B” e, em seguida, realiza o produto cartesiano com a tabela de produto. Essa execução otimiza significativamente uma consulta que possua um volume de dados significativo. Imagine que existam cadastrado na tabela de produtos 5.000 produtos e 40 categorias, em que dessas 40, 5 iniciem com a letra “B”. Se fosse realizada a execução, apenas relacionando as duas tabelas na cláusula from, o SGBD inicialmente

iria realizar um produto cartesiano retornando em memória 200.000 registros para daí sim, aplicar as regras condicionais definidas (nesse caso, `where upper(c.nomecat) like 'B%' e p.fkcodcat=cat.codigo`). Contudo, o exemplo apresentado na Figura 90 otimiza o resultado, pois o produto cartesiano possuirá um conjunto contendo um número menor de registros. O SGBD irá inicialmente executar a subconsulta definida como “cat”, em que retornará apenas 5 registros. Com base nesses 5 registros, aplicará um produto cartesiano nos outros 5.000 registros de produtos, resultando em memória 25.000 registros, que posteriormente serão submetidos à cláusula `where` do `select` principal.

Pode ser observado também que o nome das colunas de resposta está associado ao nome dos atributos que foram definidos na consulta denominada “cat”, nesse caso, os campos “codigo” e “categoria”.

Além de otimizar consultas, a aplicação desse método de busca de dados pode proporcionar a busca a dados mais complexos facilitada. Por exemplo: imagine que necessitamos listar em tela o nome de todas as categorias e duas outras colunas, sendo que a primeira listaria a quantidade de produtos dessa categoria que possuísse seu estoque superior a 20 unidades, e a outra coluna contendo a quantidade de produtos dessa categoria que possua estoque até 20 unidades.

A Figura 91 apresenta uma alternativa de resposta utilizando o método de seleção de dados na cláusula `from`.

```
select c.nomecat, t1.acima20, t2.ate20 from tbcategoria c,
(select c.pkcodcat, count(*) as acima20 from tbcategoria c inner join tbproduto p on
c.pkcodcat=p.fkcodcat where p.estoqueprod>20 group by c.pkcodcat) t1,
(select c.pkcodcat, count(*) as ate20 from tbcategoria c inner join tbproduto p on
c.pkcodcat=p.fkcodcat where p.estoqueprod<=20 group by c.pkcodcat) t2
where c.pkcodcat=t1.pkcodcat and c.pkcodcat=t2.pkcodcat
```

NOMECAT	ACIMA20	ATE20
Bebida	2	4
Enlatado	1	3
Padaria	2	3

Figura 91 Exemplo 2 de consulta como base de novas consultas

Observe que são executadas outras duas consultas, sendo a primeira listando o código da categoria e a quantidade de produtos que possuem um estoque superior a 20 unidades, e a outra consulta retornando o código de categoria e a quantidade de produtos vinculados que possuam seu estoque até 20 unidades. Como resposta, temos uma relação de categorias e outras duas colunas que representam o resultado das outras duas consultas respectivamente.

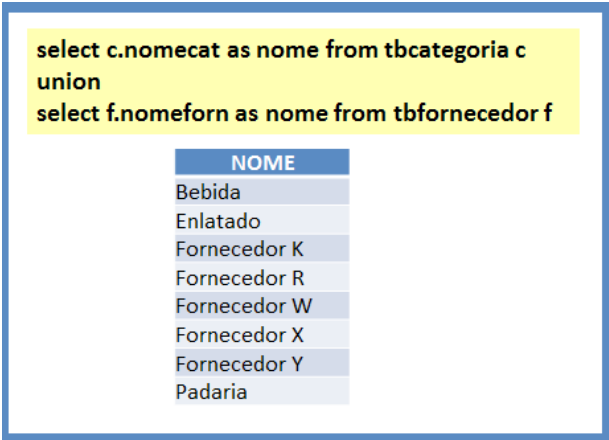
Cabe salientar que a utilização desse método também é possível nas cláusulas INNER, LEFT, RIGHT e FULL JOIN. Nesse caso, no lugar do nome das tabelas, apenas colocamos a consulta desejada.

União de consultas

Da mesma forma que a álgebra relacional une tuplas de duas relações que sejam compatíveis, a cláusula UNION realiza a

união de dois comandos SELECT que possuam colunas de resposta compatíveis.

Vamos imaginar, por exemplo, que seja necessário listar uma relação contendo o nome das categorias e o nome dos fornecedores cadastrados. Nesse caso, basta executar dois SELECT, sendo o primeiro retornando as categorias e o segundo retornando fornecedores unindo-as com a cláusula UNION. A Figura 92 apresenta a solução do exemplo apresentado.



```
select c.nomecat as nome from tbcategoria c
union
select f.nomeforn as nome from tbfornecedor f
```

NOME
Bebida
Enlatado
Fornecedor K
Fornecedor R
Fornecedor W
Fornecedor X
Fornecedor Y
Padaria

Figura 92 Exemplo de UNION

Criando visões de dados

Uma visão (ou view) é classificada como uma tabela virtual, ou seja, ela, na prática, não existe de forma física, ao contrário

das tabelas da base, cujas tuplas sempre estão armazenadas fisicamente no banco de dados. (Elmasri, 2011)

Uma view pode ser visualizada como um modo de especificar uma tabela que precisamos referenciar com frequência, embora ela possa não existir fisicamente. A partir da criação de uma view, essa pode ser acessada como se fosse uma tabela, inclusive é possível relacionar a view com outras tabelas ou outras view. (Elmasri, 2011)

A sintaxe da criação de uma view é:

```
create view <nome_da_view> as <SELECT_QUE_DEFINIRÁ_A_VIEW>
```

O Exemplo 32 cria uma view chamada “exemploview” que possui como conteúdo o código do produto, o nome do produto, o nome da categoria do produto e o nome do fornecedor do produto.

Exemplo 32: “Criando view”

create view exemploview as

```
select p.pkcodprod, p.nomeprod, c.nomecat, f.nomeform  
from tbproduto p  
inner join tbcategoria c on c.pkcodcat=p.fkcodcat  
left join tbfornecedor f on f.pkcodforn=p.fkcodforn
```

Assim que criada a view, ela pode ser acessada. Os exemplos apresentados na Figura 93 apresentam a utilização da view criada (exemploview). Observe que a view está sendo uti-

lizada como se fosse uma tabela comum, inclusive aplicando condições sobre seus atributos, e adicionando relações com outras tabelas.

a) `select * from exemploview e where e.pkcodprod>110`

PKCODPROD	NOMEPROD	NOMECAT	NOMEFORN
111	Destilado Forte	Bebida	Fornecedor X
116	Refrigerante Gasosa	Bebida	Fornecedor Y
113	Pepino em Conserva	Enlatado	Fornecedor Y
112	Vinho Sou Uva	Bebida	Fornecedor Y
114	Palmito da Floresta	Enlatado	Fornecedor K
115	Bolo de fuba	Padaria	-

b) `select e.*,p.valorvenda from exemploview e
inner join tbproduto p on p.pkcodprod=e.pkcodprod
where e.pkcodprod>110`

PKCODPROD	NOMEPROD	NOMECAT	NOMEFORN	VALORVENDA
111	Destilado Forte	Bebida	Fornecedor X	89,91
112	Vinho Sou Uva	Bebida	Fornecedor Y	15,7
113	Pepino em Conserva	Enlatado	Fornecedor Y	4,99
114	Palmito da Floresta	Enlatado	Fornecedor K	7,55
115	Bolo de fuba	Padaria	-	5,7
116	Refrigerante Gasosa	Bebida	Fornecedor Y	2,99

Figura 93 Exemplo do uso de VIEW

Caso seja necessário remover uma view, utiliza-se o comando **DROP VIEW <nome_da_view>**.

Recapitulando

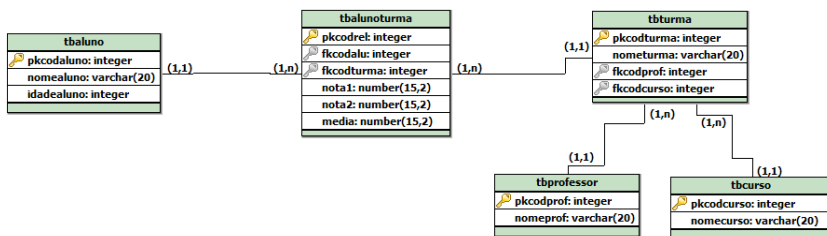
Ao concluir este capítulo, o aluno teve oportunidade de conhecer as formas de realizar uma consulta com mais de uma tabela. Foi estudada a aplicação de mais de uma tabela após a cláusula `FROM`, gerando um produto cartesiano entre as tabelas envolvidas. Nesse tópico, o aluno recordou os conceitos da álgebra relacional e entendeu de forma prática como é importante adicionar uma condição ligando as tabelas relacionadas para trazer a correta informação. Em seguida, foi abordado o uso da cláusula `JOIN` como alternativa de relacionar mais de uma tabela, onde foi estudada a partir de exemplos práticos a diferença da aplicação `INNER`, `LEFT`, `RIGHT` e `FULL JOIN`.

Em seguida, foi abordado o uso de consultas como base para realização de outras consultas. A partir de um exemplo, foi possível compreender por que esse tipo de operação é muito utilizado na otimização de consultas, pois minimiza significativamente o resultado de memória que um produto cartesiano realiza ao relacionar tabelas.

Este capítulo também abordou o uso da cláusula `UNION`, como forma de unir duas consultas que retornam o mesmo tipo de dado em uma única resposta. Por fim, também foi possível conhecer como funciona o conceito de `VIEW` como alternativa de operacionalizar dados.

Atividades

Tendo como base o diagrama lógico que segue, assinale (V) para as assertivas Verdadeiras e (F) para as Falsas.



- 1) () Ambos os comandos que seguem possuem sua sintaxe correta e retornam a mesma resposta.

Comando 1:

```
select t.nometurma,c.nomecurso from tbturma t, tb-  
curso c where t.fkcodcurso=c.pkcodcurso
```

Comando 2:

```
select t.nometurma,c.nomecurso from tbturma t in-  
ner join tbcurso c on t.fkcodcurso=c.pkcodcurso
```

- 2) () O comando que segue lista o nome do curso de todas as turmas cursadas pelos alunos que iniciam com a letra "a" cujo professor possua em seu nome a palavra "silva".

```
select distinct c.nomecurso from tbcurso c  
inner join tbturma t on t.fkcodcurso=c.pkcodcurso
```

```
inner join tbalunoturma at on at.fkcodturma=t.  
pkcodturma
```

```
inner join tbaluno a on a.pkcodaluno=at.fkcodalu
```

```
inner join tbprofessor f on f.pkcodprof=t.fkcodprof
```

```
where upper(a.nomealuno) like upper('a%') and  
upper(f.nomeprof) like upper('%silva%')
```

- 3) () O comando que segue lista o nome do curso, o nome da turma de todas as turmas de código maior do que 300, cujo nome do curso inicie em “info”, em ordem de nome de curso

```
select c.nomecurso, t.nometurma from tbturma t
```

```
left join tbcurso c on t.fkcodcurso=c.pkcodcurso and  
upper(c.nomecurso) like upper('info%')
```

```
where t.pkcodturma>300 order by t.nometurma
```

- 4) () Gere uma lista contendo o nome de todos os alunos e professores das turmas de código menor do que 250

```
(select distinct a.nomealuno as nome from tbaluno  
a, tbalunoturma at where a.pkcodaluno=at.fkcodalu  
and at.fkcodturma<250)
```

```
union
```

```
(select p.nomeprof as nome from tbprofessor p
```

```
inner join tbturma t on t.fkcodprof=p.pkcodprof
```

```
where t.pkcodturma<250)
```

- 5) () Liste todos os dados da tabela de turma além da maior e menor média dessa turma se existir.

```
select t.*, mm.maior, mm.menor from tbturma t
```

```
left join (select at.fkcodturma as cod, max(at.me-  
dia) as maior, min(at.media) as menor from tba-  
lunoturma at group by at.fkcodturma) mm on  
t.pkcodturma=mm.cod
```

Gabarito:

1 – V

2 – V

3 – F (O único erro é que está sendo ordenado pelo nome da turma e não pelo nome do curso)

4 – V

5 – V

Referências

Elmasri, Rames: Sistemas de Banco de Dados. 6º Edição. São Paulo. Editora Personal, 2011.

Date, C J,. Introdução a Banco de Dados. 8º Edição. São Paulo. Editora Campus, 2003.

Rob. Peter e Coronel, Carlos. Sistemas de Banco de Dados: Projeto implementação e administração. 8º Edição. São Paulo. Editora Cengage Learning, 2011.

MEDEIROS, L. F. Banco de Dados: princípios e prática. Curitiba: Ibpex, 2007.D