Celso Henrique Poderoso de Oliveira

SQL 99

Curso Prático

SQL

Curso Prático

SQL

Curso Prático

Celso Henrique Poderoso de Oliveira

Editor: Rubens Prates
Capa: Camila Mesquita

ISBN: 85-7522-024-1

Histórico de impressões:

Setembro/2010	Sexta reimpressão
Dezembro/2009	Quinta reimpressão
Outubro/2008	Quarta reimpressão
Outubro/2006	Terceira reimpressão
Agosto/2005	Segunda reimpressão
Maio/2004	Primeira reimpressão
Agosto/2002	Primeira edição

Novatec Editora Ltda.
Rua Luís Antônio dos Santos 110
02460-000 – São Paulo, SP – Brasil
Tel.: +55 11 2959-6529
Fax: +55 11 2950-8869
E-mail: novatec@novatec.com.br
Site: www.novatec.com.br
Twitter: twitter.com/novateceditora
Facebook: facebook.com/novatec

Sumário

Objetivo deste livro

Ao escrever este livro, tive a intenção de apresentar a linguagem SQL àqueles que estão iniciando o estudo dessa poderosa ferramenta de manipulação de banco de dados. Contudo, entendo que muitos dos usuários de banco de dados estejam habituados a uma única implementação. Isso dificulta a compreensão do padrão aberto para o qual foi concebida a linguagem SQL. Portanto, mesmo os usuários experientes poderão extrair importantes conceitos de banco de dados de uma forma genérica e ampla.

Partimos do princípio de que o leitor nada conhece de banco de dados, iniciando pela concepção básica das estruturas de um banco de dados. Exercícios são incluídos para fixar o conteúdo apresentado e para incentivá-lo, leitor, a desenvolver um caso paralelo ao apresentado.

Com minha experiência em ministrar cursos de formação para essa área, *entendo ser possível, a qualquer pessoa interessada em aprender,* desenvolver seu conhecimento em banco de dados.

Não estamos restritos ao que é essencial da linguagem SQL. Vamos além disso. Exploraremos os diversos recursos da linguagem apresentando exemplos práticos e propondo exercícios complementares.

Para vencer esse desafio, é necessário que você tenha um banco de dados instalado para testes. Pode ser Oracle, SQL Server, Sybase, DB/2, MySQL, PostgreSQL ou mesmo Access. O importante é que você possa testar os exemplos e realizar os exercícios pedidos. Sozinho ou em grupo você conseguirá dominar essa importante ferramenta.

Organização deste livro

Neste livro será utilizado o padrão SQL-92 e, quando viável, o SQL-99. Contudo, nem todos os padrões sugeridos estão devidamente implementados por todos os fornecedores de banco de dados. Alguns bancos de dados implementam a parte básica ou intermediária dos comandos e acrescentam outros comandos específicos, que só funcionam no próprio banco de dados. Não utilizaremos nenhum banco de dados específico, ainda que os exemplos mostrados neste livro tenham sido testados no banco de dados Oracle. Tentaremos mostrar as diferenças existentes entre os diversos bancos de dados, e assim você poderá pesquisar na documentação do seu banco de dados, para efetuar eventuais ajustes.

Os comandos utilizados seguirão o padrão da SQL-99 (ou SQL3) e abordaremos a criação de objetos e tipos de dados definidos pelo usuário apenas como um referencial para estudo.

Da mesma forma iremos nos concentrar na prática do dia a dia de um profissional de informática, sem nos preocuparmos demais com a teoria. Isso não quer dizer que vamos abandoná-la. Sabemos a importância que o estudo aprofundado de um assunto representa para o conhecimento, mas sabemos também que é necessário vincular a teoria à prática de forma sutil e profunda. Isso quer dizer que mesmo uma pessoa com poucos conhecimentos técnicos será capaz de acompanhar e realizar os exercícios propostos. Mas aqueles que pretendem se aprofundar no assunto também encontrarão a teoria necessária para isso.

O livro está dividido em três partes:

- Conceitos e Modelagem de Dados: parte da teoria de banco de dados, incluindo conceitos de modelagem e normalização de dados, até chegar ao modelo de dados e à criação das tabelas utilizadas para trabalhar com os comandos SQL que serão discutidos posteriormente.

- SQL Básico: aborda os comandos SELECT, INSERT, DELETE e UPDATE em profundidade, incluindo a busca em múltiplas tabelas, funções de grupo, ordenação de dados, entre outros.

- SQL Avançado: ensina a criar estruturas avançadas do SQL, como visões, e identificar os padrões para atribuição e revogação de direitos. Trata também do controle sobre transações.

Ao final de cada capítulo são propostos exercícios para a fixação dos conceitos. Conforme dito anteriormente, é importante ter um banco de dados para testar os exemplos dados e resolver os exercícios apresentados.

Para facilitar a criação do modelo-exemplo utilizado neste livro, os *scripts* para criação das tabelas e inclusão dos dados utilizados como exemplo durante todo o livro estão no site www.novateceditora.com.br/downloads.php.

Parte I

Conceitos e Modelagem de Dados

1

Introdução

Desde o início da utilização dos computadores, sabemos que um sistema é feito para aceitar entrada de dados, realizar processamentos e gerar saída das informações processadas. Com o tempo, verificou-se a necessidade de armazenar as informações geradas pelos programas de computadores. O armazenamento e a recuperação das informações passaram a desempenhar um papel fundamental na informática.

Em junho de 1970, E. F. Codd, membro do Laboratório de Pesquisa da IBM em San Jose, na Califórnia, publicou um trabalho intitulado "A Relational Model of Data for Large Shared Data Banks" (Um Modelo Relacional de Dados para Grandes Bancos de Dados Compartilhados), no jornal *Association of Computer Machinery*. Nesse trabalho, Codd estabeleceu princípios sobre gerência de banco de dados, denominando-os com o termo relacional. Essa foi a base utilizada na criação de uma linguagem-padrão para manipular informações em Banco de Dados Relacionais. E essa linguagem é a SQL (Structured Query Language).

Inicialmente chamada de SEQUEL (Structured English Query Language), a linguagem SQL foi concebida e desenvolvida pela IBM, utilizando os conceitos de Codd. Em 1979, a Relational Software Inc., hoje Oracle Corporation, lançou a primeira versão comercial da linguagem SQL.

Atualmente, a SQL pode ser considerada um padrão para manipulação de dados em banco de dados. Duas entidades, a ANSI (American National Standards Institute) e a ISO (International Standards Organization), vêm ao longo do tempo padronizando a linguagem SQL. O primeiro padrão (SQL-86) foi definido pela ANSI em 1986 e consistia basicamente na SQL da IBM, com poucas modificações. Em 1987, a ISO adotou o mesmo padrão. Em 1989, surge uma nova versão (SQL-89), com significativas modificações. Essa versão é utilizada pelos bancos de dados atuais.

Em 1992, houve uma nova versão aprimorando a anterior (SQL-92). Essa versão define as regras básicas para os bancos de dados relacionais. Em 1999, surge a SQL-99, conhecida também como SQL3, que define um modelo de banco de dados objeto-relacional. Há poucos bancos de dados com essa implementação definida, mas utilizaremos ao máximo esse padrão sem esquecer aquilo que efetivamente está disponível no mercado. Note que algumas implementações de bancos de dados existentes no mercado adotam alguns comandos de maneira diferente, pois a SQL-99 é apenas um padrão que não precisa ser necessariamente seguido pelos fabricantes.

Divide-se o padrão SQL-92 em quatro níveis: *Entry* (básico), *Transational* (em evolução), *Intermediate* (intermediário) e *Full* (completo). A maior parte dos bancos de dados utilizados atualmente atende ao nível básico. Mesmo existindo uma versão mais nova do padrão, a maior parte dos bancos de dados ainda utiliza, de forma básica, o padrão anterior. Alguns comandos, contudo, atingem os níveis intermediário e completo.

O que é SQL?

SQL (Structured Query Language) é um conjunto de comandos de manipulação de banco de dados utilizado para criar e manter a estrutura desse banco de dados, além de incluir, excluir, modificar e pesquisar informações nas tabelas dele. A linguagem SQL não é uma linguagem de programação autônoma; poderia ser chamada de "sublinguagem". Quando se escrevem aplicações para banco de dados, é necessário utilizar uma linguagem de programação tradicional (C, Java, Pascal, COBOL etc.) e embutir comandos SQL para manipular os dados.

Em um modelo relacional, apenas um tipo de estrutura de dados existe: a tabela. Novas tabelas são criadas com a junção ou combinação de outras tabelas. Utilizando apenas um comando SQL é possível pesquisar dados em diversas tabelas ou atualizar e excluir diversas linhas de tabelas.

A linguagem SQL não é *procedural*, logo é possível especificar o que deve ser feito, e não como deve ser feito. Dessa forma, um conjunto de linhas (*set*) será atingido pelo comando e não cada uma das linhas, como é feito no ambiente *procedural*. Portanto, não é necessário entender o funcionamento interno do banco de dados e como e onde estão armazenados fisicamente os dados.

Teoricamente deveria ser possível transferir facilmente os comandos SQL de um banco de dados para outro. Contudo, isso não é possível. Naturalmente, boa parte do trabalho poderá ser aproveitado, mas deve-se fazer adaptações em função do banco de dados que está sendo utilizado.

A linguagem SQL é dividida nos seguintes componentes:

- Data Definition Language (DDL): permite a criação dos componentes do banco de dados, como tabelas, índices etc.

  Principais comandos DDL:

  CREATE TABLE

  ALTER TABLE

  DROP TABLE

  CREATE INDEX

  ALTER INDEX

  DROP INDEX

- Data Manipulation Language (DML): permite a manipulação dos dados armazenados no banco de dados.

  Comandos DML:

  INSERT

  DELETE

  UPDATE

- Data Query Language (DQL): permite extrair dados do banco de dados.

  Comando:

  SELECT

- Data Control Language (DCL): provê a segurança interna do banco de dados.

  Comandos DCL:

  CREATE USER

  ALTER USER

  GRANT

  REVOKE

  CREATE SCHEMA

Com o advento da SQL-99, a linguagem SQL passou a incorporar comandos *procedurais* (BEGIN, IF, funções, procedimentos) que, na prática, já existiam como extensões da linguagem. Essas extensões, até hoje, são específicas de cada banco de dados e, portanto, a Oracle tem a sua própria linguagem *procedural* que estende a SQL, que é a PL/SQL. A Microsoft incorporou no SQL Server o Transact-SQL com o mesmo objetivo. A idéia é que, num futuro próximo, exista um padrão de programação em todos os bancos de dados (veja na Parte III — SQL Avançado algumas implementações importantes).

2

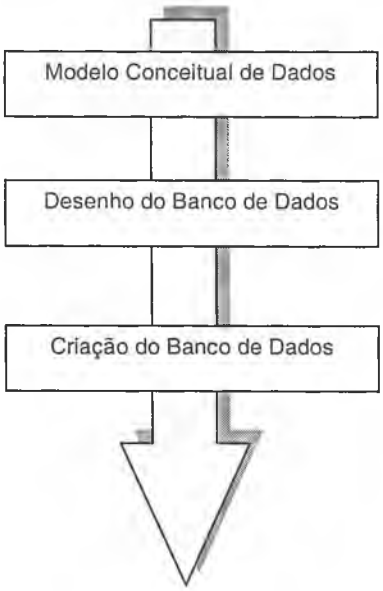
Projetando bancos de dados

Antes de utilizarmos os comandos SQL, vamos identificar a forma de planejar a criação do banco de dados. Esse planejamento é extremamente importante para a estabilidade de todo o sistema. Estudos indicam que quanto maior o tempo despendido no projeto do banco de dados, menor será o tempo despendido na manutenção do modelo.

Podemos comparar a criação de um sistema com a construção de um edifício. O projeto do banco de dados está para o sistema da mesma forma que a estrutura do prédio está para o edifício. Se não for dada a devida atenção ao desenho do banco de dados, pode-se comprometer todo o desenvolvimento do sistema. É como construir um edifício utilizando uma base inadequada: um dia o edifício cairá. De outra forma, quanto maior for o tempo dedicado ao estudo das necessidades de informação do sistema em desenvolvimento, maior será o tempo economizado no desenvolvimento do sistema. O sistema terá melhor qualidade, e será mais fácil, no futuro, implementar novas rotinas, procedimentos e agregar novas informações necessárias.

Veja um exemplo: imagine que estamos desenvolvendo um sistema para uma empresa pequena que tenha apenas uma sede. Seria normal criar um cadastro da empresa e este estaria relacionado com diversas outras tabelas do sistema, como nota fiscal, estoque etc. Podemos montar um modelo de dados que espelhe esta realidade atual. Mas digamos que amanhã essa empresa tenha suas atividades expandidas (e geralmente esquecemos que isso pode acontecer) e venha a ter diversas filiais; logo precisaremos de uma tabela para guardar os dados das filiais. Se tivéssemos pensado nisso antes (a empresa é um objeto, o local onde ela está é outro objeto), não haveria problemas. Se não pensamos, devemos criar uma nova tabela e efetuar novos relacionamentos, alterando uma série de outras tabelas para incluir o novo campo de filial, gerando trabalho de conversão das informações e modificação dos módulos relacionados à empresa.

O processo de Análise dos Dados pressupõe três fases distintas e integradas:



Banco de Dados Operacional

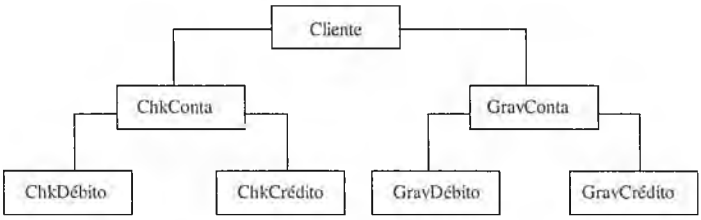
Banco de dados

Um banco de dados é um conjunto coerente e lógico de dados relacionados que possuem significância intrínseca. Esses dados representam aspectos do mundo real e devem ser mantidos para atender aos requisitos da empresa.

Esses dados estão dispostos em uma ordem predefinida para atender a determinadas necessidades dos usuários. Existem diversos objetos que podem ser armazenados em um banco de dados, como índices, visões, procedimentos e funções.

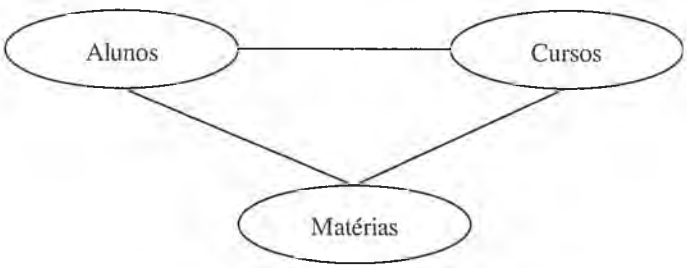
Há cinco tipos de banco de dados:

- *Hierárquico*: um gerenciador desse tipo representa dados como uma estrutura de árvore, composto de uma hierarquia de registro de dados. Exemplo:



Nesse caso, há dados específicos em CLIENTE (nome, endereço, cidade, Estado etc); os segmentos dependentes do CLIENTE são CHKCONTA e GRAVCONTA, que, por sua vez, têm cada um outros dois segmentos dependentes conforme demonstrado no gráfico. Cada um dos segmentos inferiores depende hierarquicamente dos segmentos superiores. Cada um contém dados específicos. É comum encontrá-los em ambientes de grande porte, como, por exemplo, em implementações IMS.

- *Rede*: representa os dados como registros vinculados uns aos outros, formando conjuntos comuns de dados. Existe uma similaridade muito grande entre o modelo hierárquico e o modelo de rede. Pode-se entender o modelo de rede como uma generalização do modelo hierárquico, ou este último como um caso particular do modelo de rede. No modelo de rede, um filho pode ter mais de um pai. Exemplo:



- *Relacional*: representa os dados como uma simples coleção de linhas e colunas em tabelas bidimensionais. Exemplo:

- *Objeto-Relacional*: combina o modelo orientado a objetos (união de propriedades e métodos) com o modelo relacional (linhas e colunas de tabelas).

- *Objeto*: representa os dados e processos em um único objeto. Exemplo:

Gerenciador de banco de dados

Um gerenciador de banco de dados (DBMS – *Database Management System*) é uma coleção de programas que permite criar estruturas, manter dados e gerenciar as transações efetuadas em tabelas, além de permitir a extração das informações de maneira rápida e segura.

Algumas das principais características de um gerenciador de Banco de Dados:

- *Controle de redundância*: informações devem possuir um mínimo de redundância visando estabelecer à estabilidade do modelo.

- *Compartilhamento de dados*: as informações devem estar disponíveis para qualquer número de usuários de forma concomitante e segura.

- *Controle de acesso*: necessidade de saber quem pode realizar qual função dentro do banco de dados.

- *Esquematização*: os relacionamentos devem estar armazenados no banco de dados para garantir a facilidade de entendimento e aplicação do modelo. A integridade das informações deve ser garantida pelo banco de dados.

- *Backup ou cópias de segurança*: deve haver rotinas específicas para realizar a cópia de segurança dos dados armazenados.

A segurança é um fator que muito diferencia um banco de dados de outro. Às vezes, há dois bancos de dados que utilizam o SQL como linguagem de acesso e manipulação de dados, mas que têm estruturas de controle de acesso completamente diferentes. Um gerenciador de banco de dados deve prever o controle de acesso às informações e garantir por meio de recursos internos que os dados sejam disponibilizados rapidamente.

O objetivo de um banco de dados relacional é armazenar um grupo de objetos em um dicionário de dados, de forma a tornar rápida e segura a manipulação das informações contidas nesses objetos. Como objetos, podemos entender tabelas, visões, índices e até mesmo procedimentos e funções que estejam armazenadas no banco de dados. O objeto estudado nesta seção do livro é a tabela.

Tabela

Uma tabela pode ser entendida como um conjunto de linhas e colunas. As colunas de uma tabela qualificam cada elemento (no caso, a linha) com informações relacionadas ao objeto.

Utilizando esses conceitos, é possível armazenar dados em uma ou várias tabelas, dependendo do que e como desejamos as informações.

Abordagem Relacional: o modelo de Entidade x Relacionamento

A Abordagem Relacional é a utilização de conceitos de Entidade e Relacionamento para criar as estruturas que irão compor o banco de dados. Partindo sempre da necessidade do usuário ou grupo de usuários do sistema, iniciamos a pesquisa das necessidades de informação desses usuários.

A definição do escopo do sistema é, portanto, importante para o início do trabalho de análise de dados.

É comum no início do desenvolvimento de um sistema que não tenhamos a noção exata da tarefa a ser realizada. O maior erro nessa fase é admitir que já sabemos o que deve ser feito, seja por experiência anterior, seja por falta de tempo para conversar com os usuários do sistema.

Para minimizar esse problema, devemos criar uma estrutura gráfica que permita identificar as Entidades de um sistema e como estas se relacionam.

Nessa fase é importante saber quais informações são importantes para o sistema e *o que* deve ser armazenado (note que não utilizamos o *como* deve ser armazenado; isso será discutido na fase posterior). A esta representação gráfica dá-se o nome de Modelo de Dados.

Devemos notar que o Modelo de Dados dará suporte a toda a empresa, incorporando as informações necessárias para o andamento dos negócios. Ele será composto de Entidades e Relacionamentos, daí ser conhecido por Modelo de Entidade x Relacionamento (MER).

Vantagens na utilização do Modelo de Entidade x Relacionamento:

- Sintaxe robusta: o modelo documenta as necessidades de informação da empresa de maneira precisa e clara.
- Comunicação com usuário: os usuários podem, com pouco esforço, entender o modelo.
- Facilidade de criação: os analistas podem criar e manter um modelo facilmente.
- Integração com várias aplicações: diversos projetos podem ser inter-relacionados utilizando-se o modelo de dados de cada um deles.
- Utilização universal: o modelo não está vinculado a um banco de dados específico, mas sim ao modelo da empresa, o que garante sua independência de implementação.

Objetivos da Modelagem de Dados

O principal objetivo da Modelagem de Dados é desenvolver um modelo que, contendo entidades e relacionamentos, seja capaz de representar os requerimentos das informações do negócio.

Veja o que poderia ser um exemplo de catálogo de CDs:

Cód.	Nome do CD	Nome da Música	Nome do Autor
01	Mais do Mesmo	Será	Renato Russo e...
02	Mais do Mesmo	Ainda é Cedo	Renato Russo e...
03	Mais do Mesmo	Tempo Perdido	Renato Russo
04	Bate-Boca	Meninos, Eu Vi	Tom Jobim e...
05	Bate-Boca	Eu te Amo	Tom Jobim e...

Um dos principais problemas relacionados com bancos de dados é a redundância (repetição) das informações. Sempre que houver duas informações, nunca se saberá em qual delas pode confiar.

Imagine que na tabela exemplo alguém altere o nome do CD apenas na linha 2 para Mais ou Menos. Qual dos nomes estaria correto, Mais do Mesmo ou Mais ou Menos? É por isso que devemos criar um banco de dados com um mínimo de redundância, evitando esse problema.

Exatamente para evitar a redundância é que se cria uma série de tabelas no banco de dados, e não apenas uma. Naturalmente isso aumenta a complexidade da operação, mas traz uma enorme vantagem ao evitar a redundância.

Um outro objetivo é a economia de espaço. Quando se admite a redundância, é muito comum ter que repetir nomes, descrições, datas etc. Ao isolarmos essas informações em tabelas distintas e ao relacionarmos as tabelas por um código comum estamos economizando espaço de armazenamento.

No exemplo anterior, identificamos que há diversos dados redundantes: código do CD, nome do CD, nome da Gravadora, preço e autor. Além do mais, a simples repetição desses campos representa um espaço gasto sem necessidade. Podemos separar as informações em mais de uma tabela, armazenando apenas uma vez cada informação distinta e relacionando as tabelas.

Exemplo: podemos criar uma tabela para armazenar os dados dos CDs (código do CD, nome do CD, nome da gravadora e preço) e outra para armazenar as Músicas (Número da Faixa, Nome, Autor e Tempo). Basta acrescentar o código do CD em Músicas e teríamos uma relação entre Música e CD. Contudo, somente isso não é o suficiente.

Estudaremos mais detalhadamente Entidade e Relacionamento, para montar um modelo melhor que este.

Entidade

Entidade é um agrupamento lógico de informações inter-relacionadas necessárias para a execução das atividades do sistema. Uma entidade normalmente representa um objeto do mundo real ou, quando não é, contém informações relevantes às operações da empresa.

Quando transposta ao modelo físico (ao banco de dados), chamamos a entidade de tabela. Uma entidade é entendida como um objeto concreto ou abstrato do sistema. São informações necessárias e que, portanto, devem ser armazenadas.

Ao transpor o Modelo Relacional a um modelo Orientado a Objeto, a Entidade passa a ser uma Classe ou categoria do objeto ao qual agregaremos os respectivos métodos.

Cada entidade deve conter múltiplas ocorrências ou instâncias do objeto que representa. Isso não permitirá incorrer no erro de confundir a Entidade com a Instância. A entidade é a classe ou categoria (CD), e a Instância é um objeto específico (no exemplo, Mais do Mesmo ou Bate-Boca).

Exemplos de Entidade

Físicas ou Jurídicas	Pessoas, funcionários, clientes, fornecedores e empresa.
Documentos	Ordem de compra, pedido e nota fiscal.
Local	Almoxarifado e departamento.
Tabelas	Classificação fiscal, centro de custo e UF.
Matéria	Produto e peça.

As Entidades podem ser classificadas em dois tipos:

Classificação	Descrição
Fundamental	Contém dados básicos que são resultados ou alimentadores das operações da empresa.
Associativa	É formada pelo Relacionamento de duas Entidades Fundamentais sempre que estas se relacionarem mais de uma vez. Exemplo: aluno x matéria, CD x Autor, pedido x produto etc.

Entidades Associativas

Há um caso específico para as Entidades Associativas: sempre que, além do simples relacionamento entre as duas entidades fundamentais, houver outras informações específicas da nova entidade criada (como, por exemplo, a quantidade e o valor entre pedido x produto ou bimestre, nota e faltas do aluno x matéria), ela será chamada de entidade Associativa Atributiva.

No catálogo de CD dado como exemplo, podemos identificar facilmente duas entidades: CD e Música. Observando com mais cuidado, vê-se que Gravadora e Autor também possuem uma estrutura independente. Isso porque há outras informações que, apesar de não estarem descritas na planilha, são de fato apenas da Gravadora e do Autor. Exemplo: endereço, data de nascimento, telefone etc. Para isso é importante entender o que são os atributos (características) de uma Entidade.

Atributos

Os atributos são as informações básicas que qualificam uma entidade e descrevem seus elementos ou características. Quando transpostos ao modelo físico (ao banco de dados), chamamos os atributos de campos ou colunas.

Note que todas as entidades devem possuir os atributos necessários ao andamento das operações da empresa, do contrário a entidade não será necessária para o sistema. Esses atributos devem representar o objeto na sua totalidade.

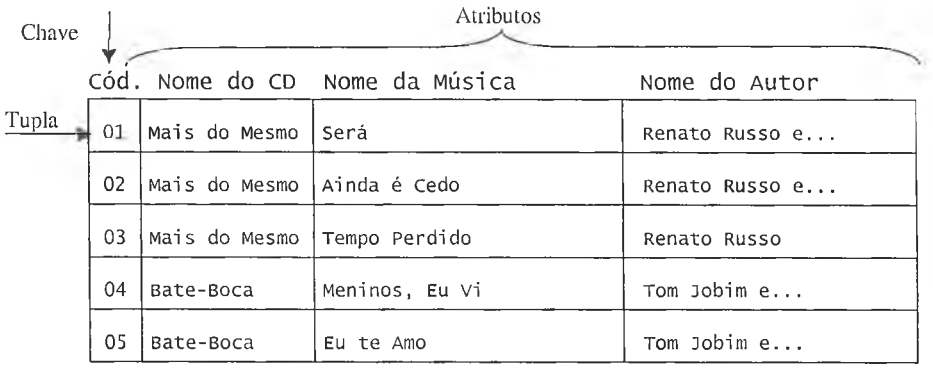
Há uma tendência a confundir Entidade e Atributo. Tenha sempre em mente que um Atributo é uma característica, logo não contém um grupo de informações. Por sua vez, uma Entidade sempre é um grupo. No mínimo são necessários dois atributos para criar uma entidade. Uma entidade com um único atributo normalmente será agregada a outra entidade existente no modelo.

Exemplos de atributos para as entidades:

- Entidade Pessoa: nome, endereço, documento, data de nascimento, telefone e e-mail.
- Entidade Nota Fiscal: série, número, data de emissão e cliente.

Nota-se, portanto, que ao utilizarmos o conceito de atributos em entidades estamos querendo qualificar ao máximo aquele objeto do mundo real. Essas informações muitas vezes não correspondem a todas as informações possíveis daquele objeto, mas sim às informações relevantes para o funcionamento do sistema.

No exemplo do catálogo de CDs, não teremos cada um dos CDs armazenados no banco de dados, mas sim as características que nos permitirão identificar qual CD o cliente quer comprar, quais músicas há naquele CD, autores, gravadoras etc. Não importa se há várias unidades do mesmo CD disponíveis para venda na loja (a menos que se esteja desenvolvendo um sistema que controle o estoque de CDs). Esta deve ser uma preocupação quando estivermos desenvolvendo um novo sistema — até onde exatamente queremos chegar com o sistema.



Cód.	Nome do CD	Nome da Música	Nome do Autor
01	Mais do Mesmo	Será	Renato Russo e...
02	Mais do Mesmo	Ainda é Cedo	Renato Russo e...
03	Mais do Mesmo	Tempo Perdido	Renato Russo
04	Bate-Boca	Meninos, Eu Vi	Tom Jobim e...
05	Bate-Boca	Eu te Amo	Tom Jobim e...

Tupla

É uma estrutura de atributos intimamente relacionados e interdependentes que residem em uma entidade.

Quando transposta ao modelo físico, uma tupla equivale a um registro ou linha da tabela.

Chave

É um atributo utilizado para indexar dados.

Há três tipos de chave:

a. Primária.

b. Estrangeira.

c. Secundária.

Chave primária

É o atributo que permite identificar uma única ocorrência de uma tupla em uma Entidade.

Dessa forma, seu conteúdo deve ser único, exclusivo e imutável para cada linha dessa Entidade. Todos os demais atributos da entidade devem depender unicamente desse atributo.

Caso não exista um atributo que possa assumir a posição de chave primária, é preciso criá-lo. Veja que nem todo campo é uma boa chave. Normalmente utilizamos campos numéricos por serem localizados mais rapidamente pelos bancos de dados. Valores alfanuméricos grandes têm acesso mais lento.

Dessa forma, fica claro que toda tabela deve conter uma chave primária. Muitas vezes encontramos o termo superchave para identificar a chave primária. Trata-se apenas de um nome diferente para designar a mesma coisa, e, portanto, não é preciso se preocupar com isso.

Eventualmente uma chave primária pode conter mais de um atributo. Nesse caso, a chave conterá mais de um atributo, mas será considerada a chave da tabela. A união dos dois atributos é que deve garantir o acesso a uma única linha da entidade. Esse caso de chave primária é chamado de Chave Concatenada. Veremos alguns desses exemplos mais adiante.

Chave estrangeira

É o atributo que estabelece a relação de uma Entidade com a Chave Primária de outra Entidade e permite uma relação entre entidades, conforme veremos no tópico Relacionamento. Isto ocorre quando uma Entidade dependente herda a chave da Entidade Fundamental exatamente para estabelecer o relacionamento entre elas.

Entidade CD

Chave Primária

01	Mais do Mesmo
02	Bate-Boca

Entidade Música

Chave Primária

Chave Estrangeira

01	Será	01
02	Ainda é Cedo	01
03	Tempo Perdido	01
04	Meninos, Eu Vi	02
05	Eu te Amo	02

Chave Secundária

Esta chave é utilizada como meio de classificação e pesquisas em entidades.

Sempre que houver a necessidade de buscar informações semelhantes, em ordem crescente ou decrescente, em função de datas, valores ou status predefinidos, criam-se chaves secundárias.

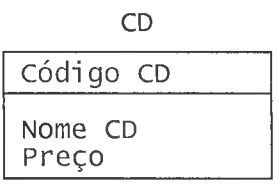
Podem também ser concatenadas a outras chaves secundárias para extrair a informação desejada.

Convenção para utilização em Diagramas

Deve-se utilizar uma caixa de qualquer dimensão com um nome único (exclusivo) em cada uma das caixas. Esses nomes devem representar as Entidades do sistema.

Alguns autores preferem utilizar a caixa com bordas arredondadas. Isso é apenas uma convenção diferente, que em nada modificará o objetivo e a compreensão da Entidade. Nos exemplos deste livro serão utilizadas as bordas arredondadas para indicar as Entidades que, de alguma forma, dependem de outras. As caixas sem as bordas arredondadas serão utilizadas apenas para as Entidades Independentes ou Fundamentais.

A seguir será utilizado o nome da Entidade fora da caixa e separada a Chave Primária dos demais atributos com uma linha horizontal:



Pratique

Para identificar as entidades, tente seguir os passos a seguir:

1. Examine os substantivos. Eles são objetos com significado próprio?
2. Dê um nome a cada Entidade.
3. Há informação relevante a respeito da entidade necessária às operações da empresa?
4. Cada instância da entidade possui um identificar único (chave)?
5. Escreva uma descrição da suposta Entidade (CD é o produto básico de venda da empresa. Exemplos de CDs são: Mais do Mesmo e Bate-Boca).
6. Faça um diagrama com, pelo menos, alguns de seus atributos.

Relacionamento

Sempre que duas entidades apresentarem interdependência (por exemplo, autor da música ou música do CD), indica-se um relacionamento entre elas.

Deve-se perguntar a cada par de entidades se elas se relacionam. Para facilitar esse trabalho, siga o esquema abaixo:

Assim, podemos dizer que:

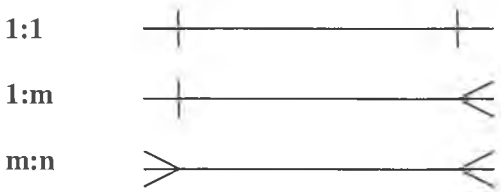
1. Cada CD deve ser gravado por uma única gravadora.
2. Cada gravadora pode ter gravado um ou mais CDs.
1. Cada autor pode ter escrito uma ou mais músicas.
2. Cada música pode ser escrita por um ou mais autores.
1. Cada música pode estar gravada em um ou mais CD.
2. Cada CD deve conter uma ou mais músicas.
1. Cada CD pode ser indicado por um ou mais CDs.
2. Cada CD pode indicar um único CD.

Conforme você pode notar, cada relacionamento contém um nome (normalmente um verbo como ser gravado, conter, ter escrito), a determinação de opcionalidade (deve ou pode) e um grau ou cardinalidade (uma única ou uma ou mais).

Convenção para utilização em diagramas

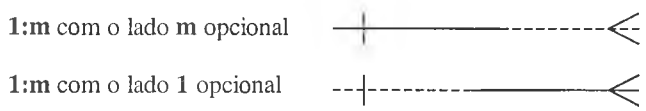
Utiliza-se uma linha para unir as duas entidades que se relacionam. Essa linha poderá ser contínua (caso seja obrigatório o critério de opcionalidade - deve) ou tracejada (caso não seja obrigatório esse critério - pode). Conterá na extremidade um tridente caso a cardinalidade seja uma ou mais ou conterá apenas um traço quando for um único.

Existem três possíveis relacionamentos:

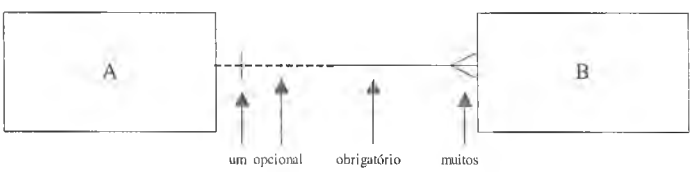


E, em cada um dos lados, o traço poderá estar tracejado, indicando que o relacionamento, é opcional.

Exemplo:



Dessa forma, essa representação gráfica, de duas entidades e um relacionamento poderia ser descrita como:



1. Cada entidade A pode conter uma ou mais entidade (s) B.

2. Cada entidade B deve estar contida em uma única entidade A.

Alguns autores preferem utilizar um círculo e outro traço vertical para identificar a ocorrência de zero (círculo) ou um (traço) em cada uma das extremidades. Mais uma vez: isso é uma convenção que em nada modifica o critério de cardinalidade proposto.

Análise dos tipos de Relacionamentos (Cardinalidade)

Conforme visto anteriormente, há três tipos de relacionamentos: um para um (1:1), um para muitos (1:n) ou muitos para muitos (m:n). A seguir, alguns detalhes de implementação de cada um deles:

Relacionamento 1:1

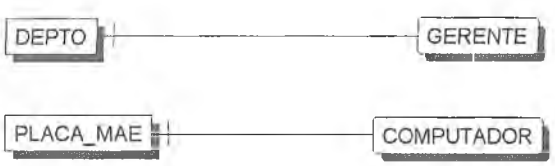
Ocorre sempre que uma entidade tiver uma única ocorrência para cada ocorrência na outra entidade.

Sempre que houver esse relacionamento, deve-se perguntar se realmente são duas entidades distintas ou se elas podem ser unidas. Normalmente, ao checarmos a chave de ambas as entidades, chegamos facilmente à conclusão se as entidades devem ou não ser unidas. Da mesma forma, deve-se perguntar se esse relacionamento sempre será um para um ou se existe a possibilidade de, amanhã, vir a ser um para muitos.

Note que esse relacionamento é efetivamente raro. Relacionamentos em que seja obrigatória em ambas as entidades são mais raros ainda.

No exemplo a seguir, cada departamento é gerenciado por um gerente, e cada gerente gerencia um departamento. As chaves são distintas (são objetos absolutamente diferentes), mas é interessante nos questionarmos, mesmo que eventualmente, se um gerente não pode gerenciar mais de um departamento. Isso pode ou não ocorrer, dependendo da empresa, mesmo que seja por um curto período de tempo. Nesse caso, o relacionamento deveria ser trocado para um para muitos (1:n).

Isso também ocorre com o relacionamento entre Computador e Placa Mãe. Essa pergunta deve ser feita quanto à possibilidade de um computador possuir mais de uma Placa Mãe no futuro.



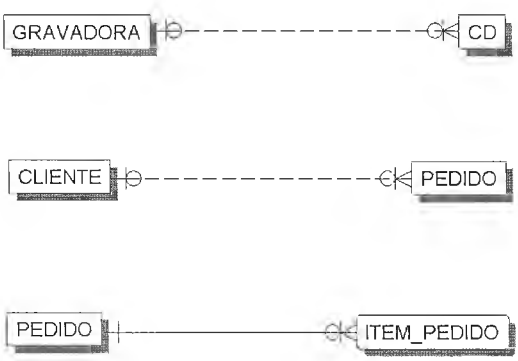
Relacionamento 1:n

Ocorre sempre que uma entidade se relacionar com uma ou mais tuplas da outra entidade e esta outra se relacionar apenas com uma tupla daquela entidade.

Esse relacionamento é o mais comum e fácil de ser analisado. Nesse caso, a parte onde o relacionamento é 1 contém os dados básicos da entidade (pois é a chave primária dessa entidade) e o lado muitos fará parte da lista de atributos não chave.

Relacionamentos desse tipo raramente são obrigatórios em ambas as entidades. A exceção é quando se trata de itens de uma entidade, como itens de nota fiscais ou pedidos. Mas esse parece ser um caso especial de muitos para muitos, conforme será visto adiante.

A seguir, exemplos em que isso ocorre: Cada Gravadora grava vários CDs e cada CD é gravado apenas por uma Gravadora. Cada Cliente possui vários Pedidos e cada Pedido é de um único Cliente.



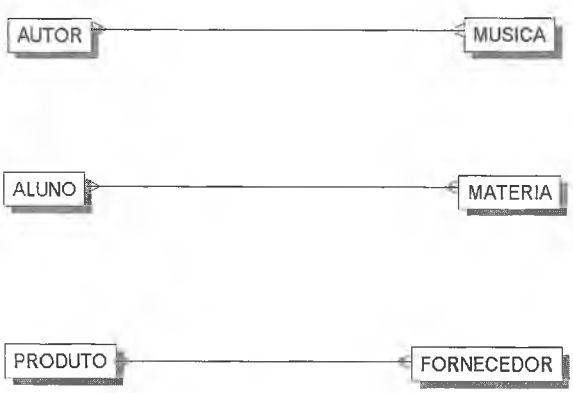
Relacionamento m:n

Ocorre sempre que uma entidade se relacionar com várias tuplas de outra entidade e esta, por sua vez, relacionar-se com várias tuplas daquela entidade.

Esse relacionamento somente é possível na modelagem lógica de dados, uma vez que não se consegue implantá-lo em bancos de dados relacionais. Ele será transformado em dois relacionamentos: um para muitos (1:n) e uma Entidade Associativa Atributiva será identificada, caso haja outras informações que devam ser agregadas a esta nova entidade - exemplo de item de pedido em que se identificam quantidade e preço, ou criada, caso seja a simples união das chaves primárias de ambas as entidades - caso de vários produtos serem fornecidos por vários fornecedores e vice-versa.

Esses relacionamentos são facilmente encontrados e normalmente são opcionais em ambas as entidades. Existem casos em que há opcionalidade apenas *em uma das duas direções.*

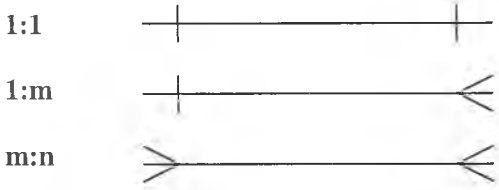
Nos exemplos a seguir, cada Música é composta por um ou vários Autores, *e cada Autor pode compor uma ou várias Músicas. Cada Produto é fornecido por vários Fornecedores e cada Fornecedor pode fornecer vários Produtos. E, finalmente, cada Aluno é inscrito em uma ou várias Matérias e cada Matéria possui vários Alunos.*



Convenção para utilização em diagramas

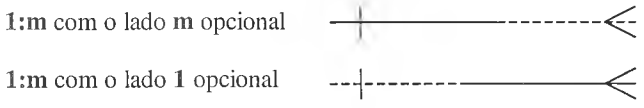
Utiliza-se uma linha para unir as duas entidades que se relacionam. Essa linha poderá ser contínua (caso seja obrigatório o critério de opcionalidade - deve) ou tracejada (caso não seja obrigatório esse critério - pode). Conterá na extremidade um tridente caso a cardinalidade seja uma ou mais ou conterá apenas um traço quando for um único.

Existem três possíveis relacionamentos:

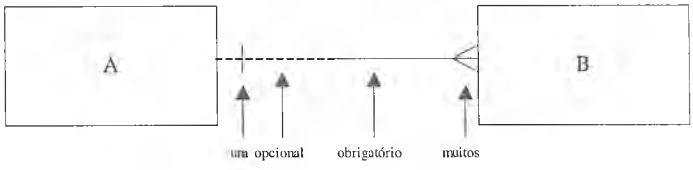


E, em cada um dos lados, o traço poderá estar tracejado, indicando que o relacionamento, é opcional.

Exemplo:



Dessa forma, essa representação gráfica, de duas entidades e um relacionamento poderia ser descrita como:



1. Cada entidade A pode conter uma ou mais entidade (s) B.

2. Cada entidade B deve estar contida em uma única entidade A.

Alguns autores preferem utilizar um círculo e outro traço vertical para identificar a ocorrência de zero (círculo) ou um (traço) em cada uma das extremidades. Mais uma vez: isso é uma convenção que em nada modifica o critério de cardinalidade proposto.

Análise dos tipos de Relacionamentos (Cardinalidade)

Conforme visto anteriormente, há três tipos de relacionamentos: um para um (1:1), um para muitos (1:n) ou muitos para muitos (m:n). A seguir, alguns detalhes de implementação de cada um deles:

Relacionamento 1:1

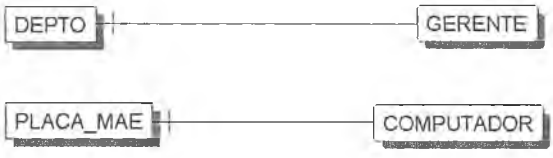
Ocorre sempre que uma entidade tiver uma única ocorrência para cada *ocorrência na outra entidade.*

Sempre que houver esse relacionamento, deve-se perguntar se realmente são duas entidades distintas ou se elas podem ser unidas. Normalmente, ao checarmos a chave de ambas as entidades, chegamos facilmente à conclusão se as entidades devem ou não ser unidas. Da mesma forma, deve-se perguntar se esse relacionamento sempre será um para um ou *se existe a possibilidade* de, amanhã, vir a ser um para muitos.

Note que esse relacionamento é efetivamente raro. Relacionamentos em que seja obrigatória em ambas as entidades são mais raros ainda.

No exemplo a seguir, cada departamento é gerenciado por um gerente, e cada gerente gerencia um departamento. As chaves são distintas (são objetos absolutamente diferentes), mas é interessante nos questionarmos, mesmo que eventualmente, se um gerente não pode gerenciar mais de um departamento. Isso pode ou *não ocorrer, dependendo* da empresa, mesmo que seja por um curto período de tempo. Nesse caso, o relacionamento deveria ser trocado para um para muitos (1:n).

Isso também ocorre com o relacionamento entre Computador e Placa Mãe. Essa pergunta deve ser feita quanto à possibilidade de um computador possuir mais de uma Placa Mãe no futuro.



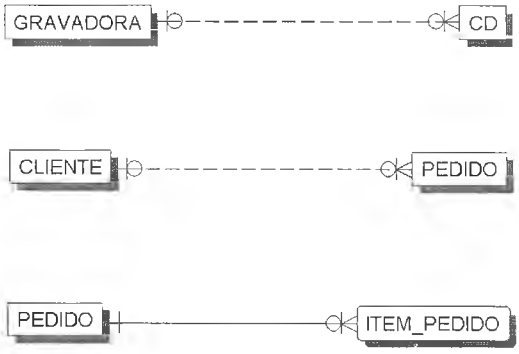
Relacionamento 1:n

Ocorre sempre que uma entidade se relacionar com uma ou mais tuplas da outra entidade e esta outra se relacionar apenas com uma tupla daquela entidade.

Esse relacionamento é o mais comum e fácil de ser analisado. Nesse caso, a parte onde o relacionamento é 1 contém os dados básicos da entidade (pois é a chave primária dessa entidade) e o lado muitos fará parte da lista de atributos não chave.

Relacionamentos desse tipo raramente são obrigatórios em ambas as entidades. A exceção é quando se trata de itens de uma entidade, como itens de nota fiscais ou pedidos. Mas esse parece ser um caso especial de muitos para muitos, conforme será visto adiante.

A seguir, exemplos em que isso ocorre: Cada Gravadora grava vários CDs e cada CD é gravado apenas por uma Gravadora. Cada Cliente possui vários Pedidos e cada Pedido é de um único Cliente.



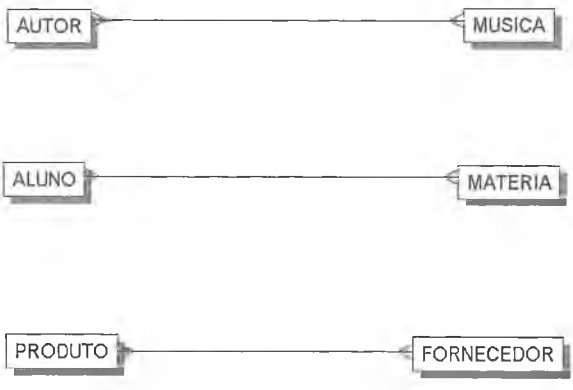
Relacionamento m:n

Ocorre sempre que uma entidade se relacionar com várias tuplas de outra entidade e esta, por sua vez, relacionar-se com várias tuplas daquela entidade.

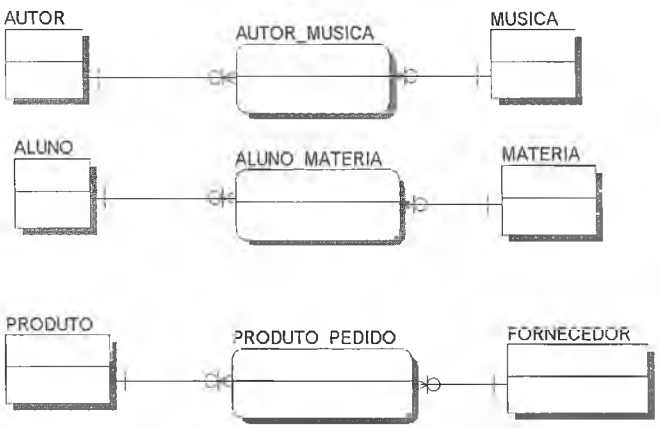
Esse relacionamento somente é possível na modelagem lógica de dados, uma vez que não se consegue implantá-lo em bancos de dados relacionais. Ele será transformado em dois relacionamentos: um para muitos (1:n) e uma Entidade Associativa Atributiva será identificada, caso haja outras informações que devam ser agregadas a esta nova entidade – exemplo de item de pedido em que se identificam quantidade e preço, ou criada, caso seja a simples união das chaves primárias de ambas as entidades – caso de vários produtos serem fornecidos por vários fornecedores e vice-versa.

Esses relacionamentos são facilmente encontrados e normalmente são opcionais em ambas as entidades. Existem casos em que há opcionalidade apenas em uma das duas direções.

Nos exemplos a seguir, cada Música é composta por um ou vários Autores, e cada Autor pode compor uma ou várias Músicas. Cada Produto é fornecido por vários Fornecedores e cada Fornecedor pode fornecer vários Produtos. E, finalmente, cada Aluno é inscrito em uma ou várias Matérias e cada Matéria possui vários Alunos.



A transformação à qual nos referimos fará com que para cada um dos relacionamentos anteriores seja criado um item que terá o relacionamento um para muitos (1:n), com cada uma das outras entidades. As novas entidades criadas serão formadas pela união das duas chaves primárias das entidades e, eventualmente, por novos atributos necessários. Não esqueça que caso novos atributos sejam identificados na entidade associativa, esta será considerada atributiva. Note que os relacionamentos das Entidades Associativas recebem de ambas as entidades fundamentais o lado "muitos" do relacionamento. Por sua vez, do lado das entidades fundamentais, fica o lado "um" do relacionamento.



Integridade referencial

É um mecanismo utilizado para manter a consistência das informações gravadas. Dessa forma, não são permitidas a entrada de valores duplicados nem a existência de uma referência a uma chave inválida em uma entidade.

É importante enfatizar o conceito de integridade referencial. É também necessário que cada valor de chave estrangeira possua uma ocorrência na outra entidade à qual faz referência. Se isso não ocorrer, fica claro que estaremos perdendo uma informação importante para o sistema. Exemplo: se tivermos um valor na entidade CD correspondente à gravadora (chave estrangeira) e não tivermos o mesmo valor (chave primária) na entidade Gravadora, estaremos diante de um problema, pois teríamos um CD sem uma Gravadora válida.

A maior parte dos bancos de dados relacionais estabelece esse tipo de relacionamento e impede que durante uma inclusão, exclusão ou alteração uma chave estrangeira de uma entidade não tenha correspondente na chave primária da outra entidade. Assim, em uma inclusão na entidade com a chave estrangeira, caso seja informado um código que não exista, correspondente na outra entidade, deve ser gerada uma mensagem de erro. Assim, caso queiramos incluir um CD com um código de Gravadora que não exista, correspondente na tabela Gravadora, deve ser enviada uma mensagem de erro e impedida a gravação da informação.

No caso de uma alteração ou exclusão na chave primária da entidade, deve-se verificar se há registros dependentes (chave estrangeira) nas demais tabelas. Se houver, deve-se excluir todos os registros dependentes ou alterá-los, dependendo do caso. Isso poderia ocorrer caso quiséssemos excluir ou alterar uma Gravadora e tivéssemos CDs armazenados com o código da Gravadora. Se fosse permitido, teríamos uma informação inválida, pois ao tentarmos localizar a Gravadora deste CD, isso não seria possível.

Caso seu banco de dados não disponha desse recurso, você deverá levar isso em consideração e criar mecanismos que evitem o problema. Quando o banco de dados dispuser desse recurso, este adotará o nome CONSTRAINT.

Em alguns bancos de dados é possível controlar até a propagação de exclusões ou alterações na chave primária: ou se apagam todos os registros dependentes ou se altera o seu conteúdo. Assim, caso excluíssemos ou alterássemos uma determinada Gravadora, todos os CDs dessa gravadora seriam automaticamente excluídos ou alterados. Note que isso é bastante perigoso, pois normalmente nenhuma mensagem será dada ao usuário. A regra será a automática modificação das informações no banco de dados.

No decorrer deste livro será utilizado um modelo de dados para criar e manipular os dados como exemplo. Esse modelo foi concebido com base no seguinte escopo:

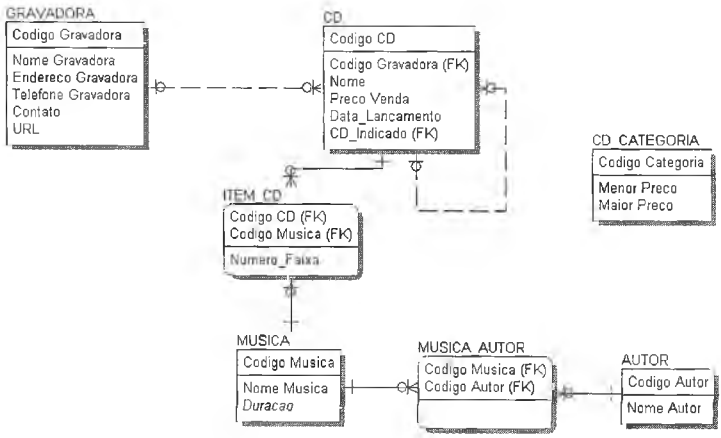
Objetivo: Criar um Catálogo de CDs

Nossa empresa necessita de um catálogo de CDs, pois constantemente recebemos ligações ou visitas à nossa loja. Os clientes costumam perguntar o nome, o preço de venda e a gravadora do CD. Cada CD contém diversas faixas em que ficam gravadas as músicas. Cada música tem seu tempo de duração e é importante sabermos os autores delas. Além disso, é importante conhecermos outras informações da gravadora, visto que, na falta de um CD, podemos localizar mais facilmente o local onde efetuar a compra. Gostaríamos que em cada CD houvesse uma indicação para outro. Classificamos os CDs com base na faixa de preços a que ele pertence.

Veja a seguir uma tabela com as informações do catálogo de CDs.

Cód CD	Nome CD	Grav.	Preço	Total Tp	Nº Faixa	Música	Autor	Tempo	Indic.
1	Mais do Mesmo	EMI	15,00		1	Será	Renato Russo e...	2:28	2
					2	Ainda é Cedo	Renato Russo e...	3:55	
					3	Geração Coca-Cola	Renato Russo	2:20	
					4	Eduardo e Monica	Renato Russo	4:32	
					5	Tempo Perdido	Renato Russo	5:00	
					6	Índios	Renato Russo	4:23	
					7	Que País é Este	Renato Russo	2:64	
					8	Faroeste Caboclo	Renato Russo	9:03	
					9	Há Tempos	Renato Russo e...	3:16	
					10	Pais e Filhos	Renato Russo e...	5:06	
					11	Meninos e Meninas	Renato Russo e...	3:22	
					12	Vento no Litoral	Renato Russo e...	6:05	
					13	Perfeição	Renato Russo e...	4:35	
					14	Giz	Renato Russo e...	3:20	
					15	Dezesseis	Renato Russo e...	5:28	
					16	Antes das Seis	Renato Russo e...	3:09	
2	Bate-Boca	PolyGram	12,00		1	Meninos, Eu Vi	Tom Jobim e...	3:25	1
					2	Eu Te Amo	Tom Jobim e...	3:06	
					3	Piano na Mangueira	Tom Jobim e...	2:23	
					4	A Violeira	Tom Jobim e...	2:54	
					5	Anos Dourados	Tom Jobim e...	2:56	
					6	Olha, Maria	Tom Jobim e...	3:55	
					7	Biscate	Chico Buarque	3:20	
					8	Retrato em Preto e	Tom Jobim e...	3:03	
					9	Falando de Amor	Tom Jobim	3:20	
					10	Pois E	Tom Jobim e...	2:48	
					11	Noite dos Mascarados	Chico Buarque	2:42	
					12	Sabiá	Tom Jobim e...	3:20	
					13	Imagina	Tom Jobim e...	2:52	
					14	Bate-Boca	Tom Jobim	4:41	

O Modelo de Entidade x Relacionamento a seguir representa o modelo do catálogo de CDs:



Veja que os relacionamentos estão exatamente de acordo com a proposta apresentada anteriormente. Devemos apenas criar as duas entidades que serão formadas entre CD e Música e Música e Autor. Na Entidade Item_CD será colocado o atributo que está no catálogo, mas que não está no modelo anterior. Esse atributo é o número da faixa da Música no CD. Observe que foi colocada a Faixa como Chave Primária, e não o Código da Música, porque, eventualmente, essa música pode estar em duas faixas de um mesmo CD (geralmente com arranjo diferente), mas jamais haverá duas faixas com o mesmo número. Isso é apenas uma convenção que foi adotada. Nada impede de ser implementada de forma diferente.

Pratique

Para identificar relacionamentos, tente seguir os passos a seguir:

1. Determine a existência de relacionamentos entre as Entidades.

2. Dê um nome a cada relacionamento identificado.

3. Determine a opcionalidade em cada direção do relacionamento.

4. Determine a cardinalidade (grau) em cada direção do relacionamento.

5. Leia e valide os relacionamentos identificados.

Até que você tenha prática na identificação de relacionamentos, pode ser útil utilizar uma matriz de relacionamento. Essa matriz nada mais é que a relação de todas as entidades em linha e coluna. Na intersecção das entidades, você deve identificar se há relacionamento. Se houver, coloque o verbo que caracteriza o relacionamento. Assim:

	CD	GRAVADORA	MÚSICA	AUTOR
CD	Indicado por	gravado por	contém	
GRAVADORA	grava			
MÚSICA	está em			é escrita
AUTOR			escreve	

Com essa nova divisão de entidades e relacionamentos do catálogo de CDs, teremos os conteúdos definidos conforme demonstrado a seguir:

Entidade: Autor

Cód Autor	Autor
1	Renato Russo
2	Tom Jobim
3	Chico Buarque
4	Dado Villa-Lobos
5	Marcelo Bonfá
6	Ico Ouro-Preto
7	Vinicius de Moraes

Entidade: Gravadora

Cód Gravadora	Nome Gravadora	Endereço	e-mail
1	EMI	Rod. Dutra km 229,8	www.emi-music.com.br
2	PolyGram		

Entidade: CD

Cód CD	Nome CD	Código Gravadora	Preço	Tempo Total	CD_Indicado
1	Mais do Mesmo	1	15,00	2	
2	Bate-Boca	2	12,00	1	

Entidade: Música

Cód	Música	Tempo
1	Será	2:28
2	Ainda é Cedo	3:55
3	Geração Coca-Cola	2:20
4	Eduardo e Mônica	4:32
5	Tempo Perdido	5:00
6	Índios	4:23
7	Que País é Este	2:64
8	Faroeste Caboclo	9:03
9	Há Tempos	3:16
10	Pais e Filhos	5:06
11	Meninos e Meninas	3:22
12	Vento no Litoral	6:05
13	Perfeição	4:35
14	Giz	3:20
15	Dezesseis	5:28
16	Antes das Seis	3:09
17	Meninos, Eu Vi	3:25
18	Eu Te Amo	3:06
19	Piano na Mangueira	2:23
20	A Violeira	2:54
21	Anos Dourados	2:56
22	Olha, Maria	3:55
23	Biscate	3:20
24	Retrato em Preto e	3:03
25	Falando de Amor	3:20
26	Pois É	2:48
27	Noite dos Mascarados	2:42
28	Sabiá	3:20
29	Imagina	2:52
30	Bate-Boca	4:41

Entidade: Item_CD

Cód CD	Nº Faixa	Cód Música
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6
1	7	7
1	8	8
1	9	9
1	10	10
1	11	11
1	12	12
1	13	13
1	14	14
1	15	15
1	16	16
2	1	17
2	2	18
2	3	19
2	4	20
2	5	21
2	6	22
2	7	23
2	8	24
2	9	25
2	10	26
2	11	27
2	12	28
2	13	29
2	14	30

Entidade: Musica_Autor

Cód Música	Cód Autor
1	1
1	4
1	5
2	1
2	4
2	5
2	6
3	1
4	1
5	1
6	1
7	1
8	1
9	1
9	4
9	5
10	1
10	4
10	5
11	1
11	4
11	5
12	1
12	4
12	5
13	1
13	4
13	5
14	1
14	4
14	5
15	1
15	4
15	5
16	1
16	4
16	5
17	2
17	3
18	2

Cód Música	Cód Autor (cont.)
18	3
19	2
19	3
20	2
20	3
21	2
21	3
22	2
22	3
22	7
23	3
24	2
24	3
25	2
26	2
26	3
27	3
28	2
28	3
29	2
29	3
30	3

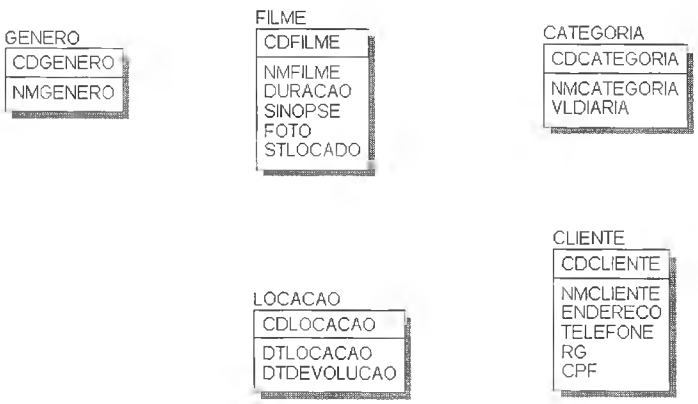
Se, por um lado, as informações ficaram mais dispersas e de certa forma até mais difíceis de serem localizadas (há muitos códigos relacionando uma informação com a outra), por outro lado a informação estará mais completa. No caso de autores de música, o catálogo apenas fornecia um dos autores. Nesse modelo, temos a oportunidade de especificar cada um dos autores de cada música.

Ao relacionarmos as entidades por meio de códigos, economizamos espaço e evitamos eventuais erros de digitação. Exemplo: alguém poderia escrever Tom Jobim e outra pessoa, T. Jobim. Como saberíamos todas as músicas de Tom Jobim de que dispomos na loja? Com o modelo anterior, basta pesquisar em MUSICA_AUTOR todos os autores com código 2, o qual corresponde a Tom Jobim.

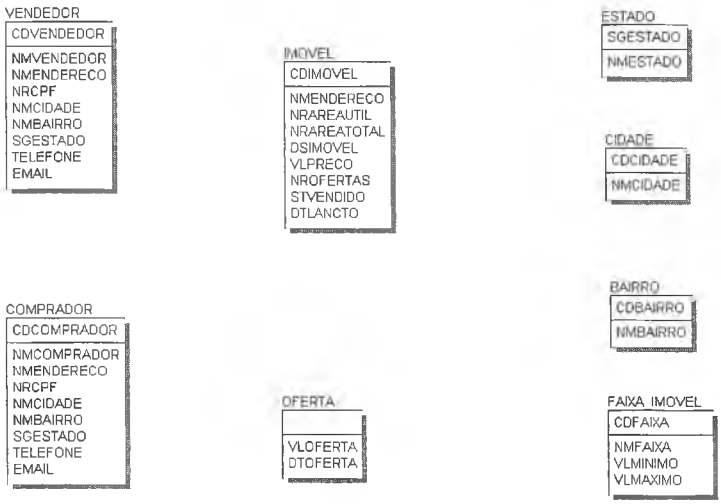
Esse modelo nos permitiria, por exemplo, adicionar informações (atributos) que hoje podem não ser relevantes, mas que amanhã poderão ser necessárias. Isso porque as entidades estão separadas por grupos de informação semelhantes (objetos), o que nos permite agregar novas informações aos objetos identificados.

Exercícios propostos

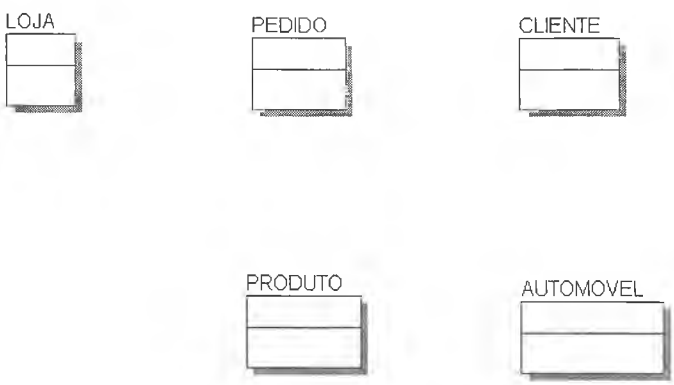
1. Veja os modelos de dados a seguir. Identifique os relacionamentos entre as entidades apresentadas. Leve em consideração que o Gênero é Drama, Comédia, Aventura etc. e Categoria é a faixa de preço do filme. Em um modelo mais completo, deveria haver várias fitas para um mesmo filme, mas imagine que, nesse sistema, não haja essa necessidade.



2. Complete os relacionamentos a seguir, levando em consideração que esse sistema é utilizado para cadastrar pessoas interessadas em vender e comprar imóveis. Portanto, imagine que há apenas um vendedor para cada imóvel, mas que vários compradores podem fazer oferta para o mesmo imóvel. Leve em consideração que o imóvel será posteriormente pesquisado por Estado, Cidade, Bairro e Faixa de Preço. Por esse motivo, não há necessidade de relacionar Estado, Cidade e Bairro com o Vendedor e Comprador. Acrescente um relacionamento para a indicação de outro Imóvel. Note que a Faixa do Imóvel representa a faixa de preço dos imóveis e que, portanto, não é um relacionamento que pode ser feito diretamente à tabela Imóvel.



3. Relacione as entidades a seguir e acrescente atributos que você julgue importantes para o sistema. Nesse caso, trata-se de uma empresa de instalação de acessórios para automóveis e, portanto, o cliente leva o seu automóvel para instalar produtos na loja.



4. Com base nos escopos a seguir, crie um Modelo de Entidade x Relacionamento para cada sistema:

   a. Sou gerente de uma empresa de treinamento que ministra vários cursos técnicos. Esses cursos são identificados por código, nome e tempo de duração. Montamos turmas com base nos cursos que oferecemos. As turmas têm dias fixos da semana, que identificamos com uma letra (S para segunda-quarta-sexta, T para terça-quinta e B para sábado), um horário específico para início e fim, e um preço. Um instrutor pode dar aulas para várias turmas e nós não trocamos os respectivos instrutores enquanto durar o curso de uma turma. É importante saber o nome, o endereço e o telefone de cada instrutor. Os alunos estão sempre vinculados a uma turma. Devemos saber o nome, o endereço e o telefone de cada aluno.

   b. Sou um gerente de Recursos Humanos. Preciso manter informações de cada funcionário da empresa. As informações de cada funcionário de que necessito são: o primeiro nome, o último nome, a função, a data de admissão e o salário. Caso o funcionário seja comissionado, preciso saber o valor médio da comissão. A empresa é dividida em departamentos. Cada funcionário é alocado em um departamento. Preciso saber o nome do departamento e a sua localização. Alguns funcionários são também gerentes, portanto preciso saber qual o gerente de cada funcionário. Note que o próprio gerente é também um funcionário.

   c. Possuo um site na Internet onde tentamos resolver questões relacionadas com informática. Para facilitar a localização das questões, segmentamos as dúvidas por plataforma e área de interesse. A partir daí, localizamos os eventos relacionados com essa plataforma (como Windows, Unix, Linux) e esse segmento (como pacotes prontos Office entre outros, sistema operacional, linguagens de programação). Com essas informações, podemos buscar os eventos relacionados à plataforma e ao segmento para mostrar ao usuário. Nos eventos armazenamos a data da ocorrência, a descrição do problema e da solução apresentada, além do usuário que levantou a dúvida. Outros usuários podem fazer comentários (um texto livre) sobre os eventos apresentados. Cadastramos todos os usuários com o nome, o endereço e o telefone. Cadastramos também os consultores que respondem às questões. Precisamos saber o nome, o endereço e o telefone dos consultores.

3

Normalização de dados

Conforme visto anteriormente, é muito melhor adotarmos uma estratégia em que se utilizem várias tabelas relacionadas em vez de uma única tabela. Basicamente, dois fatores nos levam a essa conclusão: redundância e espaço. Outros fatores também influenciam essa escolha, como a possibilidade de melhorarmos a qualidade da informação, seja fornecendo uma informação mais segura, seja pelo aumento da quantidade de dados no futuro.

A forma apresentada com o uso do Modelo de Entidade x Relacionamento é muito prática e usual, mas poderá deixar várias dúvidas quanto ao modelo adotado. Uma forma mais "científica" de realizar esse trabalho é utilizar a normalização de dados.

A normalização de dados é uma seqüência de etapas sucessivas que, ao final, apresentará um modelo de dados estável com um mínimo de redundância.

Quanto à utilização de um ou outro modelo, não há escolha preferencial. Afinal, o modelo de dados pode ser extraído de uma ou outra forma. O resultado final deverá ser igual para ambas as formas. Quando se utilizam os dois, é muito mais difícil esquecer alguma informação importante.

Devemos observar que quando desenvolvemos um sistema, dificilmente temos todas as informações estruturadas da forma como precisamos. Sempre que achamos que sabemos o que fazer no sistema, seja por experiência anterior ou por arrogância, estamos destinando nosso sistema a um modelo muitas vezes pobre, incompleto e inadequado. Por isso, é importante utilizar as duas formas para modelarmos os dados. Um modelo deve corresponder ao outro. Se não corresponder, é uma boa oportunidade de voltar e conversar com o(s) usuário(s) e entender melhor o funcionamento da estrutura do sistema.

Há cinco regras que se aplicam a bancos de dados:

a. Primeira Forma Normal (1FN).

b. Segunda Forma Normal (2FN).

c. Terceira Forma Normal (3FN).

d. Quarta Forma Normal (4FN).

e. Quinta Forma Normal (5FN).

Ao aplicarmos as regras anteriores já é possível atingirmos o objetivo ao qual se propõe a normalização de dados. Um modelo estável pode ser garantido quando atingimos a terceira forma normal. As demais formas serão utilizadas em casos específicos, desde que sejam absolutamente necessárias.

O processo de trabalhar informações para criar estruturas de entidades em formas normais é chamado de normalização de dados.

Antes de iniciarmos a análise de cada uma das etapas de normalização, verificaremos a terminologia utilizada, que envolve basicamente a teoria de conjuntos (que na realidade foi a base para a construção da teoria da normalização), o conceito de banco de dados relacional e a aplicação convencional em ambiente de processamento de dados. Observe na tabela a seguir o que um mesmo termo significa em cada uma das visões:

Conjunto	Banco de Dados	Processamento de Dados
Relação	Tabela	Arquivo
Domínio	Possíveis valores para coluna	Possíveis valores para o campo
Tupla	Linha	Registro

Como preparação para a normalização, devemos ter a relação de todos os atributos necessários e uma chave primária para a entidade.

Para estudarmos as formas normais, recorreremos ao Catálogo de CDs e utilizaremos a chave Cód. CD como a chave primária do catálogo.

Cód CD	Nome CD	Grav.	Preço	Total Tp	Nº Faixa	Música	Autor	Tempo	Indic.
1	Mais do Mesmo	EMI	15,00		1	Será	Renato Russo e...	2:28	2
					2	Ainda é Cedo	Renato Russo e...	3:55	
					3	Geração Coca-Cola	Renato Russo	2:20	
					4	Eduardo e Monica	Renato Russo	4:32	
					5	Tempo Perdido	Renato Russo	5:00	
					6	Índios	Renato Russo	4:23	
					7	Que País é Este	Renato Russo	2:64	
					8	Faroeste Caboclo	Renato Russo	9:03	
					9	Há Tempos	Renato Russo e...	3:16	
					10	Pais e Filhos	Renato Russo e...	5:06	
					11	Meninos e Meninas	Renato Russo e...	3:22	
					12	Vento no Litoral	Renato Russo e...	6:05	
					13	Perfeição	Renato Russo e...	4:35	
					14	Giz	Renato Russo e...	3:20	
					15	Dezesseis	Renato Russo e...	5:28	
					16	Antes das Seis	Renato Russo e...	3:09	
2	Bate-Boca	PolyGram	12,00		1	Meninos, Eu Vi	Tom Jobim e...	3:25	1
					2	Eu Te Amo	Tom Jobim e...	3:06	
					3	Piano na Mangueira	Tom Jobim e...	2:23	
					4	A Violeira	Tom Jobim e...	2:54	
					5	Anos Dourados	Tom Jobim e...	2:56	
					6	Olha, Maria	Tom Jobim e...	3:55	
					7	Biscate	Chico Buarque	3:20	
					8	Retrato em Preto e	Tom Jobim e...	3:03	
					9	Falando de Amor	Tom Jobim	3:20	
					10	Pois É	Tom Jobim e...	2:48	
					11	Noite dos Mascarados	Chico Buarque	2:42	
					12	Sabiá	Tom Jobim e...	3:20	
					13	Imagina	Tom Jobim e...	2:52	
					14	Bate-Boca	Tom Jobim	4:41	

Primeira Forma Normal (1FN)

Definição

Diz-se que uma entidade está na primeira forma normal quando nenhum de seus atributos (na estrutura) possuir repetições.

Isso significa que os atributos são indivisíveis ou que possuem apenas um valor por célula. Deve-se, portanto, validar cada um dos atributos de forma a identificar se este possui um único valor para cada ocorrência da entidade. Veja que o valor (conteúdo) do atributo pode, e normalmente é, ser diferente em cada ocorrência da entidade (registro). O que se deve evitar é a ocorrência de mais de um atributo com a mesma característica em um mesmo registro. Por exemplo, há várias ocorrências do atributo Produto em uma Nota Fiscal.

Não é possível determinar se será comprado um ou mais produtos em uma única Nota Fiscal. Portanto, Produto é um atributo repetitivo na entidade Nota Fiscal e deve ser aplicada a primeira regra de normalização.

Como visto no exemplo anterior, há diversos atributos que se repetem. São eles: Número da Faixa, Música, Autor e Tempo. Veja que são atributos que apresentam repetição na estrutura. Estamos identificando que para cada CD existe uma série de Faixa, Músicas, Autor e Tempo. Seria como se quiséssemos colocar todas as músicas de um CD na mesma coluna (músicas). Isso não seria viável, pois há mais de uma música no mesmo CD. Ora, isso quer dizer que esse atributo não é indivisível, visto que é possível ter mais de um valor por célula.

Solução

Nesse caso, devemos separar a informação que se repete em uma nova entidade. Devemos ainda levar a chave primária da entidade original para a nova entidade gerada (caso contrário, não haveria como relacionar as informações das duas entidades). Feito isso, criaremos a nova chave para a nova entidade. Normalmente podemos localizar um campo que, unido à chave da entidade original, formará a chave da nova tabela (nesse caso, chave concatenada), ou podemos criar um campo para esse fim, caso não exista. Há a possibilidade de criarmos simplesmente uma nova chave não concatenada e independente da entidade original. É uma questão de preferência.

Exemplo

A partir desse instante teríamos duas entidades, da seguinte forma:

Entidade: CD

Cód CD	Nome CD	Nome Gravadora	Preço	Total Tempo	CD Indicado
1	Mais do Mesmo	EMI	15,00		2
2	Bate-Boca	PolyGram	12,00		1

Entidade: Item_CD

Cód CD	Nº Faixa	Música	Autor	Tempo
1	1	Será	Renato Russo e...	2:28
1	2	Ainda é Cedo	Renato Russo e...	3:55
1	3	Geração Coca-Cola	Renato Russo	2:20
1	4	Eduardo e Monica	Renato Russo	4:32
1	5	Tempo Perdido	Renato Russo	5:00
1	6	Índios	Renato Russo	4:23
1	7	Que País é Este	Renato Russo	2:64
1	8	Faroeste Caboclo	Renato Russo	9:03
1	9	Há Tempos	Renato Russo e...	3:16
1	10	Pais e Filhos	Renato Russo e...	5:06
1	11	Meninos e Meninas	Renato Russo e...	3:22
1	12	Vento no Litoral	Renato Russo e...	6:05
1	13	Perfeição	Renato Russo e...	4:35
1	14	Giz	Renato Russo e...	3:20
1	15	Dezesseis	Renato Russo e...	5:28
1	16	Antes das Seis	Renato Russo e...	3:09
2	1	Meninos, Eu Vi	Tom Jobim e...	3:25
2	2	Eu Te Amo	Tom Jobim e...	3:06
2	3	Piano na Mangueira	Tom Jobim e...	2:23
2	4	A Violeira	Tom Jobim e...	2:54
2	5	Anos Dourados	Tom Jobim e...	2:56
2	6	Olha, Maria	Tom Jobim e...	3:55
2	7	Biscate	Chico Buarque	3:20
2	8	Retrato em Preto e	Tom Jobim e...	3:03
2	9	Falando de Amor	Tom Jobim	3:20
2	10	Pois É	Tom Jobim e...	2:48
2	11	Noite dos Mascarados	Chico Buarque	2:42
2	12	Sabiá	Tom Jobim e...	3:20
2	13	Imagina	Tom Jobim e...	2:52
2	14	Bate-Boca	Tom Jobim	4:41

Utiliza-se o campo Número da Faixa como atributo-chave junto com o Código do CD, pois não poderá haver duas faixas com o mesmo número em um único CD. Essa é, portanto, uma chave concatenada.

Segunda Forma Normal (2FN)

Definição

Diz-se que uma entidade está na segunda forma normal quando todos os seus atributos não chave dependem unicamente da chave.

Assim, deve-se perguntar a cada atributo, que não seja a chave da entidade, se ele depende apenas da chave da entidade. O que se procura com isso é medir o grau de dependência entre os atributos. Apenas coisas semelhantes podem ser unidas em grupos semelhantes (veja que entidade nada mais é do que um grupo ou conjunto).

Solução

Ao identificarmos uma situação como a descrita anteriormente, devemos separar os atributos independentes e criar ou identificar dentre os atributos separados uma nova chave para esta nova entidade. Essa chave deve ser mantida na entidade original como o atributo de relacionamento entre ambas as entidades. Dessa forma, não se perde qualquer informação no modelo.

Exemplo

No exemplo anterior, a Gravadora, o Autor e a Música são independentes de suas entidades, CD e Item_CD. Veja que há uma enorme vantagem ao separarmos esses atributos independentes, visto que, se não o fizessemos, cada alteração em uma das informações deveria ser estendida a todas as linhas da entidade. É muito mais fácil fazer isso apenas na nova entidade criada. Como a relação se dá por meio do campo-chave, não há necessidade de alterar todas as ocorrências em CD ou Item_CD, apenas nas entidades Autor, Gravadora ou Música. A inclusão de novos atributos também é facilitada. Atributos que venham a ser relevantes serão acrescentados diretamente na entidade Autor (como Data de Nascimento) ou Gravadora (como Endereço e Home Page).

Dessa forma, teremos mudanças nas entidades CD e Item_CD e criaremos as entidades Autor e Música.

Entidade: Autor

Cód Autor	Autor
1	Renato Russo
2	Tom Jobim
3	Chico Buarque
4	Dado Villa-Lobos
5	Marcelo Bonfá
6	Ico Ouro-Preto
7	Vinicius de Moraes

Entidade: Gravadora

Cód Gravadora	Nome Gravadora	Endereço	e-mail
1	EMI	Rod. Dutra km 229,8	www.emi-music.com.br
2	PolyGram		

Entidade: CD

Cód CD	Nome CD	Cód Gravadora	Preço	Total Tempo	CD Indicado
1	Mais do Mesmo	1	15,00		2
2	Bate-Boca	2	12,00		1

Entidade: Música

Cód	Música	Tempo	Autor
1	Será	2:28	Renato Russo e...
2	Ainda é Cedo	3:55	Renato Russo e...
3	Geração Coca-Cola	2:20	Renato Russo
4	Eduardo e Monica	4:32	Renato Russo
5	Tempo Perdido	5:00	Renato Russo
6	Índios	4:23	Renato Russo
7	Que País é Este	2:64	Renato Russo
8	Faroeste Caboclo	9:03	Renato Russo
9	Há Tempos	3:16	Renato Russo e...
10	Pais e Filhos	5:06	Renato Russo e...
11	Meninos e Meninas	3:22	Renato Russo e...
12	Vento no Litoral	6:05	Renato Russo e...
13	Perfeição	4:35	Renato Russo e...
14	Giz	3:20	Renato Russo e...
15	Dezesseis	5:28	Renato Russo e...
16	Antes das Seis	3:09	Renato Russo e...
17	Meninos, Eu Vi	3:25	Tom Jobim e...
18	Eu Te Amo	3:06	Tom Jobim e...
19	Piano na Mangueira	2:23	Tom Jobim e...
20	A Violeira	2:54	Tom Jobim e...
21	Anos Dourados	2:56	Tom Jobim e...
22	Olha, Maria	3:55	Tom Jobim e...
23	Biscate	3:20	Chico Buarque
24	Retrato em Preto e	3:03	Tom Jobim e...
25	Falando de Amor	3:20	Tom Jobim
26	Pois É	2:48	Tom Jobim e...
27	Noite dos Mascarados	2:42	Chico Buarque
28	Sabiá	3:20	Tom Jobim e...
29	Imagina	2:52	Tom Jobim e...
30	Bate-Boca	4:41	Tom Jobim

Entidade: Item_CD

Cód CD	Nº Faixa	Cód Música
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6
1	7	7
1	8	8
1	9	9
1	10	10
1	11	11
1	12	12
1	13	13
1	14	14
1	15	15
1	16	16
2	1	17
2	2	18
2	3	19
2	4	20
2	5	21
2	6	22
2	7	23
2	8	24
2	9	25
2	10	26
2	11	27
2	12	28
2	13	29
2	14	30

Terceira Forma Normal (3FN)

Definição

Diz-se que uma entidade está na terceira forma normal quando todos os seus atributos não chave não dependem de nenhum outro atributo não chave.

Utilizando palavras mais simples, pode-se dizer que um atributo não deve depender de outro atributo. Isso ocorre normalmente em cálculos matemáticos ou em atributos "perdidos" na entidade errada. Assim o Valor Total de uma Nota Fiscal ou Pedido dependerá do somatório do valor dos itens da Nota ou do Pedido. Por sua vez, o Valor Total do Item de uma Nota Fiscal ou o Pedido será a multiplicação do Valor Unitário do produto pela Quantidade.

Ora, não há necessidade de manter esses valores gravados, uma vez que eles são resultado de uma operação entre outros dois atributos. Se guardamos esse valor, estaremos apenas ocupando espaço e abrindo a possibilidade de termos uma informação inconsistente no banco de dados. Veja: digamos que alguém altere a Quantidade ou o Valor Unitário, ou o Valor Total do item de uma Nota Fiscal. Como saberemos qual informação é correta?

Mas isso não ocorre apenas em cálculos matemáticos. Analisemos o Local de um Departamento. Um funcionário está alocado em um Departamento. O Departamento, por sua vez, está em um determinado local. Não há, portanto, a necessidade de mantermos o Departamento e o local na tabela Funcionário. Podemos migrá-lo para a tabela Departamento. Contudo, esse tipo de análise requer treinamento e conhecimento das operações da empresa. Deve-se ter certeza de que isso não será modificado, pois, caso contrário, será necessário reconstruir toda a tabela, incorrendo em tempo de análise, processamento e redesenvolvimento de módulos. Caso haja dúvidas, deve-se optar pelo modelo que forneça maior possibilidade de adaptação, mesmo que haja eventuais redundâncias nas informações.

Em algumas situações, pode-se localizar elementos que são inter-relacionados. São atributos que dependem uns dos outros, formando uma entidade completamente nova. Se essa for a situação, deve-se criar uma nova entidade como fizemos na Segunda Forma Normal.

Solução

Ao depararmos com essa situação, devemos realizar a seguinte análise: se o atributo for resultado de um cálculo matemático, devemos simplesmente excluir esse atributo, uma vez que ele não acrescenta nada ao modelo de dados. Se for um grupo de informações relacionadas, devemos aplicar a Segunda Forma Normal. Se for um atributo "perdido", devemos reconduzi-lo à Entidade da qual depende.

Exemplo

No nosso exemplo, o atributo Tempo Total é dependente de outro atributo não chave: Tempo, na entidade música. Ao somarmos o tempo de cada música de um CD, teremos o tempo total de gravação do CD. Esse campo deve ser eliminado, sem perda de informação para o modelo de dados.

Entidade: CD

Cód CD	Nome CD	Nome Gravadora	Preço	CD Indicado
1	Mais do Mesmo	EMI	15,00	2
2	Bate-Boca	PolyGram	12,00	1

As demais entidades permanecem sem alterações.

Quarta Forma Normal (4FN)

Definição

Pode ocorrer de estarmos com um modelo na Terceira Forma Normal e mesmo assim haver alguma redundância. Isso ocorrerá quando um atributo não chave contiver valores múltiplos para uma mesma chave.

Também chamada de dependência multivalorada, tem como característica a repetição de dois ou mais atributos não chave, gerando uma redundância desnecessária no modelo.

No nosso exemplo não é possível identificar esse tipo de dependência, mas se imaginarmos que um intérprete pode cantar várias músicas e publicá-las em diferentes gravadoras, teríamos um caso prático de aplicação da Quarta Forma Normal. Veja:

Música	Intérprete	Gravadora
Será	Renato Russo	EMI
Será	Simone	PolyGram
Será	Renato Russo	PolyGram
Imagine	John Lenon	EMI
Imagine	Simone	EMI
Imagine	John Lenon	PolyGram

Note que não podemos criar uma chave com música (ela se repete), nem com música + intérprete (também há repetição), e caso coloquemos os três campos como chave, teremos uma alternativa válida, mas redundante, pois haverá repetição de música e autor ou música e gravadora.

Obs.: O exemplo anterior é meramente didático. Normalmente, os intérpretes possuem contratos de exclusividade com a gravadora.

Solução

É necessário dividir essa entidade em duas. Cada uma das entidades herdará a Música como chave e conterá o outro atributo. A primeira entidade ficaria assim:

Música	Intérprete
Será	Renato Russo
Será	Simone
Será	Renato Russo
Imagine	John Lenon
Imagine	Simone
Imagine	John Lenon

e a outra assim:

Música	Gravadora
Será	EMI
Será	PolyGram
Será	PolyGram
Imagine	EMI
Imagine	EMI
Imagine	PolyGram

Naturalmente, os nomes serão devidamente substituídos pelos respectivos códigos de música, intérprete e gravadora.

Quinta Forma Normal (5FN)

Definição

Caso raro de ocorrer. Tecnicamente, utiliza-se a Quinta Forma Normal quando uma tabela na Quarta Forma Normal pode ser subdividida em duas ou mais tabelas, para evitar eventuais redundâncias ainda existentes.

É como se tivéssemos, na aplicação da Quarta Forma Normal, deixado três campos em uma das tabelas criadas e esses campos tivessem novamente valores multivalorados.

Solução

Para solucionar essa redundância, basta dividir novamente a tabela em duas outras, carregando o campo-chave e herdando o outro atributo.

Finalização do Processo de Normalização

Após concluir o processo de normalização, devemos checar se em alguma entidade podem ser aplicadas novamente as mesmas regras. Como podemos notar, a entidade Música contém a repetição de vários autores. E como, por meio da Primeira Forma Normal, sabemos que um atributo não deve conter valores repetitivos, devemos aplicar novamente a Primeira Forma Normal na tabela Música.

Note que esse processo deve ser repetido tantas vezes quantas forem necessárias, até que não haja nenhum outro atributo ou grupo de atributos que estejam repetidos.

Agora, o nosso modelo já adaptado à nova aplicação da 1FN na entidade música ficará assim:

Entidade: Autor

Cód Autor	Autor
1	Renato Russo
2	Tom Jobim
8	Chico Buarque
9	Dado Villa-Lobos
10	Marcelo Bonfá
11	Ico Ouro-Preto
12	Vinicius de Moraes

Entidade: Gravadora

Cód Gravadora	Nome Gravadora	Endereço	e-mail
1	EMI	Rod. Dutra km 229,8	www.emi-music.com.br
2	PolyGram		

Entidade: CD

Cód CD	Nome CD	Código Gravadora	Preço	Tempo Total	CD Indicado
1	Mais do Mesmo	1	15,00		2
2	Bate-Boca	2	12,00		1

Entidade: Música

Cód	Música	Tempo
1	Será	2:28
2	Ainda é Cedo	3:55
3	Geração Coca-Cola	2:20
4	Eduardo e Monica	4:32
5	Tempo Perdido	5:00
6	Índios	4:23
7	Que País é Este	2:64
8	Faroeste Caboclo	9:03
9	Há Tempos	3:16
10	Pais e Filhos	5:06
11	Meninos e Meninas	3:22
12	Vento no Litoral	6:05
13	Perfeição	4:35
14	Giz	3:20
15	Dezesseis	5:28
16	Antes das Seis	3:09
17	Meninos, Eu Vi	3:25
18	Eu Te Amo	3:06
19	Piano na Mangueira	2:23
20	A Violeira	2:54
21	Anos Dourados	2:56
22	Olha, Maria	3:55
23	Biscate	3:20
24	Retrato em Preto e	3:03
25	Falando de Amor	3:20
26	Pois É	2:48
27	Noite dos Mascarados	2:42
28	Sabiá	3:20
29	Imagina	2:52
30	Bate-Boca	4:41

Entidade: Item_CD

Cód CD	Nº Faixa	Cód Música
1	1	1
1	2	2
1	3	3
1	4	4
1	5	5
1	6	6
1	7	7
1	8	8
1	9	9
1	10	10
1	11	11
1	12	12
1	13	13
1	14	14
1	15	15
1	16	16
2	1	17
2	2	18
2	3	19
2	4	20
2	5	21
2	6	22
2	7	23
2	8	24
2	9	25
2	10	26
2	11	27
2	12	28
2	13	29
2	14	30

Entidade: Musica_Autor

Cód Música	Cód Autor
2	1
3	4
3	5
4	1
4	4
2	5
2	6
3	1
4	1
5	1
6	1
7	1
8	1
9	1
9	4
9	5
10	1
10	4
10	5
11	1
11	4
11	5
12	1
12	4
12	5
13	1
13	4
13	5
14	1
14	4
14	5
15	1
15	4
15	5
16	1
16	4
16	5
17	2
17	3

Cód Música	Cód Autor (cont.)
18	2
18	3
19	2
19	3
20	2
20	3
22	2
21	3
22	2
22	3
22	7
23	3
24	2
24	3
25	2
26	2
26	3
27	3
28	2
28	3
29	2
29	3
30	3

Note que chegamos ao mesmo modelo de dados que havíamos chegado utilizando o Modelo de Entidade x Relacionamento. Essa é a situação ideal. Os dois modelos devem ser exatamente iguais no final do processo. Se não forem, retorne ao(s) usuário(s) do sistema e veja qual dos dois está correto. Para efeito de documentação, mantenha os dois com a versão final. Não esqueça, hoje você lembra exatamente o que fez, amanhã não lembrará.

Pratique

Para normalizar os dados, siga os passos a seguir:

1. Crie ou identifique uma chave para a entidade que você está analisando vamos chamá-la de E1.

2. Verifique se há ocorrências de atributos repetitivos. Caso haja, crie uma nova entidade e traga para esta a chave da entidade E1 (1FN).

3. Verifique se todos os atributos não chave dependem unicamente da chave. Se não dependerem, crie uma nova entidade para cada grupo interdependente e deixe a chave da nova entidade na entidade E1 (2FN).

4. Verifique se todos os atributos não chave não dependem de outros atributos não chave. Caso exista essa dependência e seja um grupo, separe-o como no item anterior. Se for resultado de operação matemática, elimine o atributo (3FN).

5. Verifique se há alguma dependência multivalorada e independente, ou seja, dois ou mais atributos não chave com conteúdo repetido em uma mesma entidade. Se houver, separe esses atributos multivalorados em outras tantas entidades quantos forem os atributos multivalor (4 e 5FN), mantendo um dos atributos para relacionamento.

6. Valide esse modelo gerado mediante a normalização com o modelo de Entidade x Relacionamento, para sanar eventuais diferenças entre ambos.

Exercícios propostos

Reveja os modelos de dados criados no Exercício do Capítulo 2 e verifique se há necessidade de alterar qualquer um dos modelos. Justifique sua resposta.

4

Criando um banco de dados

Para iniciar o trabalho de criação do banco de dados, deve-se inicialmente transformar o modelo lógico, representado pelo Modelo de Entidade x Relacionamento e pela Normalização de Dados, no modelo físico que será implementado. Essa parte do livro trata da *Data Definition Language* (DDL).

O processo básico consiste em simplesmente atribuir tipos de dado e tamanho para cada um dos atributos que foram identificados.

Observações:

1. Cada banco de dados utiliza um terminador de comando específico. Normalmente é utilizado o ponto e vírgula (;) no Oracle e DB2, mas encontra-se ponto (.) no Progress e o comando go no SQL Server. Você deverá encerrar todos os comandos mostrados daqui para frente com o padrão do seu banco de dados, para que eles sejam executados corretamente. Alguns bancos de dados permitem inclusive trocar o terminador-padrão.

2. Quanto aos comandos SQL apresentados, os bancos de dados não são sensíveis a letras maiúsculas ou minúsculas. Contudo, o conteúdo das colunas normalmente é sensível, portanto deve-se tomar cuidado ao realizar comparações entre valores alfanuméricos.

Desnormalização de dados

Em uma análise mais profunda, é muitas vezes conveniente avaliar a necessidade de alguns campos redundantes na tabela.

Esse processo é chamado de desnormalização de dados. Veja que não jogaremos fora todo o trabalho feito anteriormente. Ocorre que, quando idealizamos o modelo de dados, utilizamos o conceito de processador perfeito, em que as informações são transmitidas sem custo e instantaneamente. Ora, isso não é verdade. Mesmo os bancos de dados mais avançados possuem limitações, têm custo e não fornecem informações instantaneamente.

Imagine uma grande empresa que emita cerca de 20 mil notas fiscais por dia. No final de uma semana serão 140 mil. No final do mês, 600 mil. Veja que totalizar o valor de faturamento mensal já daria um enorme trabalho. Agora, ao aplicar as regras de normalização, elimina-se o campo Valor Total da Nota (3FN, atributo que depende de outro atributo), pois esse valor é igual ao somatório do valor dos itens da Nota Fiscal. Imaginando que cada Nota Fiscal tenha 5 itens em média, chega-se ao valor de 3 milhões de registros, consultados mensalmente para saber o valor faturado...

Em um caso como esse, por questões de *performance*, costuma-se "driblar" um pouco a regra e coloca-se o valor total da nota como um campo na tabela Nota Fiscal. Isso somente pode ser feito quando o banco de dados possui mecanismos de controle de transações, em que uma eventual gravação no banco de dados (no item da Nota Fiscal) implique necessariamente a gravação na outra ponta (na Nota Fiscal). Assim, todas as transações são gravadas ou nenhuma o será. Esse tipo de atualização deve ser feita por meio de Gatilhos (Triggers), para garantir a integridade das informações no banco de dados veja o capítulo 15 deste livro. Quando isso não for possível, aconselha-se a desistir da velocidade e manter a qualidade da informação.

Com o passar do tempo será mais fácil identificar quando é necessário desnormalizar os dados. No nosso modelo de dados não modificaremos nenhuma tabela.

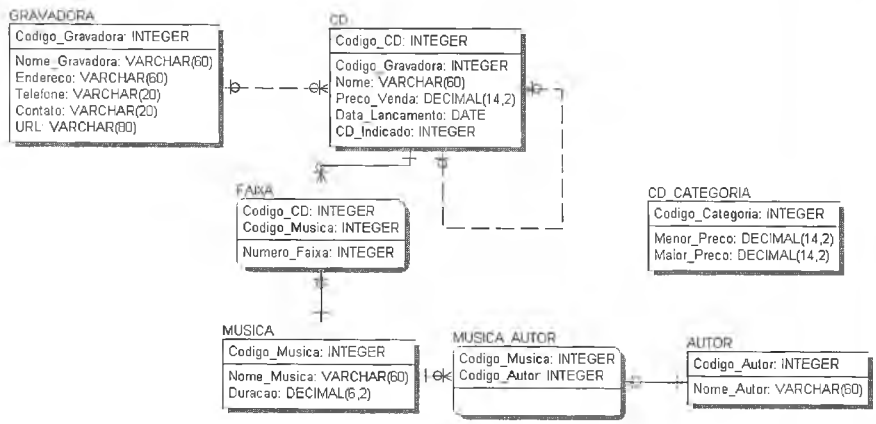
Definição de dados

Antes de criar as tabelas no nosso banco de dados, temos que definir quais são as características de cada um dos campos. As características que o SQL exige são o tipo de dado e o tamanho de cada campo. Apresentaremos o padrão utilizado pela maioria dos bancos de dados SQL e você deverá consultar as eventuais mudanças do banco de dados em que estiver trabalhando.

Tipo de Dado	Descrição
INTEGER ou INT	Número positivo ou negativo inteiro. O número de bytes que pode ser utilizado varia em função do banco de dados utilizado.
SMALLINT	Mesma função do INTEGER, mas ocupa cerca da metade do espaço.
NUMERIC	Número positivo ou negativo de ponto flutuante. Normalmente, deve-se informar o tamanho total do campo e definir quantas casas decimais devem ser armazenadas após a vírgula.
DECIMAL	Semelhante ao NUMERIC, mas, em alguns bancos de dados, poderá ter uma maior precisão após a vírgula.
REAL	Número de ponto flutuante de simples precisão. A diferença básica é que os valores serão armazenados em representação exponencial, portanto serão arredondados para o nível mais próximo de precisão.
DOUBLE PRECISION	Número de ponto flutuante de dupla precisão. Comporta-se como o REAL, mas permite maior aproximação de resultados.
FLOAT	Número de ponto flutuante em que você define o nível de precisão (número de dígitos significativos).
BIT	Armazenamento de um número fixo de *bits*. O número de *bits* deve ser indicado, do contrário o padrão será 1.
BIT VARYING	Igual ao BIT, permitindo armazenar valores maiores. Normalmente, utiliza-se para armazenamento de imagens.
DATE	Permite armazenar datas.
TIME	Permite armazenar horários.
TIMESTAMP	Permite armazenar uma combinação de data e hora.
CHARACTER ou CHAR	Permite armazenar cadeias de caracteres (letras, símbolos e números). O tamanho deve ser informado e será fixo, ou seja, mesmo que não utilizado totalmente, será ocupado o espaço fisicamente. O valor definido será o tamanho máximo da cadeia armazenada.
CHARACTER VARYING ou VARCHAR	Permite armazenar cadeias de caracteres, mas com tamanho variável. Nesse caso, especifica-se o tamanho máximo da coluna. Se for utilizado menos espaço que o máximo definido, o espaço restante não será ocupado.
INTERVAL	Intervalo de data ou hora.

Adaptando nosso modelo de dados

Baseando-nos nesses tipos de dados apresentados, adaptaremos nosso modelo de dados. Basicamente será definido para cada campo o tipo de dado e seu tamanho máximo. A seguir, visualize o modelo de dados já adaptado ao modelo Físico, com definição de tipo e tamanho de cada campo das tabelas. Para efeitos didáticos, acrescentamos uma tabela ao modelo (CD_CATEGORIA).



Criação de tabelas

Tabelas são as estruturas mais importantes de um banco de dados. Nas tabelas estará o conteúdo que representa cada objeto do mundo real, cuja importância para o funcionamento do sistema justifica a sua criação. As próprias tabelas criadas no banco de dados ficam armazenadas em tabelas internas do gerenciador de banco de dados. É o chamado Dicionário de Dados. Com a utilização do dicionário de dados, as informações das tabelas ficam sempre à disposição do usuário e permitem um controle maior do que está sendo armazenado.

O padrão SQL divide em três categorias as tabelas de um banco de dados:

a. Tabelas permanentes: ficam permanentemente armazenadas no banco de dados, a menos que sejam explicitamente excluídas.

b. Tabelas temporárias globais: utilizadas para armazenamento temporário de informações durante algum processamento específico, são eliminadas assim que for encerrado o acesso ao banco de dados. Possuem visibilidade em todo o banco de dados.

c. Tabelas temporárias locais: semelhantes às globais, são visíveis apenas em um módulo específico em que foram criadas.

Note que as tabelas temporárias possuem um conceito diferente das Visões de banco de dados (VIEWs). Mais a frente, discutiremos esse conceito.

As tabelas que criaremos para armazenar as informações do nosso sistema serão tabelas permanentes. Para criá-las, utilizaremos o comando CREATE TABLE.

Integridade Referencial - Constraints

Constraints são regras agregadas a colunas ou tabelas. Assim, pode-se definir como obrigatório o preenchimento de uma coluna que tenha um valor-padrão quando uma linha for incluída na tabela ou quando aceitar apenas alguns valores predefinidos. No caso de regras aplicadas a tabelas, tem-se a definição de chaves primárias e estrangeiras. Veremos adiante como colocar isso em prática.

CREATE TABLE

A sintaxe básica do comando é a seguinte:

```
CREATE TABLE nome-tabela
   ( nome-coluna1 tipo-de-dado-coluna1 coluna1-constraint,
   nome-coluna2 tipo-de-dado-coluna2 coluna2-constraint,
   nome-colunan tipo-de-dado-colunan colunan-constraint,
   constraint-de-tabela )
```

onde:

Argumento	Descrição
nome-tabela	Nome da tabela. Deve ser único para o usuário. Não pode coincidir com o nome de outros objetos do banco de dados de um mesmo usuário.
nome-coluna	Nome da coluna. Esse nome deve ser único e exclusivo na tabela.
tipo-de-dado-coluna	Tipo de dado conforme tabela anterior.
coluna-constraint	Regras agregadas à coluna.
constraint-de-tabela	Regras agregadas à tabela inteira.

Exemplo:

```
CREATE TABLE CD (
    Codigo_CD           INTEGER NOT NULL,
    Codigo_Gravadora    INTEGER NULL,
    Nome_CD             VARCHAR(60) NULL,
    Preco_Venda         DECIMAL(16,2) NULL,
    Data_Lancamento     DATE DEFAULT SYSDATE,
    CD_Indicado         INTEGER NULL,
    PRIMARY KEY (Codigo_CD),
    FOREIGN KEY (Codigo_Gravadora)
    REFERENCES Gravadora (Codigo_Gravadora),
    CHECK (Preco_Venda > 0)
    );
```

Tipos de Constraint mais comuns

Serão analisados alguns tipos de constraint encontrados com mais freqüência nos bancos de dados. As constraints variam muito de um banco de dados para outro. Pode ser que alguma delas não esteja presente no banco de dados que você estiver utilizando.

Chave primária

Conforme discutimos anteriormente, chave primária é a coluna, ou grupo de colunas, que permite identificar um único registro na tabela. Para especificar que uma coluna ou grupo de colunas representa a chave primária de uma tabela, deve-se acrescentar a palavra-chave PRIMARY KEY seguida do nome da(s) coluna(s). Alguns bancos de dados permitem especificar a chave primária como constraint da coluna (quando a chave primária for apenas uma coluna) ou como constraint de tabela. Outros aceitam apenas como constraint de tabela. Para facilitar, aconselha-se utilizar a chave primária como constraint de tabela.

Digamos que haja uma tabela de cliente cuja chave primária seja o campo CDCLIENTE. A criação da chave primária ficaria assim:

```
...

PRIMARY KEY (CDCLIENTE),

...
```

Chave estrangeira

Chave estrangeira é o campo que estabelece o relacionamento entre duas tabelas. Assim, uma coluna, ou grupo de colunas, de uma tabela corresponde à mesma coluna, ou grupo de colunas, que é a chave primária de outra tabela. Dessa forma, deve-se especificar na tabela que contém a chave estrangeira quais são essas colunas e à qual tabela está relacionada. Veja que o banco de dados irá verificar se todos os campos que fazem referência à tabela estão especificados.

Ao determinar esse tipo de relacionamento, fica garantida a integridade das informações. Os valores presentes na(s) coluna(s) definida(s) como chave estrangeira devem ter um correspondente na outra tabela, caso contrário o banco de dados deve retornar uma mensagem de erro. É normal que, ao tentar excluir um registro de uma tabela, o banco de dados verifique se há chaves estrangeiras relacionadas a esse registro. Se houver, também será retornado um erro impedindo essa operação.

Infelizmente, muitos programadores não utilizam esses recursos dos bancos de dados. Se não forem definidas as chaves estrangeiras nas tabelas, apenas o programa controlará o que pode ou não ser excluído ou alterado, colocando em risco informações relevantes do sistema. Por exemplo, pode-se excluir uma peça da tabela, mesmo que haja pedidos ou notas fiscais relacionadas a ela. Dessa forma, perde-se a informação. Muitos poderiam dizer que o programa não permitiria tal ocorrência. É verdade. Porém todos os bancos de dados possuem programas de manipulação direta dos dados. Não é apenas por meio do seu programa que alguém irá manipular as informações. Aliás, se alguém estiver mal-intencionado, não utilizará o seu programa para agir (...).

Para especificar uma chave estrangeira, utilize a seguinte sintaxe:

```
FOREIGN KEY nome-chave-estrangeira  ( lista-de-colunas )
   REFERENCES nome-tabela ( lista-de-colunas )
   ON UPDATE ação
   ON DELETE ação
```

Onde:

Opção	Descrição
nome-chave-estrangeira	Nome opcional da constraint.
lista-de-colunas	Lista de colunas da tabela que faz referência a outra tabela.
nome-tabela	Nome da tabela em que está a chave primária.

Opção	Descrição (cont.)
ação	Determina qual ação o banco de dados deve tomar quando for excluída ou alterada uma linha da tabela que contém referência a esta chave. Pode ser SET NULL (altera o conteúdo da coluna para Nulo, perdendo a referência, sem deixar valores inconsistentes), SET DEFAULT (altera o conteúdo da coluna para o valor especificado na cláusula DEFAULT se houver), CASCADE (exclui ou altera todos os registros que se relacionam a eles), NO ACTION (em caso de alteração, não modifica os valores que se relacionam a eles) ou RESTRICT (não permite a exclusão da chave primária).

Como exemplo, vamos fazer referência à tabela de clientes quando estamos criando a tabela de pedidos:

```
...

FOREIGN KEY  pedido_cliente_fk ( cdcliente )
    REFERENCES  cliente
    ON  UPDATE  CASCADE
    ON  DELETE  RESTRICT,

...
```

Veja que não houve a necessidade de incluir a lista de colunas após o nome da tabela que contém a chave primária, pois esta foi definida na criação da tabela Cliente.

DEFAULT

Serve para atribuir um conteúdo-padrão a uma coluna da tabela, sempre que for incluída uma nova linha na tabela. Deve-se especificar a palavra-chave DEFAULT, seguida do conteúdo-padrão. Imaginando uma coluna QUANTIDADE que deva ter o conteúdo-padrão 1, deveríamos criá-la desta forma:

```
...

QUANTIDADE  INTEGER  DEFAULT  1,

...
```

NOT NULL

Indica que o conteúdo de uma coluna não poderá ser Nulo. Se cada coluna não tiver valor atribuído durante uma inclusão, terá seu valor Nulo. Alguns outros bancos de dados não SQL atribuíam conteúdo em função do tipo de dado: campos numéricos tinham valor zero, cadeias de caracteres conteúdo branco e assim por diante.

Lembre-se: em bancos de dados SQL, colunas sem valor atribuído possuem conteúdo Nulo.

Dessa forma, ao tentar incluir uma coluna com essa restrição, que não apresenta valor, o banco de dados retornará uma mensagem de erro e não incluirá a linha. Imagine um caso em que não se possa incluir um cliente sem que seja preenchido o nome. Essa coluna seria criada desta forma:

```
...
NOME_CLIENTE  VARCHAR(50)  NOT  NULL,
...
```

UNIQUE

Indica que não pode haver repetição no conteúdo da coluna. Isso é diferente do conceito de chave primária. A chave primária, além de não permitir repetição, não pode conter valores nulos. Ao especificarmos que uma coluna deve conter valores únicos, indicamos que todos os valores não nulos devem ser exclusivos. Devemos acrescentar a cláusula UNIQUE, após a definição da coluna, ou UNIQUE, seguido dos campos que devam ter essa característica no final da criação da tabela. Ainda na tabela de clientes, vamos especificar o número do CPF como campo de valor único.

```
...
CPF  NUMERIC(11)  UNIQUE,
...
```

CHECK - Definição de domínio

Um domínio é uma expressão de valores possíveis para o conteúdo de uma coluna. Podemos, ao criarmos uma coluna, especificar quais os valores que poderão ser utilizados para preencher a coluna. Para criar um domínio para uma coluna, utilizamos a palavra-chave CHECK, seguida da condição que validará o conteúdo. Imagine um campo de sexo. Esse campo só poderá aceitar M para masculino ou F para feminino. O domínio dessa coluna é {M, F} e seria criado desta forma:

```
...

SEXO CHAR(1) CHECK ( UPPER(SEXO) = 'M' OR UPPER(SEXO) = 'F' ),

...
```

Assertivas

Uma assertiva é utilizada para estabelecer restrição no banco de dados com base em dados de uma ou mais tabelas. Dessa forma, você pode estabelecer como regra que a tabela CD sempre tenha mais de uma linha.

```
CREATE ASSERTION nome
    CHECK (expressão_lógica);

CREATE ASSERTION ha_CD
    CHECK (EXISTS select codigo_cd from CD);
```

Alteração de estrutura de tabela

Para alterar a estrutura de uma tabela, utilizamos o comando ALTER TABLE. Dependendo do banco de dados, podemos:

Acrescentar novas colunas

O comando utilizado para acrescentar novas colunas é muito semelhante ao da criação de colunas em uma tabela:

```
ALTER TABLE tabela
   ADD nome-coluna tipo-de-dado constraints [, nome-coluna tipo-de-
   dado constraints, ...]
```

Exemplo:

```
ALTER TABLE cliente
    ADD email varchar(80) UNIQUE
```

Acrescentar novas constraints

O comando utilizado para acrescentar novas constraints é muito semelhante ao da criação de constraints em uma tabela:

```
ALTER TABLE tabela
   ADD ( constraint )
```

Exemplo:

```
ALTER TABLE cliente
    ADD PRIMARY KEY (CDCLIENTE)
```

Nesse caso, estamos alterando uma constraint de tabela. Note que a colocação do parênteses após a cláusula ADD é opcional. Uma constraint de coluna deve ser alterada utilizando o comando a seguir, relacionado à própria coluna.

Modificar colunas

Podemos modificar qualquer característica de uma coluna, seja o tipo de dado (alguns bancos de dados requerem ausência de conteúdo na coluna para fazer essa alteração), o tamanho (alguns só aceitam alterações para valores maiores que o definido) e as constraints:

```
ALTER TABLE tabela
   MODIFY ( nome-coluna tipo-de-dado constraints )
```

Exemplo:

```
ALTER TABLE cliente
    MODIFY email varchar(100) NOT NULL
```

Podemos especificar apenas o que estamos modificando; não é necessário repetir o que não será alterado. Assim, se quisermos apenas modificar o tipo de dado e seu tamanho, não é necessário especificar a constraint e vice-versa. De qualquer forma, devemos sempre especificar qual coluna está sofrendo a modificação.

Excluindo elementos

Pelo padrão SQL, deveria ser possível excluir colunas ou constraints de uma tabela. Alguns bancos de dados não permitem a exclusão de colunas. No Oracle, a cláusula de exclusão é DROP e não DELETE, como definido desde a SQL-92.

```
ALTER TABLE tabela
   DELETE elemento
```

Exemplo 1 - Exclusão de coluna:

```
ALTER  TABLE  cliente
    DELETE  email
```

Exemplo 2 - Exclusão de constraint de tabela:

```
ALTER  TABLE  cliente
    DELETE  primary  key
```

Exemplo 3 - Exclusão de constraint de tabela:

```
ALTER  TABLE  cliente
    DELETE  FOREIGN  KEY  pedido_cliente_fk
```

Note que a exclusão de constraint de coluna deve ser feita como uma alteração da coluna. Veja o exemplo anterior.

Trocar nome de elementos

Podemos trocar o nome de tabelas ou colunas (alguns bancos de dados não permitem essa operação):

a. Alteração de nome de tabela

```
ALTER TABLE tabela
   RENAME nome-tabela
```

Exemplo:

```
ALTER  TABLE  cliente
    RENAME  cli
```

b. Alteração de nome de coluna

```
ALTER TABLE tabela
   RENAME coluna-velha TO coluna-nova
```

Exemplo:

```
ALTER TABLE cliente
    RENAME nome_cliente TO nmcliente
```

Laboratório

A seguir, apresentaremos a lista de comandos para a criação das tabelas e relacionamentos utilizados neste livro, para exemplificar a utilização do SQL.

```
CREATE  TABLE  AUTOR (
    Codigo_Autor  INTEGER NOT NULL,
    Nome_Autor    VARCHAR(60) NULL );

ALTER  TABLE  AUTOR
    ADD  ( PRIMARY KEY (Codigo_Autor) ) ;

CREATE  TABLE  CD (
    Codigo_CD           INTEGER NOT NULL,
    Codigo_Gravadora    INTEGER NULL,
    Nome_CD             VARCHAR(60) NULL,
    Preco_Venda         DECIMAL(16,2) NULL,
    Data_Lancamento     DATE NULL,
    CD_Indicado         INTEGER NULL );

ALTER  TABLE  CD
    ADD  ( PRIMARY KEY (Codigo_CD) ) ;

CREATE TABLE GRAVADORA (
    Codigo_Gravadora    INTEGER NOT NULL,
    Nome_Gravadora      VARCHAR(60) NULL,
    Endereço            VARCHAR(60) NULL,
    Telefone            VARCHAR(20) NULL,
    Contato             VARCHAR(20) NULL,
    URL                 VARCHAR(80) NULL );

ALTER  TABLE  GRAVADORA
    ADD  ( PRIMARY KEY (Codigo_Gravadora) ) ;

CREATE  TABLE  FAIXA (
    Codigo_Musica    INTEGER NOT NULL,
    Codigo_CD        INTEGER NOT NULL,
    Numero_Faixa     INTEGER NULL,);

ALTER  TABLE  ITEM_CD
    ADD ( PRIMARY KEY (Código_Musica, Codigo_CD) ) ;

CREATE  TABLE  MUSICA (
    Codigo_Musica    INTEGER NOT NULL,
    Nome_Musica      VARCHAR(60) NULL,
    Duracao          DECIMAL(6,2) NULL );
```

```
ALTER  TABLE  MUSICA
    ADD ( PRIMARY KEY (Codigo_Musica) ) ;

CREATE  TABLE  MUSICA_AUTOR (
    Codigo_Musica    INTEGER NOT NULL,
    Codigo_Autor     INTEGER NOT NULL );

ALTER  TABLE  MUSICA_AUTOR
    ADD ( PRIMARY KEY (Codigo_Musica, Codigo_Autor) ) ;

CREATE  TABLE CD_CATEGORIA (
    Codigo_Categoria   INTEGER    NOT NULL,
    Menor_Preco        DECIMAL(14,2) NOT NULL,
    Maior_Preco        DECIMAL(14,2) NOT NULL );

ALTER  TABLE  CD
    ADD  ( FOREIGN KEY (Codigo_Gravadora)
    REFERENCES GRAVADORA ) ;

ALTER  TABLE  CD
    ADD  ( CONSTRAINT CD_CD
    FOREIGN KEY (CD_Indicado)
    REFERENCES CD ) ;

ALTER  TABLE  FAIXA
    ADD  ( FOREIGN KEY (Codigo_Musica)
    REFERENCES MUSICA ) ;

ALTER  TABLE  FAIXA
    ADD  ( FOREIGN KEY (Codigo_CD)
    REFERENCES CD ) ;

ALTER  TABLE  MUSICA_AUTOR
    ADD  ( FOREIGN KEY (Codigo_Autor)
    REFERENCES AUTOR ) ;

ALTER  TABLE  MUSICA_AUTOR
    ADD  ( FOREIGN KEY (Codigo_Musica)
    REFERENCES MUSICA ) ;
```

Eliminando uma tabela

Para eliminar uma tabela do banco de dados, utilizamos o comando DROP TABLE seguido do nome da tabela. Alguns bancos de dados somente permitirão a exclusão da tabela desde que esta não esteja rlacionada a outras. Alguns deles podem conter cláusulas específicas para excluir a tabela, independentemente das constraints definidas (no Oracle, você pode especificar CASCADE após o nome da tabela). Exemplo:

```
DROP TABLE cliente
```

Criação de índice

O índice serve para prover um acesso rápido a linhas das tabelas. Por meio dele é possível unir uma ou mais colunas por onde o acesso é mais freqüente . Por exemplo: quando temos uma tabela de pessoas, normalmente queremos fazer buscas em ordem alfabética. Como o nome não é um boa chave primária para a tabela (por ser alfanumérica o que deixa as pesquisas mais lentas - e pela possível repetição de nomes), então criamos um índice para o nome da pessoa. Assim, mesmo que o índice não seja a chave primária, garante-se um acesso mais rápido aos nomes, uma vez que, os dados buscados estariam fora de seqüência. Os valores indexados são armazenados em um objeto do banco de dados em ordem, o que permite ao gerenciador do banco de dados pesquisar primeiro no índice, para depois buscar na tabela.

O funcionamento de um índice é muito simples. É mais ou menos como o índice disposto no final deste livro; os principais assuntos estão indexados em ordem alfabética. Você localiza ali o assunto, verifica o número da página em que ele é tratado e vai diretamente a essa página, sem precisar buscá-lo seqüencialmente nas páginas. Isso também acontece com o índice de uma tabela.

Alguns bancos de dados necessitam de índices constantemente atualizados, pois, em virtude de quedas de energia ou outros problemas no gerenciamento da informação, os índices eventualmente perdem a referência ao dado. Outros bancos de dados se incumbem de realizar esse trabalho automaticamente, sem a interferência do usuário. Você deve estar atento a esse problema para evitar que a sua aplicação perca o acesso adequado aos dados.

Observação: a criação de índices não é definida no padrão SQL-92, mas está presente na grande maioria dos bancos de dados.

Quando criar um índice

A criação de índices na tabela apresenta uma enorme vantagem quando bem dirigida, mas pode trazer enormes problemas se for aplicada sem critério. Todo trabalho em um banco de dados tem um custo! Não adianta sair criando índices para cada um dos campos da tabela, mesmo que haja pesquisas eventuais em cada um dos campos.

Quando for criado um índice, o banco de dados deverá mantê-lo atualizado a cada nova inclusão, exclusão ou alteração nos dados. Além do mais, o índice ocupará espaço no banco de dados. Lembre-se: é criado um novo objeto que manterá as informações indexadas e armazenará a referência à linha física da tabela. Se são criados muitos índices em uma tabela, pode-se prejudicar demasiadamente o desempenho do banco de dados. Se são criados poucos índices, pode-se prejudicar o desempenho da aplicação. Realmente não é fácil decidir...

Algumas dicas para ajudá-lo a decidir:

a. Chaves Primárias: se o seu banco de dados não criá-las automaticamente, crie.

b. Chaves Estrangeiras: crie.

c. Colunas utilizadas freqüentemente na cláusula WHERE do comando SELECT (veja mais adiante): crie.

d. Colunas com muito conteúdo NULO (não inicializado): não crie.

e. Colunas com muitos valores iguais: não crie.

f. Tabelas muito pequenas: não crie.

Naturalmente, sempre que uma consulta ao banco de dados estiver muito lenta, verifique a possibilidade de criar um índice para facilitar a busca.

CREATE INDEX

Para criar um índice, utilize o comando:

```
CREATE [UNIQUE] INDEX nome
   ON tabela ( coluna [, coluna, ... ]  [ASC | DESC] )
```

Onde:

Identificador	Descrição
UNIQUE	Identifica que esse índice não permite repetição de conteúdo na chave (lista de colunas). Se o índice não for especificado, admitirá repetição.
nome	Nome do objeto que será criado.
tabela	Nome da tabela que contém a(s) coluna(s).
coluna	Lista de colunas que compõe a chave de indexação.

Identificador	Descrição (cont.)
ASC	Determina que a ordem de indexação é ascendente (opção-padrão, caso não seja definido DESC).
DESC	Determina que a ordem de indexação é descendente.

A quantidade de colunas indexadas varia muito em função do banco de dados utilizado.

Exemplos:

```
CREATE  INDEX  xAutor
    ON Autor ( NMAUTOR )

CREATE  INDEX  xPedido
    ON Pedido ( CDPEDIDO DESC )

CREATE  INDEX  xCidade
    ON Cidade ( SGESTADO, CDCIDADE )

CREATE  UNIQUE  INDEX  xCliente
    ON Cliente ( CDCLIENTE )
```

DROP INDEX

Para excluir um índice, utilize o comando:

```
DROP INDEX nome
```

Exemplo:

```
DROP INDEX xAutor
```

Laboratório

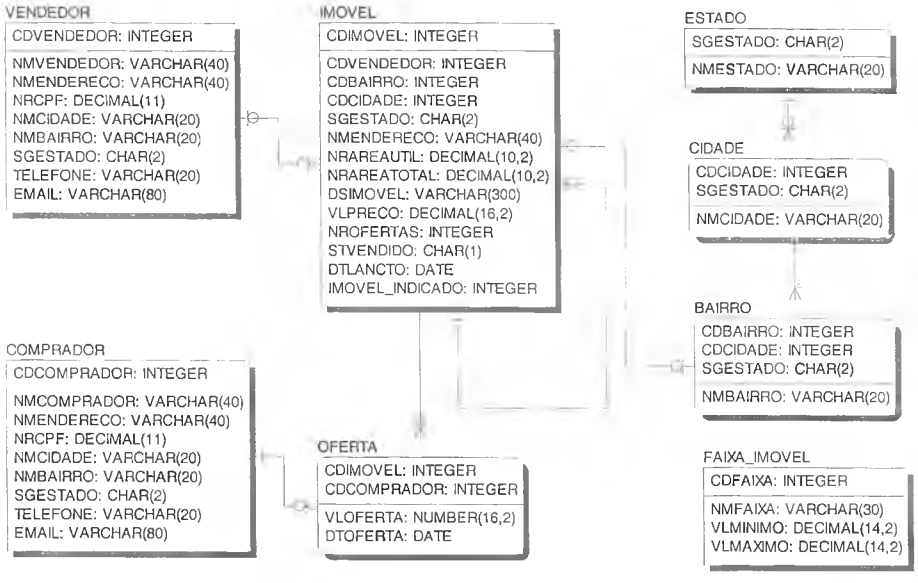
Criaremos os índices que serão utilizados nas tabelas de exemplo:

```
CREATE  INDEX  XIF9CD  ON  CD
    ( Codigo_Gravadora  ASC );

CREATE  INDEX  XIF10ITEM_CD  ON  ITEM_CD
    ( Codigo_CD  ASC );

CREATE  INDEX  XIF14ITEM_CD  ON  ITEM_CD
    ( Codigo_Musica  ASC );
```

```
CREATE  INDEX  XIF12MUSICA_AUTOR  ON  MUSICA_AUTOR
    ( Codigo_Musica  ASC );

CREATE  INDEX  XIF13MUSICA_AUTOR  ON  MUSICA_AUTOR
    ( Codigo_Autor  ASC );
```

Exercícios propostos

1. Crie as tabelas e os índices para o modelo de dados a seguir, incluindo os relacionamentos e toda a integridade referencial:



2. Crie índices para VENDEDOR.NMVENDEDOR e COMPRADOR.NMCOMPRADOR.

3. Crie índices para OFERTA.CDIMOVEL (ascendente) e OFERTA.VLOFERTA (descendente).

Observação: em todos os exercícios dos próximos capítulos, estaremos fazendo exercícios nessa base de dados. Vamos chamá-la de ImovelNet.

5

Incluindo, atualizando e excluindo linhas nas tabelas

Agora que nossas tabelas estão devidamente criadas, relacionadas e com índices para agilizar as pesquisas, vamos movimentar os dados nas tabelas para depois podermos treinar pesquisas. Essa parte do livro inicia o tratamento da *Data Manipulation Language* (DML) que será concluída com a pesquisa no capítulo 6. Há três comandos utilizados para modificar dados em tabelas: INSERT, DELETE e UPDATE. Normalmente, um sistema desenvolvido para usuário final "esconde" a complexidade desses comandos, mas o programador precisa conhecê-los para criar rotinas que cumpram esse papel.

Incluindo dados em tabelas

Para incluir linhas em tabelas, utilizamos o comando:

```
INSERT INTO tabela [ ( coluna [, coluna, ... ] ) ]
   VALUES ( conteúdo [, conteúdo, ... ] )
```

Onde:

Cláusula	Descrição
tabela	Nome da tabela em que será efetuada a inclusão da linha.
coluna	Lista de colunas que terão seus valores atribuídos.
conteúdo	Conteúdo que será atribuído a cada um dos campos. Note que estes devem corresponder (em quantidade e tipo de dado) à lista de colunas especificadas anteriormente.

Note bem

Você deve ter notado que a lista de colunas é opcional no comando INSERT. Caso você não especifique em quais colunas está incluindo valores, assume-se a inclusão de valores em todas as colunas e a ordem de inclusão (lista de conteúdo) corresponderá àquela definida na criação da tabela. Se houver uma reorganização na estrutura da tabela ou se forem acrescentadas novas colunas, o comando INSERT retornará uma mensagem de erro. É natural que o comando ficará muito mais claro caso você especifique a lista de colunas, e isto impedirá que, eventualmente, tendo sido criada a tabela em uma ordem diferente daquela que você imagina, venha a causar erro em virtude do tipo de dado ou pior, o conteúdo que deveria ir para uma coluna acabe indo para outra. Caso você especifique a lista de colunas, deverá informar os conteúdos na mesma seqüência, pois o banco de dados utilizará a posição relativa nas listas.

Note que os valores numéricos não devem vir acompanhados de aspas simples ou apóstrofos. Já os valores alfanuméricos devem conter esse separador. Colunas do tipo data devem ser tratadas conforme especificação do banco de dados. Normalmente, utiliza-se uma função para converter valores alfanuméricos em data ou são informadas entre apóstrofos, como uma coluna alfanumérica.

Sempre que você incluir uma linha no banco de dados, o gerenciador checará quaisquer restrições de integridade (*constraints*). Assim, chaves primárias, estrangeiras, domínios etc. serão verificados no momento da inclusão. Se alguma restrição for violada, a linha não será incluída e uma mensagem de erro será enviada pelo banco de dados.

Lembre-se de que colunas que não tiverem valores atribuídos terão conteúdo nulo. Caso, eventualmente, você queira explicitamente incluir um valor nulo em uma coluna da tabela, você deve informar NULL no lugar correspondente a esta coluna. Isso normalmente ocorre quando você estiver incluindo uma linha sem especificar a lista de colunas. Caso você esteja especificando as colunas, basta não informar essa coluna na lista. O banco de dados automaticamente incluirá conteúdo nulo, caso não haja restrição para valores nulos (NOT NULL).

Exemplos:

```
INSERT  INTO  Autor
    VALUES ( 1,'Renato Russo' )

INSERT  INTO  Autor ( CDAUTOR, NMAUTOR )
    VALUES ( 2, 'Tom Jobim' )
```

Incluindo várias linhas

Para incluir diversas linhas em uma tabela do banco de dados, utiliza-se o comando INSERT em conjunto com o comando SELECT. Veja que, nesse caso, apenas se copiam linhas de uma tabela para outra. Isso será útil para criar tabelas temporárias com base em outra(s) tabela(s). Veja a sintaxe:

```
INSERT INTO tabela [ ( coluna [, coluna, ... ] ) ]
   SELECT comando-select
```

Onde:

Cláusula	Descrição
tabela	Nome da tabela em que serão incluídas as linhas.
coluna	Lista de colunas que terão seus valores atribuídos.
comando-select	Comando *select* conforme veremos mais adiante.

Nesse caso, a lista de colunas do comando SELECT deve corresponder à totalidade das colunas na mesma seqüência das colunas da tabela, se não for especificada a lista de colunas, ou conter os conteúdos relativos à lista de colunas, na mesma ordem e com os mesmos tipos de dados.

No exemplo a seguir, incluiremos em um tabela chamada tmpAutor todas as linhas da tabela Autor. O exemplo seguinte mostrará o mesmo comando, mas especificando o nome de cada uma das colunas.

Exemplo:

```
INSERT  INTO  tmpAutor
    SELECT  *  FROM  Autor

INSERT  INTO  tmpAutor ( CDAUTOR, NMAUTOR )
    SELECT  CDAUTOR, NMAUTOR  FROM  Autor
```

Note bem

As linhas incluídas nas tabelas não seguem uma regra de colocação fixa. Cada banco de dados utilizará um conceito de armazenamento próprio. A maior parte manterá uma coluna interna de identificação do registro na tabela. Dessa forma, nada garante que, ao incluir uma linha, esta irá para o final da tabela. Na realidade, para onde irá a linha que estamos incluindo importa muito pouco. Quando formos extrair as informações das tabelas, definiremos a ordem de exibição das linhas. Internamente, o gerenciador do banco de dados resolverá essa questão, com maior ou menor eficiência, dependendo da sua estrutura de localização de linhas.

Laboratório

Seguem os comandos para inclusão das linhas do nosso laboratório. Observação: você poderá baixar pela Internet no site www.novateceditora.com.br/downloads.php a lista completa de comandos INSERT para colocar todos os dados nas tabelas do nosso modelo. Dessa forma, você poderá testar todos os demais comandos deste livro e comparar as respostas.

```
INSERT  INTO  AUTOR (CODIGO_AUTOR, NOME_AUTOR)
    VALUES ( 1, 'Renato Russo' );

INSERT  INTO  AUTOR (CODIGO_AUTOR, NOME_AUTOR)
    VALUES ( 2, 'Tom Jobim' );

INSERT  INTO  MUSICA (CODIGO_MUSICA, NOME_MUSICA, DURACAO )
    VALUES ( 1, 'Será',2.28 );

INSERT  INTO  MUSICA (CODIGO_MUSICA, NOME_MUSICA, DURACAO )
    VALUES ( 2, 'Ainda é Cedo', 3.55 );

INSERT  INTO  GRAVADORA (CODIGO_GRAVADORA, NOME_GRAVADORA,
    ENDERECO, URL )
    VALUES ( 1, 'EMI', 'Rod. Pres. Dutra, s/n - km 229,8',
    'www.emi-music.com.br' );

INSERT  INTO  CD (CODIGO_CD, CODIGO_GRAVADORA, NOME_CD,
    PRECO_VENDA, DATA_LANCAMENTO )
    VALUES ( 1, 1, 'Mais do Mesmo', 15.00, '01/10/1998' );

INSERT  INTO  FAIXA (CODIGO_CD, NUMERO_FAIXA, CODIGO_MUSICA )
    VALUES ( 1, 1, 1 );

INSERT  INTO  FAIXA (CODIGO_CD, NUMERO_FAIXA, CODIGO_MUSICA )
    VALUES ( 1, 2, 2 );

INSERT INTO MUSICA_AUTOR (CODIGO_MUSICA, CODIGO_AUTOR )
    VALUES ( 1, 1 );
```

Atualização de dados em tabelas

Uma vez que uma linha esteja em uma tabela, pode-se querer alterar o conteúdo de uma ou mais colunas, ou até o conteúdo de uma coluna em diversas linhas. Para isso, utilizamos o comando UPDATE:

```
UPDATE tabela
   SET coluna = conteúdo [, coluna =  conteúdo, ...]
   [ WHERE condição ]
```

Onde:

Cláusula	Descrição
tabela	Nome da tabela onde será realizada a alteração.
coluna	Coluna que terá seu valor atualizado.
conteúdo	Novo conteúdo para a coluna.
condição	Condição que indicará o escopo (limite) de atualização das linhas.

Note bem

O conteúdo deve respeitar o tipo de dado da coluna. Assim, dados alfanuméricos devem vir separados por apóstrofos ou aspas simples, dados numéricos não devem conter apóstrofos e data ou deverão ser alimentados via função de transformação de dados (CAST ou TO_DATE, veja no capítulo 9 Conversão de Tipo de Dados) ou entre apóstrofos, dependendo do banco de dados.

A condição expressa no comando servirá para definir quais linhas devem ser atualizadas. Lembre-se: o SQL não é uma linguagem procedural. Assim, podemos definir que várias linhas de uma mesma tabela sejam atualizadas ao mesmo tempo. Dessa forma, todas as linhas serão atualizadas ou nenhuma delas será.

Caso a cláusula WHERE contenha a chave primária da tabela, somente uma linha da tabela será atualizada, pois a chave primária não admite repetições. Todas as condições de restrição de integridade (*constraints*) serão avaliadas pelo banco de dados, incluindo chaves primária, estrangeira, domínios etc.

Assim, se quisermos alterar o preço de um CD, podemos utilizar o seguinte comando, filtrando na cláusula WHERE a chave primária do CD cujo preço será alterado:

```
UPDATE  cd
    SET  PRECO_VENDA = 15
    WHERE  CODIGO_CD = 1;
```

Imagine que ocorra um aumento de preço para todos os CDs de uma determinada gravadora. Nesse caso, utilizaremos como filtro da cláusula WHERE o código da gravadora, coluna que faz parte da estrutura da tabela CD:

```
UPDATE  cd
    SET  PRECO_VENDA = 15
    WHERE  CODIGO_GRAVADORA = 2;
```

Também é possível alterar o conteúdo de uma coluna com base nela mesma. Aumenta-se o preço de todos os CDs de uma gravadora em 5%, multiplicando o próprio preço do CD por 1.05:

```
UPDATE  cd
    SET  PRECO_VENDA = PRECO_VENDA * 1.05
    WHERE  CODIGO_GRAVADORA = 2;
```

Se não for especificada a cláusula WHERE, então todas as linhas serão afetadas pelo comando UPDATE. Todos os CDs serão aumentados em 5%. Veja:

```
UPDATE  cd
    SET  PRECO_VENDA = PRECO_VENDA * 1.05;
```

Mais de uma coluna poderia ser alterada ao mesmo tempo e poderíamos utilizar o conteúdo de uma coluna para alterar o conteúdo de outra. Imagine uma tabela de Item de Pedidos em que, por qualquer motivo, escolheu-se armazenar o valor total do item na linha (valor unitário multiplicado pela quantidade). O comando UPDATE ficaria assim:

```
UPDATE  ItemPedido
    SET  VLTOTALITEM = VLUNITARIO * NRQTDE
    WHERE  CDPEDIDO = 1  AND  NRITEM = 1;
```

Nesse caso, utilizamos um operador AND para determinar que somente no caso de o Código do Pedido ser 1 e o item ser 1 é que a coluna da tabela deve ser atualizada. Outro operador que podemos utilizar em outra situação é o OR, que indicará qual condição deverá ser satisfeita para que o comando seja executado.

Mais adiante, utilizaremos muito a cláusula WHERE para determinar condições e, com certeza, eventuais dúvidas que você tenha serão sanadas.

Laboratório

Seguem alguns exemplos de alteração das linhas do nosso laboratório:

```
UPDATE  cd
    SET  PRECO_VENDA = 15
    WHERE  CODIGO_CD = 1;

UPDATE  autor
    SET  NOME_AUTOR = 'B. Manilow'
    WHERE  CODIGO_AUTOR = 54;
```

```
UPDATE  cd
    SET  PRECO_VENDA = PRECO_VENDA * 1.10
    WHERE  CODIGO_GRAVADORA = 2;

UPDATE  gravadora
    SET  URL = 'www.epic-music.com.br',
    NOME_GRAVADORA = 'EPIC MUSIC'
    WHERE  CODIGO_GRAVADORA = 4;
```

Exclusão de dados em tabelas

Sempre que alguma informação deixar de ser relevante para ser armazenada, pode-se excluí-la de tabelas. O comando DELETE pode, da mesma forma que o comando UPDATE, afetar uma ou mais linhas de uma tabela. Mais uma vez, o escopo da exclusão será definido pela cláusula WHERE. Veja a sintaxe:

```
DELETE FROM tabela
   [ WHERE condição ]
```

Onde:

Cláusula	Descrição
tabela	Nome da tabela onde será excluída a linha.
condição	Condição que define o escopo (limite) de exclusão de linhas na tabela.

Note bem

Se você especificar na cláusula WHERE a chave primária de uma tabela, somente uma linha da tabela será excluída. Se for outra condição, várias linhas poderão ser excluídas. Portanto, tome cuidado com a condição que definirá esse limite. Caso não seja especificada a cláusula WHERE, todas as linhas da tabela serão excluídas.

Outro ponto importante é que o gerenciador irá sempre checar a integridade das informações. Se você tentar excluir uma linha de uma tabela que tenha "filhos" em outras tabelas (por exemplo, tentar excluir um autor que tenha músicas vinculadas a ele), será retornada uma mensagem de erro. Isso ocorre porque, de outra forma, a informação ficaria "perdida" na tabela dependente. Naturalmente isso somente ocorrerá quando essa função não tiver sido desabilitada (veja tópico de criação de tabelas).

Neste primeiro exemplo, apenas o autor com código 1 será excluído (desde que não haja relacionamentos com outras tabelas):

```
DELETE  FROM  Autor
    WHERE  CDAUTOR = 1;
```

Neste exemplo, todos os CDs da gravadora 2 serão excluídos:

```
DELETE  FROM  cd
    WHERE  CODIGO_GRAVADORA = 2;
```

Neste último exemplo, todas as músicas serão excluídas:

```
DELETE  FROM  Musica;
```

Controle básico de transações

Uma vez realizada operações de inclusão, alteração e exclusão, a modificação realizada não estará ainda no Banco de Dados. Ela é visível apenas na seção atual. Todas as modificações podem ser efetuadas de duas formas no banco de dados: ou serão gravadas para que todos os usuários tenham acesso ou serão descartadas e nenhum usuário poderá visualizá-las.

Para que os comandos de inclusão, exclusão e alteração sejam definitivamente enviados ao banco de dados, é necessário utilizarmos o comando:

```
COMMIT [WORK];
```

Para que os comandos sejam descartados utilizamos o comando:

```
ROLLBACK [WORK];
```

No capítulo 15, o controle de transações será tratado com mais detalhes.

Exercícios propostos

1. Inclua linhas na tabela ESTADO:

SGESTADO	NMESTADO
SP	SÃO PAULO
RJ	RIO DE JANEIRO

2. Inclua linhas na tabela CIDADE:

CDCIDADE	NMCIDADE	SGESTADO
1	SÃO PAULO	SP
2	SANTO ANDRÉ	SP
3	CAMPINAS	SP
1	RIO DE JANEIRO	RJ
2	NITERÓI	RJ

3. Inclua linhas na tabela BAIRRO:

CDBAIRRO	NMBAIRRO	CDCIDADE	SGESTADO
1	JARDINS	1	SP
2	MORUMBI	1	SP
3	AEROPORTO	1	SP
1	AEROPORTO	1	RJ
2	FLAMENGO	1	RJ

4. Inclua linhas na tabela VENDEDOR:

CDVENDEDOR	NMVENDEDOR	NMENDERECO	EMAIL
1	MARIA DA SILVA	RUA DO GRITO, 45	msilva@novatec.com.br
2	MARCOS ANDRADE	AV. DA SAUDADE, 325	mandrade@novatec.com.br
3	ANDRE CARDOSO	AV BRASIL, 401	acardoso@novatec.com.br
4	TATIANA SOUZA	RUA DO IMPERADOR, 778	tsouza@novatec.com.br

5. Inclua linhas na tabela IMOVEL:

CDIMOVEL	CDVENDEDOR	CDBAIRRO	CDCID	SGE	NMENDERECO	NRAREAUTIL	NRAREATOT	VLPRECO	IMOVEL_INDICADO
1	1	1	1	SP	AL TIETE, 3304 AP 101	250	400	180000	NULL
2	1	2	1	SP	AV MORUMBI, 2230	150	250	135000	1
3	2	1	1	RJ	R GAL OSORIO, 445 AP 34	250	400	185000	2
4	2	2	1	RJ	R D PEDRO I, 882	120	200	110000	1
5	3	3	1	SP	AV RUBEM BERTA, 2355	110	200	95000	4
6	4	1	1	RJ	R GETULIO VARGAS, 552	200	300	99000	5

6. Inclua linhas na tabela COMPRADOR:

CDCOMPRADOR	NMCOMPRADOR	NMENDERECO	EMAIL
1	EMMANUEL ANTUNES	R SARAIVA, 452	eantunes@novatec.com.br
2	JOANA PEREIRA	AV PORTUGAL, 52	jpereira@novatec.com.br
3	RONALDO CAMPELO	R ESTADOS UNIDOS, 790	rcampelo@novatec.com.br
4	MANFRED AUGUSTO	AV BRASIL, 351	maugusto@novatec.com.br

7. Inclua linhas na tabela OFERTA:

CDCOMPRADOR	CDIMOVEL	VLOFERTA	DTOFERTA
1	1	170000	10/01/02
1	3	180000	10/01/02
2	2	135000	15/02/02
2	4	100000	15/02/02
3	1	160000	05/01/02
3	2	140000	20/02/02

8. Aumente o preço de venda dos imóveis em 10%.

9. Abaixe o preço de venda dos imóveis do vendedor 1 em 5%.

10. Aumente em 5% o valor das ofertas do comprador 2.

11. Altere o endereço do Comprador 3 para R ANANÁS, 45 e o Estado para RJ.

12. Altere a oferta do comprador 2 no imóvel 4 para 101.000.

13. Exclua a oferta do comprador 3 no imóvel 1.

14. Exclua a cidade 3 do Estado SP.

15. Inclua linhas na tabela FAIXA_IMOVEL:

CDFAIXA	NMFAIXA	VLMINIMO	VLMAXIMO
1	BAIXO	0	105000
2	MÉDIO	105001	180000
3	ALTO	180001	999999

Parte II
SQL básico

6

Pesquisa básica em tabelas

Agora que sabemos criar, incluir, alterar e excluir informações nas tabelas, veja como podemos realizar pesquisas, ou seja, extrair informações do banco de dados. O comando utilizado para esse fim é o comando SELECT. Atrás dele há uma extensão de possibilidades que vão desde a simples extração do conteúdo de todas as linhas e colunas de uma tabela até a união de diversas tabelas, cálculos, agrupamentos, além de ordenações e filtragem de linhas e colunas. Nesta primeira parte, vamos filtrar linhas e colunas e ordenar o resultado da busca.

A sintaxe mais simples do comando SELECT é:

```
SELECT [DISTINCT | ALL] { * | coluna [, coluna, ... ] }
   FROM tabela
```

Onde:

Cláusula	Descrição
DISTINCT	Não mostra eventuais valores repetidos de colunas.
ALL	Mostra todos os valores, mesmo que repetidos. Esse é o padrão se DISTINCT não for definido.
*	Indica que devem ser mostradas todas as colunas da tabela.
coluna	Lista de colunas que devem ser mostradas.
tabela	Nome da tabela em que será realizada a busca.

Para visualizar todas as linhas e colunas da tabela Autor, devemos utilizar o seguinte comando:

```
SELECT  *
    FROM  CD;
```

```
CODIGO_CD   CODIGO_GRAVADORA   NOME_CD                      PRECO_VENDA DATA_LAN
---------   ----------------   --------------------------   ----------- -------
2           2                  Bate-Boca                    12          01/07/99
3           3                  Elis Regina - Essa Mulher    13          01/05/89
4           2                  A Força que nunca Seca       13,5        01/12/98
5           3                  Perfil                       10,5        01/05/01
6           2                  Barry Manilow Greatest Hits  9,5         01/11/91
7           2                  Listen Without Prejudice     9           01/10/91
1           1                  Mais do Mesmo                15          01/10/98
```

O exemplo a seguir demonstra como filtrar apenas algumas colunas da tabela:

```
SELECT  CODIGO_CD, NOME_CD
    FROM  CD;

CODIGO_CD   NOME_CD
---------   ----------------------------------
2           Bate-Boca
3           Elis Regina - Essa Mulher
4           A Força que nunca Seca
5           Perfil
6           Barry Manilow Greatest Hits
7           Listen Without Prejudice
1           Mais do Mesmo
```

Ordenando o resultado

Você deve ter notado que, mesmo que entremos com as linhas em determinada ordem (em ordem de CODIGO_CD, por exemplo), a ordem mostrada nem sempre é aquela que esperamos. Em linguagens de terceira geração era muito comum termos que abrir uma tabela e indicar o índice pelo qual gostaríamos de extrair a informação. Ao utilizarmos o SQL, devemos utilizar a cláusula ORDER BY para determinar a ordem em que são mostradas as linhas de uma tabela. Portanto, para alterar a ordem, você deverá acrescentar ao comando a cláusula ORDER BY seguida pela(s) coluna(s) que desejar. Sintaxe:

```
SELECT [DISTINCT | ALL] { * | coluna [, coluna, ... ] }
   FROM tabela
   ORDER BY coluna-ord [, coluna-ord, ... ]
```

Onde:

Cláusula	Descrição
coluna-ord	Lista de colunas na ordem de precedência de classificação.

Para ver a lista de autores em ordem alfabética, devemos escrever o comando da seguinte forma:

```
SELECT  CODIGO_CD, NOME_CD
    FROM  CD
    ORDER  BY  NOME_CD;

CODIGO_CD   NOME_CD
---------   ----------------------
4           A Força que nunca Seca
6           Barry Manilow Greatest Hits
2           Bate-Boca
3           Elis Regina - Essa Mulher
7           Listen Without Prejudice
1           Mais do Mesmo
5           Perfil

7 linhas selecionadas.
```

Para visualizar em ordem de código, utilizamos o código abaixo:

```
SELECT  CODIGO_CD, NOME_CD
    FROM  CD
    ORDER  BY  CODIGO_CD;

CODIGO_CD   NOME_CD
---------   ---------------------------
1           Mais do Mesmo
2           Bate-Boca
3           Elis Regina - Essa Mulher
4           A Força que nunca Seca
5           Perfil
6           Barry Manilow Greatest Hits
7           Listen Without Prejudice

7 linhas selecionadas.
```

Se você especificar mais de uma coluna para ser ordenada, o gerenciador primeiro ordenará pela primeira coluna, em seguida pela segunda e assim por diante. Por exemplo: se você quisesse fazer a mesma consulta, mas ordenando primeiro pelo código da gravadora e em seguida pela ordem alfabética do nome do CD, o comando deveria ser o seguinte:

```
SELECT  CODIGO_GRAVADORA, NOME_CD
    FROM  CD
    ORDER  BY  CODIGO_GRAVADORA, NOME_CD;
```

```
CODIGO_GRAVADORA    NOME_CD
----------------    ----------------------------
1                   Mais do Mesmo
2                   A Força que nunca Seca
2                   Barry Manilow Greatest Hits
2                   Bate-Boca
2                   Listen Without Prejudice
3                   Elis Regina - Essa Mulher
3                   Perfil

7 linhas selecionadas.
```

Veja como a ordem apresentada na listagem mostra os dados ordenados por código de gravadora, e, em seguida, como há valores que se repetem nessa coluna, estes são mostrados em ordem alfabética de nome de CD.

Filtrando linhas

Para filtrar linhas em uma pesquisa, utilizamos a cláusula WHERE. Aqui, definimos uma expressão lógica (condição) que será avaliada e mostrará apenas as linhas que atenderem ao critério estabelecido. Sintaxe:

```
SELECT [DISTINCT | ALL] { * | coluna [, coluna, ... ] }
   FROM tabela
   WHERE condição
```

Onde:

Cláusula	Descrição
condição	Condição que define o escopo (limite) de apresentação dos resultados.

Devemos saber exatamente como construir condições que satisfaçam às nossas necessidades de busca para atingir nossos objetivos. Sempre que a condição especificada for verdadeira, a linha será mostrada. Por outro lado, sempre que a condição testada na linha for falsa, então a linha não será mostrada. Há diversas formas e técnicas para fazer isso. Vamos estudá-las.

Operadores relacionais

Operadores relacionais definem um tipo de condição básica. Podemos testar igualdade, diferença, maior, menor, maior ou igual e menor ou igual. Deve-se colocar o operador entre os argumentos que estão sendo comparados. Veja uma lista de operadores relacionais, seu significado e exemplo de utilização.

Operador	Significado	Exemplo
=	igual	CODIGO_AUTOR = 2
<	menor que	PRECO_VENDA < 10
<=	menor ou igual a	PRECO_VENDA <= 10
>	maior que	PRECO_VENDA > 10
>=	maior ou igual a	PRECO_VENDA >= 10
!= ou <>	diferente	CODIGO_AUTOR != 2 ou CODIGO_AUTOR <> 2

Dessa forma, se quisermos pesquisar apenas os CDs com Preço de Venda superior a 10, devemos utilizar o seguinte comando:

```
SELECT  NOME_CD,PRECO_VENDA
    FROM  CD
    WHERE  PRECO_VENDA > 10;

NOME_CD                    PRECO_VENDA
-------------------------  -----------
Bate-Boca                  12
Elis Regina - Essa Mulher  13
A Força que nunca Seca     13,5
Perfil                     10,5
Mais do Mesmo              15
```

Note bem

Da mesma forma que podemos comparar uma coluna com um valor, podemos comparar uma coluna com outra. O funcionamento é exatamente igual. Note que será comparado o valor da coluna de uma linha com o valor da outra coluna na mesma linha. Fazer comparações com conteúdos de linhas diferentes é possível, mas será analisado no próximo capítulo.

Sempre que fazemos esse tipo de comparação, devemos obedecer ao tipo de dado que estamos comparando. Se estão sendo comparados valores numéricos, não devemos colocar qualquer símbolo separador como cifrão ou vírgula, tampouco letras. O ponto decimal, contudo, deve ser especificado.

Comparações entre alfanuméricos devem estar entre apóstrofos e normalmente são sensíveis a letras maiúsculas e minúsculas. Assim, Tom Jobim é diferente de TOM JOBIM. Colunas do tipo data devem seguir as regras do banco de dados, via conversão de tipo de dados ou utilizando apóstrofos.

Operadores lógicos

Muitas vezes, apenas uma condição não é o suficiente para determinarmos o critério de busca. Sempre que isso ocorrer, podemos utilizar operadores lógicos. Veja na tabela abaixo os operadores lógicos:

Operador	Significado	Exemplo
AND	e	condição-1 AND condição-2
OR	ou	condição-1 OR condição-2
NOT ou !	não/negação	NOT condição

AND

O operador AND indica que as duas condições devem ser verdadeiras para que seja mostrada a linha. Exemplo:

```
SELECT  NOME_CD, PRECO_VENDA, CODIGO_GRAVADORA
    FROM  CD
    WHERE  PRECO_VENDA > 10 AND CODIGO_GRAVADORA = 2;

NOME_CD                  PRECO_VENDA    CODIGO_GRAVADORA
-----------------------  -----------    ----------------
Bate-Boca                12             2
A Força que nunca Seca   13,5           2
```

Somente serão mostrados os nomes dos CDs que tiverem preço de venda maior que 10 e cuja gravadora seja a de código 2.

A seguir, uma tabela com as possibilidades que envolvem o operador AND. Nas extremidades está o retorno das condições e na interseção de linhas e colunas, a avaliação do resultado final.

AND	Verdadeiro	Falso
Verdadeiro	Verdadeiro	Falso
Falso	Falso	Falso

OR

Utilizamos o operador OR quando quisermos que o resultado final seja verdadeiro sempre que uma das duas condições forem verdadeiras (ou ambas).

Exemplo:

```
SELECT NOME_CD, PRECO_VENDA, CODIGO_GRAVADORA
    FROM CD
    WHERE PRECO_VENDA > 11 OR CODIGO_GRAVADORA = 3;
```

NOME_CD	PRECO_VENDA	CODIGO_GRAVADORA
Bate-Boca	12	2
Elis Regina - Essa Mulher	13	3
A Força que nunca Seca	13,5	2
Perfil	10,5	3
Mais do Mesmo	15	1

Somente serão mostrados os nomes dos CDs que tiverem preço de venda maior que 11 ou cuja gravadora seja a de código 3. Assim, você pode notar que o CD Perfil, mesmo custando 10,5, aparece na listagem pois o código da gravadora é 3.

A seguir, você observa uma tabela com as possibilidades que envolvem o operador OR. Nas extremidades está o retorno das condições e na interseção de linhas e colunas, a avaliação do resultado final.

OR	Verdadeiro	Falso
Verdadeiro	Verdadeiro	Verdadeiro
Falso	Verdadeiro	Falso

Cuidado

Não há limitação no uso e na combinação de condições utilizando operadores AND e OR. Mas devemos tomar cuidado na combinação de ambos. A avaliação desse tipo de condição é, no padrão, da esquerda para a direita. É conveniente utilizar parênteses para determinar o que você quer comparar. Assim, o resultado dos dois comandos a seguir pode não ser exatamente o que você está esperando:

```
SELECT  NOME_CD, CODIGO_GRAVADORA, PRECO_VENDA
    FROM  cd
    WHERE  CODIGO_GRAVADORA = 2 OR CODIGO_GRAVADORA = 3
    AND PRECO_VENDA > 11;
```

```
NOME_CD                      CODIGO_GRAVADORA   PRECO_VENDA
---------------------------  ----------------   -----------
Elis Regina - Essa Mulher    3                  13
Bate-Boca                    2                  12
A Força que nunca Seca       2                  13,5
Barry Manilow Greatest Hits  2                  9,5
Listen Without Prejudice     2                  9
```

Veja que os CDs que aparecem nessa listagem incluem dois com preço de venda inferior a 11. Isso acontece porque primeiro é verificado se o código da gravadora é 2 ou 3 e, somente depois, verifica-se o preço de venda. Veja a diferença no resultado da consulta a seguir:

```
SELECT  NOME_CD, CODIGO_GRAVADORA, PRECO_VENDA
    FROM  cd
    WHERE (CODIGO_GRAVADORA = 2 OR CODIGO_GRAVADORA = 3)
    AND PRECO_VENDA > 11;
```

```
NOME_CD                    CODIGO_GRAVADORA   PRECO_VENDA
-------------------------  ----------------   -----------
Elis Regina - Essa Mulher  3                  13
Bate-Boca                  2                  12
A Força que nunca Seca     2                  13,5
```

Nesse caso, checa-se o código da gravadora e obtém-se uma lista de linhas válidas (somente os das gravadoras 2 e 3). Depois, verifica-se o preço de venda, apenas as linhas selecionadas anteriormente, e, assim, temos o que deve ser apresentado.

NOT ou !

É utilizado para inverter o resultado de uma expressão lógica, negando o resultado da condição. Caso a condição seja verdadeira, será retornado falso e vice-versa.

```
SELECT NOME_CD, PRECO_VENDA
    FROM cd
    WHERE NOT ( PRECO_VENDA < 15 );
```

```
NOME_CD        PRECO_VENDA
-------------  -----------
Mais do Mesmo  15
```

Aqui estão apenas as linhas da tabela CD que não tenham preço menor que 15. Primeiro o banco de dados avalia o preço de venda menor que 15 e depois exclui da lista esses elementos.

Mais uma vez, apesar de opcional, o uso de parênteses deixa mais claro o que está sendo esperado do comando. Pode-se avaliar com o NOT expressões complexas incluindo as cláusulas AND e OR.

```
SELECT  NOME_CD, CODIGO_GRAVADORA, PRECO_VENDA
    FROM  cd
    WHERE NOT (CODIGO_GRAVADORA = 2 OR CODIGO_GRAVADORA = 3) AND
    PRECO_VENDA > 11;

NOME_CD         CODIGO_GRAVADORA    PRECO_VENDA
-------------   ----------------    -----------

Mais do Mesmo   1                   15
```

Nesse caso, como é excluído o grupo de CDs das gravadoras 2 e 3, pois utilizamos o NOT antes do parênteses, resta apenas o CD Mais do Mesmo com preço superior a 11. Se utilizarmos NOT em ambas as condições, teremos:

```
SELECT  NOME_CD, CODIGO_GRAVADORA, PRECO_VENDA
    FROM cd
    WHERE NOT (CODIGO_GRAVADORA = 2 OR CODIGO_GRAVADORA = 3)
    AND NOT PRECO_VENDA > 11;
```

Nenhuma linha é retornada, uma vez que somente há um CD que não é da gravadora 2 ou 3, mas o preço do CD é superior a 11.

Operadores especiais

Há alguns operadores que são utilizados para determinar melhor as linhas que queremos filtrar. São eles IS NULL, IS NOT NULL, BETWEEN, LIKE e IN.

IS NULL

Sabemos que todas as colunas que não têm valor inicializado são nulas em banco de dados SQL. Logo, esse comando é utilizado para saber se o conteúdo da coluna foi ou não inicializado. Exemplo:

```
SELECT  NOME_GRAVADORA, ENDERECO
    FROM  GRAVADORA
    WHERE  ENDERECO IS NULL;

NOME_GRAVADORA  ENDERECO
--------------  --------

SOM LIVRE

EPIC
```

Note que a coluna ENDERECO não contém valor atribuído.

IS NOT NULL

Compara a negação do operador anterior. Somente aqueles que tiverem conteúdo atribuído serão mostrados.

```
SELECT  NOME_GRAVADORA, ENDERECO
    FROM  GRAVADORA
    WHERE  ENDERECO  IS NOT NULL;
```

```
NOME_GRAVADORA      ENDERECO
--------------      ------------------------------
BMG                 Av. Piramboia, 2898 - Parte 7
EMI                 Rod. Dutra km 229,8
```

BETWEEN

Esse operador serve para determinar um intervalo de busca. Assim, sempre que quisermos realizar buscas que indiquem um intervalo de números, datas, etc., podemos utilizar o BETWEEN para simplificar a forma de escrevermos o comando. É muito utilizado para simplificar a utilização do AND.

```
SELECT  NOME_CD, DATA_LANCAMENTO
    FROM  CD
    WHERE  DATA_LANCAMENTO  BETWEEN '01/01/1999' AND '01/12/
    2001';
```

```
NOME_CD         DATA_LANCAMENTO
---------       ---------------
Bate-Boca       01/07/1999
Perfil          01/05/2001
```

```
SELECT  NOME_CD, PRECO_VENDA
    FROM  CD
    WHERE  PRECO_VENDA  BETWEEN 9 AND 11;
```

```
NOME_CD                      PRECO_VENDA
---------------------------  -----------
Perfil                       10,5
Barry Manilow Greatest Hits  9,5
Listen Without Prejudice     9
```

Caso você utilize NOT, somente as linhas fora do intervalo serão apresentadas.

```
SELECT  NOME_CD, PRECO_VENDA
    FROM  CD
    WHERE  PRECO_VENDA  NOT  BETWEEN 9 AND 11;
```

```
NOME_CD                    PRECO_VENDA
-------------------------  -----------
Bate-Boca                  12
Elis Regina - Essa Mulher  13
A Força que nunca Seca     13,5
Mais do Mesmo              15
```

LIKE

Com este operador, podemos comparar cadeias de caracteres utilizando padrões de comparação (*wildcard*) para um ou mais caracteres. Normalmente, o caracter percentual (%) substitui zero, um ou mais caracteres e sublinha (_) substitui um caractere.

Utilizando a combinação desses caracteres especiais com o que se quer localizar, pode-se conseguir uma variedade muito grande de expressões. Veja na tabela a seguir algumas possíveis combinações:

Expressão	Explicação
LIKE 'A%'	Todas as palavras que iniciem com a letra A.
LIKE '%A'	Todas que terminem com a letra A.
LIKE '%A%'	Todas que tenham a letra A em qualquer posição.
LIKE 'A_'	String de dois caracteres que tenham a primeira letra A e o segundo caractere seja qualquer outro.
LIKE '_A'	String de dois caracteres cujo primeiro caractere seja qualquer um e a última letra seja A.
LIKE '_A_'	String de três caracteres cuja segunda letra seja A, independentemente do primeiro ou do último caractere.
LIKE '%A_'	Todos que tenham a letra A na penúltima posição e a última seja qualquer outro caractere.
LIKE '_A%'	Todos que tenham a letra A na segunda posição e o primeiro caractere seja qualquer um.

No exemplo seguinte, procuraremos todos os autores cujos nomes sejam iniciados com a letra R:

```
SELECT  *  FROM  AUTOR
    WHERE  NOME_AUTOR LIKE 'R%';

CODIGO_AUTOR    NOME_AUTOR
------------    ---------------
30              Roberto Mendes
37              Roberto Carlos
39              Renato Teixeira
44              Renato Barros
53              Ronaldo Bastos
55              Richard Kerr
1               Renato Russo
```

Nessa busca serão localizados todos os autores que tenham a letra L na segunda posição da palavra. Note que utilizamos a letra minúscula porque está dentro das aspas simples e, portanto, letras maiúscula e minúscula são diferentes.

```
SELECT  *  FROM  AUTOR
    WHERE  NOME_AUTOR  LIKE  '_l%';

CODIGO_AUTOR    NOME_AUTOR
------------    ---------------
11              Aldir Blanc
15              Cláudio Tolomei
60              Elton John
```

Neste próximo exemplo, buscaremos todos os autores com a letra C no início e a letra R na terceira posição do nome.

```
SELECT  *  FROM  AUTOR
    WHERE  NOME_AUTOR  LIKE 'C_r%'

CODIGO_AUTOR    NOME_AUTOR
------------    ----------------
14              Cartola
28              Carlinhos Brown
56              Chris Arnold
```

Neste próximo exemplo, utilizaremos o LIKE com números. Apenas os preços que comecem com o algarismo 1 e tenham apenas um algarismo após o 1 serão retornados.

```
SELECT  CODIGO_CD, NOME_CD, PRECO_VENDA
    FROM  CD
    WHERE  PRECO_VENDA  LIKE '1_';

CODIGO_CD   NOME_CD                    PRECO_VENDA
--------    -------------------------  -----------
2           Bate-Boca                  12
3           Elis Regina - Essa Mulher  13
1           Mais do Mesmo              15
```

Um problema que pode surgir quando queremos fazer buscas utilizando os caracteres de substituição é tê-los na cadeia de caracteres que está sendo pesquisada. Nesse caso, devemos "avisar" o banco de dados utilizando um caracter especial denominado escape. Dessa forma, o comando a seguir seria válido e localizaria qualquer CD que tivesse no nome o caracter sublinha (_).

```
SELECT  *  FROM CD
    WHERE  NOME_CD  LIKE '%\_%'  ESCAPE '\';
```

IN

Permite comparar o valor de uma coluna com um conjunto de valores. Normalmente, utilizamos o IN para substituir uma série de comparações seguidas da cláusula OR. Note que apenas os autores cujo código seja 1, 10 ou 20 serão retornados.

```
SELECT * FROM AUTOR
    WHERE CODIGO_AUTOR IN (1,10,20);

CODIGO_AUTOR    NOME_AUTOR
------------    ----------
20              Tunai
10              João Bosco
1               Renato Russo
```

A maior utilização do IN é, contudo, quando o utilizamos em *subquery*. Essa aplicação será vista no capítulo 7.

Exercícios propostos

1. Liste todos os campos e linhas da tabela BAIRRO.

2. Liste todas as linhas e os campos CDCOMPRADOR, NMCOMPRADOR e EMAIL da tabela COMPRADOR.

3. Liste todas as linhas e os campos CDVENDEDOR, NMVENDEDOR e EMAIL da tabela VENDEDOR em ordem alfabética.

4. Repita o comando anterior em ordem alfabética decrescente.

5. Liste todos os bairros do Estado de SP.

6. Liste as colunas CDIMOVEL, CDVENDEDOR e VLPRECO de todos os imóveis do vendedor 2.

7. Liste as colunas CDIMOVEL, CDVENDEDOR, VLPRECO e SGESTADO dos imóveis cujo preço de venda seja inferior a 150 mil e sejam do Estado do RJ.

8. Liste as colunas CDIMOVEL, CDVENDEDOR, VLPRECO e SGESTADO dos imóveis cujo preço de venda seja inferior a 150 mil ou seja do vendedor 1.

9. Liste as colunas CDIMOVEL, CDVENDEDOR, VLPRECO e SGESTADO dos imóveis cujo preço de venda seja inferior a 150 mil e o vendedor não seja 2.

10. Liste as colunas CDCOMPRADOR, NMCOMPRADOR, NMENDERECO e SGESTADO da tabela COMPRADOR em que o Estado seja nulo.

11. Liste as colunas CDCOMPRADOR, NMCOMPRADOR, NMENDERECO e SGESTADO da tabela COMPRADOR em que o Estado não seja nulo.

12. Liste todas as ofertas cujo valor esteja entre 100 mil e 150 mil.

13. Liste todas as ofertas cuja data da oferta esteja entre '01/02/02' e '01/03/02'.

14. Liste todos os vendedores que comecem com a letra M.

15. Liste todos os vendedores que tenham a letra A na segunda posição do nome.

16. Liste todos os compradores que tenham a letra U em qualquer posição do endereço.

17. Liste todas as ofertas cujo imóvel seja 1 ou 2.

18. Liste todos os imóveis cujo código seja 2 ou 3 em ordem alfabética de endereço.

19. Liste todas as ofertas cujo imóvel seja 2 ou 3 e o valor da oferta seja maior que 140 mil, em ordem decrescente de data.

20. Liste todos os imóveis cujo preço de venda esteja entre 110 mil e 200 mil ou seja do vendedor 1 em ordem de área útil.

7

Cálculos e funções usuais

Outra utilização bastante importante de SQL é fazer cálculos e totalizações de valores unitários. Neste capítulo será verificado como realizar esses cálculos e totalizações, além da utilização de algumas funções de manipulação de caracteres.

Cálculos

Podemos fazer cálculos quando realizamos buscas no banco de dados, simplesmente aplicando um dos operadores aritméticos a colunas. Dessa forma, mesmo não tendo armazenado o valor total (preço unitário multiplicado pela quantidade) em um item de pedido, é possível realizar este cálculo. Como exemplo, será verificado o preço dos CDs caso estes sofram um aumento de 5%. Veja que temos uma coluna com Preço de Venda e outra com Preço de Venda multiplicado por 1.05. Os dados da tabela não foram alterados.

```
SELECT  CODIGO_CD, NOME_CD, PRECO_VENDA, PRECO_VENDA * 1.05
    FROM  CD;
```

CODIGO_CD	NOME_CD	PRECO_VENDA	PRECO_VENDA*1.05
2	Bate-Boca	12	12,6
3	Elis Regina - Essa Mulher	13	13,65
4	A Força que nunca Seca	13,5	14,175
5	Perfil	10,5	11,025
6	Barry Manilow Greatest Hits	9,5	9,975
7	Listen Without Prejudice	9	9,45
1	Mais do Mesmo	15	15,75

Os operadores aritméticos padrão para o SQL são relativamente limitados. Cada banco de dados, por sua vez, contém funções que minimizam essa limitação. Por exemplo: a função POWER faz parte das funções do Oracle e Sybase e permite calcular exponenciação.

Estes são os operadores utilizados no SQL:

Operador	Ação
+	Soma.
-	Subtração.
*	Multiplicação.
/	Divisão.

A precedência dos operadores é igual à da matemática, ou seja, multiplicação e divisão têm prioridade sobre soma e subtração. Se houver duas ou mais operações do mesmo grupo (multiplicação e divisão ou adição e subtração), a operação será realizada na ordem em que aparecer. Dentro do mesmo grupo não há prioridade. Para alterar a prioridade, devemos utilizar parênteses.

Dessa forma, o cálculo abaixo tem como resposta 9:

```
15 / 5 * 3
```

O resultado seria 1 se fosse escrito como a seguir:

```
15 / (5 * 3)
```

Numéricos

A seguir, algumas funções que retornam valores numéricos.

POSITION / INSTR

O objetivo de ambas as funções é retornar a posição do caractere de busca na cadeia de caracteres origem. Dependendo do banco de dados, encontraremos uma ou outra função.

```
POSITION( destino IN origem )
INSTR( origem, destino [, início, [fim]])
```

Tipo	Descrição
destino	Cadeia de caracteres em que se quer pesquisar.
origem	Cadeia de caracteres em que será realizada a pesquisa.
início	Posição inicial de busca (se não especificada inicia na posição 1).
fim	Ocorrência que deve ser pesquisada (se não especificada, indica que é a primeira ocorrência).

```
SELECT POSITION('Ru' IN 'Renato Russo' ) BUSCA FROM DUAL;

SELECT INSTR('Renato Russo','Ru' ) BUSCA FROM DUAL;

BUSCA
-----
8
```

CHARACTER_LENGTH / LENGTH

Retorna o número de caracteres contidos em uma cadeia de caracteres.

```
CHARACTER_LENGTH( 'cadeia_caracteres' );
LENGTH( 'cadeia_caracteres' );
```

```
SELECT CHARACTER_LENGTH( 'Renato Russo' ) TAM FROM DUAL;

SELECT LENGTH( 'Renato Russo' ) TAM FROM DUAL;

TAM
----
12
```

Alfanuméricos

O padrão SQL indica o uso de um operador para concatenar alfanuméricos e de algumas funções para manipulá-los.

Para concatenar cadeias de caracteres, utilizamos dois *pipes* (||). Dessa forma, se quisermos verificar o nome da gravadora seguido da pessoa de contato, utilizaremos o seguinte comando:

```
SELECT  NOME_GRAVADORA || ' - ' || CONTATO
    FROM  GRAVADORA;

NOME_GRAVADORA||'-'||CONTATO
----------------------------

BMG - MARIA

SOM LIVRE - MARTA

EPIC - PAULO

EMI - JOAO
```

UPPER E LOWER

Quando realizamos buscas alfanuméricas no banco de dados, devemos notar que o conteúdo do campo será comparado, literalmente, com a cadeia de caracteres informada. Isso que dizer que caso o que seja informado na busca seja composto por letras maiúsculas e o que está armazenado no banco de dados esteja com a primeira letra maiúscula e as demais minúsculas, não será retornada nenhuma linha. Veja o exemplo a seguir:

```
SELECT  *  FROM  AUTOR
    WHERE  NOME_AUTOR  LIKE 'ROBERTO%';
```

Esse comando não retorna nenhuma linha. Já utilizando a função UPPER no nome do autor, teremos:

```
SELECT  *  FROM AUTOR
    WHERE  UPPER( NOME_AUTOR )  LIKE 'ROBERTO%';

CODIGO_AUTOR    NOME_AUTOR
------------    ---------------

30              Roberto Mendes

37              Roberto Carlos
```

Conforme podemos notar, a função fez com que todos os conteúdos da coluna NOME_AUTOR fossem convertidos para caracteres maiúsculos (UPPER) e depois comparados com ROBERTO%. Isso é particularmente útil quando não sabemos exatamente como os usuários digitaram o conteúdo das colunas. O comando LOWER utilizado dessa forma produz o mesmo efeito, apenas transformando em minúsculos os caracteres a serem comparados.

Esses comandos são determinados apenas para banco de dados que sejam completamente compatíveis com o SQL-92.

TRIM

Também determinado para banco de dados compatíveis com o SQL-92, o comando TRIM serve para retirar caracteres antes e/ou depois de outra cadeia de caracteres.

Sintaxe:

```
TRIM( [{BOTH | LEADING | TRAILING} car FROM] cadeia )
```

Exemplos de utilização:

Exemplo	Resultado	Explicação
TRIM(' teste ')	'teste'	Este é o padrão. Retira espaços antes e depois da palavra teste.
TRIM(BOTH '#' FROM '#teste#')	'teste'	Retira o símbolo # que estava antes e depois (BOTH) da cadeira de caracteres.
TRIM(LEADING ' ' FROM ' teste '	'teste '	Retira o espaço que estava antes da palavra teste.
TRIM(TRAILING ' ' FROM ' teste '	' teste'	Retira o espaço que estava após a palavra teste.

Não esquecer: podemos utilizar a função TRIM em qualquer cadeia de caracteres, mesmo que esta contenha números ou símbolos.

SUBSTRING

Esta função é requisito para SQL-92 no nível intermediário. Retorna uma parte de uma cadeia de caracteres. Sintaxe:

```
SUBSTRING( cadeia_origem FROM posição_inicio FOR número_caracteres )
```

Exemplo de utilização:

```
SELECT  SUBSTRING( NOME_AUTOR FROM 1 FOR 3 ) RES, NOME_AUTOR
    FROM  AUTOR
    WHERE  NOME_AUTOR LIKE 'R%'

RES     NOME_AUTOR
---     ---------------
Ren     Renato Russo
Rob     Roberto Mendes
Rob     Roberto Carlos
Ren     Renato Teixeira
Ren     Renato Barros
Ron     Ronaldo Bastos
Ric     Richard Kerr
```

No Oracle, para chegar ao mesmo efeito, devemos utilizar o comando:

```
SELECT  SUBSTR( NOME_AUTOR,1,3 ),NOME_AUTOR
    FROM  AUTOR
    WHERE  NOME_AUTOR LIKE 'R%'
```

Nesse caso, o comando é o SUBSTR seguido da cadeia de caracteres (NOME_AUTOR), da posição de início (1) e do número de caracteres (3) de retorno.

CONVERT

Esta função serve para traduzir uma cadeia de caracteres de um conjunto de caracteres para outro. Veja que isso não quer dizer que deve haver uma tradução literal da cadeia de caracteres. Apenas os conjuntos de caracteres utilizados pelo seu banco de dados serão trocados. Consulte a documentação do seu banco de dados para saber quais conjuntos de caracteres são suportados e, dentre os que são suportados, quais admitem a conversão da cadeia de caracteres. Sintaxe:

```
CONVERT( 'cadeia_caracteres','conjunto_destino'['conjunto_origem'])
```

TRANSLATE

Serve para pesquisar e substituir caracteres em uma cadeia de caracteres.

```
TRANSLATE( caracteres, 'pesquisa','substituição' )
```

Os caracteres que estiverem em *pesquisa* serão substituídos, na mesma seqüência, pelos especificados em *substituição*.

No exemplo a seguir, as letras R serão substituídas pela letra A e as letras T serão substituídas pela letra B.

```
SELECT TRANSLATE( UPPER(NOME_AUTOR),'RT','AB' )
    FROM AUTOR
    WHERE CODIGO_AUTOR = 1;

TRANSLATE(UPPER(NOME_AUTOR),'RT','AB')
--------------------------------------

AENABO AUSSO
```

REPLACE

Substitui uma cadeia de caracteres por outra. A diferença entre essa função e TRANSLATE é que somente será substituída a cadeia de caracteres inteira. Dessa forma, se não houver uma ocorrência de toda a cadeia, nada será substituído.

```
REPLACE( caracteres, 'pesquisa', 'substituição' )
```

Veja os exemplos a seguir. No primeiro exemplo nada é modificado, pois não existe uma ocorrência de RT na cadeia de caracteres. Já no segundo, a substituição de RE é realizada:

```
SELECT REPLACE( UPPER(NOME_AUTOR),'RT','AB' )
    FROM AUTOR
    WHERE CODIGO_AUTOR = 1;

REPLACE(UPPER(NOME_AUTOR),'RT','AB')
------------------------------------

RENATO RUSSO

SELECT Replace(  UPPER(NOME_AUTOR),'RE','AB' )
    FROM AUTOR
    WHERE CODIGO_AUTOR = 1;

REPLACE(UPPER(NOME_AUTOR),'RE','AB')
------------------------------------

ABNATO RUSSO
```

Manipulação de datas

Quando criamos colunas com tipo de dado Data, podemos realizar uma série de cálculos e operações cronológicas. Podemos calcular o número de dias entre duas datas, somar e subtrair dias, meses etc.

O padrão SQL especifica quatro tipos de dados relacionados a data e hora:

Tipo	Descrição
DATE	Apenas Data.
TIME	Apenas Hora.
TIMESTAMP	Data e Hora.
INTERVAL	Intervalo entre os dois tipos de dados anteriores.

O padrão SQL definiu algumas funções para determinar a data e/ou hora atuais do sistema, acrescentando CURRENT_ antes dos três primeiros tipos anteriores: CURRENT_DATE, CURRENT_TIME e CURRENT_TIMESTAMP. Em alguns bancos de dados são utilizadas as mesmas funções, sem o (_).

Para saber quais os CDs lançados no dia de hoje, execute:

```
SELECT * FROM CD
    WHERE DATA_LANCAMENTO = CURRENT_DATE;
```

Operações aritméticas com datas

Uma coluna do tipo data é composta de seis elementos:

a. YEAR (ano).

b. MONTH (mês).

c. DAY (dia).

d. HOUR (hora).

e. MINUTE (minuto).

f. SECOND (segundo).

Na tabela a seguir, podemos visualizar quais os resultados para as operações com data:

Expressão	Resultado
DATETIME – DATETIME	INTERVAL
DATETIME ± INTERVAL	DATETIME
INTERVAL ± INTERVAL	INTERVAL
INTERVAL * número	INTERVAL
INTERVAL / número	INTERVAL

Quando escrever uma expressão que inclua INTERVAL, especifique em qual tipo de intervalo se quer visualizar a resposta (YEAR, MONTH, DAY, etc) e até indicar o período de tempo (YEAR TO MONTH ou DAY TO SECOND). Alguns bancos de dados permitem até que se façam mais variações nesses intervalos, desde que o primeiro seja sempre maior que o segundo (como no caso de YEAR - maior - e MONTH - menor). Quando quiser realizar cálculos utilizando INTERVAL, deve-se colocar a palavra-chave INTERVAL seguida do intervalo entre aspas simples e o tipo de intervalo (veja exemplos a seguir).

A manipulação de datas e os cálculos que podem ser feitos dependem e variam muito em função do banco de dados utilizado. Convém consultar a documentação do seu banco de dados para certificar-se de como utilizá-la.

Dessa forma, é possível realizar operações como encontrar o intervalo de dias entre duas datas:

```
SELECT CURRENT_DATE, DATA_LANCAMENTO,
    CURRENT_DATE - DATA_LANCAMENTO  DIFERENCA
    FROM  CD;

CURRENT_DATE    DATA_LANCAMENTO     DIFERENCA
------------    ---------------     ---------
11/02/02        01/10/98            1229
11/02/02        01/07/99            956
11/02/02        01/05/89            4669
11/02/02        01/12/98            1168
11/02/02        01/05/01            286
11/02/02        01/11/91            3755
11/02/02        01/10/91            3786
```

Adicionar dias em uma data (nos exemplos a seguir, acrescentam-se 7 dias da data de lançamento do CD):

```
SELECT  DATA_LANCAMENTO + 7  FROM  CD;

DATA_LAN
--------
08/10/98
08/07/99
08/05/89
08/12/98
08/05/01
08/11/91
08/10/91


SELECT  DATA_LANCAMENTO + INTERVAL '7' DAY  FROM CD;

DATA_LAN
--------
08/10/98
08/07/99
08/05/89
08/12/98
08/05/01
08/11/91
08/10/91
```

Subtrair meses de uma data (no exemplo a seguir, subtraem-se 2 meses da data de lançamento do CD):

```
SELECT  DATA_LANCAMENTO - INTERVAL '2' MONTH
    FROM  CD;

DATA_LAN
--------
01/08/98
01/05/99
01/03/89
01/10/98
01/03/01
01/09/91
01/08/91
```

EXTRACT

Essa função extrai e retorna um valor de um campo do tipo data. É possível extrair apenas o dia, o mês, o ano, a hora, o minuto ou o segundo.

Veja exemplos:

```
SELECT EXTRACT( MONTH FROM DATE '2002-05-01' )
    FROM DUAL;

EXTRACT(MONTHFROMDATE'2002-05-01')

----------------------------------
                                 5

SELECT EXTRACT( YEAR FROM DATE '2002-05-01' )
    FROM DUAL;

EXTRACT(YEARFROMDATE'2002-05-01')
---------------------------------
                              2002

SELECT EXTRACT( DAY FROM DATE '2002-05-01' )
    FROM DUAL;

EXTRACT(DAYFROMDATE'2002-05-01')

--------------------------------
                               1
```

Exercícios propostos

1. Escreva uma busca que mostre a data atual.

2. Escreva uma busca que mostre CDIMOVEL, VLPRECO e VLPRECO com 10% de aumento.

3. Escreva uma busca igual à anterior, porém acrescente uma coluna mostrando a diferença entre VLPRECO e VLPRECO com 10% de aumento.

4. Escreva uma busca que mostre o NMVENDEDOR em letras maiúsculas e EMAIL em letras minúsculas.

5. Escreva uma busca que mostre o NMCOMPRADOR e NMCIDADE em uma única coluna e separados por um hífen.

6. Escreva uma busca que mostre todos os compradores que tenham a letra A no nome.

7. Escreva uma busca que mostre a primeira letra do nome dos compradores e o NMBAIRRO.

8. Escreva uma busca que mostre o CDIMOVEL e o número de dias entre a data atual do sistema e DTOFERTA.

9. Escreva uma busca que mostre o CDIMOVEL, a DTLANCTO e o número de dias entre a data atual do sistema e DTLANCTO.

10. Escreva uma busca igual à anterior e mostre uma coluna com 15 dias após DTLANCTO.

8

Pesquisa em múltiplas tabelas

Até agora vimos como realizar pesquisas em uma única tabela. Contudo, na montagem do nosso Modelo de Dados sempre temos diversas tabelas. Logo é necessário sabermos como vincular a informação dessas tabelas de forma a mostrar a informação de maneira correta. A isto é dado o nome de união de tabelas (*join*).

Como visto anteriormente, a união entre as Entidades do nosso Modelo Lógico se dá por meio de chaves primárias e estrangeiras. Essas chaves são, na representação Física do Modelo, as colunas que as tabelas têm em comum. No decorrer deste capítulo, apresentaremos as diversas formas de unir colunas e como implementá-las em SQL.

União de tabelas

Para realizar a união de tabelas, basta acrescentarmos após a cláusula FROM do comando SELECT as tabelas que queremos unir. Devemos colocar na cláusula WHERE a condição de união das tabelas, ou seja, as respectivas chaves primária e estrangeira. Sintaxe:

```
SELECT [tabela1.]coluna [, [tabela2.]coluna, ... ]
   FROM tabela1, tabela 2 [, ...]
   WHERE tabela1.chave_primária = tabela2.chave_estrangeira
```

Note que é opcional colocar a identificação da tabela antes do nome das colunas na lista de campos do comando SELECT. Contudo, essa é uma prática recomendada para facilitar o entendimento do comando.

Essa regra só não é válida quando estivermos querendo mostrar uma coluna que tenha o mesmo nome em ambas as tabelas - normalmente o que acontece com as chaves primária e estrangeira. Nesse caso é necessário indicar de qual tabela se quer a informação - ainda que, de fato, o dado seja igual nas duas tabelas. Considere que na maior parte dos bancos de dados, informar em qual tabela está a coluna facilita o trabalho do banco de dados. Essa prática leva a maior agilidade na recuperação da informação.

É possível colocar diversas tabelas na cláusula FROM. Não devemos esquecer, porém, que é necessário especificar as chaves primária e estrangeira de cada relacionamento entre as tabelas.

Em caso de dúvida, consulte o Modelo de Dados para estabelecer quais são as colunas comuns entre as tabelas. A cláusula WHERE deve conter obrigatoriamente todo o canal de relacionamento entre as tabelas listadas na cláusula FROM. Se isso não acontecer, ocorrerá o produto cartesiano, conforme será mostrado a seguir.

Produto cartesiano

Ocorrerá um produto cartesiano sempre que:

- A condição de união entre as tabelas for omitida (não houver cláusula WHERE).
- A condição de união entre as tabelas for inválida (cláusula WHERE incorreta).
- Todas as linhas da primeira tabela estiverem unidas a todas as linhas da segunda tabela.

Nessa situação, as linhas da primeira tabela serão combinadas com as linhas da segunda tabela, demonstrando um resultado na maior parte das vezes indesejado. Em raríssimas situações - como quando se quer gerar uma quantidade muito grande de dados para efetuar testes de *performance*, por exemplo - isso é utilizado.

Veja o que acontece quando ocorre um produto cartesiano:

```
SELECT  CD.CODIGO_CD, CD.NOME_CD, GRAVADORA.NOME_GRAVADORA
    FROM  CD, GRAVADORA;
```

CODIGO_CD	NOME_CD	NOME_GRAVADORA
1	Mais do Mesmo	EMI
2	Bate-Boca	EMI
3	Elis Regina - Essa Mulher	EMI
4	A Força que nunca Seca	EMI
5	Perfil	EMI
6	Barry Manilow Greatest Hits	EMI
7	Listen Without Prejudice	EMI
1	Mais do Mesmo	BMG
2	Bate-Boca	BMG
3	Elis Regina - Essa Mulher	BMG
4	A Força que nunca Seca	BMG
5	Perfil	BMG
6	Barry Manilow Greatest Hits	BMG
7	Listen Without Prejudice	BMG
1	Mais do Mesmo	SOM LIVRE
2	Bate-Boca	SOM LIVRE
3	Elis Regina - Essa Mulher	SOM LIVRE
4	A Força que nunca Seca	SOM LIVRE
5	Perfil	SOM LIVRE
6	Barry Manilow Greatest Hits	SOM LIVRE
7	Listen Without Prejudice	SOM LIVRE
1	Mais do Mesmo	EPIC
2	Bate-Boca	EPIC
3	Elis Regina - Essa Mulher	EPIC
4	A Força que nunca Seca	EPIC
5	Perfil	EPIC
6	Barry Manilow Greatest Hits	EPIC
7	Listen Without Prejudice	EPIC

Como existem 4 linhas em Gravadora e 7 em CD, chega-se a um total de 28 linhas (7x4) na nossa busca. Como se nota, existe uma linha de CD para cada Gravadora. Naturalmente essa é uma informação incorreta. Pode-se querer mostrar qual é a gravadora de cada CD. Para isso, acrescente ao comando a cláusula WHERE, fazendo uma união regular entre as tabelas.

União regular (*inner join* ou *equi-join*)

Denomina-se união regular as uniões que têm a cláusula WHERE unindo a chave primária à estrangeira das tabelas afetadas pelo comando SELECT. Exemplo:

```
SELECT CD.CODIGO_CD, CD.NOME_CD, GRAVADORA.NOME_GRAVADORA
    FROM CD, GRAVADORA
    WHERE CD.CODIGO_GRAVADORA = GRAVADORA.CODIGO_GRAVADORA;
```

CODIGO_CD	NOME_CD	NOME_GRAVADORA
1	Mais do Mesmo	EMI
2	Bate-Boca	BMG
4	A Força que nunca Seca	BMG
6	Barry Manilow Greatest Hits	BMG
7	Listen Without Prejudice	BMG
3	Elis Regina - Essa Mulher	SOM LIVRE
5	Perfil	SOM LIVRE

Existem apenas sete linhas (o maior número de linhas entre as duas tabelas relacionadas - CD) mostrando apenas o nome da gravadora de cada CD. Essa é uma informação válida e correta.

O padrão SQL determina uma sintaxe alternativa para esse comando. Quando a chave primária e a chave estrangeira têm o mesmo nome em ambas as tabelas, é possível simplificar o comando utilizando:

```
SELECT  CD.CODIGO_CD, CD.NOME_CD, GRAVADORA.NOME_GRAVADORA
    FROM  CD  NATURAL JOIN  GRAVADORA;
```

Outras duas maneiras definidas no padrão SQL para realizar esse tipo de união de tabelas é determinar qual(is) coluna(s) utilizar na união (cláusula USING) e, caso o nome das colunas não seja igual determinar quais são as colunas com a cláusula ON. Todos os comandos produzem o mesmo efeito. Exemplos:

```
SELECT  CD.CODIGO_CD, CD.NOME_CD, GRAVADORA.NOME_GRAVADORA
    FROM  CD  JOIN  GRAVADORA  USING (CODIGO_GRAVADORA);

SELECT  CD.CODIGO_CD, CD.NOME_CD, GRAVADORA.NOME_GRAVADORA
    FROM  CD  JOIN GRAVADORA
    ON  CD.CODIGO_GRAVADORA = GRAVADORA.CODIGO_GRAVADORA;
```

Apelidos em tabelas

Para evitar que o comando fique extremamente extenso, é possível atribuir apelidos às tabelas utilizadas no comando SELECT. Devemos fazer isso, colocando o apelido após o nome da tabela na cláusula FROM. Dessa forma, o seguinte comando teria o mesmo efeito do comando anterior:

```
SELECT  a.CODIGO_CD, a.NOME_CD, b.NOME_GRAVADORA
    FROM  CD a, GRAVADORA b
    WHERE  a.CODIGO_GRAVADORA = b.CODIGO_GRAVADORA;
```

União de mais de duas tabelas

Freqüentemente é necessário unir mais de duas tabelas para fornecer uma informação relevante do banco de dados. No nosso caso, suponha que se queira saber o nome das músicas, a faixa e o nome do CD em que está a música. Se você consultar nosso Modelo de Dados, verá que é necessário utilizar três tabelas para chegar a essa informação: CD, FAIXA e MUSICA. Para diminuir a quantidade de linhas, faremos com que somente os CDs com código 1 ou 2 sejam mostrados. Vejamos como fica o comando:

```
SELECT  a.NOME_CD, b.NUMERO_FAIXA, c.NOME_MUSICA
    FROM  CD a, FAIXA b, MUSICA c
    WHERE  a.CODIGO_CD  IN (1,2)
    AND a.CODIGO_CD = b.CODIGO_CD
    AND b.CODIGO_MUSICA = c.CODIGO_MUSICA;
```

NOME_CD	NUMERO_FAIXA	NOME_MUSICA
Bate-Boca	1	Meninos, Eu Vi
Bate-Boca	2	Eu Te Amo
Bate-Boca	3	Piano na Mangueira
Bate-Boca	4	A Violeira
Bate-Boca	5	Anos Dourados
Bate-Boca	6	Olha, Maria
Bate-Boca	7	Biscate
Bate-Boca	8	Retrato em Preto e Branco
Bate-Boca	9	Falando de Amor
Bate-Boca	10	Pois É
Bate-Boca	11	Noite dos Mascarados
Bate-Boca	12	Sabiá
Bate-Boca	13	Imagina
Bate-Boca	14	Bate-Boca

```
NOME_CD         NUMERO_FAIXA    NOME_MUSICA (cont.)
---------       ------------    -------------------
Mais do Mesmo   1               Será
Mais do Mesmo   2               Ainda é Cedo
Mais do Mesmo   3               Geração Coca-Cola
Mais do Mesmo   4               Eduardo e Monica
Mais do Mesmo   5               Tempo Perdido
Mais do Mesmo   6               Índios
Mais do Mesmo   7               Que País é Este
Mais do Mesmo   8               Faroeste Caboclo
Mais do Mesmo   9               Há Tempos
Mais do Mesmo   10              Pais e Filhos
Mais do Mesmo   11              Meninos e Meninas
Mais do Mesmo   12              Vento no Litoral
Mais do Mesmo   13              Perfeição
Mais do Mesmo   14              Giz
Mais do Mesmo   15              Dezesseis
Mais do Mesmo   16              Antes das Seis
```

Como se pode notar, a única coisa que fizemos foi acrescentar o relacionamento entre todas as tabelas envolvidas no comando SELECT. Fazendo isso, sempre teremos o resultado esperado e correto. Mais uma vez convém lembrar que, de acordo com o padrão SQL, o comando a seguir produziria o resultado igual:

```
SELECT  a.NOME_CD, b.NUMERO_FAIXA, c.NOME_MUSICA
    FROM  CD a  NATURAL JOIN  FAIXA b  NATURAL JOIN  MUSICA c
    WHERE  a.CODIGO_CD  IN (1,2);
```

Utilizando essa sintaxe, podemos determinar e influenciar o tipo de recuperação de informações no banco de dados. No padrão SQL, a ordem em que colocamos a seqüência das tabelas na cláusula FROM determina quais tabelas serão pesquisadas primeiro. Logo, se colocarmos as tabelas menores primeiro a sua busca ficará mais rápida. Deixe as tabelas maiores, sempre que possível, para o final da cláusula.

União de tabelas sem colunas em comum (*non-equijoin*)

Existem situações em que, mesmo não havendo um relacionamento explícito entre colunas de tabelas, há relacionamento de uma coluna com o intervalo de outras colunas em outra tabela. Esse é o caso do nosso modelo de dados. Veja a tabela CD_CATEGORIA. Nela temos a menor e a maior faixa de preços de CD dividida por categoria. Como temos o preço dos CDs na tabela CD, fica claro que podemos definir a qual categoria pertence cada um dos CDs realizando uma busca nessas tabelas.

Esse tipo de relacionamento não é definido por uma igualdade, e sim por um intervalo. Para não gerar mais de uma linha para cada CD, certifique-se de que as faixas estejam bem definidas (CD_CATEGORIA). Caso haja a possibilidade de um único CD estar em duas faixas (preço maior e de uma faixa coincidir com o preço menor da próxima faixa), ele será mostrado duas vezes.

Veja como fica o comando para checar as categorias de preço dos CDs:

```
SELECT  a.NOME_CD, a.PRECO_VENDA, b.CODIGO_CATEGORIA
    FROM  CD a, CD_CATEGORIA b
    WHERE  a.PRECO_VENDA  BETWEEN  b.MENOR_PRECO AND
    b.MAIOR_PRECO;
```

```
NOME_CD                    PRECO_VENDA    CODIGO_CATEGORIA
-------------------------  -----------    ----------------
Barry Manilow Greatest Hits  9,5          1
Listen Without Prejudice     9            1
Bate-Boca                    12           2
Perfil                       10,5         2
Mais do Mesmo                15           3
Elis Regina - Essa Mulher    13           3
A Força que nunca Seca       13,5         3
```

Poderíamos reescrever o mesmo comando utilizando os sinais >= e <= ou ainda utilizando o operador IN para determinar o intervalo de classificação dos CDs.

Uniões Externas (*outer-join*)

Quando uma linha não satisfaz a condição de união entre as tabelas, a linha não será mostrada no resultado da busca. Isso acontece porque o banco de dados, não podendo estabelecer a relação entre as colunas que estão sendo unidas na busca, coloca NULL onde o dado não existe. Veja que a gravadora de código número 4 não aparece na busca a seguir pois não há nenhum CD cadastrado com essa gravadora.

```
SELECT
    a.CODIGO_CD,a.NOME_CD,a.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM  CD a, GRAVADORA b
    WHERE  a.CODIGO_GRAVADORA = b.CODIGO_GRAVADORA;
```

CODIGO_CD	NOME_CD	CODIGO_GRAVADORA	NOME_GRAVADORA
1	Mais do Mesmo	1	EMI
2	Bate-Boca	2	BMG
4	A Força que nunca Seca	2	BMG
6	Barry Manilow Greatest Hits	2	BMG
7	Listen Without Prejudice	2	BMG
3	Elis Regina - Essa Mulher	3	SOM LIVRE
5	Perfil	3	SOM LIVRE

Define-se União Externa como aquela que inclui linhas no resultado da busca mesmo que não haja relação entre as duas tabelas que estão sendo unidas. Dessa forma, ao executarmos o comando SELECT utilizando a União Externa, deveríamos ter a EPIC na coluna NOME_GRAVADORA e o algarismo 4 em CODIGO_GRAVADORA. Existem três formas de realizar a União Externa:

União Externa à Esquerda (*Left Outer Join*)

Como o nome diz, a união pela esquerda incluirá linhas da primeira tabela na expressão de união. Note que, caso você esteja fazendo uma união com chave primária de uma tabela com a respectiva chave estrangeira na outra e deixar a chave estrangeira à esquerda, será igual fazer uma união regular (*inner join*), uma vez que não haverá linhas opcionais na tabela da esquerda (pois ali está a chave primária).

```
tabela1  LEFT OUTER JOIN  tabela2
```

```
SELECT
    a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a LEFT OUTER JOIN GRAVADORA b;
```

Alguns bancos de dados permitem utilizar a sintaxe dessa coluna, pois pressupõem a coluna de união entre as tabelas. Quando isso não ocorrer, é necessário indicar qual é essa coluna:

```
SELECT  a.CODIGO_CD,a.NOME_CD,CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM  CD a  LEFT OUTER JOIN  GRAVADORA b
    USING (CODIGO_GRAVADORA);
```

Observe que neste exemplo retiramos o apelido da tabela na coluna CODIGO_GRAVADORA, uma vez que na cláusula USING não devemos utilizar o qualificador de tabela. Caso o nome da coluna seja diferente nas tabelas, podemos utilizar a cláusula ON:

```
SELECT
    a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM  CD  a LEFT OUTER JOIN  GRAVADORA b
    ON (a.CODIGO_GRAVADORA = b.CODIGO_GRAVADORA);
```

Em todos os casos, o resultado final seria:

CODIGO_CD	NOME_CD	CODIGO_GRAVADORA	NOME_GRAVADORA
1	Mais do Mesmo	1	EMI
7	Listen Without Prejudice	2	BMG
6	Barry Manilow Greatest Hits	2	BMG
4	A Força que nunca Seca	2	BMG
2	Bate-Boca	2	BMG
5	Perfil	3	SOM LIVRE
3	Elis Regina - Essa Mulher	3	SOM LIVRE

Note que, nesse caso, a união funciona como uma União Regular, pois a chave estrangeira está à esquerda da cláusula de união (veja explicação anterior).

União Externa à Direita (*Right Outer Join*)

Realiza a união pela direita, ou seja, as linhas da segunda tabela serão incluídas na busca, mesmo sem haver coluna correspondente na primeira tabela. Da mesma forma que a união à esquerda, se você colocar a tabela da chave estrangeira à direita da busca, terá uma união regular entre as tabelas.

```
tabela1 RIGHT OUTER JOIN tabela2
```

```
SELECT a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a RIGHT OUTER JOIN GRAVADORA b;
```

ou

```
SELECT a.CODIGO_CD,a.NOME_CD,CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a RIGHT OUTER JOIN GRAVADORA b
    USING (CODIGO_GRAVADORA);
```

ou

```
SELECT a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a RIGHT OUTER JOIN GRAVADORA b
    ON (a.CODIGO_GRAVADORA = b.CODIGO_GRAVADORA);
```

Produziria o seguinte resultado:

```
CODIGO_CD  NOME_CD                      CODIGO_GRAVADORA  NOME_GRAVADORA
---------  ---------------------------  ----------------  --------------
1          Mais do Mesmo                1                 EMI
2          Bate-Boca                    2                 BMG
3          Elis Regina - Essa Mulher    3                 SOM LIVRE
4          A Força que nunca Seca       2                 BMG
5          Perfil                       3                 SOM LIVRE
6          Barry Manilow Greatest Hits  2                 BMG
7          Listen Without Prejudice     2                 BMG
                                        4                 EPIC
```

Note que, nesse caso, aparece a EPIC na última linha e sem correspondente em CODIGO_CD, NOME_CD.

União Externa Total (*Full Outer Join*)

Realiza uma união, independente de a coluna opcional estar à direita ou à esquerda. Tome cuidado ao utilizar esse tipo de união, uma vez que, na busca, serão geradas linhas que não existem em uma, outra ou em ambas as tabelas.

```
tabela1 FULL OUTER JOIN tabela2
```

```
SELECT a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a FULL OUTER JOIN GRAVADORA b;
```

ou

```
SELECT a.CODIGO_CD,a.NOME_CD,CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a FULL OUTER JOIN GRAVADORA b

 USING (CODIGO_GRAVADORA);
```

ou

```
SELECT a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a FULL OUTER JOIN GRAVADORA b
    ON (a.CODIGO_GRAVADORA = b.CODIGO_GRAVADORA);
```

Produziria o seguinte resultado:

```
CODIGO_CD   NOME_CD                    CODIGO_GRAVADORA   NOME_GRAVADORA
---------   ---------------------------   ---------------   --------------
1           Mais do Mesmo              1                  EMI
2           Bate-Boca                  2                  BMG
3           Elis Regina - Essa Mulher  3                  SOM LIVRE
4           A Força que nunca Seca     2                  BMG
5           Perfil                     3                  SOM LIVRE
6           Barry Manilow Greatest Hits 2                 BMG
7           Listen Without Prejudice   2                  BMG
                                       4                  EPIC
```

Observação: para usuários do banco de dados Oracle, pode-se utilizar, além da sintaxe do padrão SQL, uma outra forma de União Externa. Basta colocar (+) na cláusula WHERE ao lado da tabela cujo conteúdo da coluna pode ser nulo. Dessa forma, para ter o mesmo resultado anterior, seria utilizado o comando:

```
SELECT a.CODIGO_CD,a.NOME_CD,b.CODIGO_GRAVADORA,b.NOME_GRAVADORA
    FROM CD a, GRAVADORA b
    WHERE a.CODIGO_GRAVADORA (+) = b.CODIGO_GRAVADORA;
```

União de tabela com ela mesma (*self join*)

Algumas vezes é necessário unir uma tabela a ela mesma por diversas razões. Veja o caso no qual se quer saber o nome do CD que foi indicado ao fazermos uma busca na tabela CD. Como esse é um auto-relacionamento, é necessário que façamos uma busca na própria tabela, para sabermos o nome do CD que foi indicado.

Para fazer isso, coloca-se duas vezes o nome da tabela CD, mas com apelidos diferentes. A partir daí, basta colocar a cláusula de união entre as tabelas na cláusula WHERE:

```
SELECT a.CODIGO_CD,a.NOME_CD,a.CD_INDICADO,b.NOME_CD
    FROM CD a, CD b
    WHERE a.CD_INDICADO = b.CODIGO_CD;
```

CODIGO_CD	NOME_CD	CD_INDICADO	NOME_CD
1	Mais do Mesmo	5	Perfil
2	Bate-Boca	3	Elis Regina - Essa Mulher
3	Elis Regina - Essa Mulher	1	Mais do Mesmo
4	A Força que nunca Seca	1	Mais do Mesmo
5	Perfil	2	Bate-Boca
6	Barry Manilow Greatest Hits	7	Listen Without Prejudice

Exercícios propostos

1. Faça uma busca que mostre CDIMOVEL, CDVENDEDOR, NMVENDEDOR e SGESTADO.

2. Faça uma busca que mostre CDCOMPRADOR, NMCOMPRADOR, CDIMOVEL e VLOFERTA.

3. Faça uma busca que mostre CDIMOVEL,VLPRECO e NMBAIRRO, cujo código do vendedor seja 3.

4. Faça uma busca que mostre todos os imóveis que tenham ofertas cadastradas.

5. Faça uma busca que mostre todos os imóveis e ofertas mesmo que não haja ofertas cadastradas para o imóvel.

6. Faça uma busca que mostre todos os compradores e as respectivas ofertas realizadas por eles.

7. Faça a mesma busca, porém acrescentando os compradores que ainda não fizeram ofertas para os imóveis.

8. Faça uma busca que mostre todos os endereços de imóveis e os endereços dos imóveis indicados.

9. Acrescente à busca anterior o nome dos vendedores tanto do imóvel quanto do imóvel indicado.

10. Faça uma busca que mostre o endereço do imóvel, o bairro e o nível de preço do imóvel.

9

Funções de grupo e agrupamentos

Funções de Grupo operam conjuntos de linhas visando a fornecer um resultado para o grupo. Até agora trabalhamos apenas com funções que tratavam apenas uma linha de cada vez. A diferença básica é que serão utilizados grupos de linhas. Esses grupos podem ser constituídos desde toda a tabela até subgrupos da tabela.

Funções de grupo

Existem diversas funções de grupo que são implementadas pelo padrão SQL. Essas funções auxiliam a computar uma variedade de medidas baseadas em valores das colunas do banco de dados. As principais funções de grupo são:

Função	Ação
COUNT	Retorna o número de linhas afetadas pelo comando.
SUM	Retorna o somatório do valor das colunas especificadas.
AVG	Retorna a média aritmética dos valores das colunas.
MIN	Retorna o menor valor da coluna de um grupo de linhas.
MAX	Retorna o maior valor da coluna de um grupo de linhas.
STDDEV	Retorna o desvio-padrão da coluna.
VARIANCE	Retorna a variância da coluna.

COUNT

Diferentemente das outras funções de grupo, o COUNT retorna o número de linhas que atende a uma determinada condição. Podemos utilizá-lo com um asterisco entre parênteses, para indicar que queremos saber a quantidade total de linhas, independentemente de haver linhas com colunas nulas ou não.

Caso queiramos saber quantas linhas existem e quais destas não têm valor nulo em determinada coluna, especificamos essa coluna entre parênteses.

Nesse exemplo aparece o total de registros na tabela Gravadora:

```
SELECT  COUNT(*) FROM GRAVADORA;

COUNT(*)
--------

4
```

Aqui o total de registros que possuem endereço:

```
SELECT  COUNT(ENDERECO) FROM GRAVADORA;

COUNT(ENDERECO)
---------------

2
```

Efeito semelhante poderíamos ter com o comando:

```
SELECT COUNT(*) FROM GRAVADORA
    WHERE ENDERECO IS NOT NULL;

COUNT(*)
--------

2
```

Uma outra forma interessante de utilizar o COUNT é acrescentando a ele uma cláusula DISTINCT. Veja o exemplo a seguir. Têm-se um total de 140 registros na tabela MUSICA_AUTOR. Se quisermos saber quantas são as músicas distintas, ou seja, sem repetições, visto que pode haver mais de um autor para uma mesma música, podemos utilizar o comando:

```
SELECT COUNT(*)
    FROM MUSICA_AUTOR;

COUNT(*)
--------

140


SELECT COUNT( DISTINCT CODIGO_MUSICA )
    FROM MUSICA_AUTOR;

COUNT(DISTINCTCODIGO_MUSICA)
----------------------------

86
```

SUM

Retorna o valor total de uma determinada coluna em um determinado grupo de linhas. Assim, se quisermos saber o total do preço de venda dos CDs, utilizamos o comando:

```
SELECT SUM( PRECO_VENDA ) FROM CD;

SUM(PRECO_VENDA)
----------------

82,5
```

Podemos realizar outros cálculos baseados na somatória ou mesmo incluir outras colunas e operações no comando. Imaginando que se queira saber como ficariam os preços após um aumento de 20%, teríamos o comando:

```
SELECT SUM( PRECO_VENDA ) * 1.2 FROM CD;

SUM(PRECO_VENDA)*1.2
--------------------

99
```

AVG

Extrai a média aritmética de um determinado grupo de linhas. Para saber o preço médio dos CDs da loja, execute o seguinte comando:

```
SELECT AVG( PRECO_VENDA ) FROM CD;

AVG(PRECO_VENDA)
----------------

11,7857143
```

MIN

Retorna o menor valor de uma coluna em um grupo de linhas. Podemos utilizá-la para colunas do tipo data ou alfanuméricas. Para saber o preço de venda do CD mais barato da loja, execute o seguinte comando:

```
SELECT MIN( PRECO_VENDA ) FROM CD;

MIN(PRECO_VENDA)
----------------

9
```

```
SELECT MIN( DATA_LANCAMENTO ) FROM CD;

MIN(DATA)
---------

01/05/89

SELECT MIN( NOME_CD ) FROM CD;

MIN(NOME_CD)
--------------------

A Força que nunca Seca
```

MAX

Retorna o maior valor de uma coluna em um grupo de linhas. Igualmente ao MIN, pode-se utilizá-la para colunas do tipo data ou alfanuméricas. Para saber qual é o CD mais caro da loja, execute o seguinte comando:

```
SELECT MAX( PRECO_VENDA ) FROM CD;

MAX(PRECO_VENDA)
----------------

15

SELECT MAX( DATA_LANCAMENTO ) FROM CD;

MAX(DATA)
---------

01/05/01

SELECT MAX( NOME_CD ) FROM CD;

MAX(NOME_CD)
------------

Perfil
```

STDDEV

Retorna o desvio-padrão de uma determinada coluna. Para saber o desvio-padrão do preço dos CDs da loja, execute o seguinte comando:

```
SELECT STDDEV( PRECO_VENDA ) FROM CD;

STDDEV(PRECO_VENDA)
-------------------

2,2146697
```

VARIANCE

Retorna a variância de uma determinada coluna. Para saber a variância do preço dos CDs da loja, execute o seguinte comando:

```
SELECT VARIANCE( PRECO_VENDA ) FROM CD;

VARIANCE(PRECO_VENDA)
---------------------

4,9047619
```

Conversão de tipos de dados

Freqüentemente é necessário realizar uma conversão entre tipos de dados para determinadas situações. Para isso, utilizamos a função CAST.

Sintaxe:

```
CAST( dado_origem AS tipo_dado_destino )
```

No exemplo anterior, a média de preço de venda dos CDs utilizava diversas casas decimais. Seria desejável que aparecessem apenas duas casas decimais. Isso pode ser feito convertendo o resultado para o formato decimal e utilizando apenas 2 casas depois da vírgula. O comando ficaria assim:

```
SELECT CAST( AVG(PRECO_VENDA) AS DECIMAL (10,2))
    COLUNA FROM CD;

COLUNA
-------

11,79
```

Há outras situações em que isso pode ser muito importante, como na conversão de alfanuméricos para data:

```
SELECT CAST( '11/02/2002' AS DATE ) COLUNA
    FROM DUAL;

COLUNA
--------

11/02/02
```

Podemos também converter alfanuméricos em numéricos:

```
SELECT CAST( '12' AS INTEGER ) * 5     COLUNA
    FROM DUAL;

COLUNA
---------
60
```

No banco de dados Oracle, há funções específicas para transformação de alfanuméricos (TO_CHAR) e numéricos (TO_NUMBER) em data (TO_DATE). Em todos os casos, o primeiro argumento da função é o que se quer transformar e o segundo é o formato (opcional). Dessa forma, temos:

```
SELECT TO_DATE( SYSDATE,'DD-MM-YYYY' ) FROM DUAL;

SELECT TO_NUMBER( '1234' ) FROM DUAL;

SELECT TO_CHAR( 555,'09999' ) FROM DUAL;
```

Agrupando resultados

Uma característica muito importante do SQL é o poder de agrupar linhas com base em valores de determinadas colunas. Dessa forma, não estaremos trabalhando na pesquisa em todas as linhas da tabela, como fizemos anteriormente, mas sim em grupos menores. Para isso, utilizamos as funções de grupo já mostradas, com a cláusula GROUP BY no comando SELECT. A cláusula GROUP BY deve vir antes da cláusula ORDER BY e depois do WHERE (se houver necessidade de utilizá-los).

Sintaxe:

```
SELECT coluna [, coluna, ... ], função_de_grupo, [função_de_grupo,
   ...]
   FROM tabela
   GROUP BY coluna [, coluna, ... ]
```

Onde:

Cláusula	Descrição
coluna	Lista de colunas pela qual se quer agrupar (deve corresponder à mesma seqüência da cláusula GROUP BY).
função_de_grupo	COUNT, SUM, AVG, MIN ou MAX.

Como exemplo, podemos querer saber quantas músicas há em cada CD. Devemos realizar a busca na tabela FAIXA e agrupar por CODIGO_CD a quantidade (COUNT) de músicas. Assim:

```
SELECT CODIGO_CD,COUNT(*)
    FROM FAIXA
    GROUP BY CODIGO_CD;
```

```
CODIGO_CD       COUNT(*)
---------       --------
1               16
2               14
3               10
4               14
5               13
6               10
7               11
```

Para saber o preço médio de venda de cada CD agrupado por Gravadora, têm-se:

```
SELECT CODIGO_GRAVADORA, AVG(PRECO_VENDA)
    FROM CD
    GROUP BY CODIGO_GRAVADORA;
```

```
CODIGO_GRAVADORA    AVG(PRECO_VENDA)
----------------    ----------------
1                   15
2                   11
3                   11,75
```

Podemos realizar mais de uma função de grupo dentro de um mesmo SELECT. Aqui, além da média do preço de venda, temos a quantidade de CDs de cada gravadora:

```
SELECT CODIGO_GRAVADORA, AVG(PRECO_VENDA), COUNT(*)
    FROM CD
    GROUP BY CODIGO_GRAVADORA;
```

```
CODIGO_GRAVADORA        AVG(PRECO_VENDA)    COUNT(*)
----------------        ----------------    --------
1                       15                      1
2                       11                      4
3                       11,75                   2
```

Agrupamento com mais de uma tabela

Podemos unir mais de uma tabela seguindo as mesmas regras vistas no capítulo 8, mesmo quando se utilizam funções de grupo. Veja o exemplo a seguir:

```
SELECT CD.CODIGO_GRAVADORA, GRAVADORA.NOME_GRAVADORA,
    AVG(PRECO_VENDA)
    FROM CD, GRAVADORA
    WHERE CD.CODIGO_GRAVADORA = GRAVADORA.CODIGO_GRAVADORA
    GROUP BY CD.CODIGO_GRAVADORA, GRAVADORA.NOME_GRAVADORA;
```

```
CODIGO_GRAVADORA    NOME_GRAVADORA      AVG(PRECO_VENDA)
----------------    --------------      ----------------
1                   EMI                 15
2                   BMG                 11
3                   SOM LIVRE           11,75
```

Note bem

Devemos colocar TODAS as colunas que fazem parte do comando SELECT na cláusula GROUP BY, exceto, naturalmente, a função de grupo. Adote isso como regra para evitar problemas com o comando. Não é obrigatório, contudo, a utilização das colunas da cláusula GROUP BY no comando SELECT. De qualquer forma, é aconselhável utilizar a coluna no comando SELECT para facilitar o entendimento do resultado da busca. Dessa forma, o comando a seguir é válido:

```
SELECT AVG(PRECO_VENDA), COUNT(*)    FROM CD
    GROUP BY CODIGO_GRAVADORA;
```

Ordenando resultados

O resultado da busca pelas colunas que foram alvo da função de grupo pode ser ordenado. Veja o exemplo a seguir:

```
SELECT CODIGO_GRAVADORA, AVG(PRECO_VENDA), COUNT(*)    FROM CD
    GROUP BY CODIGO_GRAVADORA
    ORDER BY AVG(PRECO_VENDA);
```

```
CODIGO_GRAVADORA AVG(PRECO_VENDA)   COUNT(*)
---------------- ---------------- --------
2                11                 4
3                11,75              2
1                15                 1
```

Restringindo resultados

Os resultados dos dados agrupados podem ser restritos. Até agora foi visto que toda a tabela era afetada pelo comando GROUP BY. Contudo, nem sempre isso é desejável. Há duas maneiras de fazer isto: uma é utilizar a cláusula WHERE em conjunto com o GROUP BY; a outra é utilizar HAVING. Veja no exemplo a seguir como fica:

Queremos saber o total de autores agrupados por Código de Música (extraindo da tabela MUSICA_AUTOR) e queremos que seja o código da música menor que 15:

```
SELECT CODIGO_MUSICA,COUNT(*)
    FROM MUSICA_AUTOR
    WHERE CODIGO_MUSICA < 15
    GROUP BY CODIGO_MUSICA;
```

CODIGO_MUSICA	COUNT(*)
1	1
2	2
3	3
4	2
5	1
6	1
7	1
8	1
9	3
10	3
11	3
12	3
13	3
14	3

Podemos utilizar a cláusula HAVING após o GROUP BY, para obter o mesmo resultado. Assim:

```
SELECT  CODIGO_MUSICA,COUNT(*)
    FROM MUSICA_AUTOR
    GROUP BY  CODIGO_MUSICA
    HAVING  CODIGO_MUSICA < 15;
```

```
CODIGO_MUSICA       COUNT(*)
-------------       --------
1                   1
2                   2
3                   3
4                   2
5                   1
6                   1
7                   1
8                   1
9                   3
10                  3
11                  3
12                  3
13                  3
14                  3
14 linhas selecionadas.
```

Você deve estar pensando na diferença em se utilizar um ou outro comando. Na realidade, como se pode ver, o resultado final é exatamente igual. O tempo de execução também é igual, pois a base de dados é muito pequena. Mas há uma diferença básica entre os dois métodos:

Ao utilizarmos a cláusula WHERE, as linhas são filtradas ANTES do agrupamento. Ao utilizarmos o HAVING, as linhas são filtradas DEPOIS do agrupamento.

Veja o que acontece quando fazemos o filtro das linhas antes (WHERE), que invalida as linhas agrupadas (HAVING):

```
SELECT CODIGO_MUSICA,COUNT(*)
    FROM MUSICA_AUTOR
    WHERE CODIGO_MUSICA > 15
    GROUP BY CODIGO_MUSICA
    HAVING CODIGO_MUSICA < 15;

não há linhas selecionadas
```

Isso porque o filtro da cláusula WHERE manda mostrar apenas as linhas com código de música maior que 15 e o HAVING filtra as menores que 15. Não há o que mostrar nessa situação. Já no comando a seguir isso não acontece:

```
SELECT CODIGO_MUSICA,COUNT(*)
    FROM MUSICA_AUTOR
    WHERE CODIGO_MUSICA < 15
    GROUP BY CODIGO_MUSICA
    HAVING CODIGO_MUSICA > 10;
```

```
CODIGO_MUSICA       COUNT(*)
-------------       --------
11                  3
12                  3
13                  3
14                  3
```

O filtro da cláusula WHERE mostra todas as linhas com código de música menor que 15 e o HAVING, as linhas com códigos maiores que 10. Apenas os códigos 11 a 14 aparecem.

A única restrição é que a cláusula HAVING só pode utilizar as colunas que fazem parte do GROUP BY para filtrar as linhas. Para o WHERE, isso não é necessário.

Note bem

a. Dependendo do volume de informação, devemos utilizar um ou outro comando e eventualmente ambos. Caso a maior parte das linhas deva fazer parte do seu agrupamento e os dados devam ser mostrados para a maior parte dos grupos, então a busca será mais eficiente se os grupos forem formados primeiro e depois filtrados. Caso o agrupamento seja realizado na menor parte das linhas, será melhor filtrá-las e depois agrupá-las.

b. Nunca utilize na cláusula WHERE uma função de grupo para filtrar os grupos. Dessa forma, o comando abaixo é inválido:

```
SELECT CODIGO_GRAVADORA, AVG(PRECO_VENDA)
    FROM CD
    WHERE AVG(PRECO_VENDA)>12
    GROUP BY CODIGO_GRAVADORA;
```

c. Isso pode ser feito utilizando a cláusula HAVING. Dessa forma, o comando a seguir é válido:

```
SELECT CODIGO_GRAVADORA, AVG(PRECO_VENDA)
    FROM CD
    GROUP BY CODIGO_GRAVADORA
    HAVING AVG(PRECO_VENDA)>12;

CODIGO_GRAVADORA        AVG(PRECO_VENDA)
----------------        ----------------
1                       15
```

Exercícios propostos

1. Verifique a maior, a menor e o valor médio das ofertas da tabela.

2. Verifique o desvio-padrão e a variância do preço de venda dos imóveis.

3. Refaça o comando anterior mostrando o resultado em formato decimal com duas casas depois da vírgula.

4. Mostre o maior, o menor, o total e a média de preço de venda dos imóveis.

5. Modifique o comando anterior para que sejam mostrados os mesmos índices por bairro.

6. Faça uma busca que retorne o total de imóveis por vendedor. Apresente em ordem total de imóveis.

7. Verifique a diferença de preços entre o maior e o menor imóvel da tabela.

8. Mostre o código do vendedor e o menor preço de imóvel dele no cadastro. Exclua da busca os valores de imóveis inferiores a 10 mil.

9. Mostre o código e o nome do comprador e a média do valor das ofertas e o número de ofertas deste comprador.

10. Faça uma busca que retorne o total de ofertas realizadas nos anos de 2000, 2001 e 2002.

10

Subqueries

Neste capítulo você verá como utilizar uma busca dentro de outra busca. Esse é um recurso avançado e extremamente útil do comando SELECT. Na prática, será colocado um SELECT dentro de outro. Isso quer dizer que serão colocados vários SELECTs internos. *Subqueries* faz parte do padrão SQL-86, logo todos os bancos de dados relacionais que utilizam SQL devem permitir essa utilização.

Esta é a forma básica do comando:

```
SELECT colunas
   FROM tabela
   WHERE expressão operador ( SELECT colunas
   FROM tabela
   WHERE ... )
```

Função	Descrição
colunas	Colunas que serão mostradas.
tabela	Nome da tabela.
expressão	Condição ou predicado da busca.
operador	Pode ser de linha (>, <, >=, <=, =, <>) ou de grupo (IN, ANY, ALL).

A *subquery* pode ser colocada na cláusula HAVING, como será visto adiante.

Há três tipos de *subquery*:

1. *Subquery* de uma linha: o retorno do SELECT interno será uma única linha.

2. *Subquery* de múltiplas linhas: o retorno do SELECT interno será mais de uma linha.

3. *Subquery* de múltiplas colunas: o retorno do SELECT interno conterá mais de uma linha e coluna.

Subquery de uma linha

Nesse caso, o resultado do SELECT mais interno servirá de base para o primeiro SELECT. Ao utilizarmos essa forma de busca, poderemos até melhorar o desempenho do banco de dados. Uma alternativa para essa forma de realizar a busca seria fazer uma União Regular de tabelas. Dependendo do que queremos buscar nas tabelas, esse comando é mais indicado em virtude da velocidade. Note que fazendo a união regular de tabelas, estamos combinando todas as linhas de ambas as tabelas. Primeiro é feita a busca em uma tabela e, com base no resultado, pesquisada a outra tabela. Veja um exemplo:

```
SELECT NOME_CD, PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA > (SELECT AVG(PRECO_VENDA)
    FROM CD);
```

```
NOME_CD                  PRECO_VENDA
-----------------------  -----------
Mais do Mesmo            15
Bate-Boca                12
Elis Regina - Essa Mulher  13
A Força que nunca Seca   13,5
```

Aqui é utilizada uma *subquery* para verificar a média do preço dos CDs e, com base nesse resultado (11,785714), é extraído o resultado da busca do primeiro SELECT (nome e preço do CD). Note que apenas os CDs com preço maior que a média efetivamente aparecem na busca.

Veja o exemplo abaixo:

```
SELECT CODIGO_GRAVADORA, NOME_CD, PRECO_VENDA
    FROM CD a
    WHERE PRECO_VENDA > (SELECT AVG(PRECO_VENDA)
    FROM CD
    WHERE CODIGO_GRAVADORA = a.CODIGO_GRAVADORA);
```

```
CODIGO_GRAVADORA  NOME_CD                  PRECO_VENDA
----------------  -----------------------  -----------
2                 Bate-Boca                12
3                 Elis Regina - Essa Mulher  13
2                 A Força que nunca Seca   13,5
```

Aqui é utilizada uma técnica interessante. Note que são buscados apenas os CDs que tenham preço de venda superior à média de preço da própria gravadora. Isso ocorre porque foi colocado um apelido na tabela CD da primeira busca (a) e este comparado no segundo SELECT com o código da gravadora.

Dessa forma, o segundo SELECT retorna a média de preço de venda dos CDs da gravadora do primeiro SELECT e somente os que tiverem preço de venda maiores que essa média é que aparecerão na listagem. Cuidado ao utilizar esse tipo de construção porque ele tende a consumir muito recurso do banco de dados, pois a cada linha do primeiro SELECT esse tipo de construção vai extrair a média no segundo SELECT. Assim é necessário saber qual a gravadora do primeiro SELECT, para poder calcular o segundo SELECT.

Note bem

Para realizar *subqueries* tenha em mente:

1. Coloque as *subqueries* entre parênteses.
2. Coloque a *subquery* à direita do operador.
3. Não coloque a cláusula ORDER BY em uma *subquery*. Não esqueça que deve haver apenas uma cláusula ORDER BY em todo comando SELECT. Logo, se for necessário ordenar o resultado, faça isso no SELECT principal.
4. Utilize operadores de linha apenas em buscas que retornem uma única linha.
5. Utilize operadores de grupo apenas em buscas que potencialmente retornem mais de uma linha.

Veja outro exemplo onde são utilizadas mais de uma *subquery*:

```
SELECT CODIGO_GRAVADORA,NOME_CD,PRECO_VENDA  FROM CD
    WHERE CODIGO_GRAVADORA = (SELECT CODIGO_GRAVADORA   FROM CD
    WHERE CODIGO_CD = 2)AND
    PRECO_VENDA > (SELECT PRECO_VENDA FROM CD WHERE CODIGO_CD =
    5);
```

```
CODIGO_GRAVADORA    NOME_CD                 PRECO_VENDA
----------------    ----------------------  -----------

2                   Bate-Boca               12

2                   A Força que nunca Seca  13,5
```

O CD de código 2 é da gravadora número 2 (resultado do primeiro SELECT aninhado) e o preço de venda do CD código 5 é 10,50 (resultado do segundo SELECT aninhado). Os CDs que atendem aos dois requisitos (ser da gravadora 2 e custar mais de 10,50) são aqueles mostrados como resultado.

Utilizando *subquery* em cláusula HAVING

A utilização é realizada da mesma forma que na cláusula WHERE. A *subquery* será executada primeiro e o resultado da busca servirá de base para filtrar as linhas do GROUP BY. Veja no exemplo a seguir:

```
SELECT CODIGO_GRAVADORA,MIN(PRECO_VENDA)
    FROM CD
    GROUP BY CODIGO_GRAVADORA
    HAVING MIN(PRECO_VENDA) >
    (SELECT PRECO_VENDA FROM CD WHERE CODIGO_CD = 6);

CODIGO_GRAVADORA        MIN(PRECO_VENDA)
----------------        ----------------
1                       15
3                       10,5
```

Veja que o preço de venda do CD de código 6 é R$ 9,50. Esse é o valor utilizado para que somente as gravadoras que possuam preços mínimos superiores a este apareçam na busca. Veja outro exemplo:

```
SELECT CODIGO_GRAVADORA,MAX(PRECO_VENDA)
    FROM CD a
    GROUP BY CODIGO_GRAVADORA
    HAVING MAX(PRECO_VENDA) > (SELECT AVG(PRECO_VENDA)
    FROM CD
    WHERE CODIGO_GRAVADORA = a.CODIGO_GRAVADORA);

CODIGO_GRAVADORA        MAX(PRECO_VENDA)
----------------        ----------------
2                       13,5
3                       13
```

A diferença entre esse comando e o anterior é que estamos buscando dados da primeira busca (a.CODIGO_GRAVADORA no segundo SELECT) para realizar a busca do segundo SELECT. O que pedimos é que seja comparada a média de preço dos CDs da gravadora da primeira busca (por esse motivo, utilizamos o apelido na cláusula FROM do primeiro SELECT). Somente as gravadoras que tiverem CDs com preço de venda superiores à média aparecem no resultado. Note que a gravadora de código 1 não aparece, pois só há um único CD dessa gravadora. Logo seu preço máximo não pode ser superior à média de preços.

EXISTS

Verifica o número de linhas retornadas pela *subquery*. Caso ela contenha uma ou mais linhas, então o resultado será mostrado. Do contrário, não o será.

Será utilizado como exemplo a situação das gravadoras. Sabemos que há uma gravadora que não tem CDs cadastrados. Logo, para mostrarmos apenas as gravadoras que têm CDs cadastrados, podemos executar o seguinte comando:

```
SELECT CODIGO_GRAVADORA,NOME_GRAVADORA     FROM GRAVADORA
    WHERE EXISTS (SELECT *FROM CD
    WHERE CD.CODIGO_GRAVADORA = GRAVADORA.CODIGO_GRAVADORA);

CODIGO_GRAVADORA        NOME_GRAVADORA
----------------        --------------
1                       EMI
2                       BMG
3                       SOM LIVRE
```

Note que quando você está utilizando esse operador, não importa o que o comando SELECT interno irá buscar. Interessa apenas se ele retorna ou não linhas. Por esse motivo, utilizamos o *em vez de uma coluna em especial.

Subquery de múltiplas linhas

Neste caso, o SELECT interno retorna mais de uma linha. Não se pode utilizar operadores simples como igualdade, diferença, maior ou menor. Deve-se utilizar um operador de grupo para realizar a comparação. Esses operadores são ANY, ALL e IN.

IN

Imagine que gostaríamos de saber quais CDs têm o preço igual ao menor preço de cada gravadora. Inicialmente devemos saber qual o menor preço de cada gravadora. Isso pode ser feito com o comando:

```
SELECT MIN(PRECO_VENDA)     FROM CD
    GROUP BY CODIGO_GRAVADORA;

MIN(PRECO_VENDA)
----------------
15
9
10,5
```

Para saber quais são os CDs com esses preços, poderíamos escrever o seguinte comando:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA IN (15,9,10.5);
```

```
CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
1           Mais do Mesmo              15
5           Perfil                     10,5
7           Listen Without Prejudice   9
```

Para fazer isso tudo em um único comando, escrevemos o comando da seguinte forma:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA IN
    ( SELECT MIN( PRECO_VENDA )
    FROM CD
    GROUP BY CODIGO_GRAVADORA );
```

```
CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
7           Listen Without Prejudice   9
5           Perfil                     10,5
1           Mais do Mesmo              15
```

Uma outra situação em que poderíamos utilizar esse operador é quando queremos saber em qual CD e faixa há músicas que tenham AMOR no nome. Para isso, temos que utilizar as linhas da tabela MUSICA e compará-las com AMOR. Não esquecer de colocar a função UPPER do nome do campo pois, para efetuar a comparação com sucesso, devemos ter certeza de que o conteúdo da coluna esteja em letras maiúsculas, uma vez que escrevemos AMOR em letras maiúsculas. Veja a seguir:

```
SELECT CODIGO_MUSICA
    FROM MUSICA
    WHERE UPPER(NOME_MUSICA) LIKE '%AMOR%';
```

```
CODIGO_MUSICA
-------------
25
41
```

Sabendo quais são os códigos da música, podemos realizar uma busca na tabela FAIXA que nos informe em quais CDs e Faixas estão essas músicas. Assim:

```
SELECT CODIGO_CD,NUMERO_FAIXA
     FROM FAIXA
     WHERE CODIGO_MUSICA IN (25,41);

CODIGO_CD       NUMERO_FAIXA
---------       ------------
4               1
3               9
```

Da mesma forma, podemos simplificar o comando realizando tudo isso em uma única operação:

```
SELECT CODIGO_CD,NUMERO_FAIXA
     FROM FAIXA
     WHERE CODIGO_MUSICA IN
     (SELECT CODIGO_MUSICAFROM MUSICA
     WHERE UPPER(NOME_MUSICA) LIKE '%AMOR%');

CODIGO_CD       NUMERO_FAIXA
---------       ------------
2               9
4               1
```

ANY

Esse operador permite comparar operadores simples (=, >, <, !=) com um grupo de linhas. Será feita a comparação com cada valor retornado do SELECT interno.

Imagine que desejemos saber quais CDs têm preço inferior a qualquer outro da gravadora com código 2, mas que não sejam da gravadora 2. Inicialmente é necessário saber qual é o preço de venda de cada CD da gravadora 2:

```
SELECT PRECO_VENDA
     FROM CD
     WHERE CODIGO_GRAVADORA = 2;

PRECO_VENDA
-----------
12
13,5
9,5
9
```

Agora realizamos a pesquisa na tabela de CD para comparar os preços inferiores a esses:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA < ANY (12,13.5,9.5,9)
    AND CODIGO_GRAVADORA != 2;
```

```
CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
3           Elis Regina - Essa Mulher  13
5           Perfil                     10,5
```

Mais uma vez, isso pode ser simplificado, colocando em um único comando o preço de venda da tabela CD em que o código da gravadora é 2:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA < ANY (
    SELECT PRECO_VENDA
    FROM CD
    WHERE CODIGO_GRAVADORA = 2 )
    AND CODIGO_GRAVADORA != 2;
```

```
CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
3           Elis Regina - Essa Mulher  13
5           Perfil                     10,5
```

Podemos ainda utilizar o mesmo operador ANY com os outros operadores simples. Veja os exemplos a seguir:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA > ANY (
    SELECT PRECO_VENDA
    FROM CD
    WHERE CODIGO_GRAVADORA = 2 )
    AND CODIGO_GRAVADORA != 2;
```

```
CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
1           Mais do Mesmo              15
3           Elis Regina - Essa Mulher  13
5           Perfil                     10,5
```

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA = ANY (
    SELECT PRECO_VENDA
    FROM CD
    WHERE CODIGO_GRAVADORA = 2 );
```

```
CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
7           Listen Without Prejudice   9
6           Barry Manilow Greatest Hits 9,5
2           Bate-Boca                  12
4           A Força que nunca Seca     13,5
```

Observe que nesse último exemplo não existe mais a condição do código da gravadora ser diferente de 2, caso contrário, não existiriam linhas de retorno. Apenas os CDs da gravadora 2 aparecem na última listagem.

ALL

Este operador é utilizado em combinação com operadores simples (>, <) para que os valores retornados de todas as linhas do SELECT interno sejam comparados com o SELECT externo.

Imagine que queiramos saber quais CDs têm o preço de venda menor que a média de preço de venda de todas as gravadoras. Inicialmente é necessário saber a média de preços por gravadora:

```
SELECT AVG(PRECO_VENDA)
    FROM CD
    GROUP BY CODIGO_GRAVADORA;

AVG(PRECO_VENDA)
----------------
15
11
11,75
```

Depois de verificarmos quais são os CDs com preço inferior a todos anteriores. O comando seria o seguinte:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
    FROM CD
    WHERE PRECO_VENDA < ALL ( 15,11,11.75 );

CODIGO_CD   NOME_CD                    PRECO_VENDA
---------   -------------------------  -----------
5           Perfil                     10,5
6           Barry Manilow Greatest Hits 9,5
7           Listen Without Prejudice   9
```

Em um único comando teríamos:

```
SELECT CODIGO_CD,NOME_CD,PRECO_VENDA
     FROM CD
     WHERE PRECO_VENDA < ALL (
     SELECT AVG(PRECO_VENDA)
     FROM CD
     GROUP BY CODIGO_GRAVADORA );
```

```
CODIGO_CD   NOME_CD                      PRECO_VENDA
---------   ---------------------------  -----------
5           Perfil                       10,5
6           Barry Manilow Greatest Hits  9,5
7           Listen Without Prejudice     9
```

Subquery de múltiplas colunas

Alguns bancos de dados permitem realizar a busca em mú tiplas colunas utilizando uma *subquery*.

A primeira técnica consiste em colocar as colunas unidas na cláusula WHERE do SELECT externo e realizar a busca no SELECT interno dessas colunas unidas da mesma forma. Essa técnica tende a ser muito lenta, mas pode ser feita em qualquer banco de dados. Veja o exemplo a seguir:

Temos vários CDs indicados no cadastro e queremos conhecer os dados do CD dentre os menores CDs indicados de cada gravadora. Para isso, temos que saber o menor CD indicado de cada gravadora. Observe que tivemos que unir as duas colunas com ||.

```
SELECT CODIGO_GRAVADORA || MIN(CD_INDICADO)
     FROM CD
     GROUP BY CODIGO_GRAVADORA;
```

```
CODIGO_GRAVADORA||MIN(CD_INDICADO)
----------------------------------
15
21
31
```

Agora devemos realizar a busca dos dados dos CDs que atendem a essa característica:

```
SELECT CODIGO_CD,NOME_CD,CODIGO_GRAVADORA,CD_INDICADO
     FROM CD
     WHERE CODIGO_GRAVADORA || CD_INDICADO IN (15,21,31);
```

```
CODIGO_CD   NOME_CD                    CODIGO_GRAVADORA    CD_INDICADO
---------   -------------------------  ----------------    -----------
1           Mais do Mesmo              1                   5
3           Elis Regina - Essa Mulher  3                   1
4           A Força que nunca Seca     2                   1
```

Note como a combinação das colunas CODIGO_GRAVADORA e CD_INDICADO é a mesma do comando SELECT anterior. Agora veremos tudo num único comando:

```
SELECT CODIGO_CD,NOME_CD,CODIGO_GRAVADORA,CD_INDICADO
    FROM CD
    WHERE CODIGO_GRAVADORA || CD_INDICADO IN
    (SELECT CODIGO_GRAVADORA || MIN(CD_INDICADO)
    FROM CD
    GROUP BY CODIGO_GRAVADORA );
```

```
CODIGO_CD   NOME_CD                    CODIGO_GRAVADORA    CD_INDICADO
---------   -------------------------  ----------------    -----------
1           Mais do Mesmo              1                   5
4           A Força que nunca Seca     2                   1
3           Elis Regina - Essa Mulher  3                   1
```

A segunda técnica é colocar os argumentos de busca entre parênteses no SELECT externo. Isso pode não estar disponível no seu banco de dados, mas o desempenho da consulta, caso esteja disponível, será sensivelmente melhor. Isso porque o banco de dados gasta algum tempo tendo que combinar os valores das colunas durante a busca. Utilizando o mesmo exemplo anterior, teríamos:

```
SELECT  CODIGO_CD,NOME_CD,CODIGO_GRAVADORA,CD_INDICADO
    FROM  CD
    WHERE (CODIGO_GRAVADORA,CD_INDICADO) IN
    (SELECT CODIGO_GRAVADORA,MIN(CD_INDICADO)
    FROM CD
    GROUP BY  CODIGO_GRAVADORA );
```

```
CODIGO_CD   NOME_CD                    CODIGO_GRAVADORA    CD_INDICADO
---------   -------------------------  ----------------    -----------
1           Mais do Mesmo              1                   5
4           A Força que nunca Seca     2                   1
3           Elis Regina - Essa Mulher  3                   1
```

Note bem

Há duas considerações que devem ser feitas sobre as *subqueries*:

a. NULL.

b. Pares de comparação.

NULL

Quando um SELECT interno contiver valores nulos, não deve ser utilizado NOT IN como operador de comparação. Isso porque qualquer comparação com nulo retorna valor nulo. Exemplo: em nosso cadastro, temos um CD (código 7) que não tem CD indicado. No SELECT a seguir não há problema, pois é utilizado o operador IN:

```
SELECT CODIGO_CD,NOME_CD,CD_INDICADO
     FROM CD
     WHERE CD_INDICADO IN
     (SELECT CD_INDICADO
     FROM CD );
```

```
CODIGO_CD   NOME_CD                    CD_INDICADO
---------   -------------------------  -----------
3           Elis Regina - Essa Mulher  1
4           A Força que nunca Seca     1
5           Perfil                     2
2           Bate-Boca                  3
1           Mais do Mesmo              5
6           Barry Manilow Greatest Hits 7
```

Quando é utilizado o operador NOT IN, nada é retornado:

```
SELECT CODIGO_CD,NOME_CD,CD_INDICADO
     FROM CD
     WHERE CD_INDICADO NOT IN
     (SELECT CD_INDICADO
     FROM CD );
não há linhas selecionadas
```

Pares de comparação

Sempre que são realizadas comparações em *subquery* de múltiplas colunas, devemos saber se realmente o que queremos é a comparação entre pares de colunas, ou não. Veja as listagens a seguir:

CODIGO_CD	CODIGO_GRAVADORA	CD_INDICADO	CODIGO_CD	CODIGO_GRAVADORA	CD_INDICADO
1	1	5	1	1	5
2	2	3	2	2	3
3	3	1	3	3	1
4	2	1	4	2	1
5	3	2	5	3	2
6	2	7	6	2	7

Observe que no caso à direita são combinados também o código da gravadora 3 com o CD indicado 1, além dos códigos 2 e 3 de gravadoras com o mesmo CD indicado. À esquerda são comparados os CDs indicados e as gravadoras em uma mesma linha. A diferença entre os comandos que seriam escritos é a seguinte:

a. Esquerda:

```
SELECT CODIGO_CD,NOME_CD,CD_INDICADO,CODIGO_GRAVADORA
    FROM CD
    WHERE (CODIGO_GRAVADORA,CD_INDICADO) IN
    (SELECT CODIGO_GRAVADORA,CD_INDICADO
    FROM CD
    WHERE CODIGO_CD = 3 );
```

CODIGO_CD	NOME_CD	CD_INDICADO	CODIGO_GRAVADORA
3	Elis Regina - Essa Mulher	1	3

b. Direita:

```
SELECT CODIGO_CD,NOME_CD,CD_INDICADO,CODIGO_GRAVADORA
    FROM CD
    WHERE CODIGO_GRAVADORA IN (
    SELECT CODIGO_GRAVADORA
    FROM CD
    WHERE CODIGO_CD = 3 )
    OR CD_INDICADO IN (
    SELECT CD_INDICADO
    FROM CD
    WHERE CODIGO_CD = 3 );
```

```
CODIGO_CD   NOME_CD                    CD_INDICADO     CODIGO_GRAVADORA
---------   -------------------------  -----------     ----------------
3           Elis Regina - Essa Mulher  1               3
4           A Força que nunca Seca     1               2
5           Perfil                     2               3
```

Subquery na cláusula FROM

Podemos utilizar uma *subquery* na cláusula FROM de um comando SELECT. Essa estrutura não está disponível para todos os bancos de dados, mas faz parte do padrão SQL.

A utilização é muito parecida com as visões (este objeto será visto na parte III deste livro). Uma *subquery* na cláusula FROM define a fonte de dados para esse SELECT em particular. Imagine que desejemos extrair o nome do CD, o seu preço de venda e o preço médio da gravadora, e o preço médio da gravadora é extraído por uma *subquery* na própria cláusula FROM. Veja como ficaria:

```
SELECT a.NOME_CD, a.PRECO_VENDA, b.PRECO_MEDIO
    FROM CD a, (SELECT CODIGO_GRAVADORA, AVG(PRECO_VENDA)
    PRECO_MEDIO
    FROM CD
    GROUP BY CODIGO_GRAVADORA) b
    WHERE a.CODIGO_GRAVADORA = b.CODIGO_GRAVADORA
    AND a.PRECO_VENDA > b.PRECO_MEDIO;
```

```
NOME_CD                    PRECO_VENDA     PRECO_MEDIO
-------------------------  -----------     -----------
Bate-Boca                  12              11
A Força que nunca Seca     13,5            11
Elis Regina - Essa Mulher  13              11,75
```

Veja que foi substituída a segunda tabela da cláusula FROM por um comando SELECT e que a tabela recebeu o apelido de b nesse comando. Depois, comparamos o código da gravadora de a (CD da cláusula FROM) com b (*subquery*) e foi dada a condição de que o preço de venda deve ser maior que o preço médio. Observe que foi criada uma coluna virtual, resultado da média de preço de venda, no SELECT interno, e foi possível compará-la na cláusula WHERE do SELECT externo.

Exercícios propostos

1. Faça uma lista de imóveis do mesmo bairro do imóvel 2. Exclua o imóvel 2 da sua busca.

2. Faça uma lista que mostre todos os imóveis que custam mais que a média de preço dos imóveis.

3. Faça uma lista com todos os compradores que tenham ofertas cadastradas com valor superior a 70 mil.

4. Faça uma lista com todos os imóveis com oferta superior à média do valor das ofertas.

5. Faça uma lista com todos os imóveis com preço superior à média de preço dos imóveis do mesmo bairro.

6. Faça uma lista dos imóveis com maior preço agrupado por bairro, cujo maior preço seja superior à média de preços dos imóveis.

7. Faça uma lista com os imóveis que têm o preço igual ao menor preço de cada vendedor.

8. Faça uma lista com as ofertas dos imóveis com data de lançamento do imóvel inferior a 30 dias e superior a 180 dias, a contar de hoje e cujo código vendedor seja 2.

9. Faça uma lista com os imóveis que têm o preço igual ao menor preço de todos os vendedores, exceto os imóveis do próprio vendedor.

10. Faça uma lista com as ofertas menores que todas as ofertas do comprador 2, exceto as ofertas do próprio comprador.

11. Faça uma lista de todos os imóveis cujo Estado e cidade sejam os mesmos do vendedor 3, exceto os imóveis do vendedor 3.

12. Faça uma lista com todos os nomes de bairro cujos imóveis sejam do mesmo Estado, cidade e bairro do imóvel código 5.

11

Pesquisa avançada

Já vimos muitas formas de pesquisa em tabelas. Agora vamos analisar algumas formas que permitem trabalhar com vários comandos SELECT. Serão criados conjuntos que poderão ser unidos e separados da mesma forma como trabalhamos com os conjuntos na matemática.

Há três formas de operação:

1. União.
2. Intersecção.
3. Exceção.

UNION

A união entre tabelas cria uma saída com todas as linhas de uma tabela unida com as linhas da outra tabela. Diferentemente da união regular entre tabelas, aqui as linhas não são combinadas às linhas da outra tabela. As linhas da segunda tabela são colocadas em seqüência às linhas da primeira tabela.

Veja como ocorre uma união de tabelas. Esta seria a primeira tabela:

```
1  Renato Russo
2  Tom Jobim
3  Chico Buarque
4  Dado Villa-Lobos
5  Marcelo Bonfá
6  Ico Ouro-Preto
7  Vinicius de Moraes
8  Baden Powell
9  Paulo Cesar Pinheiro
```

Esta seria a segunda tabela:

```
1  Será
2  Ainda é Cedo
3  Geração Coca-Cola
4  Eduardo e Monica
5  Tempo Perdido
6  Índios
7  Que Pais é Este
8  Faroeste Caboclo
9  Há Tempos
```

Ao unirmos a primeira com a segunda tabela teríamos:

```
1  Renato Russo
2  Tom Jobim
3  Chico Buarque
4  Dado Villa-Lobos
5  Marcelo Bonfá
6  Ico Ouro-Preto
7  Vinicius de Moraes
8  Baden Powell
9  Paulo Cesar Pinheiro
1  Será
2  Ainda é Cedo
3  Geração Coca-Cola
4  Eduardo e Monica
5  Tempo Perdido
6  Índios
7  Que Pais é Este
8  Faroeste Caboclo
9  Há Tempos
```

Para que isso aconteça, é necessário que haja uma compatibilidade entre as colunas. Na prática, isso quer dizer que as colunas devem ser do mesmo tipo e estar na mesma seqüência para que se possa unir as duas tabelas. Mesmo que o conteúdo das tabelas não seja coerente, a união entre as tabelas Autores e Músicas é possível porque as colunas têm o mesmo tipo de dado e tamanho.

Para realizar essa união, serão escritos dois comandos SELECT com a cláusula UNION entre eles.

Sintaxe:

```
SELECT colunas  FROM tabela(s)
   [WHERE predicado]
   [GROUP BY colunas [HAVING condição]]
UNION
SELECT colunas  FROM tabela(s)
   [WHERE predicado]
   [GROUP BY colunas [HAVING condição]]
```

É importante que as colunas estejam na mesma seqüência, com o mesmo tipo de dados nos dois SELECTs, do contrário o comando não funcionará. Veja que não é necessário que as colunas tenham o mesmo nome, uma vez que apenas o tipo de dado e o tamanho são comparados e, normalmente, quando é feita essa união, trabalha-se com tabelas diferentes. Veja o exemplo:

```
SELECT CODIGO_AUTOR,NOME_AUTOR   FROM AUTOR
    WHERE CODIGO_AUTOR < 10
    UNION
    SELECT CODIGO_MUSICA,NOME_MUSICA   FROM MUSICA
    WHERE CODIGO_MUSICA < 15;
```

CODIGO_AUTOR	NOME_AUTOR
1	Renato Russo
1	Será
2	Ainda é Cedo
2	Tom Jobim
3	Chico Buarque
3	Geração Coca-Cola
4	Dado Villa-Lobos
4	Eduardo e Monica
5	Marcelo Bonfá
5	Tempo Perdido
6	Ico Ouro-Preto
6	Índios
7	Que País é Este
7	Vinicius de Moraes
8	Baden Powell
8	Faroeste Caboclo
9	Há Tempos
9	Paulo Cesar Pinheiro
10	Pais e Filhos
11	Meninos e Meninas
12	Vento no Litoral
13	Perfeição
14	Giz

Veja que o nome da coluna no resultado da busca ficou com o nome da coluna do primeiro SELECT.

Note também que as linhas estão combinadas, ou seja, aparece o código 1 do autor, e na linha debaixo o código 1 da música. Isso porque em toda união existe a possibilidade de haver o mesmo conteúdo no outro SELECT. A união não permitirá que duas linhas se repitam no resultado da busca. Isso pode parecer muito bom, mas gera um trabalho adicional ao banco de dados, uma vez que terá que ser verificado se há linhas repetidas. Veja uma outra união que, por motivos didáticos, realizaremos na mesma tabela, mas com predicado diferente:

```
SELECT CODIGO_AUTOR,NOME_AUTOR
    FROM AUTOR
    WHERE CODIGO_AUTOR < 10
    UNION
    SELECT CODIGO_AUTOR,NOME_AUTOR
    FROM AUTOR
    WHERE CODIGO_AUTOR < 15;
```

CODIGO_AUTOR	NOME_AUTOR
1	Renato Russo
2	Tom Jobim
3	Chico Buarque
4	Dado Villa-Lobos
5	Marcelo Bonfá
6	Ico Ouro-Preto
7	Vinicius de Moraes
8	Baden Powell
9	Paulo Cesar Pinheiro
10	João Bosco
11	Aldir Blanc
12	Joyce
13	Ana Terra
14	Cartola

Veja que as linhas de 1 a 10 não se repetem no resultado da busca. Claro que em pequenas tabelas, isso não representará perda de desempenho, mas em tabelas muito grandes, com certeza causará problemas. Uma alternativa a esta situação é a utilização da cláusula UNION ALL presente em alguns bancos de dados relacionais.

UNION ALL

Esta cláusula permite unir as duas ou mais tabelas sem nos preocuparmos com a repetição do conteúdo. Isso leva a um melhor desempenho, pois não será necessário comparar o conteúdo das tabelas. Deve-se, naturalmente, tomar cuidado ao utilizá-lo, uma vez que é possível gerar informação errada. Aconselhamos utilizar o UNION ALL apenas quando se trabalha com tabelas distintas, em que os dados não se repetirão. Veja o exemplo:

```
SELECT CODIGO_AUTOR,NOME_AUTOR
    FROM AUTOR
    WHERE CODIGO_AUTOR < 10
    UNION ALL
    SELECT CODIGO_MUSICA,NOME_MUSICA
    FROM MUSICA
    WHERE CODIGO_MUSICA < 15;
```

CODIGO_AUTOR	NOME_AUTOR
1	Renato Russo
2	Tom Jobim
3	Chico Buarque
4	Dado Villa-Lobos
5	Marcelo Bonfá
6	Ico Ouro-Preto
7	Vinicius de Moraes
8	Baden Powell
9	Paulo Cesar Pinheiro
1	Será
2	Ainda é Cedo
3	Geração Coca-Cola
4	Eduardo e Monica
5	Tempo Perdido
6	Índios
7	Que País é Este
8	Faroeste Caboclo
9	Há Tempos
10	Pais e Filhos
11	Meninos e Meninas
12	Vento no Litoral
13	Perfeição
14	Giz

Veja como aparecem inicialmente todas as linhas da primeira tabela e depois as linhas da segunda tabela. Elas não estão mais combinadas, como no primeiro exemplo.

A cláusula UNION é utilizada freqüentemente para substituir a cláusula OR no predicado do SELECT. Veja o exemplo:

```
SELECT NOME_CD,NOME_MUSICA
    FROM CD, FAIXA, MUSICA
    WHERE CD.CODIGO_CD = FAIXA.CODIGO_CD
    AND FAIXA.CODIGO_MUSICA = MUSICA.CODIGO_MUSICA
    AND CD.CODIGO_CD = 1
    UNION
    SELECT NOME_CD,NOME_MUSICA
    FROM CD, FAIXA, MUSICA
    WHERE CD.CODIGO_CD = FAIXA.CODIGO_CD
    AND FAIXA.CODIGO_MUSICA = MUSICA.CODIGO_MUSICA
    AND MUSICA.CODIGO_MUSICA = 20;
```

```
NOME_CD          NOME_MUSICA
-------------    -----------------
Bate-Boca        A Violeira
Mais do Mesmo    Ainda é Cedo
Mais do Mesmo    Antes das Seis
Mais do Mesmo    Dezesseis
Mais do Mesmo    Eduardo e Monica
Mais do Mesmo    Faroeste Caboclo
Mais do Mesmo    Geração Coca-Cola
Mais do Mesmo    Giz
Mais do Mesmo    Há Tempos
Mais do Mesmo    Meninos e Meninas
Mais do Mesmo    Pais e Filhos
Mais do Mesmo    Perfeição
Mais do Mesmo    Que País é Este
Mais do Mesmo    Será
Mais do Mesmo    Tempo Perdido
Mais do Mesmo    Vento no Litoral
Mais do Mesmo    Índios
```

Esse mesmo resultado seria produzido pelo comando:

```
SELECT NOME_CD,NOME_MUSICA
    FROM CD, FAIXA, MUSICA
    WHERE CD.CODIGO_CD = FAIXA.CODIGO_CD
    AND FAIXA.CODIGO_MUSICA = MUSICA.CODIGO_MUSICA
    AND (CD.CODIGO_CD = 1 OR MUSICA.CODIGO_MUSICA = 20);
```

Contudo, ainda que não seja esse o caso, não corremos o risco de ter linhas duplicadas na tabela, quando utilizamos UNION em vez do OR. Se utilizarmos a segunda opção e desejamos ter certeza de que não haverá duplicidade nas linhas de resultado, teremos que utilizar a cláusula DISTINCT, o que, na prática, terá o mesmo efeito de um UNION.

CORRESPONDING BY

Há uma sintaxe alternativa prevista no padrão SQL para fazer com que a união entre as duas tabelas seja compatível. Deve ser utilizada quando as duas tabelas têm colunas com os mesmos nomes, ainda que não tenham exatamente todas as colunas, tipos de dados e tamanhos iguais.

Para utilizar essa cláusula, basta utilizarmos o SELECT * de cada uma das tabelas e então indicarmos as colunas utilizadas para união na cláusula CORRESPONDING BY. Assim:

```
SELECT *
   FROM tabela
   [WHERE predicado]

UNION CORRESPONDING BY (colunas)

SELECT *
   FROM tabela
   [WHERE predicado]
```

Para realizar a consulta, o banco de dados fará a busca em todas as colunas de ambas as tabelas, mas descartará as que não estiverem na cláusula CORRESPONDING BY.

EXCEPT DISTINCT/MINUS

Utilizamos a cláusula EXCEPT DISTINCT, padrão SQL ou MINUS, em alguns bancos de dados, sempre que quisermos conhecer as linhas que existem em um SELECT e não existem em outro. É a exceção de elementos de um conjunto em relação ao outro. Para demonstrar esse comando, vamos realizar dois SELECTs em uma mesma tabela. Veja:

```
SELECT CODIGO_AUTOR,NOME_AUTOR
     FROM AUTOR
     WHERE CODIGO_AUTOR < 10
     EXCEPT DISTINCT
     SELECT CODIGO_AUTOR,NOME_AUTOR
     FROM AUTOR
     WHERE CODIGO_AUTOR > 5 AND CODIGO_AUTOR < 15;

CODIGO_AUTOR    NOME_AUTOR
------------    ----------------

1               Renato Russo

2               Tom Jobim

3               Chico Buarque

4               Dado Villa-Lobos

5               Marcelo Bonfá
```

Veja que o primeiro SELECT busca apenas as linhas cujos códigos de autor são menores que 10 e o segundo SELECT, as linhas cujos códigos de autor são maiores que 5 e menores que 15. Logo, como se quer saber os elementos do conjunto que estão no primeiro SELECT e não no segundo, temos o retorno apenas das 5 primeiras linhas. Se invertermos os SELECT, teremos:

```
SELECT CODIGO_AUTOR,NOME_AUTOR
    FROM AUTOR
    WHERE CODIGO_AUTOR > 5 AND CODIGO_AUTOR < 15
    EXCEPT DISTINCT
    SELECT CODIGO_AUTOR,NOME_AUTOR
    FROM AUTOR
    WHERE CODIGO_AUTOR < 10;
```

```
CODIGO_AUTOR    NOME_AUTOR
------------    ----------
10              João Bosco
11              Aldir Blanc
12              Joyce
13              Ana Terra
14              Cartola
```

Agora, apenas as linhas do primeiro SELECT que não estão no segundo é que aparecem (de 10 a 14).

Uma outra utilização prática é quando desejamos conhecer ocorrências de uma tabela que ainda não tem relacionamento com outra. Exemplo: para sabermos qual gravadora não tem CD cadastrado basta utilizarmos o comando:

```
SELECT CODIGO_GRAVADORA
    FROM GRAVADORA
    EXCEPT DISTINCT
    SELECT CODIGO_GRAVADORA
    FROM CD;
```

```
CODIGO_GRAVADORA
----------------
4
```

INTERSECT

Esta cláusula permite checar as linhas que estão em ambos os SELECTs. Ela demonstra a intersecção dos conjuntos formados pelos comandos SELECT.

Veja o exemplo:

```
SELECT CODIGO_AUTOR,NOME_AUTOR
     FROM AUTOR
     WHERE CODIGO_AUTOR < 10
     INTERSECT
     SELECT CODIGO_AUTOR,NOME_AUTOR
     FROM AUTOR
     WHERE CODIGO_AUTOR > 5 AND CODIGO_AUTOR < 15;

CODIGO_AUTOR    NOME_AUTOR
------------    ----------------------
6               Ico Ouro-Preto
7               Vinicius de Moraes
8               Baden Powell
9               Paulo Cesar Pinheiro
```

Veja que somente as linhas em comum são mostradas, ou seja, aquelas que são maiores que 5 e menores que 10. No exemplo a seguir podemos visualizar quais gravadoras têm CDs cadastrados:

```
SELECT CODIGO_GRAVADORA
     FROM GRAVADORA
     INTERSECT
     SELECT CODIGO_GRAVADORA
     FROM CD;

CODIGO_GRAVADORA
----------------
1
2
3
```

Expressões CASE

O padrão SQL prevê a possibilidade de utilizar a expressão CASE, presente em diversas linguagens de programação. Utilizando essa cláusula é possível criar complexas estruturas de controle tanto nas diversas linguagens de programação como nos comandos SQL. Ao ser utilizada a cláusula CASE em comandos SQL é possível economizar diversas linhas de código, pois não é necessário criar blocos de programação para testar condições.

A expressão CASE será testada em tempo de execução do comando SELECT ou UPDATE. Como o CASE faz parte de outro comando, será possível colocá-lo em qualquer situação em que um valor deva ser testado.

Veja que em linguagens de programação usuais a expressão CASE, que é autônoma, não tem a mesma forma de avaliação e executará um bloco de instruções. Dessa forma, um único comando testará diversas linhas retornando, em cada linha, o comando correspondente à linha analisada.

SELECT

É possível utilizar o comando SELECT prevendo diversas condições para extração dos dados. A sintaxe é a seguinte:

```
SELECT colunas,
   CASE
   WHEN condição THEN ação
   ...
   [ELSE condição  padrão]
   END
   FROM tabela;
```

Como exemplo, podemos imaginar uma situação em que desejamos dar um desconto sobre o preço de venda dos CDs em função do preço. Quanto maior o preço de venda, maior o desconto. Veja:

```
SELECT  NOME_CD,PRECO_VENDA,
    CASE
    WHEN  PRECO_VENDA < 10 THEN PRECO_VENDA * .9
    WHEN  PRECO_VENDA >=10 AND PRECO_VENDA < 13 THEN PRECO_VENDA
    * .8
    ELSE  PRECO_VENDA * .7
    END  VENDA
    FROM  CD;
```

NOME_CD	PRECO_VENDA	VENDA
Mais do Mesmo	15	10,5
Bate-Boca	12	9,6
Elis Regina - Essa Mulher	13	9,1
A Força que nunca Seca	13,5	9,45
Perfil	10,5	8,4
Barry Manilow Greatest Hits	9,5	8,55
Listen Without Prejudice	9	8,1

Veja que foi acrescentado após o END do comando o nome que queremos para a coluna (VENDA). O banco de dados faz os cálculos com base nas condições descritas em CASE, ou seja, quando o preço de venda for menor que 10, será dado um desconto de 10%; se for maior ou igual a 10 e menor que 13, será dado 20% de desconto e se for maior ou igual a 13, será dado um desconto de 30%.

UPDATE

Podemos utilizar a mesma cláusula com o comando UPDATE. Assim, poderemos realizar atualizações com base em condições, simplificando a lógica de atualização dos dados. Veja a seguir um exemplo de como ficaria o banco de dados ao realizarmos a atualização proposta no comando SELECT anterior:

```
UPDATE CD
    SET PRECO_VENDA =
    CASE
    WHEN PRECO_VENDA < 10 THEN PRECO_VENDA * .9
    WHEN PRECO_VENDA >=10 AND PRECO_VENDA < 13 THEN
    PRECO_VENDA * .8
    ELSE PRECO_VENDA * .7
    END;

7 linhas atualizadas.
```

Vamos verificar como ficou o conteúdo do campo que atualizamos:

```
SELECT NOME_CD,PRECO_VENDA FROM CD;

NOME_CD                    PRECO_VENDA
-------------------------  -----------
Mais do Mesmo              10,5
Bate-Boca                  9,6
Elis Regina - Essa Mulher  9,1
A Força que nunca Seca     9,45
Perfil                     8,4
Barry Manilow Greatest Hits 8,55
Listen Without Prejudice   8,1
```

CASE compacto

É possível utilizar um teste CASE mais compacto que o demonstrado anteriormente. Para isso, basta colocarmos a coluna que queremos avaliar após a cláusula CASE e testarmos os valores após a cláusula WHEN. Observe que nesse caso é possível testar apenas igualdade, uma vez que nenhum operador poderá ser colocado ao lado do valor avaliado.

O exemplo a seguir é apenas didático, pois o correto seria unir as tabelas CD e GRAVADORA, e mostra o nome do CD e o nome da gravadora.

```
SELECT NOME_CD,
    CASE CODIGO_GRAVADORA
    WHEN 1 THEN 'EMI'
    WHEN 2 THEN 'BMG'
    WHEN 3 THEN 'SOM LIVRE'
    END
    FROM CD;
```

```
NOME_CD                    CASECODIG
-------------------------  ---------------
Mais do Mesmo              EMI
Bate-Boca                  BMG
Elis Regina - Essa Mulher  SOM LIVRE
A Força que nunca Seca     BMG
Perfil                     SOM LIVRE
Barry Manilow Greatest Hits BMG
Listen Without Prejudice   BMG
```

Expressões NULLIF

É comum precisar atualizar alguma coluna com conteúdo nulo (NULL) quando o valor da coluna não é mais válido. Para isso, utilizamos a expressão NULLIF. Imagine que não haja mais o CD de código 1 no estoque e, por esse motivo, queiramos anular as indicações a este CD. Veja como ficaria o comando:

```
UPDATE CD
    SET CD_INDICADO = NULLIF( CD_INDICADO,1 );

7 linhas atualizadas.
```

Note que as sete linhas foram "atualizadas". Na realidade, o que a expressão NULLIF faz é avaliar o conteúdo da coluna. Se o conteúdo de CD_INDICADO for 1, o comando NULLIF retornará NULL. Se for diferente de 1, será mantido o conteúdo da coluna. Veja como ficou a tabela após a atualização:

```
SELECT NOME_CD,CD_INDICADO FROM CD;
```

```
NOME_CD                    CD_INDICADO
-------------------------  -----------
Mais do Mesmo              5
Bate-Boca                  3
Elis Regina - Essa Mulher
A Força que nunca Seca
Perfil                     2
Barry Manilow Greatest Hits 7
Listen Without Prejudice
```

Expressões COALESCE

A função COASLESCE funciona como uma expressão CASE em que são testados valores diferentes de NULL. Dessa forma, o primeiro valor diferente de NULL será retornado pela função. Imagine a seguinte situação: caso queiramos a indicação de um CD e este esteja com valor NULL, então devemos retornar o código do mesmo CD. O comando ficaria assim:

```
SELECT CODIGO_CD,NOME_CD,CD_INDICADO,
    COALESCE(CD_INDICADO,CODIGO_CD) "INDICADO"
    FROM CD;
```

CODIGO_CD	NOME_CD	CD_INDICADO	INDICADO
1	Mais do Mesmo	5	5
2	Bate-Boca	3	3
3	Elis Regina - Essa Mulher		3
4	A Força que nunca Seca		4
5	Perfil	2	2
6	Barry Manilow Greatest Hits	7	7
7	Listen Without Prejudice		7

Veja que, onde há CD_INDICADO, a expressão retorna o conteúdo original da coluna. Onde não há nada cadastrado, a função retorna o código do CD. Podemos fazer isso com mais de um campo. Nesse caso, se o primeiro valor do COALESCE for nulo, será restado o segundo. Caso este seja nulo, será testado o terceiro e assim sucessivamente, até que não haja valores a serem testados. Nesse caso, será retornado o último conteúdo da função.

Exercícios propostos

1. Faça uma lista com todos os códigos e nomes de compradores e códigos e nomes de vendedores.

2. Faça uma lista com todos os códigos e nomes de compradores e códigos e nomes de vendedores, e admita que não há repetição das linhas.

3. Faça uma lista com todos os nomes de vendedores e endereço dos respectivos imóveis, e o comprador tenha o código 2 ou o bairro seja código 1.

4. Faça uma lista com todos os compradores que não tenham ofertas cadastradas.

5. Faça uma lista de todos os vendedores que não tenham imóveis cadastrados.

6. Faça uma lista com todos os vendedores que tenham imóveis cadastrados.

7. Faça uma lista com todos os compradores que tenham ofertas cadastradas.

8. Faça uma lista que mostre o código e o endereço do imóvel e aplique um desconto no preço do imóvel seguindo estas condições:

    a. Caso o preço do imóvel seja maior que 100 mil, desconto de 10%.

    b. Caso seja maior que 50 mil e menor ou igual a 100 mil, desconto de 5%.

    c. Caso seja maior que 30 mil e menor ou igual a 50 mil, desconto de 3%.

    d. Não aplicar desconto a imóveis com valor igual ou menor a 30 mil.

9. Faça uma lista que mostre o endereço do imóvel e o código do vendedor, e que, caso o campo STVENDIDO tenha conteúdo S, escreva VENDIDO, e do contrário, escreva DISPONÍVEL.

10. Faça uma lista que mostre o código do comprador e, caso a oferta tenha data inferior a 30 dias, apareça ANTIGA e, do contrário apareça NOVA.

Parte III
SQL avançado

12

Objetos avançados

Um banco de dados SQL não é composto apenas de tabelas, índices e comandos de manipulação de dados; há outros objetos igualmente importantes que permitem um controle maior das informações armazenadas no banco de dados. Neste capítulo serão identificados alguns desses elementos e para que servem.

Catálogo de sistema

O Catálogo de um sistema de banco de dados são as tabelas e visões em que estão armazenadas todas as informações importantes do banco de dados. Normalmente, a estrutura de um banco de dados e as informações do dicionário de dados são definidas na criação do banco de dados. A atualização do dicionário de dados é realizada pelo próprio gerenciador do banco de dados. Somente é possível consultar o que existe no catálogo de um sistema.

Dentro do banco de dados estão os objetos como tabelas, visões, índices, procedimentos e funções armazenadas. Também estão informações relacionadas ao desempenho do banco de dados. Todo administrador de banco de dados (DBA) necessita conhecer essas estruturas para desempenhar suas funções. Não há um padrão para definir o que deve ou não estar no catálogo do sistema e isso diferencia o que podemos ou não realizar em uma determinada implementação de banco de dados.

As principais informações armazenadas no catálogo de um sistema são as seguintes:

1. Usuários e direitos.

2. Estatísticas de desempenho do sistema.

3. Objetos (tabelas, visões, índices etc.).

4. Estrutura de Objetos (colunas de tabelas, visões, procedimentos, funções etc.).

5. Sessões de usuários.

6. Informações de auditoria.

7. Parâmetros de inicialização do banco de dados.

8. Localização física de arquivos de dados.

No banco de dados Oracle, existem algumas dezenas de estruturas desse tipo. Normalmente são visões que começam por ALL_, DBA_ e USER_ para os principais objetos do banco e V$ para as visões que determinam o desempenho do banco de dados.

No SQL Server e Sybase são as visões que começa com SYS.

Dessa forma, para obter as tabelas no banco de dados Oracle, digitamos:

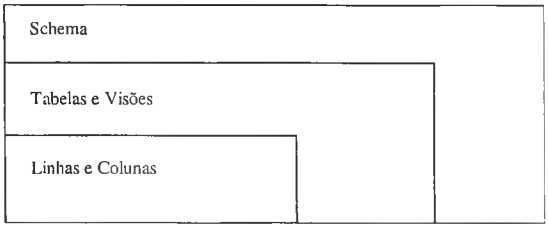
```
SELECT * FROM USER_TABLES;
```

No SQL Server ou Sybase:

```
SELECT * FROM SYSTABLES
    GO
```

Hierarquia de banco de dados

Os objetos de um banco de dados são armazenados seguindo a seguinte estrutura hierárquica:



As visões (objetos que veremos logo a frente neste capítulo) e as tabelas são compostas de linhas e colunas. Esses são objetos únicos que permitem armazenar informações necessárias ao funcionamento das operações da empresa. Por sua vez, essas tabelas e visões estão vinculadas a um *schema*. O *schema* seria o proprietário desses objetos. Acima desses níveis estaria a estrutura física de armazenamento (arquivos do sistema operacional) onde os *schemas* estariam vinculados.

O padrão SQL estabelece uma forma de armazenamento físico para bancos de dados efetivamente grandes. Isso ocorre porque nesse tipo de banco de dados é possível haver a duplicidade de *schemas* em um mesmo banco de dados. Para isso é criado um *catálogo* acima do *schema*. Dessa forma, o *catálogo* é um conjunto de *schemas*.

Essa estrutura contém implícita a necessidade de se manter a seguinte estrutura de nomes para os objetos de banco de dados:

a. Nomes de coluna devem ser únicos em cada tabela.

b. Nomes de objetos devem ser únicos em cada *schema*.

c. Nomes de *schema* devem ser únicos no banco de dados.

Em alguns bancos de dados, o usuário confunde-se com o *schema*. De qualquer forma, o conceito relacionado ao *schema* como proprietário de objetos de banco de dados é exatamente igual.

Notação de ponto

Para acessar a informação procurada e determinar exatamente qual *schema*, objeto e coluna, utilizamos a notação de ponto. Assim, informamos na seqüência proposta pelo padrão SQL qual será o catálogo, o *schema*, o objeto e a coluna. Assim:

```
catálogo.schema.objeto.coluna
```

Dessa forma, podemos concluir que duas tabelas podem ser criadas com o mesmo nome desde que em *schemas* diferentes. Isso também acontece com colunas em uma tabela, como visto anteriormente neste livro.

Nomes que podem ser associados à estrutura apresentada podem conter letras, números e o símbolo sublinha (_). Como dito anteriormente, SQL não é uma linguagem sensível a letras maiúsculas ou minúsculas, sendo portanto indiferente à forma como criamos os elementos.

Alguns bancos de dados permitem a utilização de outros símbolos, como cifrão ($) e "jogo da velha" (#). Estes, porém, devem ser evitados por não serem reconhecidos em pesquisas no SQL.

Schemas e usuários

Schema

Um *schema* representa um local onde se armazenam os objetos do banco de dados. A pessoa responsável pelo projeto do banco de dados coloca os elementos de significado comum dentro de um mesmo *schema*.

O *schema* pode ser entendido como um usuário que detém o poder sobre os objetos do banco de dados. Inclusive, em alguns bancos de dados a criação do usuário determina o *schema* com o mesmo conceito que utilizamos aqui. No caso do Oracle, apesar de possuir o comando CREATE SCHEMA, a utilização de *schema* não é a mesma definida aqui. O *schema* é o mesmo usuário do banco de dados.

Para criar um *schema*, utilizamos o comando:

```
CREATE SCHEMA nome_schema
   [AUTHORIZATION usuário];
```

O padrão é que o *schema* pertença ao próprio usuário que o criou. Contudo, podemos atribuir os direitos para outros usuários do banco de dados, utilizando a cláusula AUTHORIZATION. Em caso de não atribuir direitos a outros usuários, somente quem criou o *schema* é que poderá modificá-lo.

Quando se cria um *schema* é possível criar as tabelas ou visões relacionadas a ele, bastando colocar o respectivo comando de criação após a identificação do nome do *schema* ou usuário que recebe a autorização. Em algumas implementações de banco de dados SQL, devemos colocar o símbolo chaves entre os comandos de criação.

Exemplo:

```
CREATE SCHEMA teste
    AUTHORIZATION cd {
    CREATE TABLE tab1 (colChave INTEGER, colAlfa VARCHAR(50))
    }
```

Para trocar de *schema*, utilizamos o comando SET SCHEMA. O usuário deve ter direitos de trabalhar com esse *schema* para que o comando tenha sucesso:

```
SET SCHEMA schema;
```

Apenas bancos de dados que atendam todos os requisitos do padrão SQL possuem esse comando.

Usuários

Quando não há *schema* no banco de dados ou quando há, mas com outra significação, devemos criar usuários para que estes possuam os objetos do banco de dados. A sintaxe para criação de usuários é:

```
CREATE USER usuário
   IDENTIFIED BY senha;
```

Em alguns bancos de dados é permitido criar usuários independentes do sistema operacional, como Sybase e SQL-Server. Em outros, os usuários são dependentes do sistema operacional, como o DB2. O Oracle permite tanto criar usuários específicos para o banco de dados quanto utilizar o usuário definido no sistema operacional.

Quando criamos ou excluímos algum objeto do banco de dados, desde que tenhamos direito de fazê-lo, podemos especificar o usuário ou *schema* antes do nome do objeto. Funciona como na notação de ponto explicada anteriormente neste capítulo. Exemplo:

```
CREATE TABLE  cd.tab1
    (colChave  INTEGER,
    colAlfa  VARCHAR(50));
```

Domínios

Um domínio é uma expressão que permite a atribuição de determinados valores às colunas. O padrão SQL introduziu o conceito de utilizar um conjunto de valores possíveis para determinar o conteúdo das colunas de uma tabela. Isso é particularmente útil para conteúdos predeterminados como sexo (Masculino ou Feminino), estágio de pedidos, notas fiscais etc.

Alguns bancos de dados permitem definir o domínio na criação da própria tabela. Dessa forma, o domínio é uma CONSTRAINT.

Para criar um domínio, utilizamos o comando CREATE DOMAIN:

```
CREATE DOMAIN nome tipo_de_dado
   CHECK (expressão);
```

Para detalhes na utilização da cláusula CHECK, veja no capítulo 4 o tópico Criação de Tabelas.

Depois de criar um domínio, qualquer coluna de tabelas poderá ser do tipo de dado criado. Em vez de criar a coluna como INTEGER, VARCHAR etc., podemos criar com o domínio. Dessa forma, a coluna terá o mesmo tipo de dado e tamanho do domínio e terá seu conteúdo limitado ao que estiver definido no domínio.

Visões

Visões são um modo especial de enxergar os dados de uma ou várias tabelas.

Uma visão (*view*) é um objeto de banco de dados criado a partir de um comando SELECT. É armazenado no dicionário de dados do banco de dados e possui a mesma estrutura de uma tabela. A diferença consiste na ausênciade de linhas na visão. A visão faz referências a linhas e colunas de uma ou mais tabelas, ou até de outras visões, com base no comando SELECT que a criou.

Por ser criada com base em um comando SELECT, seus dados sempre estarão atualizados. Ao buscar as linhas e colunas de uma visão estamos, na realidade, realizando a busca diretamente nas tabelas de origem. Qualquer mudança nas tabelas base é refletida imediatamente na visão.

Por que utilizar visões?

a. Restringir acesso aos dados: utilizando o comando SELECT, podemos filtrar linhas e colunas que não devam ser mostradas a todos os usuários.

b. Buscas complexas tornam-se simples: em vez dos usuários terem que realizar complexos comandos SELECT para localizar as informações de que eles necessitam, nós, analistas, podemos criar visões desses comandos complexos. Assim, quando os usuários precisarem da informação, farão a busca na visão.

c. Independência de dados: as visões podem ser criadas visando a atender necessidades genéricas e não apenas às relacionadas a pessoas ou programas. Podemos buscar informações de diversas tabelas.

d. Dados mostrados de maneira diferente: usuários ou grupo de usuários pode(m) ver as mesmas informações armazenadas no banco de dados, mas cada um com a visão que lhe interessa.

e. Eliminação de códigos: normalmente os códigos não devem ser mostrados aos usuários, pois são utilizados apenas para estabelecer relacionamentos entre tabelas. Ao eliminar os códigos nas visões, podemos simplificar o que os usuários estão vendo e facilitar a compreensão dos dados.

Classificação

As visões podem ser classificadas por:

a. Simples:

- Os dados são extraídos de uma única tabela.
- Não contêm funções.
- Não possuem dados agrupados.
- Podem utilizar comandos DML para manipular os dados.

b. Complexas:

- Os dados são extraídos de várias tabelas.
- Podem conter funções.
- Podem conter dados agrupados.
- Não podem utilizar comandos DML para manipulação dos dados. No banco de dados Oracle é possível realizar comandos DML em visões complexas, desde que utilizando um gatilho específico. Maiores detalhes veja o Guia Oracle 8*i* PL/SQL deste mesmo autor e editora.

Criação da visão

Para criar uma visão, utilizamos o comando CREATE VIEW:

```
CREATE VIEW nome
   AS subquery;
```

Exemplo:

```
CREATE VIEW  vCD  AS
    SELECT  CODIGO_CD, NOME_CD, PRECO_VENDA
    FROM  CD;
```

A visão criada conterá apenas as colunas definidas no comando SELECT. As colunas terão o mesmo nome das colunas da tabela CD. Para modificar o nome, podemos utilizar o comando a seguir:

```
CREATE VIEW  vCD (CODIGO,NOME,PRECO) AS
    SELECT  CODIGO_CD, NOME_CD, PRECO_VENDA
    FROM  CD;
```

Assim a coluna CODIGO_CD terá nome CODIGO, NOME_CD será NOME e PRECO_VENDA apenas PRECO. Quando queremos trocar o nome de qualquer coluna na visão, devemos informar o nome de todas as colunas entre parênteses. Não é possível informar apenas um ou outro nome de coluna.

Como especificado anteriormente, essa é uma visão simples. Como uma visão é sempre baseada em um comando SELECT, podemos criar visões extremamente complexas.

Exemplos:

```
CREATE VIEW  vPRECO_CD AS
     SELECT CODIGO_GRAVADORA,NOME_CD,PRECO_VENDA
     FROM CD
     WHERE CODIGO_GRAVADORA = (SELECT CODIGO_GRAVADORA
     FROM CD
     WHERE CODIGO_CD = 2)
     AND PRECO_VENDA > (SELECT PRECO_VENDA
     FROM CD
     WHERE CODIGO_CD = 5);

CREATE VIEW  vCD_FAIXA AS
     SELECT CODIGO_CD,NUMERO_FAIXA
     FROM FAIXA
     WHERE CODIGO_MUSICA IN
     (SELECT CODIGO_MUSICA
     FROM MUSICA
     WHERE UPPER(NOME_MUSICA) LIKE '%AMOR%');

CREATE VIEW  vMAX_GRAVADORA AS
     SELECT CODIGO_GRAVADORA,MAX(PRECO_VENDA) VENDA
     FROM CD a
     GROUP BY CODIGO_GRAVADORA
     HAVING MAX(PRECO_VENDA) >
     (SELECT AVG(PRECO_VENDA)
     FROM CD
     WHERE CODIGO_GRAVADORA = a.CODIGO_GRAVADORA);
```

Observe que nesse último comando precisamos acrescentar o apelido para a coluna MAX(PRECO_VENDA). Isso porque toda coluna necessita ter um nome. Como não é possível criar colunas com esse nome, devemos indicar ao banco de dados o nome da coluna.

Buscando conteúdo de visões

Utilizamos o comando SELECT para buscar o conteúdo de visões exatamente como fazemos com qualquer tabela. Podemos, inclusive, combinar visões com tabelas ou com outras visões, criando comandos efetivamente complexos.

Exemplos:

```
SELECT * FROM vCD;

CODIGO  NOME                          PRECO
------  ----------------------------  -----
1       Mais do Mesmo                 15
2       Bate-Boca                     12
3       Elis Regina - Essa Mulher     13
4       A Força que nunca Seca        13,5
5       Perfil                        10,5
6       Barry Manilow Greatest Hits   9,5
7       Listen Without Prejudice      9

SELECT * FROM vCD_FAIXA;

CODIGO_CD   NUMERO_FAIXA
---------   ------------
2           9
4           1
```

E podemos acrescentar qualquer outra cláusula do comando SELECT, mesmo em pesquisas realizadas em visões:

```
SELECT * FROM vPRECO_CD
    WHERE PRECO_VENDA > 12;

CODIGO_GRAVADORA    NOME_CD                 PRECO_VENDA
----------------    ----------------------  -----------
2                   A Força que nunca Seca  13,5
```

Utilização de comandos DML em Visões

Conforme explicado anteriormente, somente em visões simples é possível utilizar comandos DML. O padrão SQL determina as condições em que uma visão pode ser atualizada:

- Deve ser criada com base em uma única tabela.
- Deve conter apenas um SELECT (excluem-se cláusulas UNION, por exemplo).
- Se foi criada com base em outra visão, a primeira visão deve ser passível de atualização.
- O comando SELECT não pode conter colunas calculadas, somente colunas originais da tabela.
- Não deve utilizar GROUP BY no comando SELECT.
- Não deve conter a cláusula DISTINCT.
- Pode conter uma *subquery* desde que o SELECT interno tenha como base a mesma tabela do SELECT externo.
- Inclusões só podem ser feitas caso a visão contenha a chave primária da tabela base.
- Exclusões e alterações em visões sem a chave primária da tabela base são permitidas, porém desaconselháveis, pois não se sabe ao certo o resultado que pode aparecer.

Excluindo uma visão

Para excluir uma visão, utilizamos o comando DROP VIEW. Observe que ao excluir uma visão, não estamos excluindo os dados (linhas e colunas), visto que as linhas e colunas permanecem nas tabelas.

```
DROP VIEW view;
```

Localizando TOP-N ocorrências

A técnica demonstrada a seguir é útil para obtermos as maiores ou menores ocorrências de uma coluna em uma tabela. Para isso, utilizamos o conceito de visão de linha. Veja um exemplo em que foram extraídos os CDs com os três maiores preços de venda e determinado um "ranking" para o de maior e o de menor preço:

```
SELECT ROWNUM as RANK, NOME_CD, PRECO_VENDA
  FROM (SELECT NOME_CD,PRECO_VENDA
          FROM CD
         ORDER BY PRECO_VENDA DESC)
 WHERE ROWNUM <= 3;
```

```
RANK    NOME_CD                    PRECO_VENDA
----    -------------------------  -----------
1       Mais do Mesmo              15
2       A Força que nunca Seca     13,5
3       Elis Regina - Essa Mulher  13
```

Para solucionar problemas comuns de paginação de informações na Internet, por exemplo, podemos utilizar uma combinação destes comandos:

```
SELECT ROWNUM as RANK, NOME_CD, PRECO_VENDA
    FROM (SELECT NOME_CD,PRECO_VENDA
    FROM CD
    ORDER BY PRECO_VENDA DESC)
    WHERE ROWNUM <=5
    MINUS
    SELECT ROWNUM as RANK, NOME_CD, PRECO_VENDA
    FROM (SELECT NOME_CD,PRECO_VENDA
    FROM CD
    ORDER BY PRECO_VENDA DESC)
    WHERE ROWNUM <=3;
```

```
RANK    NOME_CD     PRECO_VENDA
----    ----------  -----------
4       Bate-Boca   12
5       Perfil      10,5
```

Esse comando foi testado no banco de dados Oracle, onde ROWNUM representa o número da linha apresentada. Caso essa função não exista no seu banco de dados, verifique outra que faça o mesmo.

Tabelas temporárias

Uma tabela temporária funciona como uma tabela normal, porém como sua existência é temporária, depois de encerrada a seção no banco de dados, ela será automaticamente destruída. Aparentemente ela pode ser confundida com visões, mas existem várias diferenças entre elas:

Visão	Temporária
Existência condicionada ao comando SELECT	Existe enquanto a seção estiver ativa.
Conteúdo preenchido automaticamente	Deve-se acrescentar conteúdo com comando INSERT.
Critério de atualização de dados rígido	Critério de atualização igual ao de qualquer outra tabela.
Conteúdo sempre atualizado	Reflete o conteúdo do instante em que foi buscada a informação.

A decisão de utilizar uma ou outra construção deve ser concentrada na questão desempenho e necessidade de atualização. Como a visão tem seu conteúdo atualizado automaticamente por buscar o conteúdo no exato momento em que se executa o comando SELECT, o desempenho tende a ser menor que o das tabelas temporárias. Contudo, as tabelas temporárias não terão o conteúdo atualizado automaticamente, o que melhora o desempenho, mas pode demonstrar dados desatualizados.

Para buscas que demorem muito, ou que sejam repetidas e não haja necessidade do "último dado", a utilização das tabelas temporárias pode ser mais eficiente.

Criação de tabela temporária

Para criar uma tabela temporária, utilizamos o comando CREATE TEMPORARY TABLE. Uma tabela temporária pode ser GLOBAL ou LOCAL. Quando definida como GLOBAL, todos os usuários poderão ter acesso a ela. Se for LOCAL, somente a seção que a criou terá acesso.

```
CREATE {GLOBAL|LOCAL} TEMPORARY TABLE tabela
   (coluna tipo_dado, ... );
```

Exemplo:

```
CREATE GLOBAL TEMPORARY TABLE tempCD
    (CODIGO INTEGER,
    NOME   VARCHAR(50),
    PRECO  DECIMAL(16,2),
    PRIMARY KEY (CODIGO));
```

Manipulando linhas na tabela temporária

Devemos utilizar os comandos DML como em qualquer outra tabela do banco de dados. Exemplo:

```
INSERT INTO tempCD
    SELECT CODIGO_CD, NOME_CD, PRECO_VENDA
    FROM CD;

SELECT * FROM TEMPCD;

CODIGO  NOME                          PRECO
------  ----------------------------  -----
1       Mais do Mesmo                 15
2       Bate-Boca                     12
3       Elis Regina - Essa Mulher     13
4       A Força que nunca Seca        13,5
5       Perfil                        10,5
6       Barry Manilow Greatest Hits   9,5
7       Listen Without Prejudice      9
```

Exercícios propostos

1. Explique como funciona a notação de ponto para determinar *schema*, tabela e coluna.

2. Crie um *schema* chamado LOJA e, dentro do *schema*, crie uma tabela chamada SEDE, com código e nome da loja.

3. Crie um domínio para determinar o sexo.

4. Crie uma visão simples com o código, nome e endereço dos vendedores.

5. Mostre o conteúdo dessa visão.

6. Faça uma visão que contenha o código, o endereço e o preço de venda dos imóveis do bairro 1. Dê-lhe o nome de vIMOVEL_BAIRRO1.

7. Faça uma visão que mostre o nome do vendedor, o endereço, o bairro, a cidade e o Estado do imóvel. Não deixe nenhum código na visão. Dê-lhe o nome de vIMOVEL.

8. Faça uma visão com o preço médio de venda dos imóveis agrupados por cidade. Dê-lhe o nome de vIMOVEL_CIDADE.

9. Faça uma visão com os nomes dos compradores, o valor de oferta e o endereço do imóvel. Não coloque nenhum código na visão. Dê-lhe o nome de vOFERTA.

10. Faça uma tabela temporária global igual à do exercício 4.

11. Inclua todos os vendedores com código menor que 5 nessa tabela temporária.

12. Mostre o conteúdo dessa tabela.

13. Faça um *ranking* com os 3 imóveis mais baratos.

14. Faça um *ranking* com as 5 melhores ofertas, paginando-as de 2 em 2.

13

Segurança de bancos de dados e transações

No capítulo 12, verificamos como criar usuários no banco de dados e estabelecer um critério de segurança de acesso às informações. Uma vez criado o usuário, ele não possui qualquer direito sobre o banco de dados. Para que se possa acrescentar e revogar direitos aos usuários, devemos utilizar comandos específicos.

Segurança é um item primordial que diferencia banco de dados SQL de outros bancos de dados. Por meio de critérios rígidos de controle, podemos ter uma informação segura e que apenas usuários autorizados podem alterá-la.

Embora os direitos variem de um banco de dados para outro, os comandos relacionados à atribuição e à revogação de direitos são os mesmos. Vamos nos concentrar, portanto, nos comandos e não no tipo de direito.

Normalmente os bancos de dados vêm com um ou dois usuários (SYS, SYSTEM, DBA ou SA são exemplos desses usuários) que têm poder sobre todo o banco de dados. Portanto, devemos utilizar esses usuários para criar outros usuários e estes, por sua vez, receberão direitos de sistema e concederão direitos de objeto a outros usuários.

Classificação dos direitos

Existem duas classificações para os direitos no banco de dados:

1. Objeto: diz respeito aos objetos que um *schema* possui.
2. Sistema: diz respeito ao que se pode realizar no banco de dados.

Direitos de objeto

Como é o *schema* que possui os objetos, somente ele poderá conceder os direitos sobre seus objetos a outros usuários do banco de dados. Assim, os demais usuários somente poderão realizar o que lhes for permitido pelo proprietaário do objeto.

Privilégio de Objeto	Tabela	Visão
ALTER	SIM	
DELETE	SIM	SIM
INSERT	SIM	SIM
REFERENCES	SIM	
RENAME	SIM	SIM
SELECT	SIM	SIM
UPDATE	SIM	SIM

Tipos de direitos

Existem seis tipos básicos de direitos de acesso (podendo haver mais ou menos, dependendo do banco de dados):

1. SELECT: permite extrair dados das tabelas ou visões.
2. INSERT: permite incluir dados em tabelas ou visões. Alguns bancos de dados permitem que apenas algumas colunas possam ser atualizadas.
3. UPDATE: permite modificar linhas nas tabelas. Alguns bancos de dados permitem que apenas algumas colunas possam ser atualizadas.
4. DELETE: permite excluir linhas de tabelas ou visões.
5. REFERENCES: permite criar CONSTRAINT de chave estrangeira.
6. ALL PRIVILEGES: concede todos os privilégios anteriores.

Atribuindo direitos de objeto

Para atribuir direitos a outros usuários, utilizamos o comando GRANT.

```
GRANT tipo_direito
   ON tabela
   TO {usuário|PUBLIC}
   [WITH GRANT OPTION];
```

Quando utilizamos a cláusula WITH GRANT OPTION, significa que será transferida ao usuário que recebe o direito a possibilidade de ele dar os mesmos direitos que recebeu a outros usuários. Exemplo:

```
GRANT SELECT
    ON CD
    TO SCOTT;

GRANT SELECT
    ON CD
    TO SCOTT WITH GRANT OPTION;
```

Caso desejemos atribuir direitos de pesquisa em toda tabela e alteração somente na coluna NOME_CD, utilizaremos o seguinte comando:

```
GRANT SELECT, UPDATE (NOME_CD)
    ON CD
    TO SCOTT;
```

Caso queiramos atribuir direitos a todos os usuários do banco de dados, utilizamos PUBLIC em vez do nome do usuário:

```
GRANT SELECT
    ON CD
    TO PUBLIC;
```

Revogando direitos de objeto

Para revogar direitos anteriormente concedidos, utilizamos o comando REVOKE:

```
REVOKE tipo_direito
   ON tabela
   FROM usuário;
```

Para revogar o direito de pesquisa na tabela CD, utilizamos o comando:

```
REVOKE SELECT
    ON CD
    FROM SCOTT;
```

Direitos de sistema

Estes direitos também variam muito em função do banco de dados utilizado. Contudo, os mais comuns são:

a. CONNECT: permite ao usuário conectar-se ao banco de dados. Usuários que não tem esse direito sequer podem acessar o banco de dados.

b. RESOURCE: permite ao usuário, além de conectar-se, criar objetos no banco de dados.

c. DBA: permite ao usuário controlar todos os objetos do banco de dados. É um direito que deve ser concedido com muito critério para evitar problemas ao banco de dados. Recomenda-se que apenas o administrador do banco de dados o possua.

Privilégio de Sistema	Usuário	Índice	Procedimento	Tabela	Visão
ALTER	SIM	SIM	SIM	SIM	
CREATE	SIM	SIM	SIM	SIM	SIM
DROP	SIM	SIM	SIM	SIM	SIM
SELECT				SIM	
EXECUTE			SIM		
UPDATE				SIM	

Atribuindo direitos de sistema

Utilizamos o mesmo comando GRANT:

```
GRANT privilégio
   TO usuário
   [WITH ADMIN OPTION];
```

Exemplo:

```
GRANT CONNECT TO SCOTT;
```

Ao utilizarmos a cláusula WITH ADMIN OPTION, estamos dando direito ao usuário de transferir o mesmo direito recebido a outros usuários.

Revogando direitos de sistema

Utilizamos o mesmo comando REVOKE:

```
REVOKE privilégio
   FROM usuário;
```

Sessões de banco de dados

Quando um usuário interage com o banco de dados, ele precisa estar conectado a ele. O tempo compreendido entre a entrada e a saída do banco de dados é denominado sessão do banco de dados.

Para conectar-se ao banco de dados, utilizamos o comando:

```
CONNECT
```

Como cada banco de dados possui sua própria ferramenta de acesso, quando não forem pedidos usuário e senha em uma janela-padrão, utilizar o comando CONNECT deve ser o suficiente. Normalmente, são solicitados usuário e senha após esse comando.

Para desconectar-se do banco de dados, utilizamos o comando:

```
DISCONNECT
```

Transações e níveis de isolamento

Uma transação é uma unidade de trabalho submetida ao banco de dados. É comum que mais de um usuário realize transações concorrentes, ou seja, ao mesmo tempo. Teoricamente, seria possível determinar transações que fossem realizadas em *série* (cada transação completa o trabalho antes que a outra inicie) ou *inter-relacionada* (em que as ações de todas as transações são alternadas).

O resultado de operações inter-relacionadas pode não ocasionar nenhum problema e concluir-se como em uma operação em série. Mas imagine que duas pessoas realizem operações iguais (consulta de estoque de material, por exemplo) em frações de segundo diferentes. Corre-se o risco de a última das consultas retornar a quantidade errada em função da atualização do primeiro usuário. Para isso, deve haver um controle de bloqueio de linhas.

Contudo, esse bloqueio pode afetar o desempenho do banco de dados, visto que o segundo usuário seria deixado em *wait state* (estado de espera) até que a primeira transação fosse concluída. Esse tipo de bloqueio é conhecido por *exclusive lock*. Há ainda bloqueios do tipo *shared* e *read*. O padrão é *shared*, para permitir o máximo de operações concorrentes sem que haja bloqueio das operações.

Dependendo do banco de dados, pode haver bloqueios de linha, páginas e tabelas. Quanto menor o nível do bloqueio (no caso, linha) mais operações concorrentes podem ocorrer sem haver perda de disponibilidade do banco de dados. Há bancos de dados que prometem bloqueio de colunas em futuro próximo. Entretanto, isso naturalmente faz com que o gerenciador do banco de dados tenha que trabalhar mais e, conseqüentemente, consumir mais tempo para realizar a tarefa.

Foi visto anteriormente neste livro que uma transação no banco de dados pode ser encerrada de duas formas: gravando ou restaurando a situação anterior. O comando para gravar é o COMMIT e para restaurar é o ROLLBACK. Deve-se considerar que o ROLLBACK recuperará todas as transações efetuadas desde o último COMMIT. Logo, é conveniente que constantemente sejam gravadas as informações no banco de dados, pois não há como selecionar quais operações não devem ser gravadas.

SAVEPOINT e ROLLBACK TO

Este comando identifica um ponto de identificação para transações. Ao utilizá-lo é possível reverter a transação até o ponto especificado. Isso faz com que não seja necessário reverter toda a transação desde o último COMMIT.

```
SAVEPOINT nome_savepoint;
```

O nome utilizado para identificar a transação deve ser exclusivo para as transações associadas.

Para que uma transação seja revertida até o ponto especificado pelo comando SAVEPOINT, utilizamos o comando:

```
ROLLBACK TO nome_savepoint;
```

Exemplo de utilização:

```
INSERT INTO AUTOR (CODIGO_AUTOR,NOME_AUTOR)
    VALUES ( 61, 'Zé Ramalho');

1 linha criada.
```

```
SAVEPOINT svp1;
Ponto de salvamento criado.

INSERT INTO MUSICA(CODIGO_MUSICA,NOME_MUSICA)
    VALUES (89,'Canção da América');
1 linha criada.

SAVEPOINT svp2;
Ponto de salvamento criado.

DELETE FROM MUSICA
    WHERE CODIGO_MUSICA NOT IN
    (SELECT CODIGO_MUSICA
    FROM FAIXA);
1 linha deletada.

ROLLBACK TO svp2;
Rollback completo.

SELECT * FROM MUSICA WHERE CODIGO_MUSICA = 89;

CODIGO_MUSICA   NOME_MUSICA
-------------   ----------------
89              Canção da América

SELECT * FROM AUTOR WHERE CODIGO_AUTOR = 61;

CODIGO_AUTOR    NOME_AUTOR
------------    ----------
61              Zé Ramalho
```

Observe que, como foi feito um ROLLBACK até o ponto SVP2, o autor permanece na tabela, pois o INSERT está antes do ponto SVP2. A música também está na tabela porque ela foi incluída antes do SVP2 e foi excluída depois desse ponto. Como voltamos à transação, apenas a inclusão da linha foi mantida. A exclusão não foi considerada.

Níveis de isolamento

O padrão SQL determina que se possa especificar quatro níveis de transações:

1. SERIALIZABLE: uma transação é totalmente isolada das outras transações. Caso a transação tenha comandos DML que tentem atualizar dados não gravados de outra transação, essa transação não será efetuada.

2. READ COMMITED: caso a transação utilize DML que precise do bloqueio de linhas que outras transações estão utilizando, a operação somente será concluída após a liberação da linha da outra transação.

3. REPEATABLE READ: os dados podem ser lidos mais de uma vez, e se outra transação tiver incluído ou atualizado linhas e estas forem gravadas no banco de dados entre uma e outra leitura dos dados, então os dados retornados da última busca serão diferentes dos dados da busca anterior. Esse efeito é conhecido como *fantasma*.

4. READ UNCOMMITED: conforme o nome diz, serão lidos conteúdos não gravados ainda pelo banco de dados (transações passíveis de ROLLBACK). Há um enorme risco nessas operações, visto que o usuário que está bloqueando a informação pode descartá-la.

O comando utilizado para determinar o nível de isolamento é:

```
SET TRANSACTION ISOLATION LEVEL nível;
```

Exemplo:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Observe que devemos utilizar esse comando antes de efetuar as transações no banco de dados.

É possível determinar a forma que as transações vão ocorrer no banco de dados. O padrão é permitir a leitura e a gravação dos dados, mas deixar o banco de dados em modo só de leitura, quando se extraem relatórios por exemplo, em que as transações jamais vão envolver bloqueios. Isso é interessante para permitir que mais usuários possam trabalhar concorrentemente.

Isso é feito utilizando o mesmo comando, mas com cláusulas específicas somente para leitura e leitura e gravação:

```
SET TRANSACTION READ ONLY;
SET TRANSACTION READ WRITE;
```

O padrão SQL não determina um comando para início ou fim de transações. Normalmente, isso é implícito assim que uma nova transação é iniciada na seção do banco de dados. Alguns bancos de dados, contudo, utilizam os comandos START WORK ou START TRANSACTION para determinar o início de uma transação.

Determinando quando CONSTRAINTs são verificadas

O padrão SQL determina que os bancos de dados tenham a habilidade de checar quando as CONSTRAINTs devem ser verificadas: em cada comando SQL ou no final da transação. Isso pode ser particularmente útil quando queremos realizar alterações que envolvam várias tabelas em cascata. Ao colocar o banco de dados em verificação, apenas no final da transação elimina-se o problema dessas modificações.

Para isso, utilizamos o seguinte comando:

```
SET CONSTRAINTS [MODE]
   {constraint|ALL}
   {IMMEDIATE|DEFERRED};
```

O padrão dos bancos de dados é IMMEDIATE. Quando especificamos uma *constraint,* apenas ela será afetada pelo comando, as demais permanecerão com o padrão do banco de dados.

Exercícios propostos

1. Crie um usuário no banco de dados.
2. Atribua direitos de conexão a esse usuário.
3. Atribua direitos de consulta na tabela comprador e vendedor.
4. Atribua direitos de inclusão e atualização nas tabelas oferta e imóvel.
5. Atribua direitos de exclusão nas tabelas cidade, Estado e bairro.
6. Revogue os direitos concedidos no exercício 5.
7. Altere sua sessão para que as transações sejam em série.
8. Altere sua sessão para que as constraints sejam verificadas apenas no final das transações.

14

Entendendo o padrão SQL

Até aqui foi vista a parte essencial do padrão SQL. Vamos nos aprofundar um pouco nos motivos que tornaram esse padrão um sucesso. Iniciamos com a evolução até o ambiente Cliente-Servidor, o novo paradigma que são as aplicações para Internet e o novo padrão SQL-99 ou SQL-3.

Cliente-Servidor

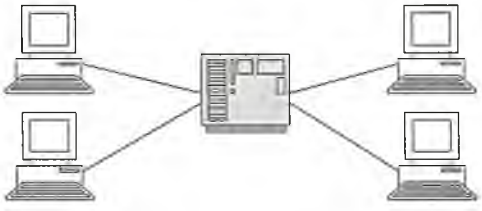
Arquitetura baseada em Hosts

Nas décadas de 1960 e 1970 a utilização da informática nas poucas empresas que utilizavam computadores era baseada em um servidor de grande porte ou *mainframes*. Esse computador centralizava todas as operações e armazenava todos os dados de todas as aplicações. Os usuários tinham acesso às informações do servidor por meio de terminais compostos de um vídeo e um teclado. Nesses terminais não havia processadores que efetuassem processamento local, todo o processamento era transferido para o computador central.

Era comum que um único fornecedor colocasse toda a infra-estrutura de informática na empresa. Assim, a empresa dependia de um único fornecedor para todo setor de informática. Os sistemas de outros fabricantes não se comunicavam uns com os outros.

Como é natural, a computação centralizada oferece ganhos significativos de controle e acesso a dados e aplicações. A administração do servidor, sendo centralizada, oferecia condições muito melhores.

A grande desvantagem dessa arquitetura é que a centralização das informações compromete a disponibilidade do sistema como um todo. Da mesma forma, o distanciamento do usuário final causava atrasos na entrega de documentos e relatórios importantes para o gerenciamento da empresa. Sistemas simples e necessários para a empresa demoravam muito tempo para serem disponibilizados.



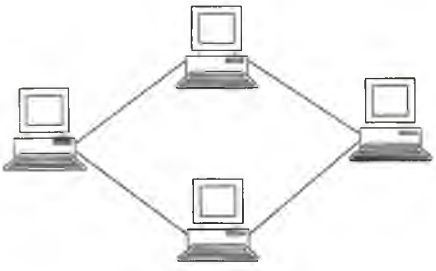
Arquitetura baseada em redes de PCs

A partir da década de 1980, quando começaram a surgir os primeiros computadores pessoais (PC), isso começou a mudar. Os usuários passaram a ter terminais que eram capazes de realizar pequenos processamentos locais. Empresas que não podiam adquirir computadores de grande porte começaram a adquirir esses equipamentos menores e mais baratos, mas que eram capazes de realizar algumas atividades importantes para agilizar o fluxo de informações na empresa.

Logo esses computadores começaram a ser interligados por meio de redes locais. Assim, recursos desses computadores começaram a ser compartilhados (discos, impressoras, fax etc.). As empresas maiores trabalhavam com dois ambientes distintos: *mainframes* e redes de PCs.

A descentralização das informações e processos é a tônica desse novo ambiente. Os usuários passam a contar com uma interface gráfica muito mais atraente que nos antigos terminais. Como os equipamentos são autônomos, a falha em um dos computadores não afetava os demais. A disponibilidade da informação passou a ser maior que no modelo anterior.

A maior desvantagem está na dificuldade de administração, pois os dados e as aplicações estão distribuídos e nem sempre são desenvolvidos de maneira técnica e confiável. Dessa forma, era muito comum que informações e processos fossem repetidos na empresa. Várias pessoas faziam o mesmo trabalho em setores diferentes.



Arquitetura baseada em cliente-servidor

Os dois ambientes apresentavam vantagens e desvantagens. Uma nova necessidade surgia: como integrar esses ambientes? Como possibilitar aos usuários usufruir o melhor dos dois mundos?

A arquitetura cliente-servidor baseia-se na distribuição dos vários componentes de um sistema ou aplicação. Essa distribuição obedece ao critério de que os componentes consumidores de recursos são os Clientes e os fornecedores desses recursos são os Servidores.

O equipamento desse ambiente é composto de computadores que consomem serviços e outros que os fornecem. Os vários computadores são conectados por meio de uma rede local ou remota. Deve-se permitir a conexão dos mais diversos tipos de computadores, desde os PCs até os *mainframes*. Deve ser possível utilizar diferentes tipologias de rede e qualquer protocolo de comunicação.

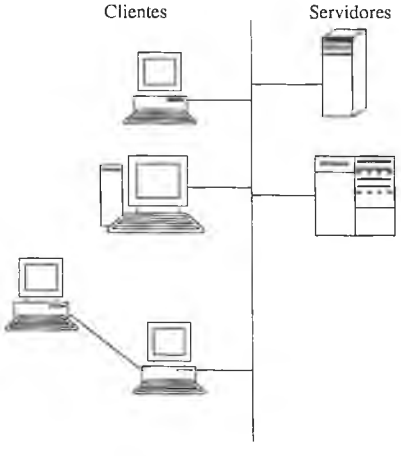
Por sua vez, redes menores podem ser montadas com base nos computadores que compõem a rede. Assim, redes maiores interligam-se a redes menores para fornecer informações departamentais integradas à empresa.

Uma outra característica importante nesse ambiente é a possibilidade de evoluir com o equipamento de acordo com as necessidades da empresa. Uma empresa pequena começa com um único computador. À medida que aumenta a necessidade de processamento e informação, a empresa pode adquirir novos computadores e montar sua rede.

Outra característica diz respeito ao *software* utilizado nesse ambiente. Ele deve ser capaz de realizar parte da aplicação no servidor e outra parte no cliente. Assim, extrai-se o que há de melhor nos dois ambientes (redes e *host*).

Como parte da aplicação é executada no cliente e parte no servidor, obtém-se a produtividade dos dois ambientes: controle centralizado e produtividade distribuída. Regras de integridade dos dados e de gerência do negócio devem ficar no servidor e as regras da aplicação e interação com usuário, no cliente.

Esse ambiente traz também a idéia de sistemas abertos. Assim, não é necessário ficar preso a um único fornecedor de soluções. Pode-se comprar o sistema operacional de um fornecedor, o banco de dados de outro e a linguagem de programação de outro. Com essa possibilidade, nasceu a filosofia de padronizar alguns elementos de hardware e software. O padrão SQL surgiu com essa iniciativa: padronizar a manipulação e o acesso aos bancos de dados.



Arquitetura de n-camadas – Internet/Intranet

Com o advento da Internet, os sistemas cliente-servidor passaram a ter mais uma possibilidade. Um novo nível de servidor de sistema surgiu com a oportunidade de ligar diversos computadores em uma rede única. Observe que essa rede não está mais limitada às conexões físicas locais. O mundo tem o tamanho da rede. Ainda que no ambiente cliente-servidor as redes pudessem ser interligadas utilizando os diversos meios de comunicação disponíveis, nesse novo ambiente, com uma simples linha telefônica, é possível conectar-se a qualquer computador que esteja disponível em qualquer lugar do planeta.

Uma intranet nada mais é do que uma pequena Internet. Contém as mesmas características, apenas não sendo disponibilizadas as informações para a rede mundial de computadores.

Middleware é o nome que se dá a esse novo nível que surgiu com a Internet. Esse bloco assume algumas das responsabilidades antes delegadas ora ao cliente, ora ao servidor. Assim, tanto o cliente quanto o servidor podem concentrar-se nas suas próprias operações, mantendo-se mais leves do que seriam caso não houvesse esse componente.

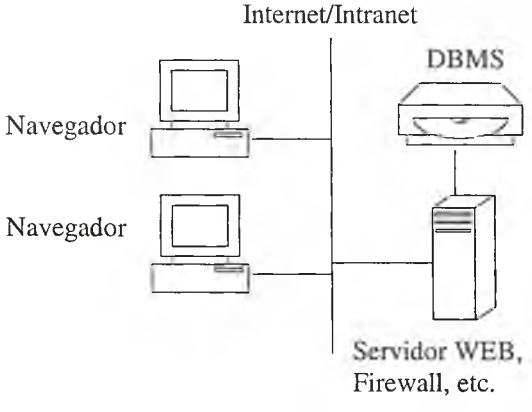
Com clientes mais leves, pode-se estender a utilização da informática a qualquer parte do mundo. Rodando apenas um *software* básico (um navegador do tipo Internet Explorer ou Netscape), é possível acessar informações e executar aplicações na Internet.

Com esse novo ambiente, é possível extrair ainda mais vantagens da computação baseada em *hosts* em virtude da centralização das operações em servidores e aumentar a disponibilidade das informações ao usuário final, que nem por isso tem que abdicar de uma interface agradável.

Um aspecto extremamente importante a se notar quando se disponibilizam informações pela Internet é a questão segurança. Para isso, deve ser montada uma estrutura de controle de acesso. Veja alguns pontos que devem ser levados em consideração antes de implementar um banco de dados na Internet:

1. Banco de dados com segurança própria, senhas, usuários e encriptação dos dados (método de codificar os dados para dificultar a compreensão destes).

2. Firewall: programa que realiza a segurança de conexões não autorizadas à rede da empresa.



SQL e Internet

Desde a sua criação, o SQL foi projetado para funcionar e trabalhar em rede. Sejam as redes locais, remotas ou mesmo toda a Internet, pode utilizar esse padrão para acessar informações em qualquer lugar com rapidez e confiabilidade. Normalmente os bancos de dados necessitam que a aplicação seja executada no servidor e que uma camada de aplicação seja executada no cliente. Essa camada costuma ser relativamente leve e contém uma série de ferramentas que permitem o acesso à base de dados.

Como na Internet não é possível instalar esta camada de aplicação para todos os possíveis usuários da aplicação, devemos criar entre o servidor WEB e o servidor de banco de dados as ferramentas de acesso às informações. Assim, o cliente envia suas requisições ao servidor WEB e este, por sua vez, "conversa" com o banco de dados. O banco de dados retorna ao servidor WEB o que foi pedido e este retornará àquele que fez a requisição.

É comum hoje em dia haver servidores dedicados a essas operações. São os servidores de aplicação. Eles possuem capacidade de processamento aliviando o serviço tanto do servidor WEB quanto do servidor de banco de dados.

ODBC e JDBC

ODBC (Open Database Connectivity) é uma interface-padrão dos bancos de dados para comunicação entre os clientes e o servidor. Dessa forma, qualquer aplicação cliente (também conhecida como *front-end*) pode acessar as informações do banco de dados (também conhecido como *back*-end) utilizando o SQL. Com isso, as informações requisitadas pelo cliente serão transformadas em algo que o servidor entenda. Essa transformação ocorre no servidor. É utilizada uma biblioteca de funções para realizar este trabalho.

Todos os fornecedores de banco de dados incluem em seus produtos uma API (Application Programmatic Interface) que realiza esse trabalho de tradução das requisições em comandos que os usuários possam entender. Desta forma, cada fornecedor de banco de dados possui sua própria API, o que, na prática, leva a diferenças de utilização e desempenho entre eles.

JDBC (Java Database Connectivity) é uma interface-padrão para utilização com bancos de dados, caracterizada por estar vinculada à linguagem Java.

Como se sabe, Java tornou-se padrão no desenvolvimento de aplicações, inclusive para WEB. Como o Java foi concebido para trabalhar na WEB, o JDBC também possui essa característica. Como o Java é independente de plataforma (Windows, Linux, Unix etc.), esse padrão de comunicação com bancos de dados também herda a mesma característica. Assim, não é necessário preocupar-se com qual banco de dados será feita a comunicação, pois todos os bancos que suportam o JDBC utilizam esse mesmo padrão.

O que mudou no padrão SQL-99 (ou SQL-3)

Nos últimos 20 anos, os bancos de dados têm sido determinados pelo padrão relacional. Atualmente, a maior parte dos bancos de dados segue esse padrão, em maior ou menor grau. Com o passar do tempo, as necessidades foram sendo aumentadas. Até algum tempo atrás não se falava em manipular imagens e outros tipos de dados que não fossem textos ou números. Esses tipos os bancos de dados relacionais manipulam com extraordinária eficiência.

Para atender a essa demanda por aplicações que possam armazenar sons e imagens, permitir recuperações de dados cada vez mais complexas e em diversos níveis de detalhamento, para atender à necessidade de tomada de decisões na empresa, é que surgiu esse novo padrão.

ORDBMS (Object-Relational Database Management System)

São bancos de dados que mantêm total compatibilidade com o modelo relacional, mas incorporam funções de orientação a objetos. Isso os torna mais completos que seus parentes mais antigos. A linguagem SQL é estendida para atender aos requisitos de novos tipos de dados, muito mais complexos. Com isso é possível agregar aos bancos de dados liguagens poderosas como C++ e Java.

Problemas com a normalização

Ao utilizarmos alguns conceitos de Orientação a Objetos, como encapsulamento, herança e polimorfismo, no padrão SQL podemos ter problemas com determinadas regras de normalização de dados. O fato é que muitas vezes o custo de normalizar uma base de dados é extremamente alto.

Podemos utilizar os conceitos da OOP sem perda de consistência e segurança. Isso está causando uma modificação no padrão SQL, que será visto e acompanhado ao longo do tempo.

Objetos

O novo padrão SQL introduz o conceito de Objeto com seus métodos (semelhantes a funções e procedimentos utilizados anteriormente pelas diversas linguagens SQL) e propriedades. Junto com os objetos, a SQL-99 introduziu a possibilidade de criar tipos definidos pelo usuário, introduzir *arrays* em colunas de bancos de dados e criar um tipo com base em outro tipo definido anteriormente.

Claro que nesse primeiro instante não é necessário utilizarmos esses conceitos para a utilização dos bancos de dados SQL. Podemos sequer notar que isso foi acrescentado, visto que a maioria das implementações ainda não possui esses recursos incorporados.

Padronização de programas

Está sendo feita uma tentativa de padronizar as diversas linguagens de programação que surgiram em função do uso do SQL. Veja que, hoje em dia, cada banco de dados relacional procura seguir em maior ou menor grau o padrão SQL para armazenamento, criação de estruturas e recuperação de dados. Isso, porém, não acontece com as linguagens de programação. Cada uma possui sua própria estrutura, estendendo a linguagem SQL-padrão.

O que se tenta é adotar um padrão também para a forma de programar nos diversos bancos de dados padrão SQL. O capítulo 15 traz uma análise de algumas das extensões existentes nos bancos de dados mais populares do mercado.

15

Estendendo o padrão SQL

Ao compreendermos o desenvolvimento do SQL, é natural que agora você entenda que há necessidade de avançar um pouco mais nesse padrão. Cada implementação de banco de dados tem suas próprias extensões. Essas extensões são linguagens de programação muitas vezes semelhantes ou completamente diferentes. Para aqueles que possuem conhecimento em programação, fica muito mais fácil entender e diferenciar uma implementação da outra.

Algumas implementações importantes

Alguns dos mais populares bancos de dados da atualidade são o Oracle e o SQL Server. O SQL Server, como foi derivado do Sybase, mantém uma série de estruturas comuns a ambos.

Transact-SQL

É uma linguagem procedural que estende o SQL para o banco de dados SQL Server. Por meio dessa implementação é possível criar procedimentos, funções e gatilhos (*triggers*).

PL/SQL

É a extensão SQL do banco de dados Oracle. Da mesma forma que o Transact-SQL, o PL/SQL é uma linguagem procedural presente em diversas ferramentas Oracle, e não apenas no banco de dados. No PL/SQL existem três partes que definem um bloco de programa.

A primeira parte é a seção de declaração de variáveis (identificada pela cláusula DECLARE), a segunda é a seção de processamento (cláusula BEGIN) e a terceira, o tratamento de erros (cláusula EXCEPTION). Com o PL/SQL é possível criar procedimentos, funções, gatilhos (*triggers*) e objetos.

Recursos da extensão SQL

Vamos analisar neste capítulo quais são as principais implementações e os recursos que cada uma traz consigo. Os nossos destaques são:

1. Procedimentos e funções armazenadas.
2. Cursores.
3. Gatilhos.
4. SQL dinâmica.
5. SQL embutida.
6. ORDBMS.

Procedimentos e funções armazenadas

São blocos de comando que têm como objetivo executar uma série de comandos SQL no servidor de banco de dados. A característica básica é que esses procedimentos estarão armazenados no servidor, sendo, portanto, muito mais rápida a sua execução. A utilização de procedimentos armazenados contribui para diminuir o tráfego de dados na rede, melhorando o desempenho dos sistemas.

Assim, sempre que possível, devemos utilizar esse recurso. A sintaxe de programação e as estruturas são muito semelhantes às diversas linguagens de programação. Com os procedimentos, dizemos ao banco de dados *como* deve ser realizada determinada função. Como visto no início do livro, o SQL-padrão (DML, DCL, DDL) não é uma linguagem procedural, logo não podemos dizer como deve ser feito.

Para criar um procedimento, utilizamos os seguintes comandos:

```
CREATE PROCEDURE nome
   [( parâmetro tipo_dado, parâmetro tipo_dado,...)]
   {IS|AS}
   bloco_comandos

CREATE FUNCTION nome
   [( parâmetro tipo_dado, parâmetro tipo_dado,...)]
   RETURN tipo_dado
   {IS|AS}
   bloco_comandos
```

Para eliminar os procedimentos e funções que criamos, utilizamos o comando:

```
DROP PROCEDURE nome

DROP FUNCTION nome
```

Naturalmente de nada adiantaria existirem esses comandos se não houvesse comandos de controle de fluxo e repetição. Resumidamente, vamos comentar cada um deles:

IF ... THEN ... ELSE ... END IF

Presente em diversas linguagens de programação, esta estrutura é responsável pelo desvio condicional com base no teste de condição da cláusula IF. Assim temos:

```
IF condição THEN
  Ação 1;
  Ação n;
ELSE
  Ação 1;
  Ação n;
END IF;
```

Caso a *condição* seja verdadeira, as ações após a cláusula THEN serão executadas. Caso não seja verdadeira, as ações após o ELSE serão executadas. É muito comum encontrarmos implementações que façam novos testes. Nesse caso, antes do ELSE, podemos colocar outras condições acrescentando ELSEIF ou ELSIF, dependendo da implementação.

LOOP ... END LOOP

Esse comando serve para repetir ações que estão entre o LOOP e o END LOOP. Assim, quando o fluxo do programa encontra o comando END LOOP, retorna até o comando LOOP e repete todos os comandos que estão entre ambos. Normalmente é incluída uma cláusula de saída, do contrário a repetição não teria fim. Em algumas implementações, em vez de utilizar o EXIT, devemos utilizar o LEAVE.

```
LOOP
  Ação 1;
  Ação n;
  EXIT WHEN condição;
END LOOP;
```

WHILE ... DO/LOOP ... END WHILE/LOOP

Esta instrução faz com que uma série de comandos seja repetida com base em uma condição. Da mesma forma que a anterior, todos os comandos entre o DO ou LOOP e o END WHILE ou LOOP serão repetidos.

```
WHILE condição LOOP
  Ação 1;
  Ação n;
END LOOP;
WHILE condição DO
  Ação 1;
  Ação n;
END WHILE;
```

Note que em algumas implementações devemos utilizar uma ou outra sintaxe.

Veja o exemplo de um bloco escrito para utilização no Oracle:

```
CREATE PROCEDURE manAutor (cOperacao VARCHAR2,
    nCDAUTOR NUMBER, cNMAUTOR VARCHAR2 ) IS

BEGIN

 IF cOperacao = 'I' THEN
```

```
    INSERT INTO AUTOR (CODIGO_AUTOR,NOME_AUTOR)
      VALUES ( nCDAUTOR, cNMAUTOR) ;

  ELSE

   UPDATE AUTOR
      SET NOME_AUTOR = cNMAUTOR
      WHERE CODIGO_AUTOR = nCDAUTOR;

   END IF;

  END;
```

Cursores

É uma área de memória do banco de dados que armazena as linhas recuperadas do comando SELECT. Isso quer dizer que um cursor sempre é criado com base em um comando SELECT. Implicitamente, toda vez que se realiza uma consulta no banco de dados, ao retornar as linhas da consulta, está realizando uma operação com cursores.

Como as linguagens procedurais não podem trabalhar com blocos de informação (conjunto de linhas retornadas pelo comando SELECT), é necessário criar um cursor para ter acesso a cada uma das linhas da consulta. Dessa forma é criado um ponteiro que indica em que linha é utilizada a informação. O cursor é semelhante ao *recordset* presente em diversas linguagens de programação. Na tabela a seguir, observamos um Cursor com o ponteiro na segunda linha:

Cód	Nome CD	Cód Gravadora	Preco Venda
01	Mais do Mesmo	01	15.00
→ 02	Bate-Boca	02	12.00
03	Essa Mulher	03	9.00
04	Perfil	03	8.00

Há quatro fases distintas para utilização de um cursor:

1. Declaração.
2. Abertura.
3. Busca de Dados.
4. Fechamento.

Declaração

Na mesma seção onde foram declaradas as variáveis do procedimento ou função, declara-se o cursor. Observe que nesse instante estamos apenas declarando a estrutura do cursor. O conteúdo será alimentado apenas na abertura deste. A primeira opção declara um cursor no SQL Server e a segunda, no Oracle:

```
DECLARE nome CURSOR FOR comando_SELECT
DECLARE CURSOR nome IS comando_SELECT
```

Abertura

Para abrir um cursor, utilizamos o comando OPEN. Nesse instante será executado o comando SELECT. No instante em que o cursor é aberto, os dados ficam disponíveis na memória do banco de dados.

```
OPEN nome
```

Busca de dados

Para recuperar as informações da linha onde está posicionado o ponteiro do cursor, utilizamos o comando FETCH. As variáveis utilizadas no comando devem estar previamente declaradas para serem utilizadas.

```
FETCH nome INTO lista_variáveis
```

Fechamento

Para fechar um cursor depois da utilização das informações nele contidas, utilizamos o comando CLOSE.

```
CLOSE nome
```

Alguns atributos de cursor podem ser testados durante a execução do programa:

Atributo	Significado
%FOUND	Retorna TRUE caso seja possível retornar alguma linha, FALSE se não conseguir e NULL se não houver sido executado um FETCH.
%NOTFOUND	Inverso ao anterior e igualmente NULL se não tiver sido executado um FETCH.
%ROWCOUNT	Retorna o número de linhas processadas pelo cursor.
%ISOPEN	Retorna TRUE se o cursor foi aberto (OPEN) e FALSE do contrário.

Veja um exemplo de utilização de cursor utilizando um procedimento escrito para Oracle:

```
CREATE PROCEDURE cursorTeste IS
 CURSOR oAUTOR IS SELECT * FROM AUTOR;
 nCDAUTOR INTEGER; cNMAUTOR VARCHAR2(60);
BEGIN
 OPEN oAUTOR;
 WHILE oAUTOR%FOUND LOOP
  FETCH oAUTOR INTO nCDAUTOR, cNMAUTOR;
 END LOOP;
 CLOSE oAUTOR;
END;
```

Uma alternativa mais simples à utilização formal de um cursor é utilizar o comando FOR ... LOOP ... END LOOP. Nesse caso, o argumento *x* do comando FOR representa o ponteiro onde está a linha do cursor. O nome dos campos é relacionado ao argumento *x* seguido do nome do campo. Por meio dele é possível declarar, abrir, recuperar e fechar um cursor automaticamente. Veja como ficaria o exemplo anterior:

```
CREATE PROCEDURE cursorTeste IS
 CURSOR oAUTOR IS SELECT * FROM AUTOR;
 nCDAUTOR INTEGER; cNMAUTOR VARCHAR2(60);
BEGIN
 FOR x IN oAUTOR LOOP
  nCDAUTOR:= x.CODIGO_AUTOR;
  cNMAUTOR:= x.NOME_AUTOR;
 END LOOP;
END;
```

Esse procedimento poderia ficar ainda mais simples caso fosse criado o Cursor no próprio comando FOR. Assim:

```
CREATE PROCEDURE cursorTeste IS

 nCDAUTOR INTEGER; cNMAUTOR VARCHAR2(60);

BEGIN

 FOR x IN ( SELECT * FROM AUTOR )

  LOOP

  nCDAUTOR:= x.CODIGO_AUTOR;

  cNMAUTOR:= x.NOME_AUTOR;

 END LOOP;

END;
```

Gatilho

Gatilho é um procedimento SQL armazenado em um banco de dados e que depende de um evento (INSERT, DELETE ou UPDATE) para que seja disparado. O gatilho pode ser disparado antes ou depois de um dos eventos citados e, no caso de violar alguma regra de negócio, pode reverter toda transação pendente.

Utilizam-se gatilhos para acrescentar regras de negócio ao banco de dados. Assim, caso um cliente esteja em situação irregular, um gatilho pode impedir que uma venda seja realizada para esse cliente. Note que será dado um comando INSERT na tabela de Pedidos. Ao interceptarmos o comando INSERT, podemos programar o gatilho para checar a situação do cliente e, em caso de ele estar inadimplente, bloquear a operação.

Veja a sintaxe:

```
CREATE TRIGGER nome
   BEFORE {INSERT | UPDATE | DELETE}
   ON tabela
   [FOR EACH {ROW | STATEMENT}]
   [WHEN condição]
   bloco_comandos
```

Caso seja especificada a cláusula FOR EACH ROW, o gatilho será disparado a cada linha afetada pelo comando DML. Se nada for especificado ou for especificado STATEMENT, o gatilho será disparado apenas no final do comando DML, mesmo que diversas linhas sejam afetadas pelo comando.

Caso especifique uma condição, o gatilho somente será executado se essa condição for verdadeira.

Sempre que criamos um gatilho, temos o valor dos campos da tabela em que ele está sendo executado antes e depois da atualização. Isso quer dizer que, em caso de inclusão, temos o valor que está sendo incluído em cada uma das colunas. Em caso de exclusão temos o valor anterior à exclusão. Em caso de atualização temos o valor anterior (antes de atualização) e o valor posterior (o valor que está sendo informado). Fazemos referência a esses valores utilizando as cláusulas :NEW e :OLD, onde :NEW representa o novo conteúdo para cada uma das colunas e :OLD o conteúdo anterior da coluna.

É comum utilizar gatilhos para situações de "desnormalização" de dados. Quando, por questões de desempenho, resolvemos violar a terceira forma normal, por exemplo, costuma-se criar um gatilho para manter a coluna de total sempre atualizada. Veja um exemplo de gatilho para o Oracle:

```
CREATE TRIGGER t_CD

 AFTER INSERT

 ON FAIXA

 FOR EACH ROW

DECLARE

 nTEMPO NUMBER;

BEGIN

 SELECT DURACAO INTO nTEMPO
    FROM MUSICA
    WHERE CODIGO_MUSICA = :NEW.CODIGO_MUSICA;

  UPDATE CD
    SET TTEMPO = TTEMPO + nTEMPO
    WHERE CODIGO_CD = :NEW.CODIGO_CD;

END;
```

Para excluir um gatilho, utilizamos o comando:

```
DROP TRIGGER nome
```

SQL dinâmica

Com esse recurso é possível executar comandos SQL em tempo de execução. Nesse caso, estaremos criando um comando em tempo de execução. Até aqui os comandos eram digitados e executados no banco de dados. Com a SQL dinâmica é possível fazer com que o comando seja "montado" durante a execução do programa.

Na SQL dinâmica, a consulta pode ser adaptada às necessidades reais do usuário. Sempre que possível, devemos utilizar a SQL estática, pois ela requer menos recursos do banco de dados.

Depois de executar o comando enviado ao banco de dados, este retorna o conteúdo nas variáveis do programa, e essas variáveis são associadas em tempo de execução do programa.

É comum utilizar a SQL dinâmica para embutir comandos SQL em linguagens procedurais, como C, COBOL, etc. e em muitas linguagens de *script* para Internet, como ASP.NET e PHP. Uma vez passado o texto para o banco de dados, este, por sua vez, retorna o valor nas variáveis definidas no comando. Normalmente utilizamos o comando EXEC SQL para essa função.

A maneira mais fácil de utilizar a SQL Dinâmica é com o comando EXECUTE IMMEDIATE. Por meio dele podemos montar o comando SQL e enviá-lo ao servidor de banco de dados para que seja executado.

O banco de dados Oracle possui um comando para montar e executar o código SQL. Esse código pode ser utilizado dentro de procedimentos criados no próprio banco de dados.

SQL embutida

Ocorre sempre que colocamos um código SQL dentro de outra linguagem de programação procedural, como C, COBOL ou mesmo ASP.NET e PHP. A linguagem SQL fica embutida na linguagem de programação e é executada apenas quando encontra o comando EXEC SQL no meio do código.

As instruções e controles de fluxo, decisão, variáveis, etc. são controlados pela linguagem de programação. Na prática, não há grande diferença entre o SQL embutido e o dinâmico quando estamos utilizando ambos dentro de linguagens de programação.

Dessa forma, o pré-processador é capaz de processar seus comandos normalmente até que o comando de interação com o SQL seja encontrado. A partir daí o controle é enviado ao banco de dados e este retorna o controle e eventuais valores ao programa original.

Veja um exemplo de utilização:

```
String cString;

cString = "SELECT * FROM AUTOR WHERE";

IF (cOpcao = "CODIGO")

{

  cString = cString + " CODIGO_AUTOR = " + nCodigo

}

IF (cOpcao = "NOME")

{

  cString = cString + " NOME_AUTOR = " + cNome

}

EXEC SQL EXECUTE IMMEDIATE :cString;
```

ORDBMS (Object Relational Database Management System)

Os bancos de dados objeto-relacionais padronizados com a SQL-99 ou SQL-3 trazem consigo uma série de novos conceitos:

1. Tipos de dados criados pelo usuário.
2. Objeto.
3. Tabelas Tipo.

Tipos de dados criados pelo usuário

Com esse novo padrão, os usuários não estão mais restritos aos tipos básicos definidos pelos bancos de dados. É possível combinar novos tipos para formar um novo tipo, específico para o usuário. Os tipos podem ser ARRAY, TABLE ou RECORD. Os tipos ARRAY têm tamanho fixo e são de um único tipo de dado. TABLE não têm tamanho fixo e são de um único tipo de dado. RECORD podem ter várias colunas e cada uma delas pode apresentar um tipo de dado diferente.

Para criar um Tipo de Dado utilizamos o comando:

```
CREATE TYPE nome IS TABLE OF elemento

CREATE TYPE nome IS {VARRAY | VARYING ARRAY} (limite)
   OF elemento

CREATE TYPE nome IS RECORD (def_campo[,def_campo]...)
```

Objeto

Objeto nada mais é do que a união de propriedades (dados) e métodos (processos). A criação de objetos, com herança, polimorfismo, abstração e encapsulamento passou a ser uma realidade proposta pela SQL-3. A criação de um objeto se dá pelo comando:

```
CREATE TYPE nome
   {IS | AS} OBJECT ( propriedade tipo-de-dado[, propriedade tipo-de-
   dado]...)
```

Tabelas Tipo

As tabelas tipo são aquelas derivadas de tipos de dado estruturados, como os definidos nos objetos. A tabela tipo é a estrutura que armazena os dados e métodos definidos em um tipo objeto. Cada linha receberá uma identidade exclusiva para identificar a instância da classe (objeto). A criação se dá pelo comando:

```
CREATE TABLE tabela OF tipo_objeto
```

Para definir o identificador único do objeto existe o tipo REF. Ele sempre estará associado a uma tabela tipo específica. Assim, desejemos vincular um objeto a outro, devemos fazê-lo utilizando esse tipo.

Apêndice A

Principais comandos SQL

Alteração de estrutura de tabelas

```
ALTER TABLE tabela
   ADD nome-coluna tipo-de-dado constraints [, nome-coluna tipo-de-
   dado constraints, ...]

ALTER TABLE tabela
   MODIFY nome-coluna tipo-de-dado constraints

ALTER TABLE tabela
   DELETE elemento

ALTER TABLE tabela
   RENAME nome-tabela

ALTER TABLE tabela
   RENAME coluna-velha TO coluna-nova
```

Transações

```
COMMIT [WORK];
```

Conexão ao banco de dados

```
CONNECT
```

Fechamento de Cursor

```
CLOSE nome
```

Assertivas

```
CREATE ASSERTION nome
   CHECK (expressão_lógica);
```

Domínio

```
CREATE DOMAIN nome tipo_de_dado
   CHECK (expressão);
```

Funções

```
CREATE FUNCTION nome [( parâmetro tipo_dado, parâmetro
   tipo_dado,...)]
   RETURN tipo_dado
   {IS|AS}
   bloco_comandos
```

Índices

```
CREATE [UNIQUE] INDEX nome
   ON tabela ( coluna [, coluna, ... ]  [ASC | DESC] )
```

Procedimentos

```
CREATE PROCEDURE nome [( parâmetro tipo_dado, parâmetro
   tipo_dado,...)]
   {IS|AS}
   bloco_comandos
```

Schema

```
CREATE SCHEMA nome_schema
   [AUTHORIZATION usuário];
```

Tabelas

```
CREATE TABLE nome-tabela
   ( nome-coluna1 tipo-de-dado-coluna1 coluna1-constraint,
   nome-coluna2 tipo-de-dado-coluna2 coluna2-constraint,
   nome-colunan tipo-de-dado-colunan colunan-constraint,
   constraint-de-tabela )
```

Tabelas Tipo

```
CREATE TABLE tabela OF tipo_objeto
```

Tabelas Temporárias

```
CREATE {GLOBAL|LOCAL} TEMPORARY TABLE tabela
   (coluna tipo_dado, ... );
```

Gatilho

```
CREATE TRIGGER nome
   BEFORE {INSERT | UPDATE | DELETE}
   ON tabela
   [FOR EACH {ROW | STATEMENT}]
   [WHEN condição]
   bloco_comandos
```

Objeto

```
CREATE TYPE nome
   {IS | AS} OBJECT ( propriedade tipo-de-dado[, propriedade tipo-de-
   dado]...)
```

Tipo de dado criado pelo usuário

```
CREATE TYPE nome IS TABLE OF elemento

CREATE TYPE nome IS {VARRAY | VARYING ARRAY} (limite) OF elemento

CREATE TYPE nome IS RECORD (def_campo[,def_campo]...)
```

Usuários

```
CREATE USER usuário
   IDENTIFIED BY senha;
```

Visão

```
CREATE VIEW nome
   AS subquery;
```

Declaração de cursor

```
DECLARE nome CURSOR FOR comando_SELECT
```

```
DECLARE CURSOR nome IS instrução_SELECT
```

Exclusão de linhas

```
DELETE FROM tabela
   [ WHERE condição ]
```

Exclusão de índices

```
DROP INDEX nome
```

Desconexão ao banco de dados

```
DISCONNECT
```

Exclusão de função

```
DROP FUNCTION nome
```

Exclusão de procedimento

```
DROP PROCEDURE nome
```

Exclusão de tabelas

```
DROP TABLE nome
```

Exclusão de gatilho (trigger)

```
DROP TRIGGER nome
```

Exclusão de visão

```
DROP VIEW view;
```

Busca de dados em cursor

```
FETCH nome INTO lista_variáveis
```

Chave estrangeira

```
FOREIGN KEY nome-chave-estrangeira ( lista-de-colunas )
   REFERENCES nome-tabela ( lista-de-colunas )
   ON UPDATE ação
   ON DELETE ação
```

Concessão de direitos

```
GRANT tipo_direito
   ON tabela
   TO {usuário|PUBLIC}
   [WITH GRANT OPTION];

GRANT privilégio
   TO usuário
   [WITH ADMIN OPTION];
```

Estrutura de controle IF

```
IF condição THEN
 Ação 1;
 Ação n;
ELSE
 Ação 1;
 Ação n;
END IF;
```

Inclusão de linhas

```
INSERT INTO tabela [ ( coluna [, coluna, ... ] ) ]
   VALUES ( conteúdo [, conteúdo, ... ] )

INSERT INTO tabela [ ( coluna [, coluna, ... ] ) ]
   SELECT comando-select
```

Estrutura de controle LOOP

```
LOOP
 Ação 1;
 Ação n;
 EXIT WHEN condição;
END LOOP;
```

Abertura de cursor

```
OPEN nome
```

Revogação de direitos

```
REVOKE tipo_direito
   ON tabela
   FROM usuário;

REVOKE privilégio
   FROM usuário;
```

Desfazer transações

```
ROLLBACK [WORK];

ROLLBACK TO nome_savepoint;
```

Ponto de salvamento de transações

```
SAVEPOINT nome_savepoint;
```

Comando SELECT

```
SELECT [DISTINCT | ALL] { * | coluna [, coluna, ... ] }
   FROM tabela


SELECT [DISTINCT | ALL] { * | coluna [, coluna, ... ] }
   FROM tabela
   ORDER BY coluna-ord [, coluna-ord, ... ]


SELECT [DISTINCT | ALL] { * | coluna [, coluna, ... ] }
   FROM tabela
   WHERE condição
```

```
SELECT [tabela1.]coluna [, [tabela2.]coluna, ... ]
   FROM tabela1, tabela 2 [, ...]
   WHERE tabela1.chave_primária = tabela2.chave_estrangeira
```

```
SELECT coluna [, coluna, ... ], função_de_grupo, [função_de_grupo,
   ...]
   FROM tabela
   GROUP BY coluna [, coluna, ... ]
```

```
SELECT colunas
   FROM tabela
   WHERE expressão operador ( SELECT colunas
   FROM tabela
   WHERE ... )
```

```
SELECT colunas
   FROM tabela(s)
   [WHERE predicado]
   [GROUP BY colunas [HAVING condição]]
{UNION | INTERSECT | MINUS}
SELECT colunas
   FROM tabela(s)
   [WHERE predicado]
   [GROUP BY colunas [HAVING condição]]
```

```
SELECT *
   FROM tabela
   [WHERE predicado]
UNION CORRESPONDING BY (colunas)
SELECT *
   FROM tabela
   [WHERE predicado]
```

```
SELECT colunas,
   CASE
   WHEN condição THEN ação
   ...
   [ELSE condição padrão]
   END
   FROM tabela;
```

Controle de transações

```
SET CONSTRAINTS [MODE]
   {constraint|ALL}
   {IMMEDIATE|DEFERRED};
```

Mudança de schema

```
SET SCHEMA schema;
```

Controle de transações

```
SET TRANSACTION ISOLATION LEVEL nível;
```

Atualização de dados

```
UPDATE tabela
   SET coluna = conteúdo [, coluna =  conteúdo, ...]
   [ WHERE condição ]
```

```
UPDATE tabela
   SET coluna =
   CASE
   WHEN condição THEN ação
   ...
   [ELSE condição padrão]
   END
```

Estrutura de controle WHILE

```
WHILE condição LOOP
  Ação 1;
  Ação n;
END LOOP;
```

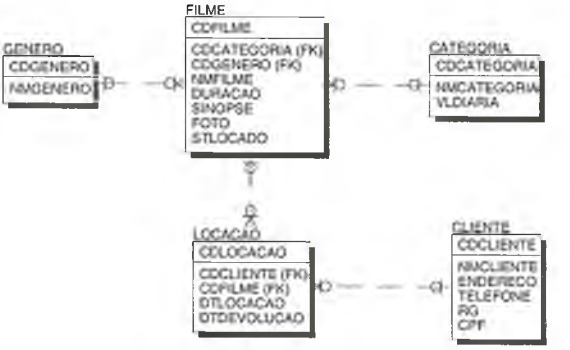
```
WHILE condição DO
  Ação 1;
  Ação n;
END WHILE;
```

Apêndice B

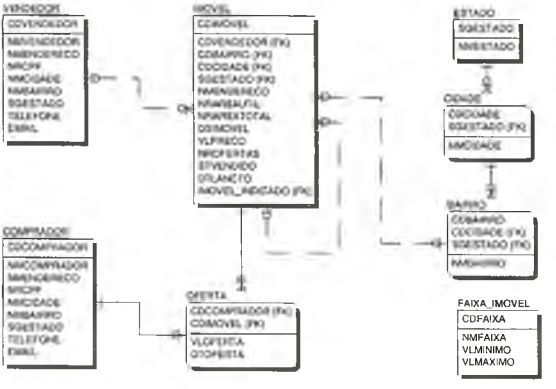
Resposta aos exercícios

Capítulo 2

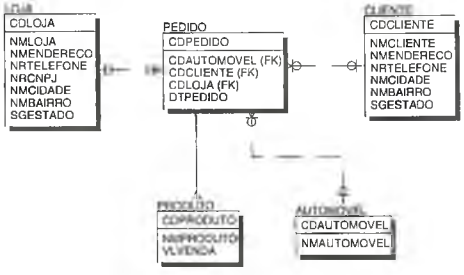
1.



2.

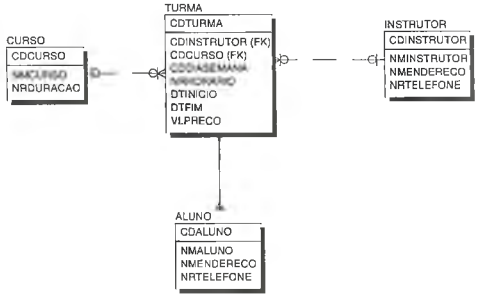


3.

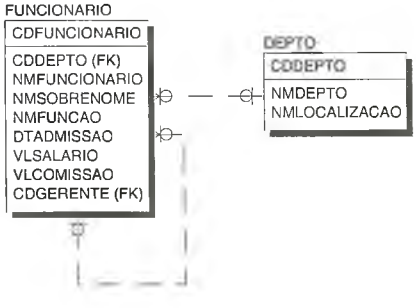

LOJA
CDLOJA
NMLOJA
NMENDERECO
NRTELEFONE
NRCNPJ
NMCIDADE
NMBAIRRO
SGESTADO
PEDIDO
CDPEDIDO
CDAUTOMOVEL (FK)
CDCLIENTE (FK)
CDLOJA (FK)
DTPEDIDO
CLIENTE
CDCLIENTE
NMCLIENTE
NMENDERECO
NRTELEFONE
NMCIDADE
NMBAIRRO
SGESTADO
PRODUTO
CDPRODUTO
NMPRODUTO
VLVENDA
AUTOMOVEL
CDAUTOMOVEL
NMAUTOMOVEL

4.

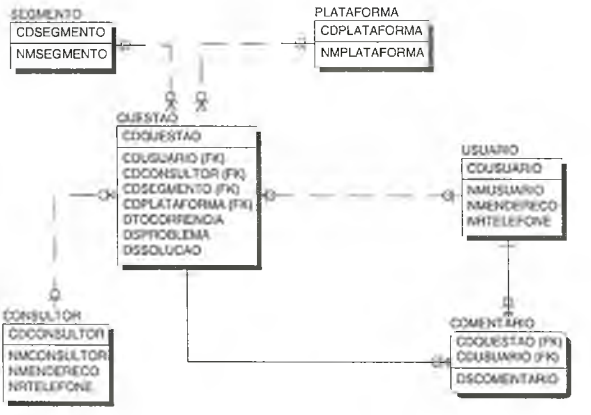
a.)


CURSO
CDCURSO
NMCURSO
NRDURACAO
TURMA
CDTURMA
CDINSTRUTOR (FK)
CDCURSO (FK)
CDDIASEMANA
NRHORARIO
DTINICIO
DTFIM
VLPRECO
INSTRUTOR
CDINSTRUTOR
NMINSTRUTOR
NMENDERECO
NRTELEFONE
ALUNO
CDALUNO
NMALUNO
NMENDERECO
NRTELEFONE

b.)


FUNCIONARIO
CDFUNCIONARIO
CDDEPTO (FK)
NMFUNCIONARIO
NMSOBRENOME
NMFUNCAO
DTADMISSAO
VLSALARIO
VLCOMISSAO
CDGERENTE (FK)
DEPTO
CDDEPTO
NMDEPTO
NMLOCALIZACAO

c.)



Nota importante: estas soluções são as mais usuais. Dependendo do alcance (escopo) do sistema, podemos ter diversas respostas para as mesmas questões. Um exercício interessante é tentar aprimorar os modelos apresentados buscando acrescentar novas funcionalidades.

Capítulo 3

Não há necessidade de alteração em nenhum dos modelos apresentados, visto que todos atendem a todas as regras de normalização de dados.

Capítulo 4

1.

```
CREATE TABLE BAIRRO (
     CDBAIRRO       INTEGER NOT NULL,
     NMBAIRRO       VARCHAR(20) NULL,
     CDCIDADE       INTEGER NOT NULL,
     SGESTADO       CHAR(2) NOT NULL );

CREATE INDEX XIF8BAIRRO ON BAIRRO (
     CDCIDADE       ASC,
     SGESTADO       ASC );

ALTER TABLE BAIRRO
     ADD  ( CONSTRAINT XPKBAIRRO PRIMARY KEY (CDBAIRRO, CDCIDADE,
     SGESTADO) ) ;
```

```
CREATE TABLE CIDADE (
     CDCIDADE       INTEGER NOT NULL,
     NMCIDADE       VARCHAR(20) NULL,
     SGESTADO       CHAR(2) NOT NULL );

CREATE INDEX XIF7CIDADE ON CIDADE ( SGESTADO  ASC );

ALTER TABLE CIDADE
     ADD  ( CONSTRAINT XPKCIDADE PRIMARY KEY (CDCIDADE,
     SGESTADO));

CREATE TABLE COMPRADOR (
     CDCOMPRADOR    INTEGER NOT NULL,
     NMCOMPRADOR    VARCHAR(40) NULL,
     NMENDERECO     VARCHAR(40) NULL,
     NRCPF          DECIMAL(11) NULL,
     NMCIDADE       VARCHAR(20) NULL,
     NMBAIRRO       VARCHAR(20) NULL,
     SGESTADO       CHAR(2) NULL,
     TELEFONE       VARCHAR(20) NULL,
     EMAIL          VARCHAR(80) NULL );

ALTER TABLE COMPRADOR
     ADD  ( CONSTRAINT XPKCOMPRADOR PRIMARY KEY (CDCOMPRADOR) ) ;

CREATE TABLE ESTADO (
     SGESTADO       CHAR(2) NOT NULL,
     NMESTADO       VARCHAR(20) NULL );

ALTER TABLE ESTADO
     ADD  ( CONSTRAINT XPKESTADO PRIMARY KEY (SGESTADO) ) ;

CREATE TABLE FAIXA_IMOVEL (
     CDFAIXA        INTEGER NOT NULL,
     NMFAIXA        VARCHAR(30) NULL,
     VLMINIMO       DECIMAL(14,2) NULL,
     VLMAXIMO       DECIMAL(14,2) NULL );

ALTER TABLE FAIXA_IMOVEL
     ADD  ( CONSTRAINT XPKFAIXA_IMOVEL PRIMARY KEY (CDFAIXA) ) ;

CREATE TABLE IMOVEL (
     CDIMOVEL          INTEGER NOT NULL,
     CDVENDEDOR        INTEGER NULL,
     CDBAIRRO          INTEGER NULL,
     CDCIDADE          INTEGER NULL,
     SGESTADO          CHAR(2) NULL,
     NMENDERECO        VARCHAR(40) NULL,
     NRAREAUTIL        DECIMAL(10,2) NULL,
     NRAREATOTAL       DECIMAL(10,2) NULL,
     DSIMOVEL          VARCHAR(300) NULL,
     VLPRECO           DECIMAL(16,2) NULL,
     NROFERTAS         INTEGER NULL,
     STVENDIDO         CHAR(1) NULL,
     DTLANCTO          DATE NULL,
     IMOVEL_INDICADO   INTEGER NULL );
```

```
CREATE INDEX XIF12IMOVEL ON IMOVEL ( CDVENDEDOR  ASC );

CREATE INDEX XIF13IMOVEL ON IMOVEL ( IMOVEL_INDICADO  ASC );

CREATE INDEX XIF9IMOVEL ON IMOVEL (
    CDBAIRRO      ASC,
    CDCIDADE      ASC,
    SGESTADO      ASC );

ALTER TABLE IMOVEL
    ADD  ( CONSTRAINT XPKIMOVEL PRIMARY KEY (CDIMOVEL) ) ;

CREATE TABLE OFERTA (
    CDCOMPRADOR   DECIMAL(16,2) NOT NULL,
    CDIMOVEL      INTEGER NOT NULL,
    VLOFERTA      NUMBER(16,2) NULL,
    DTOFERTA      DATE NULL);

CREATE INDEX XIF10OFERTA ON OFERTA ( CDCOMPRADOR  ASC );

CREATE INDEX XIF11OFERTA ON OFERTA ( CDIMOVEL  ASC );

ALTER TABLE OFERTA
    ADD  ( CONSTRAINT XPKOFERTA PRIMARY KEY (CDCOMPRADOR,
    CDIMOVEL) ) ;

CREATE TABLE VENDEDOR (
    CDVENDEDOR    INTEGER NOT NULL,
    NMVENDEDOR    VARCHAR(40) NULL,
    NMENDERECO    VARCHAR(40) NULL,
    NRCPF         DECIMAL(11) NULL,
    NMCIDADE      VARCHAR(20) NULL,
    NMBAIRRO      VARCHAR(20) NULL,
    SGESTADO      CHAR(2) NULL,
    TELEFONE      VARCHAR(20) NULL,
    EMAIL         VARCHAR(80) NULL );

ALTER TABLE VENDEDOR
    ADD  ( CONSTRAINT XPKVENDEDOR PRIMARY KEY (CDVENDEDOR) ) ;

ALTER TABLE BAIRRO
    ADD  ( CONSTRAINT CIDADE_BAIRRO
    FOREIGN KEY (CDCIDADE, SGESTADO)
    REFERENCES CIDADE ) ;

ALTER TABLE CIDADE
    ADD  ( CONSTRAINT ESTADO_CIDADE
    FOREIGN KEY (SGESTADO)
    REFERENCES ESTADO ) ;

ALTER TABLE IMOVEL
    ADD  ( CONSTRAINT IMOVEL_INDICADO
    FOREIGN KEY (IMOVEL_INDICADO)
    REFERENCES IMOVEL ) ;
```

```
ALTER TABLE IMOVEL
    ADD  ( CONSTRAINT VENDEDOR_IMOVEL
    FOREIGN KEY (CDVENDEDOR)
    REFERENCES VENDEDOR ) ;

ALTER TABLE IMOVEL
    ADD  ( CONSTRAINT BAIRRO_IMOVEL
    FOREIGN KEY (CDBAIRRO, CDCIDADE, SGESTADO)
    REFERENCES BAIRRO ) ;

ALTER TABLE OFERTA
    ADD  ( CONSTRAINT IMOVEL_OFERTA
    FOREIGN KEY (CDIMOVEL)
    REFERENCES IMOVEL ) ;

ALTER TABLE OFERTA
    ADD  ( CONSTRAINT COMPRADOR_OFERTA
    FOREIGN KEY (CDCOMPRADOR)
    REFERENCES COMPRADOR ) ;
```

2.

```
CREATE INDEX XIFNMVENDEDOR ON VENDEDOR ( NMVENDEDOR  ASC );

CREATE INDEX XIFNMCOMPRADOR ON COMPRADOR ( NMCOMPRADOR  ASC );
```

3.

```
CREATE INDEX XIFCDIMOVEL ON OFERTA
    ( CDIMOVEL    ASC );

CREATE INDEX XIFVLIMOVEL ON OFERTA
    ( VLOFERTA    DESC );
```

Capítulo 5

Atenção! Não esqueça que pode haver diferenças entre a forma de entrada de dados nos campos do tipo Data. Caso seja necessário, realize a conversão do tipo de dado com o comando CAST.

1.

```
INSERT INTO ESTADO (SGESTADO,NMESTADO)
    VALUES ('SP','SÃO PAULO');

INSERT INTO ESTADO (SGESTADO,NMESTADO)
    VALUES ('RJ','RIO DE JANEIRO');
```

2.

```
INSERT INTO CIDADE (CDCIDADE, SGESTADO, NMCIDADE)
    VALUES (1,'SP','SÃO PAULO');
```

```
INSERT INTO CIDADE (CDCIDADE, SGESTADO, NMCIDADE)
    VALUES (2,'SP','SANTO ANDRÉ');

INSERT INTO CIDADE (CDCIDADE, SGESTADO, NMCIDADE)
    VALUES (3,'SP','CAMPINAS');

INSERT INTO CIDADE (CDCIDADE, SGESTADO, NMCIDADE)
    VALUES (1,'RJ','RIO DE JANEIRO');

INSERT INTO CIDADE (CDCIDADE, SGESTADO, NMCIDADE)
    VALUES (2,'RJ','NITERÓI');
```

3.

```
INSERT INTO BAIRRO (CDBAIRRO,CDCIDADE,SGESTADO,NMBAIRRO)
    VALUES (1,1,'SP','JARDINS');

INSERT INTO BAIRRO (CDBAIRRO,CDCIDADE,SGESTADO,NMBAIRRO)
    VALUES (2,1,'SP','MORUMBI');

INSERT INTO BAIRRO (CDBAIRRO,CDCIDADE,SGESTADO,NMBAIRRO)
    VALUES (3,1,'SP','AEROPORTO');

INSERT INTO BAIRRO (CDBAIRRO,CDCIDADE,SGESTADO,NMBAIRRO)
    VALUES (1,1,'RJ','AEROPORTO');

INSERT INTO BAIRRO (CDBAIRRO,CDCIDADE,SGESTADO,NMBAIRRO)
    VALUES (2,1,'RJ','FLAMENGO');
```

4.

```
INSERT INTO VENDEDOR (CDVENDEDOR, NMVENDEDOR, NMENDERECO, EMAIL)
    VALUES (1,'MARIA DA SILVA','RUA DO GRITO,
    45','msilva@novatec.com.br');

INSERT INTO VENDEDOR (CDVENDEDOR, NMVENDEDOR, NMENDERECO, EMAIL)
    VALUES (2,'MARCOS ANDRADE','AV. DA SAUDADE,
    325','mandrade@novatec.com.br');

INSERT INTO VENDEDOR (CDVENDEDOR, NMVENDEDOR, NMENDERECO, EMAIL)
    VALUES (3,'ANDRE CARDOSO','AV BRASIL,
    401','acardoso@novatec.com.br');

INSERT INTO VENDEDOR (CDVENDEDOR, NMVENDEDOR, NMENDERECO, EMAIL)
    VALUES (4,'TATIANA SOUZA','RUA DO IMPERADOR,
    778','tsouza@novatec.com.br');
```

5.

```
INSERT INTO IMOVEL (CDIMOVEL, CDVENDEDOR, CDBAIRRO, CDCIDADE,
    SGESTADO, NMENDERECO, NRAREAUTIL, NRAREATOTAL, VLPRECO,
    DTLANCTO)
```

```
    VALUES (1,1,1,1,'SP','AL TIETE, 3304 AP
    101',250,400,180000,'05/11/2001');

INSERT INTO IMOVEL (CDIMOVEL, CDVENDEDOR, CDBAIRRO, CDCIDADE,
    SGESTADO, NMENDERECO, NRAREAUTIL, NRAREATOTAL, VLPRECO,
    DTLANCTO)
    VALUES (2,1,2,1,'SP','AV MORUMBI, 2230',150,250,135000,'05/
    12/2001');

INSERT INTO IMOVEL (CDIMOVEL, CDVENDEDOR, CDBAIRRO, CDCIDADE,
    SGESTADO, NMENDERECO, NRAREAUTIL, NRAREATOTAL, VLPRECO,
    DTLANCTO)
    VALUES (3,2,1,1,'RJ','R GAL OSORIO, 445 AP
    34',250,400,185000,'15/01/2002');

INSERT INTO IMOVEL (CDIMOVEL, CDVENDEDOR, CDBAIRRO, CDCIDADE,
    SGESTADO, NMENDERECO, NRAREAUTIL, NRAREATOTAL, VLPRECO,
    DTLANCTO)
    VALUES (4,2,2,1,'RJ','R D PEDRO I, 882',120,200,110000,'15/
    01/2002');

INSERT INTO IMOVEL (CDIMOVEL, CDVENDEDOR, CDBAIRRO, CDCIDADE,
    SGESTADO, NMENDERECO, NRAREAUTIL, NRAREATOTAL, VLPRECO,
    DTLANCTO)
    VALUES (5,3,3,1,'SP','AV RUBEM BERTA,
    2355',110,200,95000,'05/02/2002');

INSERT INTO IMOVEL (CDIMOVEL, CDVENDEDOR, CDBAIRRO, CDCIDADE,
    SGESTADO, NMENDERECO, NRAREAUTIL, NRAREATOTAL, VLPRECO,
    DTLANCTO)
    VALUES (6,4,1,1,'RJ','R GETULIO VARGAS,
    552',200,300,99000,'10/02/2002');
```

6.

```
INSERT INTO COMPRADOR( CDCOMPRADOR, NMCOMPRADOR, NMENDERECO,
    EMAIL)
    VALUES (1,'EMMANUEL ANTUNES', 'R SARAIVA, 452',
    'eantunes@novatec.com.br');

INSERT INTO COMPRADOR( CDCOMPRADOR, NMCOMPRADOR, NMENDERECO,
    EMAIL)
    VALUES (2,'JOANA PEREIRA', 'AV PORTUGAL, 52',
    'jpereira@novatec.com.br');

INSERT INTO COMPRADOR( CDCOMPRADOR, NMCOMPRADOR, NMENDERECO,
    EMAIL)
    VALUES (3,'RONALDO CAMPELO', 'R ESTADOS UNIDOS, 790',
    'rcampelo@novatec.com.br');

INSERT INTO COMPRADOR( CDCOMPRADOR, NMCOMPRADOR, NMENDERECO,
    EMAIL)
    VALUES (4,'MANFRED AUGUSTO', 'AV BRASIL, 351',
    'maugusto@novatec.com.br');
```

7.

```
INSERT INTO OFERTA (CDCOMPRADOR, CDIMOVEL, VLOFERTA, DTOFERTA)
    VALUES (1,1,170000,'10/01/2002');

INSERT INTO OFERTA (CDCOMPRADOR, CDIMOVEL, VLOFERTA, DTOFERTA)
    VALUES (1,3,180000,'10/01/2002');

INSERT INTO OFERTA (CDCOMPRADOR, CDIMOVEL, VLOFERTA, DTOFERTA)
    VALUES (2,2,135000,'15/02/2002');

INSERT INTO OFERTA (CDCOMPRADOR, CDIMOVEL, VLOFERTA, DTOFERTA)
    VALUES (2,4,100000,'15/02/2002');

INSERT INTO OFERTA (CDCOMPRADOR, CDIMOVEL, VLOFERTA, DTOFERTA)
    VALUES (3,1,160000,'05/01/2002');

INSERT INTO OFERTA (CDCOMPRADOR, CDIMOVEL, VLOFERTA, DTOFERTA)
    VALUES (3,2,140000,'20/02/2002');
```

8.

```
UPDATE IMOVEL
    SET VLPRECO = VLPRECO * 1.1;
```

9.

```
UPDATE IMOVEL
    SET VLPRECO = VLPRECO * .95
    WHERE CDVENDEDOR = 1;
```

10.

```
UPDATE OFERTA
    SET VLOFERTA = VLOFERTA * 1.05
    WHERE CDCOMPRADOR = 2;
```

11.

```
UPDATE COMPRADOR
    SET NMENDERECO = 'R ANANAS, 45', SGESTADO = 'RJ'
    WHERE CDCOMPRADOR = 3;
```

12.

```
UPDATE OFERTA
    SET VLOFERTA = 101000
    WHERE CDCOMPRADOR = 2 AND CDIMOVEL = 4;
```

13.

```
DELETE FROM OFERTA
    WHERE CDCOMPRADOR = 3 AND CDIMOVEL = 1;
```

14.

```
DELETE FROM CIDADE
    WHERE CDCIDADE = 3 AND SGESTADO = 'SP';
```

15.

```
INSERT INTO FAIXA_IMOVEL (CDFAIXA,NMFAIXA,VLMINIMO,VLMAXIMO)
    VALUES (1,'BAIXO',0,105000);

INSERT INTO FAIXA_IMOVEL (CDFAIXA,NMFAIXA,VLMINIMO,VLMAXIMO)
    VALUES (2,'MÉDIO',105001,180000);

INSERT INTO FAIXA_IMOVEL (CDFAIXA,NMFAIXA,VLMINIMO,VLMAXIMO)
    VALUES (3,'ALTO',180001,999999);
```

Capítulo 6

1.

```
SELECT * FROM BAIRRO;
```

2.

```
SELECT CDCOMPRADOR, NMCOMPRADOR, EMAIL
    FROM COMPRADOR;
```

3.

```
SELECT CDVENDEDOR, NMVENDEDOR, EMAIL
    FROM VENDEDOR
    ORDER BY NMVENDEDOR;
```

4.

```
SELECT CDVENDEDOR, NMVENDEDOR, EMAIL
    FROM VENDEDOR
    ORDER BY NMVENDEDOR DESC;
```

5.

```
SELECT *
    FROM BAIRRO
    WHERE SGESTADO = 'SP';
```

6.

```
SELECT CDIMOVEL, CDVENDEDOR, VLPRECO
    FROM IMOVEL
    WHERE CDVENDEDOR = 2;
```

7.

```
SELECT CDIMOVEL, CDVENDEDOR, VLPRECO, SGESTADO
    FROM IMOVEL
    WHERE VLPRECO < 150000 AND SGESTADO = 'RJ';
```

8.

```
SELECT CDIMOVEL, CDVENDEDOR, VLPRECO, SGESTADO
    FROM IMOVEL
    WHERE VLPRECO < 150000 OR CDVENDEDOR = 1;
```

9.

```
SELECT CDIMOVEL, CDVENDEDOR, VLPRECO, SGESTADO
    FROM IMOVEL
    WHERE VLPRECO < 150000 AND CDVENDEDOR != 2;
```

10.

```
SELECT CDCOMPRADOR,NMCOMPRADOR,NMENDERECO,SGESTADO
    FROM COMPRADOR
    WHERE SGESTADO IS NULL;
```

11.

```
SELECT CDCOMPRADOR,NMCOMPRADOR,NMENDERECO,SGESTADO
    FROM COMPRADOR
    WHERE SGESTADO IS NOT NULL;
```

12.

```
SELECT *
    FROM OFERTA
    WHERE VLOFERTA BETWEEN 100000 AND 150000;
```

13.

```
SELECT *
    FROM OFERTA
    WHERE DTOFERTA BETWEEN '01/02/02' AND '01/03/02';
```

14.

```
SELECT *
    FROM VENDEDOR
    WHERE NMVENDEDOR LIKE 'M%';
```

15.

```
SELECT *
    FROM VENDEDOR
    WHERE NMVENDEDOR LIKE '_A%';
```

16.

```
SELECT *
    FROM COMPRADOR
    WHERE NMENDERECO LIKE '%U%';
```

17.

```
SELECT *
    FROM OFERTA
    WHERE CDIMOVEL IN (1,2);
```

18.

```
SELECT *
    FROM IMOVEL
    WHERE CDIMOVEL IN (2,3)
    ORDER BY NMENDERECO;
```

19.

```
SELECT *
    FROM OFERTA
    WHERE CDIMOVEL IN (2,3) AND VLOFERTA > 140000
    ORDER BY DTOFERTA DESC;
```

20.

```
SELECT *
    FROM IMOVEL
    WHERE VLPRECO BETWEEN 110000 AND 200000 OR CDVENDEDOR = 1
    ORDER BY NRAREAUTIL;
```

Capítulo 7

1.

```
SELECT SYSDATE FROM DUAL;

SELECT CURRENT_DATE FROM DUAL;
```

2.

```
SELECT CDIMOVEL,VLPRECO,VLPRECO*1.1
    FROM IMOVEL;
```

3.

```
SELECT CDIMOVEL,VLPRECO,VLPRECO*1.1,VLPRECO*.1
    FROM IMOVEL;
```

4.

```
SELECT UPPER(NMVENDEDOR),LOWER(EMAIL)
    FROM VENDEDOR;
```

5.

```
SELECT NMCOMPRADOR || '-' || NMCIDADE
    FROM COMPRADOR;
```

6.

```
SELECT NMCOMPRADOR
    FROM COMPRADOR
    WHERE NMCOMPRADOR LIKE '%A%';
```

7.

```
SELECT SUBSTR(NMCOMPRADOR,1,1),NMBAIRRO
    FROM COMPRADOR;
```

8.

```
SELECT CDIMOVEL,CURRENT_DATE - DTOFERTA
    FROM OFERTA;
```

9.

```
SELECT CDIMOVEL,DTLANCTO,CURRENT_DATE - DTLANCTO
    FROM IMOVEL;
```

10.

```
SELECT CDIMOVEL,DTLANCTO + 15,CURRENT_DATE - DTLANCTO
    FROM IMOVEL;
```

Capítulo 8

1.

```
SELECT i.CDIMOVEL,v.CDVENDEDOR,v.NMVENDEDOR,i.SGESTADO
    FROM IMOVEL i, VENDEDOR v
    WHERE i.CDVENDEDOR = v.CDVENDEDOR;
```

ou

```
SELECT
    IMOVEL.CDIMOVEL,CDVENDEDOR,VENDEDOR.NMVENDEDOR,IMOVEL.SGESTADO
    FROM IMOVEL JOIN VENDEDOR USING (CDVENDEDOR);
```

2.

```
SELECT c.CDCOMPRADOR,c.NMCOMPRADOR,o.CDIMOVEL,o.VLOFERTA
    FROM COMPRADOR c, OFERTA o
    WHERE c.CDCOMPRADOR = o.CDCOMPRADOR;

ou

SELECT
    CDCOMPRADOR,COMPRADOR.NMCOMPRADOR,OFERTA.CDIMOVEL,OFERTA.VLOFERTA
    FROM COMPRADOR JOIN OFERTA USING (CDCOMPRADOR);
```

3.

```
SELECT i.CDIMOVEL,i.VLPRECO,b.NMBAIRRO
    FROM IMOVEL i, BAIRRO b
    WHERE i.CDVENDEDOR = 3
    AND i.CDBAIRRO = b.CDBAIRRO
    AND i.SGESTADO = b.SGESTADO
    AND i.CDCIDADE = b.CDCIDADE;

ou

SELECT IMOVEL.CDIMOVEL,IMOVEL.VLPRECO,BAIRRO.NMBAIRRO
    FROM IMOVEL NATURAL JOIN BAIRRO
    WHERE IMOVEL.CDVENDEDOR = 3;
```

4.

```
SELECT DISTINCT i.CDIMOVEL
    FROM IMOVEL i, OFERTA o
    WHERE i.CDIMOVEL = o.CDIMOVEL;
```

5.

```
SELECT i.CDIMOVEL, o.VLOFERTA
    FROM IMOVEL i, OFERTA o
    WHERE i.CDIMOVEL = o.CDIMOVEL (+);

ou

SELECT CDIMOVEL, VLOFERTA
    FROM IMOVEL FULL OUTER JOIN OFERTA USING (CDIMOVEL);
```

6.

```
SELECT c.CDCOMPRADOR,c.NMCOMPRADOR,o.CDIMOVEL,o.VLOFERTA
    FROM COMPRADOR c, OFERTA o
    WHERE c.CDCOMPRADOR = o.CDCOMPRADOR;
```

ou

```
SELECT
    CDCOMPRADOR,COMPRADOR.NMCOMPRADOR,OFERTA.CDIMOVEL,OFERTA.VLOFERTA
    FROM COMPRADOR JOIN OFERTA USING( CDCOMPRADOR );
```

7.

```
SELECT c.CDCOMPRADOR,c.NMCOMPRADOR,o.CDIMOVEL,o.VLOFERTA
    FROM COMPRADOR c, OFERTA o
    WHERE c.CDCOMPRADOR = o.CDCOMPRADOR (+);
```

ou

```
SELECT
    CDCOMPRADOR,COMPRADOR.NMCOMPRADOR,OFERTA.CDIMOVEL,OFERTA.VLOFERTA
    FROM COMPRADOR FULL OUTER JOIN OFERTA USING( CDCOMPRADOR );
```

8.

```
SELECT a.CDIMOVEL,a.NMENDERECO,b.NMENDERECO INDICADO
    FROM IMOVEL a, IMOVEL b
    WHERE a.IMOVEL_INDICADO = b.CDIMOVEL;
```

ou

```
SELECT a.CDIMOVEL,a.NMENDERECO,b.NMENDERECO INDICADO
    FROM IMOVEL a JOIN IMOVEL b ON (a.IMOVEL_INDICADO =
    b.CDIMOVEL)
```

9.

```
SELECT a.CDIMOVEL,a.NMENDERECO,v.NMVENDEDOR,b.NMENDERECO
    INDICADO,v1.NMVENDEDOR V_INDICADO
    FROM IMOVEL a, IMOVEL b, VENDEDOR v, VENDEDOR v1
    WHERE a.IMOVEL_INDICADO = b.CDIMOVEL
    AND a.CDVENDEDOR = v.CDVENDEDOR
    AND b.CDVENDEDOR = v1.CDVENDEDOR;
```

10.

```
SELECT i.NMENDERECO,b.NMBAIRRO,i.VLPRECO,f.NMFAIXA
    FROM IMOVEL i, BAIRRO b, FAIXA_IMOVEL f
    WHERE i.SGESTADO = b.SGESTADO
    AND i.CDCIDADE = b.CDCIDADE
    AND i.CDBAIRRO = b.CDBAIRRO
    AND i.VLPRECO BETWEEN f.VLMINIMO AND f.VLMAXIMO;
```

Capítulo 9

1.

```
SELECT MAX(VLOFERTA) MAIOR,MIN(VLOFERTA) MENOR, AVG(VLOFERTA)
    MEDIA
    FROM OFERTA;
```

2.

```
SELECT STDDEV(VLPRECO) DESVIO, VARIANCE(VLPRECO) VARIANCIA
    FROM IMOVEL;
```

3.

```
SELECT CAST(STDDEV(VLPRECO) AS DECIMAL(16,2)) DESVIO,
    CAST(VARIANCE(VLPRECO) AS DECIMAL(16,2)) VARIANCIA
    FROM IMOVEL;
```

4.

```
SELECT MAX(VLPRECO) MAIOR,MIN(VLPRECO) MENOR, AVG(VLPRECO)
    MEDIA, SUM(VLPRECO) TOTAL
    FROM IMOVEL;
```

5.

```
SELECT CDBAIRRO, MAX(VLPRECO) MAIOR,MIN(VLPRECO) MENOR,
    AVG(VLPRECO) MEDIA, SUM(VLPRECO) TOTAL
    FROM IMOVEL
    GROUP BY CDBAIRRO;
```

6.

```
SELECT CDVENDEDOR, COUNT(*)
    FROM IMOVEL
    GROUP BY CDVENDEDOR
    ORDER BY COUNT(*);
```

7.

```
SELECT MAX(VLPRECO) - MIN(VLPRECO) DIFERENCA
    FROM IMOVEL;
```

8.

```
SELECT CDVENDEDOR,MIN(VLPRECO)
    FROM IMOVEL
    WHERE VLPRECO >= 10000
    GROUP BY CDVENDEDOR;
```

9.

```
SELECT CDCOMPRADOR,AVG(VLOFERTA) MEDIA,COUNT(*)
    FROM OFERTA
    GROUP BY CDCOMPRADOR;
```

10.

```
SELECT COUNT(*)
    FROM OFERTA
    WHERE TO_CHAR(DTOFERTA,'YYYY') IN ('2000','2001','2002');

ou

SELECT TO_CHAR(DTOFERTA,'YYYY') ANO, COUNT(*) OFERTAS
    FROM OFERTA
    GROUP BY TO_CHAR(DTOFERTA,'YYYY')
    HAVING TO_CHAR(DTOFERTA,'YYYY') IN ('2000','2001','2002');
```

Capítulo 10

1.

```
SELECT CDIMOVEL,NMENDERECO
    FROM IMOVEL
    WHERE CDBAIRRO = (SELECT CDBAIRRO
    FROM IMOVEL
    WHERE CDIMOVEL = 2) AND CDIMOVEL != 2;
```

2.

```
SELECT CDIMOVEL,NMENDERECO,VLPRECO
    FROM IMOVEL
    WHERE VLPRECO > (SELECT AVG(VLPRECO)
    FROM IMOVEL);
```

3.

```
SELECT c.NMCOMPRADOR, o.VLOFERTA
    FROM COMPRADOR c, OFERTA o
    WHERE o.CDCOMPRADOR = c.CDCOMPRADOR
    AND o.VLOFERTA > 70000;

ou

SELECT NMCOMPRADOR
    FROM COMPRADOR
    WHERE CDCOMPRADOR IN (SELECT CDCOMPRADOR
    FROM OFERTA
    WHERE VLOFERTA > 70000);
```

4.

```
SELECT CDIMOVEL,CDCOMPRADOR,VLOFERTA
    FROM OFERTA
    WHERE VLOFERTA > (SELECT AVG(VLOFERTA)
    FROM OFERTA);
```

5.

```
SELECT CDIMOVEL,NMENDERECO,VLPRECO
     FROM IMOVEL i
     WHERE VLPRECO > (SELECT AVG(VLPRECO)
     FROM IMOVEL
     WHERE CDBAIRRO = i.CDBAIRRO);
```

6.

```
SELECT CDBAIRRO,MAX(VLPRECO)
     FROM IMOVEL
     GROUP BY CDBAIRRO
     HAVING MAX(VLPRECO) > (SELECT AVG(VLPRECO) FROM IMOVEL);
```

7.

```
SELECT CDIMOVEL,CDVENDEDOR,VLPRECO
     FROM IMOVEL i
     WHERE VLPRECO = (SELECT MIN(VLPRECO)
     FROM IMOVEL
     WHERE CDVENDEDOR = i.CDVENDEDOR);
```

8.

```
SELECT i.CDIMOVEL,o.CDCOMPRADOR,i.DTLANCTO
     FROM IMOVEL i, OFERTA o
     WHERE i.CDIMOVEL = o.CDIMOVEL
     AND i.DTLANCTO BETWEEN CURRENT_DATE - 180 AND CURRENT_DATE -
     30
     AND i.CDVENDEDOR = 2;
```

ou

```
SELECT CDIMOVEL,CDCOMPRADOR
     FROM OFERTA
     WHERE CDIMOVEL IN (SELECT CDIMOVEL
     FROM IMOVEL
     WHERE DTLANCTO BETWEEN CURRENT_DATE - 180 AND CURRENT_DATE -
     30 AND CDVENDEDOR = 2);
```

9.

```
SELECT CDIMOVEL,NMENDERECO,VLPRECO
     FROM IMOVEL i
     WHERE VLPRECO = (SELECT MIN(VLPRECO)
     FROM IMOVEL
     WHERE CDVENDEDOR != i.CDVENDEDOR);
```

10.

```
SELECT CDIMOVEL,CDCOMPRADOR,VLOFERTA
     FROM OFERTA
     WHERE VLOFERTA < (SELECT MAX(VLOFERTA)
     FROM OFERTA
     WHERE CDCOMPRADOR = 2)
     AND CDCOMPRADOR !=2;
```

11.

```
SELECT CDIMOVEL,SGESTADO,CDCIDADE,CDVENDEDOR
    FROM IMOVEL
    WHERE (SGESTADO,CDCIDADE) IN (SELECT SGESTADO,CDCIDADE
    FROM IMOVEL
    WHERE CDVENDEDOR = 3)
    AND CDVENDEDOR != 3;
```

12.

```
SELECT NMBAIRRO
    FROM BAIRRO
    WHERE (SGESTADO,CDCIDADE,CDBAIRRO) =
    (SELECT SGESTADO,CDCIDADE,CDBAIRRO
    FROM IMOVEL
    WHERE CDIMOVEL = 5);
```

Capítulo 11

1.

```
SELECT CDCOMPRADOR CODIGO,NMCOMPRADOR NOME
    FROM COMPRADOR
    UNION ALL
    SELECT CDVENDEDOR CODIGO,NMVENDEDOR NOME
    FROM VENDEDOR;
```

2.

```
SELECT CDCOMPRADOR CODIGO,NMCOMPRADOR NOME
    FROM COMPRADOR
    UNION ALL
    SELECT CDVENDEDOR CODIGO,NMVENDEDOR NOME
    FROM VENDEDOR;
```

3.

```
SELECT v.NMVENDEDOR, i.NMENDERECO
    FROM VENDEDOR v, IMOVEL i, COMPRADOR c, OFERTA o
    WHERE i.CDIMOVEL = o.CDIMOVEL
    AND i.CDVENDEDOR = v.CDVENDEDOR
    AND o.CDCOMPRADOR = c.CDCOMPRADOR
    AND o.CDCOMPRADOR = 2
    UNION ALL
    SELECT v.NMVENDEDOR, i.NMENDERECO
    FROM VENDEDOR v, IMOVEL i, COMPRADOR c, OFERTA o
    WHERE i.CDIMOVEL = o.CDIMOVEL
    AND i.CDVENDEDOR = v.CDVENDEDOR
    AND o.CDCOMPRADOR = c.CDCOMPRADOR
    AND i.CDBAIRRO = 1;
```

ou

```
SELECT v.NMVENDEDOR, i.NMENDERECO
    FROM VENDEDOR v, IMOVEL i, COMPRADOR c, OFERTA o
    WHERE i.CDIMOVEL = o.CDIMOVEL
    AND i.CDVENDEDOR = v.CDVENDEDOR
    AND o.CDCOMPRADOR = c.CDCOMPRADOR
    AND (o.CDCOMPRADOR = 2 OR i.CDBAIRRO = 1);
```

4.

```
SELECT CDCOMPRADOR
    FROM COMPRADOR
    MINUS
    SELECT CDCOMPRADOR
    FROM OFERTA;
```

5.

```
SELECT CDVENDEDOR
    FROM VENDEDOR
    MINUS
    SELECT CDVENDEDOR
    FROM IMOVEL;
```

6.

```
SELECT CDCOMPRADOR
    FROM COMPRADOR
    INTERSECT
    SELECT CDCOMPRADOR
    FROM OFERTA;
```

7.

```
SELECT CDVENDEDOR
    FROM VENDEDOR
    INTERSECT
    SELECT CDVENDEDOR
    FROM IMOVEL;
```

8.

```
SELECT CDIMOVEL,NMENDERECO,VLPRECO,
    CASE
    WHEN VLPRECO > 100000 THEN VLPRECO * .9
    WHEN VLPRECO > 50000 AND VLPRECO <= 100000 THEN .95
    WHEN VLPRECO > 30000 AND VLPRECO <= 50000  THEN .97
    ELSE VLPRECO
    END DESCONTO
    FROM IMOVEL;
```

9.

```
SELECT NMENDERECO,CDVENDEDOR,
    CASE STVENDIDO
    WHEN 'S' THEN 'VENDIDO'
    ELSE 'DISPONÍVEL'
    END STATUS
    FROM IMOVEL;
```

10.

```
SELECT CDCOMPRADOR,
    CASE
    WHEN DTOFERTA < CURRENT_DATE - 30 THEN 'ANTIGA'
    ELSE 'NOVA'
    END TIPO
    FROM OFERTA;
```

Capítulo 12

1. Deve-se especificar o nome do *schema* ou usuário, seguido do nome da tabela e depois da coluna. Asim: celso.comprador.nmcomprador.

2.

```
CREATE SCHEMA LOJA

AUTHORIZATION SENHA {

CREATE TABLE SEDE (CDLOJA INTEGER, NMLOJA VARCHAR(50))

};
```

3.

```
CREATE DOMAIN SEXO CHAR(1)

 CHECK (SEXO IN ('M','F'));
```

4.

```
CREATE VIEW vVENDEDOR AS

 SELECT CDVENDEDOR,NMVENDEDOR,NMENDERECO

 FROM VENDEDOR;
```

5.

```
SELECT * FROM vVENDEDOR;
```

6.

```
CREATE VIEW vIMOVEL_BAIRRO1 AS

 SELECT CDIMOVEL,NMENDERECO,VLPRECO
    FROM IMOVEL
    WHERE CDBAIRRO = 1;
```

7.

```
CREATE VIEW vIMOVEL AS

SELECT v.NMVENDEDOR, i.NMENDERECO, b.NMBAIRRO, c.NMCIDADE,
    e.NMESTADO
    FROM IMOVEL i, VENDEDOR v, BAIRRO b, CIDADE c, ESTADO e
    WHERE i.CDVENDEDOR = v.CDVENDEDOR
    AND i.SGESTADO   = e.SGESTADO
    AND i.SGESTADO   = c.SGESTADO
    AND i.CDCIDADE   = c.CDCIDADE
    AND i.SGESTADO   = b.SGESTADO
    AND i.CDCIDADE   = b.CDCIDADE
    AND i.CDBAIRRO   = b.CDBAIRRO;
```

8.

```
CREATE VIEW vIMOVEL_CIDADE AS

SELECT SGESTADO,CDCIDADE,AVG(VLPRECO) MEDIA

  FROM IMOVEL

 GROUP BY SGESTADO,CDCIDADE;
```

9.

```
CREATE VIEW vOFERTA AS

SELECT c.NMCOMPRADOR, o.VLOFERTA, i.NMENDERECO

  FROM OFERTA o, IMOVEL i, COMPRADOR c

 WHERE o.CDIMOVEL = i.CDIMOVEL

   AND o.CDCOMPRADOR = c.CDCOMPRADOR;
```

10.

```
CREATE GLOBAL TEMPORARY TABLE tempVENDEDOR

(CDVENDEDOR INTEGER PRIMARY KEY,

 NMVENDEDOR VARCHAR(60),

 NMENDERECO VARCHAR(60));
```

11.

```
INSERT INTO tempVENDEDOR
    SELECT CDVENDEDOR,NMVENDEDOR,NMENDERECO
    FROM VENDEDOR
    WHERE CDVENDEDOR < 5;
```

12.

```
SELECT * FROM tempVENDEDOR;
```

13.

```
SELECT ROWNUM AS RANK, CDIMOVEL, NMENDERECO, VLPRECO
    FROM (SELECT CDIMOVEL,NMENDERECO,VLPRECO
    FROM IMOVEL
    ORDER BY VLPRECO)
    WHERE ROWNUM <=3;
```

14.

```
SELECT ROWNUM AS RANK, CDIMOVEL, CDCOMPRADOR, VLOFERTA
    FROM (SELECT CDIMOVEL, CDCOMPRADOR, VLOFERTA
    FROM OFERTA
    ORDER BY VLOFERTA DESC)
    WHERE ROWNUM <=5
    MINUS
    SELECT ROWNUM AS RANK, CDIMOVEL, CDCOMPRADOR, VLOFERTA
    FROM (SELECT CDIMOVEL, CDCOMPRADOR, VLOFERTA
    FROM OFERTA
    ORDER BY VLOFERTA DESC)
    WHERE ROWNUM <=3;
```

Capítulo 13

1.

```
CREATE USER LOJA

 IDENTIFIED BY LOJA;
```

2.

```
GRANT CONNECT TO LOJA;
```

3.

```
GRANT SELECT ON COMPRADOR TO LOJA;

GRANT SELECT ON VENDEDOR TO LOJA;
```

4.

```
GRANT INSERT,UPDATE ON OFERTA TO LOJA;

GRANT INSERT,UPDATE ON IMOVEL TO LOJA;
```

5.

```
GRANT DELETE ON CIDADE TO LOJA;

GRANT DELETE ON ESTADO TO LOJA;

GRANT DELETE ON BAIRRO TO LOJA;
```

6.

```
REVOKE DELETE ON CIDADE FROM LOJA;

REVOKE DELETE ON ESTADO FROM LOJA;

REVOKE DELETE ON BAIRRO FROM LOJA;
```

7.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

8.

```
SET CONSTRAINTS ALL DEFERRED;
```

Índice Remissivo

Símbolos

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

Y

Curso Prático SQL

SQL (Structured Query Language) é a linguagem-padrão para consulta, inclusão, exclusão e alteração de dados em bancos de dados relacionais e objeto-relacionais. Quem pretende utilizar os principais bancos de dados do mercado, como Oracle, Microsoft SQL Server, Sybase, DB/2, MySQL, PostgreSQL, entre outros, deve conhecer a linguagem SQL. Mesmo que cada banco de dados tenha sua própria implementação do padrão SQL, existe uma parte básica comum a todos eles. Este livro não está vinculado a um único banco de dados, sendo possível utilizá-lo em todos eles, com poucas adaptações.

Este livro é indicado a usuários iniciantes e avançados ou desenvolvedores que queiram conhecer a linguagem SQL e/ou se aprofundar nela. Demonstra os principais conceitos com clareza e simplicidade, sem perder a profundidade técnica necessária à compreensão da linguagem. O livro está dividido em três partes: a primeira trata dos conceitos relacionados à criação de modelos de dados estáveis (incluindo técnicas de modelagem e normalização de dados); a segunda apresenta os comandos básicos de manipulação de dados utilizando SQL; e a terceira é direcionada a usuários avançados que desejam utilizar recursos adicionais da linguagem.

Inclui exercícios e exemplos que irão ajudar o leitor a consolidar o conhecimento da linguagem. Ao final do livro, há dois casos montados e implementados: um será o caso de uma loja de CDs (utilizado para desenvolver a matéria) e o outro, uma imobiliária virtual (criado com base nos exercícios propostos).

Sobre o autor

Celso Henrique Poderoso de Oliveira (henrique@nq.com.br), economista especializado em Sistemas de Informação, é diretor técnico de Desenvolvimento de Sistemas e Consultoria da Net Quality Informática (www.nq.com.br). É professor de banco de dados Oracle na FIAP - Faculdade de Informática e Administração Paulista (www.fiap.com.br).

ISBN 85-7522-024-1

9 788575 220242

www.novateceditora.com.br

novatec editora