FABRÍCIO AUGUSTO FERRARI

CRIE BANCO DE DADOS EM MySQL

DESVENDE OS RECURSOS DESTA PODEROSA FERRAMENTA

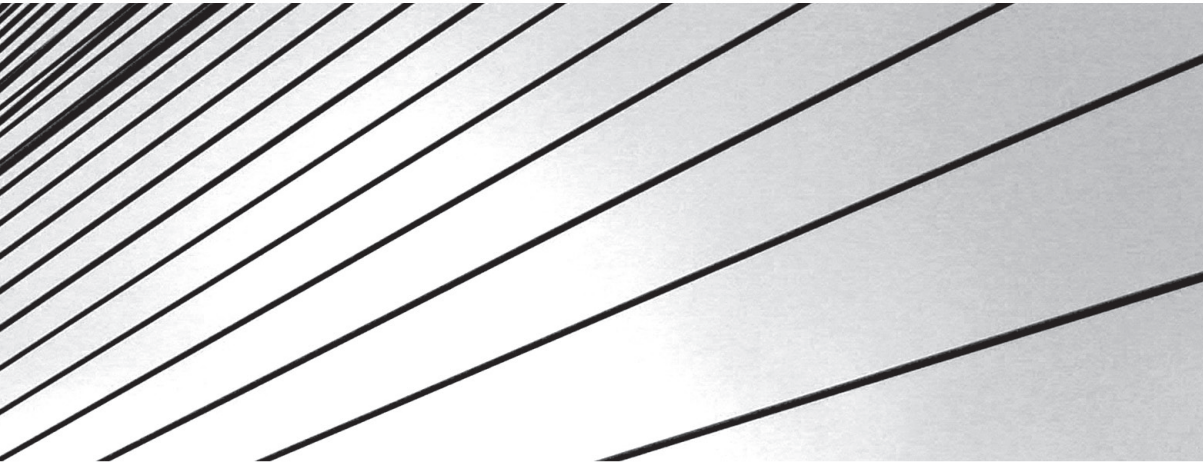
- PRINCIPAIS COMANDOS SQL
- INSTALAÇÃO DO MYSQL
- CRIAÇÃO DE TABELAS E INCLUSÃO DE REGISTROS
- BANCOS DE DADOS RELACIONAIS
- CRIAÇÃO DE RELACIONAMENTOS
- INSERÇÃO DE DADOS EM UMA TABELA
- REALIZAÇÃO DE CONSULTAS SQL
- EXEMPLOS PRÁTICOS DE CONSULTAS E MANIPULAÇÃO DE DADOS
- SISTEMAS DE GERENCIAMENTO DE BANCOS DE DADOS RELACIONAIS

E MUITO MAIS...

CRIE BANCO DE DADOS EM MySQL

DIGERATI BOOKS

CRIE BANCO DE DADOS EM MySQL



FABRÍCIO AUGUSTO FERRARI

DIGERATI BOOKS

Diretor Editorial
Luis Matos

Assistência Editorial
Monalisa Neves
Erika Sá

Edição de Arte e Projeto Gráfico
Daniele Fátima

Preparação dos originais
Ricardo Dino de Freitas

Revisão
Guilherme Laurito Summa

Diagramação
Rogério Chagas

Capa
FullCASE Comunicação

Dados Internacionais de Catalogação na Publicação (CIP)
(Câmara Brasileira do Livro, SP, Brasil)

F375c Ferrari, Fabrício Augusto.

Crie banco de dados em MySQL / Fabrício Augusto Ferrari. – São Paulo: Digerati Books, 2007.
128 p.

ISBN 978-85-60480-25-8

1. MySQL (Linguagem de programação de computadores). 2. Bancos de dados. I. Título.

CDD 005.13

Universo dos Livros Editora Ltda.
Rua Tito, 1.609
CEP 05051-001 • São Paulo/SP
Telefone: (11) 3648-9090 • Fax: (11) 3648-9083
www.universodoslivros.com.br
e-mail: editor@universodoslivros.com.br

Conselho Administrativo: Alessandro Gerardi, Alessio Fon Melozo, Luis Afonso G. Neira, Luis Matos e William Nakamura.

Sumário

Capítulo 1

Introdução aos bancos de dados

Antes de nos aprofundarmos no conhecimento do SQL, é fundamental entendermos o que é e como é estruturado um banco de dados. Neste tópico, veremos algumas noções importantes para criação de um banco de dados eficiente e bem planejado.

Apesar de o termo "banco de dados" parecer um tanto quanto técnico para a maioria das pessoas, trata-se de um conceito amplamente conhecido e empregado por quase toda população mundial. De fato, a grande maioria das pessoas hoje tem acesso a equipamentos, cuja função (principal ou secundária) é o armazenamento de informações. Quem, hoje em dia, não usa, por exemplo, um telefone celular? Desde o seu surgimento, esse tipo de aparelho possui uma agenda, na qual podemos gravar nomes e telefones para, em um segundo momento, acessá-los. Uma lista telefônica impressa também é um exemplo válido de banco de dados, pois nela são relatados todos os nomes, endereços e números de telefone das empresas e dos moradores da sua cidade e, eventualmente, dos arredores.

Tudo isso remete ao conceito de banco de dados, ou seja, um local no qual é possível armazenar informações, para consulta ou utilização, quando necessário.

Independentemente do aplicativo que se deseja usar para o armazenamento e manipulação das informações, todos os bancos de dados são constituídos por três elementos básicos: campos, registros e tabelas.

O que são campos?

Num banco de dados, é chamado de campo o espaço reservado para a inserção de um determinado dado.

O que são registros?

Um registro nada mais é que um conjunto de campos, ou seja, de dados sobre um determinado assunto.

O que são tabelas?

Todos os registros, isto é, as fichas que contêm os dados que armazenamos, são agrupados nas tabelas do banco de dados.

Introdução ao SQL

Como vimos no tópico anterior, todos os sistemas de computação para o gerenciamento de informações são baseados em registros (ou *record*, em inglês). Um registro é um conjunto simples de dados com o qual se identifica determinado elemento (uma pessoa,

um objeto, um evento etc.), associando a ele uma série de atributos que o caracterizam de maneira unívoca, por exemplo: nome, sobrenome e endereço, e número de telefone, no caso de uma pessoa; local, data e hora em que certo evento será realizado; código, peso, dimensões e cor, caso se trate de um objeto ou produto. Cada informação é inserida separadamente dentro de um espaço a ela reservada, chamado de campo (ou *field*, em inglês). Um exemplo de estrutura de dados pode ser vista na tabela do exemplo a seguir: cada célula da tabela é um campo, que contém apenas um tipo de informação; cada linha da tabela constitui um registro:

Augusto	J. Vésica	(62)9876-5432	Goiânia	GO
Nayana	Couto	(34)3216-5498	Patos de Minas	MG
Cláudio	Vésica	(62)3789-6545	Ap. de Goiânia	GO
Daniela	Savoi	(11)3558-9899	São Paulo	SP

Tabela 1.1: Registros.

A década de 1970 foi um marco na evolução dos bancos de dados digitais, pois viu o nascimento dos bancos de dados relacionais, definidos como conjuntos auto-explicativos de registros integrados. Antes disso, os bancos de dados utilizavam estruturas hierárquicas, o que tornava sua estrutura muito articulada e complexa de se consultar.

Para saber um pouco mais...

O modelo relacional para gerência de bancos de dados (SGBD) é um modelo de dados baseado em lógica de predicados e na teoria de conjuntos.
Historicamente, ele é o sucessor do modelo hierárquico e do modelo em rede. Estas arquiteturas antigas são até hoje utilizadas em alguns Data Centers com alto volume de dados, em que a migração é inviabilizada pelo custo que ela demandaria; existem ainda os novos modelos baseados em orientação ao objeto, que, na maior parte das vezes, são encontrados como kits de construção de SGBD, ao invés de um SGBD propriamente dito.
O modelo relacional foi o primeiro modelo de banco de dados formal. Somente depois, seus antecessores, os bancos de dados hierárquicos e em rede, passaram a ser também descritos em linguagem formal.

O modelo relacional foi inventado pelo **Dr. Ted Codd** e posteriormente mantido e aprimorado por **Chris Date** e **Hugh Darwen** como um modelo geral de dados. No Terceiro Manifesto (1995), eles mostraram como o modelo relacional pode ser estendido com características de orientação ao objeto sem comprometer os seus princípios fundamentais.

A linguagem padrão para os bancos de dados relacionais, **SQL**, do inglês *Structured Query Language*, é apenas vagamente remanescente do modelo matemático. Atualmente ela é adotada, apesar de suas restrições, porque é antiga e muito mais popular que qualquer outra linguagem de banco de dados.

A principal proposição do modelo relacional é que todos os dados sejam representados como relações matemáticas, isto é, um subconjunto do produto Cartesiano de n conjuntos. No modelo matemático (diferentemente do SQL), a análise dos dados é feita em uma lógica de predicados de dois valores (ou seja, sem o valor nulo), isto significa que existem dois possíveis valores para uma proposição verdadeira ou falsa. Os dados são tratados pelo cálculo relacional ou álgebra relacional.

O modelo relacional permite ao projetista criar um modelo lógico consistente da informação a ser armazenada. Este modelo lógico pode ser refinado através de um processo de normalização. Um banco de dados construído puramente baseado no modelo relacional estará inteiramente normalizado. O plano de acesso, outras implementações e detalhes de operação são tratados pelo sistema DBMS, e não devem ser refletidos no modelo lógico. Isto se contrapõe à prática comum para DBMSs SQL nos quais o ajuste de desempenho freqüentemente requer mudanças no modelo lógico.

Os blocos básicos do modelo relacional são: o domínio, ou tipo de dado; uma tupla, que é um conjunto de atributos que são ordenados em pares de domínio e valor; uma relvar (variável relacional), que é um conjunto de pares ordenados de domínio e nome que serve como um cabeçalho para uma relação; e uma relação, que é um conjunto desordenado de tuplas. Apesar destes conceitos matemáticos, eles correspondem basicamente aos conceitos tradicionais dos bancos de dados. Uma relação é similar ao conceito de tabela e uma tupla é similar ao conceito de linha.

O princípio básico do modelo relacional é o princípio da informação: toda informação é representada por valores em relações.

Assim, as relvars não são relacionadas umas às outras no momento do projeto. Entretanto, os projetistas utilizam o mesmo domínio em vários relvars, e se um atributo é dependente de outro, esta dependência é garantida através da integridade referencial.

Como todos os bancos de dados, o relacional também tem sua estrutura baseada em registros relacionados e organizados em tabelas. Essas relações tornam os registros integrados.

Um subconjunto específico de dados constitui um dicionário de dados, que determina a maneira em que os registros de dados são relacionados, quais vínculos existem para a sua utilização e outras características. Esses dados descritivos são chamados "metadados" e é graças a eles que um banco de dados é autodescritivo, ou seja, contém uma descrição de sua própria estrutura.

A presença de todos esses elementos (dados e metadados) transforma o banco de dados em uma estrutura informática acessível por qualquer aplicação que seja capaz de explorar os metadados, extrair informações sobre os dados e tratá-los para copiá-los, alterá-los e criar novos.

A característica autodescrevente de um banco de dados faz com que a aplicação que o acessa não precise gerenciar a estrutura dos registros, e sim, limite-se a utilizá-los, pois o próprio banco de dados se encarregará de criar espaço para novos registros, alterando seu conteúdo de acordo com as solicitações da aplicação que está acessando-o.

Assim, os bancos de dados com essas características não são simplesmente conjuntos de dados, mas contêm mecanismos automatizados que se encarregam da gestão dos registros, por esse motivo são chamados **Sistemas Gerenciadores de Banco de Dados Relacionais** (SGBDR), ou **Relational Database Management Systems** (RDMS).

Em um banco de dados relacional, as informações são estruturadas em tabelas, divididas em linhas e colunas: as linhas são os registros e as colunas constituem os campos.

Há diversas razões para que o modelo de banco de dados relacional substituisse o modelo hierárquico: primeiramente porque o SGBDR permite adicionar com facilidade novas tabelas em um banco de dados, relacionando seus dados e sem afetar substancialmente a estrutura do banco de dados e sem a necessidade de alterar os aplicativos que trabalharão com as tabelas; em segundo lugar, num

banco de dados relacional é muito simples alterar a estrutura de uma ou mais tabelas, inserindo ou excluindo linhas e colunas de acordo com as necessidades sem comprometer a funcionalidade do banco de dados; outra razão relevante do sucesso dos SGBDR é que todos eles podem ser acessados utilizando uma única ferramenta, chamada Structured Query Language (SQL), disponível praticamente para todas as plataformas de hardware, desde os mainframes até computadores portáteis.

Neste livro, veremos como acessar e manipular um banco de dados relacional utilizando os comandos oferecidos pelo SQL.

A linguagem SQL deriva diretamente do SEQUEL (*Structured English Query Language*).

O SEQUEL tinha como objetivo se tornar uma linguagem universal (cujos comandos originaram do idioma inglês), usada para acessar os dados armazenados em um banco de dados, ignorando as complicações derivadas da estrutura interna de funcionamento de cada um.

A primeira versão a ser usada foi distribuída em 1981 pela IBM e foi denominada Structured Query Language / Data System, ou simplesmente SQL/DS.

Esta nova linguagem despertou o interesse dos maiores construtores de computadores de grande porte (mainframes) e das empresas que desenvolviam softwares para aqueles equipamentos, pois, graças ao SQL/DS, seria possível simplificar os processos de manipulação de grandes quantidades de dados, que naquela época ainda empregavam estruturas hierárquicas.

Nos anos seguintes, o SQL deixou de ser um produto exclusivo da IBM e se tornou, de fato, uma linguagem de acesso a bancos de dados muito articulada e funcional, que pode ser empregada em computadores de arquiteturas totalmente diferentes.

Conceitos básicos do SQL

Os computadores trabalham executando programas escritos em uma linguagem de programação específica. As linguagens de programação são conjuntos de comandos e palavras-chave que devem ser utilizados respeitando regras de composição predefinidas, que constituem a sintaxe.

No decorrer dos anos, desde o surgimento da informática, foram criadas inúmeras linguagens de programação, cada qual com seus comandos e sintaxe, para tornar mais viável a interação do usuário

com o computador. Entre as linguagens mais conhecidas, podemos citar: FORTRAN, CLIPPER, COBOL, PASCAL, BASIC, VB, DELPHI, C (e seus derivados), JAVA etc. Em todos os casos, estamos falando de linguagens que permitem ao computador realizar qualquer tipo de operação (cálculos, armazenar e manipular dados, desenhar, digitar textos etc.). Para isso, nos oferecem comandos por meio dos quais o programador deve instruir o computador sobre como cada tarefa deve ser realizada.

No caso do SQL, estamos falando de uma linguagem um pouco diferente, antes de tudo porque pode ser usada exclusivamente para acessar dados em um banco de dados, isto é, trata-se de uma linguagem específica para a manipulação de tabelas de dados. Além disso, o SQL não serve para criar rotinas de procedimentos a serem executados pelo computador, e sim, para informar quais dados (ou conjuntos de dados) queremos manipular. De fato, a finalidade do SQL é acessar dados, independentemente do tipo de hardware ou software que estamos usando.

Tabelas, consultas e views

A linguagem SQL é baseada no conceito de views (visões, em português). Uma view é uma tabela virtual cujos registros são constituídos por campos que, no banco de dados físico, podem pertencer a registros localizados em tabelas diferentes.

Utilizando o SQL, podemos criar views realizando queries, ou seja, pesquisas, nas tabelas físicas existentes no banco de dados. O termo query significa: pergunta, interrogação e, na linguagem SQL, indica uma requisição precisa de dados a serem extraídos a partir de um banco de dados. Em outras palavras, uma query é uma lista de elementos (campos) que contém os dados que desejamos obter.

A partir da extração de dados de uma ou mais tabelas, é gerada uma nova tabela virtual (view), que contém os campos filtrados de acordo com os critérios especificados. Definimos uma view como uma tabela virtual porque, quando é gerada, não é armazenada em disco, mas apenas visualizada. Essa característica permite imprimila em papel ou torná-la uma tabela física pertencente ao banco de dados, gravando-a junto com as demais tabelas que constituem a estrutura do banco de dados. O SQL permite também operar não apenas com dados existentes: é possível criar novas tabelas criando novos campos ou, ainda, formá-las por campos obtidos a partir de

cálculos ou operações feitas em campos de outras tabelas. Essas tabelas são chamadas de "tabelas de base" e são físicas, ou seja, ao serem geradas são gravadas fisicamente no banco de dados.

A linguagem SQL é um conjunto de comandos (ou instruções) que permitem gerar enunciados, ou seja, linhas de comandos compostas por uma ou mais instruções. Alguns comandos permitem ou até mesmo exigem o uso de parâmetros adicionais, chamados de cláusulas e predicados. Basicamente, todos os comandos do SQL são verbos em inglês (`select`, `create`, `alter` etc.), e as cláusulas são preposições do mesmo idioma (`from`, `as`, `by`, `in` etc.).

Grupos de comandos SQL

Os comandos do SQL são classificados em três grupos, de acordo com suas principais funções:

- **DDL (Definition Data Language):** são todos aqueles comandos usados para criar e alterar tabelas que compõem o banco de dados, ou seja, os comandos que definem a estrutura dos dados;
- **DML (Data Manipulation Language):** pertencem a este grupo todos os comandos usados para extrair informações das tabelas, ou seja, para manipular os dados existentes;
- **DCL (Data Control Language):** trata-se de um conjunto de comandos usado em sistemas multiusuário para definir os privilégios de acesso aos dados a cada usuário. Os comandos de controle de acesso aos dados são usados para implementar segurança e privacidade em bancos de dados.

Os dialetos do SQL

Como já dissemos, o SQL é uma linguagem que possui seus comandos, seus parâmetros e sua sintaxe específicos. A integridade dessas características é constantemente monitorada e mantida por um comitê, o *American National Standards Institute* (algo como Instituto Americano de Padrões Nacionais), mais conhecido como ANSI. O primeiro padrão para o SQL foi publicado em 1986, e em 1989 teve sua primeira atualização. Em 1992, houve outra atualização, que se tornou conhecida como SQL92, ou simplesmente SQL2. A última atualização é de 1999, mas foi publicada oficialmente em 2000, com o nome SQL99 (ou SQL3).

Essa última versão foi assumida como padrão também por outra instituição, a ISO (*International Standards Organization* – Organização de Padrões Internacionais).

A definição de padrões para os comandos, os parâmetros e a sintaxe do SQL é, sem dúvida alguma, um fato positivo, pois evita divergências entre as atualizações lançadas que devem respeitar as diretrizes estabelecidas pelo ANSI e pela ISO. Contudo, os produtores de softwares lançam com freqüência versões próprias desta linguagem, adicionando ao SQL novos comandos e funcionalidades. Essas versões são chamadas "dialetos" e, além dos comandos básicos do SQL (cerca de 50 instruções), trazem outros que, em alguns casos, fogem do propósito inicial do SQL, que nasceu como linguagem para a manipulação de dados em bancos de dados. Assim, é possível encontrar dialetos SQL que trazem comandos para a criação de estruturas condicionais (`IF...THEN...ELSE...`), típicos das linguagens de programação para desenvolvimento de softwares, ou instruções para estruturas de controle (`WHILE...`) etc.

Para os puristas do SQL, tratam-se de alterações que descaracterizam a linguagem, pois tentam transformá-la em algo para o qual não foi criada. De qualquer forma, os dialetos apresentam novas funcionalidades que visam sempre aprimorar a manipulação de bancos de dados, desde alterações simples até mais complexas.

Sobre os Sistemas de Gerenciamento de Bancos de Dados Relacionais (SGBDR)

Para utilizar as características e o funcionamento do SQL é preciso se servir de um Sistema de Gerenciamento de Bancos de Dados Relacionais (SGBDR), isto é, de um ambiente no qual possamos utilizar os comandos desta linguagem para manipular dados.

A discussão sobre qual é o melhor SGBDR é ampla e praticamente sem fim, pois há legiões de usuários prontos a jurar que o sistema que usam é melhor que qualquer outro. De fato, no mercado existem diversos SGBDRs nos quais podemos operar usando o SQL.

Existem SGBDRs criados para trabalhar em servidores de redes locais (LANs), tais como o Oracle, o Microsoft SQL Server, o Sybase. Tratam-se de sistemas muito difusos, porém, não ao alcance de todos, pois seu custo inicial de implantação é de aproximadamente 10 mil dólares e pode chegar facilmente a valores bem mais altos, de acordo com as exigências.

Para os exemplos e as demonstrações deste livro, usaremos um SGBDR, que foi desenvolvido tendo como referência os sistemas mais utilizados no mercado (Microsoft SQL Server e Oracle), capaz de operar com os comandos SQL e, o mais importante, é gratuito: trata-se do MySQL, um SGBDR que pode ser obtido facilmente em sites de download pela Internet.

Por exemplo, no site http://www.mysqlbrasil.com.br/ é possível não só baixar a última versão do MySQL (tanto para Windows como para sistemas Linux), mas também obter manuais, participar de fóruns, eventos e treinamentos promovidos pela comunidade MySQL do nosso país.

Como veremos ao longo dos capítulos deste livro, para trabalhar com o SQL bastam poucas linhas de comandos (ou enunciados), que serão digitados em uma janela a qual servirá de interface entre nós e o banco de dados no qual queremos trabalhar.

Instalando o MySQL 5 no Windows XP

Apesar de estar disponível também para outras plataformas, neste tópico veremos como instalar o MySQL em um microcomputador com sistema operacional Windows XP – Service Pack 2, devido à maior difusão desse sistema operacional em ambientes tanto corporativos quanto domésticos.

No momento da redação deste livro, foi utilizada a versão mais atual (5.0.27), que pode ser obtida gratuitamente a partir do endereço:

http://dev.mysql.com/get/Downloads/MySQL-5.0/mysql-5.0.27-win32.zip/from/pick#mirrors

Neste site, basta escolher um mirror, ou seja, um dos links por meio dos quais é possível baixar o arquivo **mysql-5.0.27-win32.zip** (com aproximadamente 40 MB), que contém o instalador do MySQL 5 para Windows XP.

Uma vez feito o download, é preciso descompactar os arquivos baixados em uma pasta qualquer do disco rígido do computador; para isso, basta usar um software de compactação/descompactação como WinZip, WinRar, WinAce ou, ainda, recorrer à opção de descompactação do próprio Windows XP.

Após descompactar o arquivo baixado, basta dar clique duplo sobre o arquivo Setup.exe para iniciar a instalação do MySQL. A partir

daí, é preciso seguir as instruções fornecidas pelo Windows Installer para continuar com a instalação. Veja as etapas em detalhes:

1. A primeira janela apresentada nos dá as boas-vindas; para continuar, clique no botão **Next** (próximo), como mostra a **Figura 1.1**.

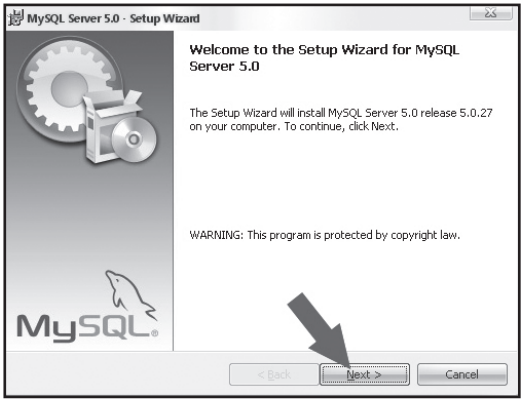


Figura 1.1: Na tela de boas-vindas, clique em **Next** para proceder com a instalação.

2. Na segunda etapa, podemos escolher entre instalação típica (**Typical**), completa (**Complete**) ou personalizada (**Custom**). A primeira opção instala as funções mais comuns, indicadas para uso geral; a segunda, todas as funcionalidades do MySQL, o que exigirá mais espaço no HD do computador, mas proporcionará maior flexibilidade no trabalho; a terceira opção permite decidir quais características desejamos instalar e quais descartar. Para o nosso exemplo, mantenha a primeira opção e clique em **Next** novamente (**Figura 1.2**).

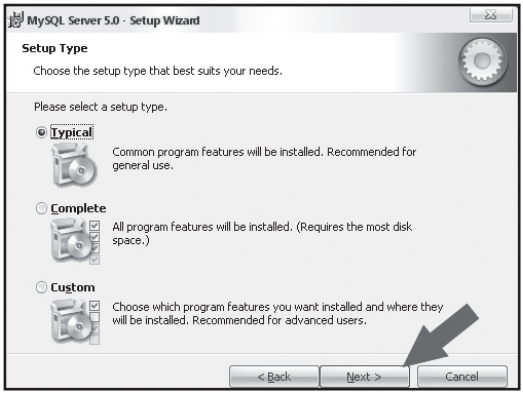


Figura 1.2: Na segunda etapa, selecione o tipo de instalação desejada e clique em **Next** para avançar.

3. Ao optar pela instalação típica ou completa, a terceira etapa mostrará apenas um resumo das opções de instalação escolhidas; para continuar, basta clicar no botão **Install** para iniciar a instalação do MySQL, como mostra a **Figura 1.3**. Se, na etapa anterior, tivesse sido marcardo a opção **Custom**, seria apresentada uma janela adicional, na qual deveremos escolher as características a serem instaladas (**Figura 1.4**). Vale a pena ressaltar que a instalação personalizada deve ser realizada apenas por usuários que já conhecem o MySQL e compreendem o funcionamento de cada elemento do programa, caso contrário, poderá apresentar falhas no funcionamento.

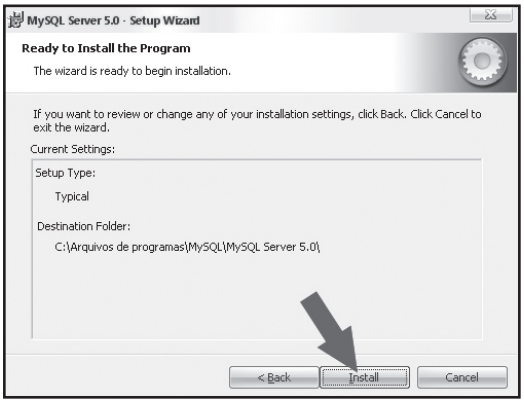


Figura 1.3: Na terceira etapa, são exibidas as informações sobre o tipo de instalação escolhida e a pasta do sistema em que o software será instalado. Clicando em **Install**, será iniciada a cópia dos arquivos.

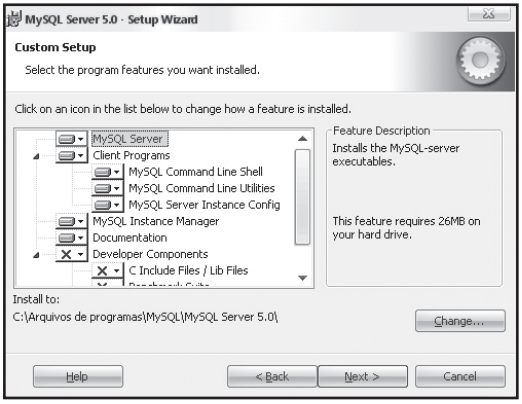


Figura 1.4: Caso selecionássemos a instalação personalizada – **Custom** – usaríamos esta janela para especificar quais características do MySQL deveriam ser instaladas no computador. Feito isso, bastaria clicar em **Install** para finalizar a instalação.

4. Durante algum tempo, que depende da configuração do computador em uso, será exibida uma barra de progresso, que nos mantém informados sobre o andamento da instalação (**Figura 1.5**). Ao terminar a cópia dos arquivos necessários para o funcionamento do MySQL, é exibida uma janela na qual podemos nos registrar no site MySQL.com para receber informações sobre as atualizações do aplicativo (**Figura 1.6**). Para o nosso exemplo, pule essa parte marcando a opção **Skip Sign-Up** (pular registro) e clique em **Next**.

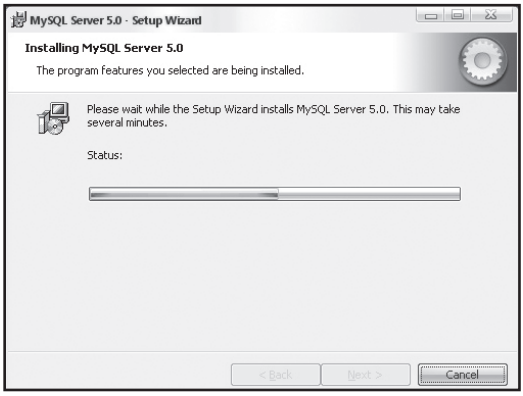


Figura 1.5: Esta barra de progresso nos informa sobre o andamento da instalação.

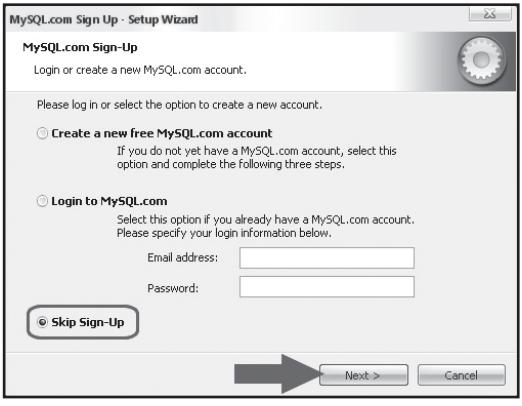


Figura 1.6: Se desejar, poderá efetuar o cadastro no site MySQL.com para receber informações atualizadas a respeito do MySQL. Marque a opção **Skip Sign-Up** para não efetuar o registro.

5. Enfim, uma tela final nos informa que a instalação foi concuída com êxito (**Figura 1.7**). Agora é preciso configurar o MySQL para podermos trabalhar, portanto, mantenha marcada a opção **Configure the MySQL Server now** e clique no botão **Finish** (Concluir).

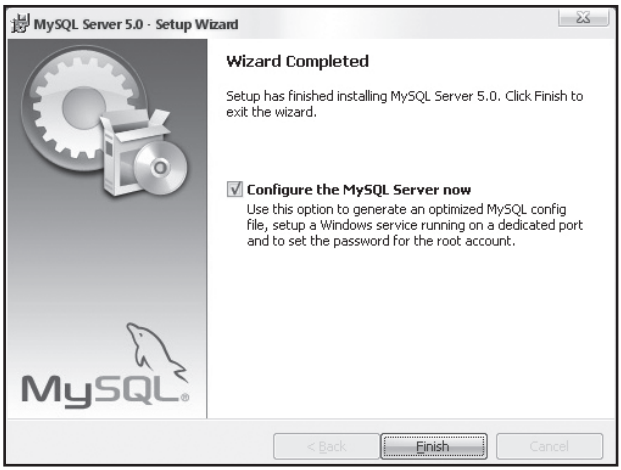


Figura 1.7: A instalação está concluída. Agora é preciso configurar o MySQL Server.

6. É iniciado, então, um novo assistente, que se encarrega de nos guiar no processo de configuração do MySQL Server. Clique em **Next** (**Figura 1.8**).

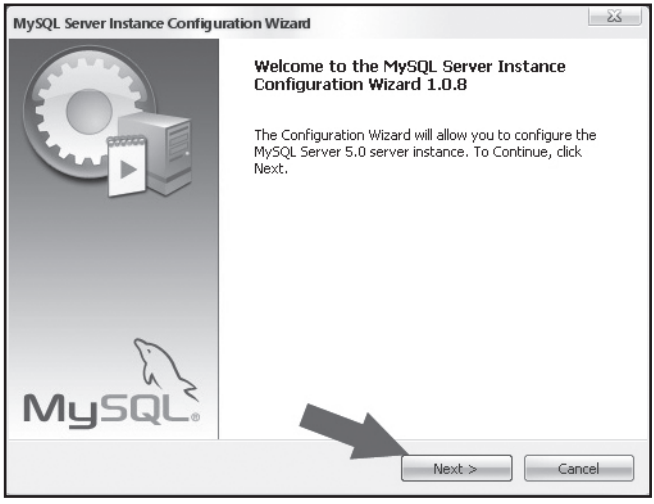


Figura 1.8: Um novo assistente nos ajudará na configuração do servidor do MySQL.

7. Na etapa sucessiva, podemos optar pela configuração detalhada ou padrão. A primeira permite definir a configuração mais adequada ao sistema em uso, a segunda adota uma configuração que se adequa à maioria dos sistemas. Marque a segunda opção e clique em **Next** (**Figura 1.9**).

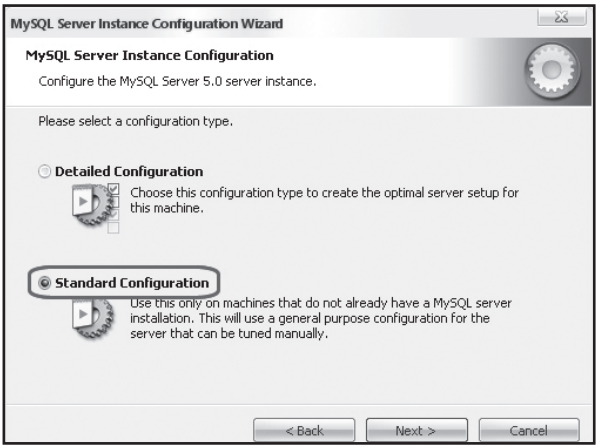


Figura 1.9: Marque a opção **Standard Configuration** para iniciar a configuração padrão do MySQL Server; em seguida clique em **Next**.

8. Na terceira etapa de configuração, são apresentadas três opções: a primeira (**Install As Windows Service**) registra o MySQL como um serviço do sistema operacional, o que o torna mais seguro e menos vulnerável; a segunda (**Launch the MySQL Server automatically**), faz com que o MySQL Server seja executado automaticamente ao inicializar o Windows; a terceira (**Include Bin Directory in Windows PATH**), inclui a pasta de trabalho do MySQL nas variáveis de PATH do Windows, para que os comandos SQL possam ser executados a partir do prompt de comando do Windows no modo MS-DOS (**Figura 1.10**). Mantenha as opções padrão e clique novamente em **Next** para proceder.

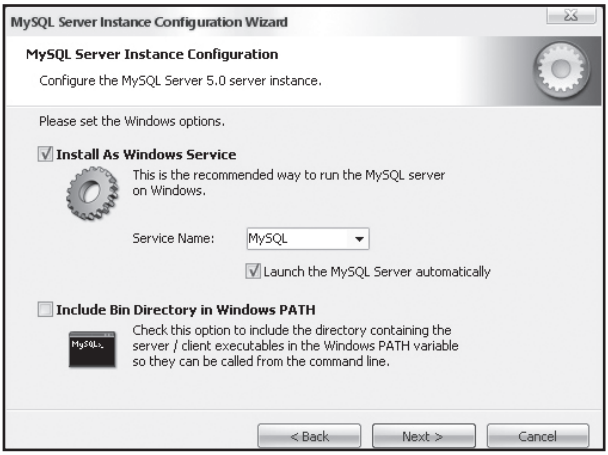


Figura 1.10: Nesta etapa, mantenha as opções padrão e clique em **Next**.

9. Na etapa seguinte (**Figura 1.11**), é necessário definir as opções de segurança, criando contas de usuários para ter acesso ao servidor do MySQL. A primeira opção, **Modify Security Settings**, serve para definir a senha de acesso do usuário Root, o administrador do servidor: digite então uma senha de acesso no campo **Enter the root password** (digite a senha de root) e confirme-a, digitando-a novamente no campo **Retype the password** (redigite a senha). Se marcarmos a opção **Enable root access from remote machines**, habilitaremos o acesso ao servidor a partir de outros computadores ligados em rede com aquele em que o MySQL server está instalado. É possível também criar uma conta de usuário anônimo (**Create An Anonymous Account**), que não necessitará de senha para acessar o servidor. Trata-se de uma opção perigosa, pois qualquer usuário do computador poderá acessar o servidor e mudar suas configurações. Mantenha as opções padrão e clique em **Next**.

Figura 1.11: Após definir a senha de acesso ao servidor, clique em **Next** para proceder com a configuração.

10. Definidas as configurações, o assistente pede para clicar no botão **Execute** para aplicar as opções especificadas (**Figura 1.12**). Então, clique e aguarde a conclusão do processo.

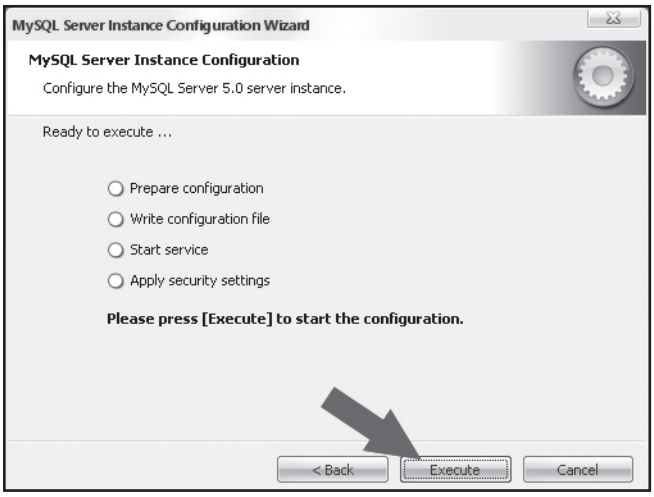


Figura 1.12: Clicando no botão **Execute**, aplicamos as opções de configuração definidas nas etapas anteriores.

11. Após alguns instantes, somos notificados de que o processo foi concluído. Clique em **Finish** para sair do assistente e terminar.

Pronto! Agora o MySQL está instalado e pronto para ser usado!

Como funciona o MySQL

Após a instalação, o MySQL 5.0 encontra-se armazenado na pasta **Arquivos de Programas**, dentro de uma subpasta específica, chamada **MySQL Server 5.0**.

> **Observação:** obviamente, trata-se do caminho padrão usado pelo instalador (que vimos em detalhes no tópico anterior). Vale lembrar que a pasta de instalação pode ter sido alterada durante o processo de instalação; neste caso, os arquivos do MySQL 5.0 estarão na pasta especificada pelo usuário nas etapas de instalação do software.

Dentro da pasta em que o MySQL foi instalado, encontra-se uma subpasta chamada **Docs**, na qual está gravado o arquivo **manual.chm**, que é o manual para o uso do MySQL; para abri-lo e ler seu conteúdo, dê clique duplo sobre o arquivo. Nele, podemos encontrar as informações que desejamos, procurando em uma lista de tópicos ou realizando pesquisas, como ocorre geralmente nos sistemas de ajuda de outros softwares. Infelizmente, toda a documentação é em inglês, por isso, o conhecimento desse idioma é indispensável.

Na pasta **MySQL Server 5.0** há também uma subpasta chamada **Data**, na qual são armazenados os bancos de dados criados com o MySQL. Cada banco de dado é salvo dentro de uma pasta específica que traz como rótulo o nome dado ao banco de dados na hora de gerá-lo.

Para iniciar o MySQL, basta localizar seu grupo de ícones no menu **Iniciar** do Windows, como mostram os procedimentos a seguir:

1. Clique em **Iniciar** > **Programas** (ou **Todos os Programas**, dependendo da versão e da configuração do seu sistema operacional), localize no grupo de programas o **MySQL 5.0** e, nele, clique no ícone rotulado como **MySQL Command Line Client**.

2. Será aberta uma janela do prompt do MS-DOS na qual é solicitada a senha para o acesso ao sistema. Digite a senha que foi definida durante a instalação e pressione **Enter**.

3. Imediatamente é exibido o texto de boas-vindas (em inglês) e é disponibilizando o ambiente para a digitação de comandos, chamado comumente de "Prompt do MySQL", representado desta forma:

```
mysql>
```

É aqui que digitaremos todas as linhas de comando SQL para o gerenciamento dos nossos bancos de dados.

Antes de iniciar o uso do MySQL, é preciso verificar se o serviço está ativo no sistema. Nas versões mais recentes do Windows, o serviço MySQL é ativado automaticamente na instalação do software. Todavia, é possível que seja necessário ativá-lo manualmente. Para verificar se o MySQL foi iniciado no sistema, faça o seguinte:

1. Caso esteja aberta, feche a janela do prompt do MySQL.

2. Abra o menu **Iniciar**, localize e acesse o **Painel de Controle** do Windows.

3. Clique no ícone **Ferramentas administrativas.**

Observação: no Windows XP, o painel de controle pode ser apresentado de duas formas: Modo Clássico ou Exibição por categorias. No primeiro caso, o ícone **Ferramentas Administrativas** se encontra junto com as demais opções; em Exibição por categorias, esta opção está dentro do grupo **Desempenho e manutenção**.

4. Abra a opção **Serviços**, também com clique duplo.

5. Na parte à direita da janela, são listados todos os serviços instalados no sistema. Use a barra de rolagem para navegar nas opções até encontrar o item **MySQL**. Os serviços são listados em ordem alfabética crescente, portanto, não será difícil encontrar o item desejado.

6. Clique sobre o item e verifique se à sua direita, na coluna **Status**, aparece a indicação **Iniciado**, que significa que o serviço está ativo.

7. Caso não esteja, no centro da mesma janela estão os comandos para iniciar o MySQL, sob forma de um link (texto azul sublinhado). Clique em **Iniciar o serviço** e feche todas as janelas em seguida.

Feito isso, o MySQL está ativo e pronto para ser usado.

Características dos bancos de dados relacionais

Como vimos, o SQL é utilizado para criar e manipular bancos de dados relacionais. Isto quer dizer que os comandos do grupo DDL (*Data Definition Language*) do SQL permitem gerar facilmente tabelas que formam um banco de dados. A estrutura e o conteúdo dessas tabelas devem ser definidos antes, utilizando um software específico para esse fim.

Para que um banco de dados possa ser considerado relacional, as tabelas que o compõem devem ser construídas respeitando certos vínculos, chamados *Normal Forms* (formas normais, em português),

por isso, as tabelas que respeitam esse esquema são chamadas tabelas normalizadas. A teoria que está por trás do gerenciamento de tabelas normais é bastante complexa, todavia, podemos resumi-la em alguns conceitos essenciais, que veremos ao longo deste capítulo.

Chaves primárias e univocidade dos dados

Cada registro de uma tabela deve ser unívoco, isto é, cada linha da tabela deve ser diferente, em pelo menos uma coluna, das demais. Registros idênticos dentro de uma tabela causariam ambigüidade e redundâncias inúteis e, portanto, devem ser evitados.

Por esse motivo, é de fundamental importância, após a entrada de dados, verificar a redundância de dados e eliminar os registros duplicados.

Os registros de uma tabela não são numerados e a ordem na qual são listados na tabela é aleatória, respeitando geralmente a ordem cronológica da inserção dos dados, assim, para diferenciar um registro do outro, é preciso analisar com cuidado o conteúdo de cada campo, e, visto que cada registro pode conter um número considerável de campos, é imprescindível identificar os campos da tabela que conterão dados unívocos, ou seja, que jamais serão repetidos dentro da mesma tabela. Os campos (ou colunas) da tabela que possuem essa característica de univocidade são chamados de "chaves candidatas" e, entre eles, é preciso eleger um que será a chave primária, que identificará sem ambigüidade cada registro da tabela.

Em um cadastro de pessoas, por exemplo, deveremos verificar quais dados jamais poderiam ser repetidos: podem existir várias pessoas com o mesmo nome e sobrenome, filiação, naturalidade etc.; mas, com certeza, jamais existirão duas pessoas com o mesmo CPF ou com o mesmo número de RG. Neste caso, os campos CPF e RG seriam chaves candidatas e escolheremos uma delas para servir de chave primária, um dado que usaremos para identificar cada pessoa de maneira exclusiva.

Uma tabela pode ter somente uma chave primária, cuja presença garante a univocidade dos registros. Uma chave primária pode ser:

- **Simples:** formada por um só campo da tabela (por exemplo, o CPF, no caso de um banco de dados de pessoas);
- **Composta:** formada por mais de um campo da tabela (por exemplo, CPF e RG). Geralmente, as chaves primárias compos-

tas são usadas quando um só campo não é suficiente para garantir a univocidade dos registros.

Quando uma tabela não tem, em sua estrutura, nenhum campo que possa ser usado como chave primária, é aconselhável adicionar uma coluna para esse fim. Por exemplo, em uma tabela de produtos podemos ter vários produtos com o mesmo nome, o mesmo preço etc. Entretanto, podemos criar um campo Código, que contenha um valor (numérico ou alfanumérico) diferente para cada registro. Assim, cada código identificará um determinado produto de maneira unívoca e poderemos usar este campo como chave primária da nossa tabela.

Vale a pena ressaltar que não é obrigatório definir uma chave primária em uma tabela, contudo, é aconselhável fazê-lo para não comprometer o correto funcionamento do banco de dados.

Chaves externas e relacionamentos entre tabelas

Um banco de dados é criado para poder gerenciar informações contidas em tabelas diferentes, sendo que cada uma delas agrupa dados relativos a um mesmo contexto. Pensemos em um sistema de gerenciamento de uma empresa da área do comércio de produtos e serviços: provavelmente será necessário manter um cadastro de clientes, um de fornecedores, um de produtos comercializados e outro de pedidos.

Cada um desses cadastros será estruturado em uma tabela separada, porém, essas tabelas estão diretamente relacionadas, pois, por exemplo, cada pedido envolverá um determinado cliente e diversos produtos, dados esses que constam em outras tabelas do mesmo banco de dados.

Quando determinados dados de uma tabela estão relacionados a dados do mesmo tipo de outra tabela, é possível estabelecer um relacionamento entre eles. Para entender melhor o conceito de relacionamento entre dados de tabelas diferentes, veja o exemplo a seguir:

Suponhamos que uma empresa utiliza um banco de dados para manter o controle de seus clientes, fornecedores, produtos e pedidos; esse banco de dados seria estruturado em quatro tabelas distintas, mostradas a seguir:

Nome_ Completo	CPF	Endereço	CEP	Cidade	Estado
Augusto Vésica	789888245-65	Rua 1 nº 999	12345-678	Goiânia	GO
Daniela Savoi	659666232-45	Av. Z nº 888	12345-789	São Paulo	SP
Marianna Favesi	788332698-87	Pça. C12 nº 777	12345-789	Patos de Minas	MG

Tabela 1.2: Clientes.

Razão_Social	CNPJ	Endereço	CEP	Cidade	Estado	Telefone
New-Line Imp. Exp. Ltda.	78945612345654	Av. C33 nº 555	12345-678	Rio de Janeiro	RJ	3789-9874
TopTech Ltda.	65498732145654	Rua 98 nº 211	12345-789	São Paulo	SP	3456-9878
Teratron Ltda.	12365478921123	Av. 65 nº 444	12345-789	São Paulo	SP	3112-9874

Tabela 1.3: Fornecedores.

Código	Descrição	Valor_Unitário
00121	DVD +R 4.7 GB	1,99
00232	PenDrive 2 GB	178,00
00454	CD-R 700 MB	0,99

Tabela 1.4: Produtos.

Num_Pedido	Nome_Cliente	Produto	Valor_Pedido
Linha 1 em branco	Linha 1 em branco	Linha 1 em branco	Linha 1 em branco
Linha 2 em branco	Linha 2 em branco	Linha 2 em branco	Linha 2 em branco
Linha 3 em branco	Linha 3 em branco	Linha 3 em branco	Linha 3 em branco

Tabela 1.5: Pedidos.

Como podemos observar, a tabela Pedidos contém informações presentes em duas outras tabelas: o campo Nome_Cliente, que corresponde ao campo Nome_Completo da tabela Clientes, e o campo Produto, que corresponde ao campo Descrição, da tabela Produtos. Em casos como esse, dizemos que se trata de campos relacionados.

Geralmente, o relacionamento é estabelecido no banco de dados usando a chave primária de uma tabela, relacionando-a com o campo correspondente de outra: o campo relacionado à chave primária é chamado "chave externa" e as duas tabelas são chamadas "tabela-mãe" (que contém a chave primária), e "tabela-filha" (que contém a chave externa).

Os relacionamentos podem ser de vários tipos:

- **Um-para-um:** quando para cada registro da tabela-mãe corresponde um único registro na tabela-filha;
- **Um-para-muitos:** quando para cada registro na tabela-mãe há vários registros na tabela-filha;
- **Muitos-para-muitos:** para cada valor do campo de uma tabela, pode haver n valores no campo da outra tabela e vice-versa. Em bancos de dados relacionais, isso normalmente é feito por meio de uma tabela de ligação.

Vejamos isso com maiores detalhes nos tópicos a seguir.

Relacionamentos um-para-um

Dissemos que duas tabelas possuem relacionamento um-para-um quando, para cada linha da tabela principal (ou tabela-mãe), existe apenas uma linha na tabela-filha, e vice-versa. Trata-se de um tipo de relacionamento pouco empregado.

Relacionamentos um-para-muitos

Duas tabelas estão relacionadas na modalidade um-para-muitos quando, para cada registro da tabela principal, podem ter zero, um ou mais registros relacionados na tabela secundária, mas, para cada registro da tabela secundária, há apenas um registro na tabela principal.

Ao contrário do anterior, esse tipo de relacionamento é muito usado nas estruturas dos bancos de dados, por isso, é de suma importância entender seu funcionamento. Veja um exemplo a seguir:

O nosso banco de dados possui uma tabela na qual inserimos os dados relativos aos clientes da nossa empresa, com a estrutura apresentada na **Tabela 1.2**: Clientes.

Suponhamos que seja necessário inserir nesta tabela os dados relativos aos números de telefone de cada cliente. Algumas pessoas possuem apenas um número, outras dois (residencial e celular), três (residencial, celular e comercial), ou até mesmo quatro ou mais números para contato. Para contemplar todas as possibilidades, seria necessário adicionar pelo menos quatro colunas à nossa tabela, como mostrado a seguir:

Telefone_residencial	Telefone_celular	Telefone_comercial	Telefone_outro
Linha 1 em branco	Linha 1 em branco	Linha 1 em branco	Linha 1 em branco
Linha 2 em branco	Linha 2 em branco	Linha 2 em branco	Linha 2 em branco

Tabela 1.6: Números de telefone.

Seria uma solução simples, mas não a melhor, pois diversos registros poderiam apresentar campos vazios, pois nem todas as pessoas possuem quatro números de telefone, ou, ainda, se algum de

nossos clientes tiver mais de quatro números para contato seríamos obrigados a desconsiderar parte deles, porque não há campos onde inseri-los.

A solução ideal para casos como esse é adicionar ao banco de dados uma nova tabela, chamada, por exemplo, Telefones, cuja estrutura poderia ser como a mostrada a seguir:

Código_Telefone	ID_Cliente	N_Telefone
Linha 1 em branco	Linha 1 em branco	Linha 1 em branco
Linha 2 em branco	Linha 2 em branco	Linha 2 em branco

Tabela 1.7: Telefones.

Nesta tabela, o campo Código_Telefone, é a chave primária e o campo ID_Cliente é a chave externa, ou seja, um campo relacionado com a chave primária da tabela Clientes (que poderia ser o campo CPF) que estabelece um relacionamento entre cada cliente e seus respectivos números de telefone.

Para cada pessoa listada na tabela Clientes, pode haver nenhum, um ou mais números na tabela Telefones; a correspondência entre os registros das duas tabelas é realizada por maio do campo ID_ Cliente da tabela Telefones, que, estando relacionado com o campo CPF (chave primária da tabela Clientes), identifica de maneira unívoca cada cliente da tabela e associa a ele os números de telefone digitados na tabela.

Não é obrigatório que para cada cliente da tabela Clientes haja um número correspondente na tabela Telefones, porém, para cada número de telefone na tabela Telefones pode existir apenas UM cliente relacionado. Em outras palavras, UM cliente pode possuir MUITOS números de telefone.

A **Figura 1.13** mostra, de maneira esquemática, um relacionamento um-para-muitos.

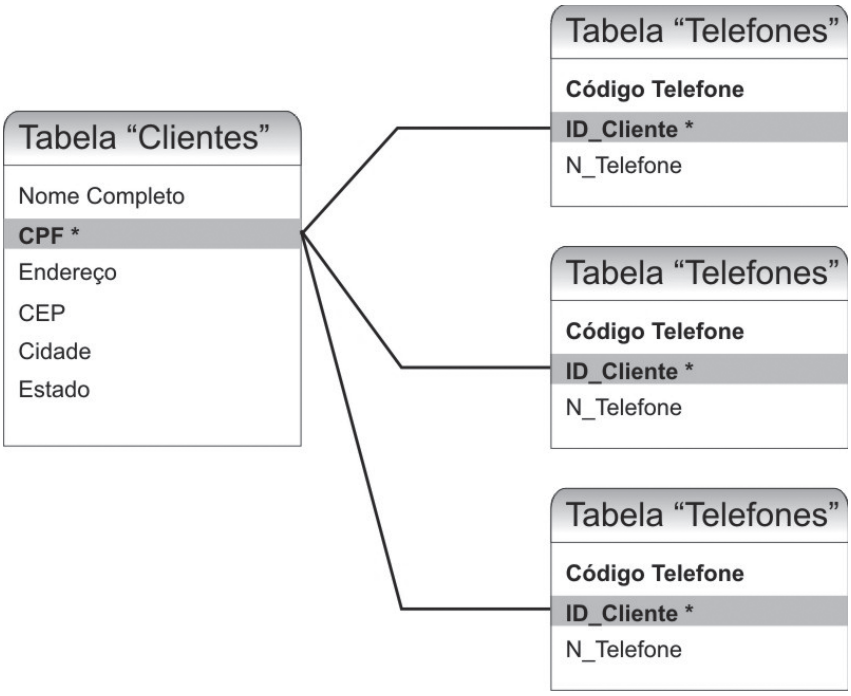


Figura 1.13: O esquema de um relacionamento um-para-muitos: à esquerda, a tabela Clientes, o lado um, e, à direita, a tabela Telefones, o lado muitos. Para cada cliente, identificado de maneira unívoca pelo campo CPF – a chave primária – podem existir vários registros na tabela Telefones. Os dados são relacionados graças ao campo ID_Cliente, que é o CPF do cliente do qual desejamos cadastrar o(s) número(s).

Relacionamentos muitos-para-muitos

Temos um relacionamento muitos-para-muitos quando, para cada registro da tabela-mãe, existem vários registros correspondentes na tabela-filha, e vice-versa. Esse tipo de relacionamento só pode ser criado com a ajuda de uma outra tabela, denominada "tabela de associação", que se torna uma intermediação entre as outras duas. Esta tabela deverá possuir duas chaves externas, uma para a tabela principal (mãe) e outra para a tabela secundária (filha). As duas tabelas deverão estar ligadas com a terceira por um relacionamento do tipo um-para-muitos.

Para ficar mais claro, utilizaremos como exemplo um banco de dados para o controle de livros de uma biblioteca; podemos organizar nossos dados em três tabelas: Livros, Autores e Editoras. A estrutura dessas tabelas seria mais ou menos assim:

ISBN	Título	Ano	Código_Editora
Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco

Tabela 1.8: Livros.

Código_Autor	Nome_Autor
Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco

Tabela 1.9: Autores.

Código_Editora	Nome	Cidade	Estado
Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco	Linha 1 em branco Linha 2 em branco

Tabela 1.10: Editoras.

A tabela Livros contém informações sobre os livros disponíveis e usa como chave primária o campo ISBN, um código que identifica todos os livros publicados no mundo de maneira unívoca.

Sabemos que, para cada editora, podemos ter muitos livros, então estabelecemos um relacionamento um-para-muitos entre as tabelas Editora (um) e Livros (muitos); o campo responsável pelo relacionamento é Código_Editora, presente em ambas as tabelas e que na tabela Editoras é a chave primária, enquanto na tabela Livros é a chave externa.

O relacionamento entre as tabelas Livros e Autores é um pouco mais complexo, pois, para cada autor, podemos ter mais de um livro e vice-versa, ou seja, um mesmo livro pode ter sido escrito por mais de um autor; trata-se de um relacionamento muitos-para-muitos.

Para estabelecer esse relacionamento, será necessário criar uma quarta tabela, que chamaremos Títulos_e_Autores, a qual será usada para realizar a associação entre as tabelas Livros e Autores. Sua estrutura pode ser assim:

ISBN	Código_Autor
Linha 1 em branco	Linha 1 em branco
Linha 2 em branco	Linha 2 em branco

Tabela 1.11: Títulos e autores.

Como podemos observar, esta tabela possui um campo ISBN, relacionado com a chave primária da tabela Livros, e um campo Código_Autor, relacionado com o mesmo campo que é a chave primária da tabela Autores.

Assim, a tabela Títulos_e_Autores funciona como uma ponte entre as tabelas Livros e Autores, realizando o relacionamento muitos-para-muitos. O esquema desse relacionamento pode ser visto na **Figura 1.14**.

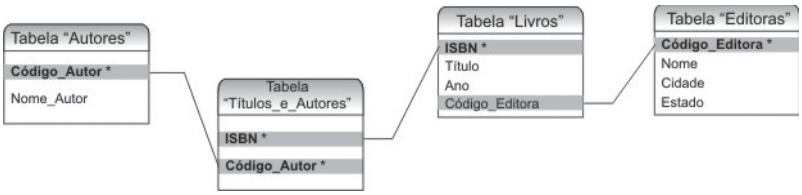


Figura 1.14: Representação esquemática de um relacionamento muitos-para-muitos. Veja como a tabela Títulos_e_Autores – que possui duas chaves primárias – funciona como uma ponte de ligação entre as tabelas Livros e Autores.

Normalização de formas normais

As Formas Normais são critérios rigorosos usados para auxiliar no planejamento das tabelas de um banco de dados relacional e são

baseadas nas teorias dos conjuntos e da lógica matemática. Ao longo do tempo, foram encontradas e definidas várias formas normais, que estabelecem vínculos para níveis progressivos, ou seja, uma tabela construída de acordo com a terceira Forma Normal, por exemplo, respeita também a primeira e a segunda.

A Primeira Forma Normal – (1FN)

Vejamos, no exemplo a seguir, como funciona a Primeira Forma Normal (ou *First Normal Form* – 1NF, como é chamada em inglês). Utilizaremos como exemplo a tabela mostrada a seguir:

Nome Completo	Endereço
Doodle Moraz	Av. C32 nº 666, Goiânia – GO
Fernando de Castro Junqueira	Rua B nº 456, Goiânia – GO
Marianna Favesi	Pça. C12 nº 777, Patos de Minas – MG
Augusto Vésica	Rua 1 nº 999, Goiânia – GO
Margarida Ferreira dos Santos	Rua 7 nº 234, Ribeirão Preto – SP
Daniela Savoi	Av. Z nº 888, São Paulo – SP

Tabela 1.12: Nome completo e endereço.

Os itens listados nas linhas da tabela são pessoas, e as colunas contêm apenas duas informações para cada uma delas: o nome completo e o endereço

Observação: os dados apresentados na tabela anterior e em outras tabelas são, obviamente, fictícios. Ao acompanhar os procedimentos desse livro, não será necessário copiá-los, use dados de sua preferência e, quando necessário, ajuste os critérios utilizados nas explicações para adequá-los às informações inseridas. Preocupe-se apenas em manter a estrutura das tabelas mostradas, isto é, a quantidade e o tipo de campo, as chaves primárias etc.

Se desejássemos classificar os dados desta tabela em ordem alfabética crescente pelo sobrenome, seria impossível, pois o sobrenome está junto com o nome dentro do campo Nome Completo e, além disso, encontra-se depois do nome.

Dada a estrutura da tabela, não conseguiríamos também extrair, por exemplo, as pessoas que moram na mesma cidade ou estado, pois esses dados se encontram misturados com a rua e o número, no campo Endereço.

Se organizarmos os dados, como mostrado na tabela a seguir, poderemos realizar as operações citadas anteriormente (e muitas outras) com extrema facilidade:

Nome Completo	Sobrenome	Endereço	Cidade	Estado
Doodle	Moraz	Av. C32 nº 666	Goiânia	GO
Fernando	de Castro Junqueira	Rua B nº 456	Goiânia	GO
Marianna	Favesi	Pça. C12 nº 777	Patos de Minas	MG
Augusto	Vésica	Rua 1 nº 999	Goiânia	GO
Margarida	Ferreira dos Santos	Rua 7 nº 234	Ribeirão Preto	SP
Daniela	Savoi	Av. Z nº 888	São Paulo	SP

Tabela 1.13: Detalhes dos clientes.

Veja como todas as informações, com base nas quais poderemos querer trabalhar (filtrar, excluir, classificar etc.), encontram-se separadas em campos diferentes.

Uma tabela com essa estrutura respeita a Primeira Forma Normal, que determina que cada coluna deva conter apenas um atributo, ou seja, uma informação, relacionada ao item relacionado naquela linha; em outras palavras, o conteúdo de uma coluna deve ser um único dado não divisível.

A Segunda Forma Normal – (2FN)

Vamos analisar agora um exemplo um pouco mais complexo. A tabela a seguir, que faz parte de um banco de dados de uma em-

presa, tem como propósito reunir as informações sobre a quantidade de horas trabalhadas por cada funcionário em cada projeto em desenvolvimento.

Matrícula	Nome do Funcionário	Projeto	Horas Trabalhadas
FV-911	Doodle Moraz	Projeto-AX; Projeto-BX; Projeto-CX	28; 30; 12
FV-465	Fernando de Castro Junqueira	Projeto-BX; Projeto-DX	41; 13
FV-732	Marianna Favesi	Projeto-DX	9
FV-245	Augusto Vésica	Projeto-AX; Projeto-CX	32; 18
FV-348	Margarida Ferreira dos Santos	Projeto-BX	8
FV-611	Daniela Savoi	Projeto-CX; Projeto-DX	26; 31

Tabela 1.14: Horas trabalhadas.

Observando a tabela, notamos que não respeita a Primeira Forma Normal, pois os campos Nome do Funcionário, Projeto e Horas Trabalhadas podem ser divididos em campos separados, como mostrado a seguir:

Matrícula	Nome do Funcionario	Sobrenome do Funcionario	Projeto 1	Hs Projeto 1	Projeto 2	Hs Projeto 2	Projeto 3	Hs Projeto 3	Projeto 4	Hs Projeto 4
FV-911	Doodle	Moraz	Projeto-AX	28	Projeto-BX	30	Projeto-CX	12		
FV-465	Fernando	de Castro Junqueira			Projeto-BX	41			Projeto-DX	13
FV-732	Marianna	Favesi							Projeto-DX	9
FV-245	Augusto	Vésica	Projeto-AX	32			Projeto-CX	18		
FV-348	Margarida	Ferreira dos Santos			Projeto-BX	8				
FV-611	Daniela	Savoi					Projeto-CX	26	Projeto-DX	31

Tabela 1.15: Detalhes de horas trabalhadas por projeto.

Esta nova estrutura respeita a Primeira Forma Normal, porém, não ficou fácil de se consultar, pois deixa muitas células vazias. Além disso, se surgisse um quinto projeto (e um sexto, um sétimo etc.) seria preciso modificar a estrutura da tabela adicionando duas novas colunas para cada novo projeto (Projeto 5 e Hs Projeto 5, e assim por diante).

Para organizar melhor esses dados, é recomendável criar duas tabelas distintas, ambas de acordo com a Primeira Forma Normal, e relacionadas entre si. Portanto, a solução seria criar uma tabela que contenha apenas os dados relativos aos funcionários (que chamaremos de tabela Funcionários), assim:

Matrícula	Nome do Funcionário	Sobrenome do Funcionário
FV-911	Doodle	Moraz
FV-465	Fernando	de Castro Junqueira
FV-732	Marianna	Favesi
FV-245	Augusto	Vésica
FV-348	Margarida	Ferreira dos Santos
FV-611	Daniela	Savoi

Tabela 1.16: Funcionários.

Nesta tabela, definiremos uma chave primária que identifique de maneira unívoca cada funcionário, neste caso, o campo Matrícula.

Então, criaremos uma segunda tabela, chamada Andamento de Projetos, na qual inseriremos os projetos em andamento e a quantidade de horas, relacionando cada projeto ao respectivo funcionário por meio da chave externa, que é o número de matrícula. A estrutura dessa tabela poderia ser a seguinte:

Matrícula	Projeto	Horas Trabalhadas
FV-911	Projeto-AX	28
FV-245	Projeto-AX	32
FV-911	Projeto-BX	30
FV-465	Projeto-BX	41
FV-348	Projeto-BX	8
FV-911	Projeto-CX	12
FV-245	Projeto-CX	18
FV-611	Projeto-CX	26

Matrícula	Projeto	Horas Trabalhadas
FV-465	Projeto-DX	13
FV-732	Projeto-DX	9
FV-611	Projeto-DX	31

Tabela 1.17: Andamento de Projetos.

Nesta tabela, definiremos uma chave primária composta, ou seja, formada por mais de um campo que, neste caso, serão Matrícula e Projeto.

Agora a nossa tabela, ou melhor, a estrutura dela, foi adequada à Segunda Forma Normal (2FN), segundo a qual as colunas de uma tabela devem conter atributos relacionados a um único item da tabela, identificado pela chave primária. Em palavras mais simples, uma tabela não deve conter campos com dados redundantes.

Suponhamos, agora, que seja preciso acrescentar na tabela Andamento de Projetos um campo no qual será inserida uma breve descrição do projeto. Provavelmente modificaremos a estrutura da tabela da seguinte forma:

Matrícula	Projeto	Horas Trabalhadas	Descrição
FV-911	Projeto-AX	28	Realização de material publicitário para a empresa X
FV-245	Projeto-AX	32	Realização de material publicitário para a empresa X
FV-911	Projeto-BX	30	Desenvolvimento do website da empresa X
FV-465	Projeto-BX	41	Desenvolvimento do website da empresa X
FV-348	Projeto-BX	8	Desenvolvimento do website da empresa X
FV-911	Projeto-CX	12	Realização da abertura do programa de TV 123
FV-245	Projeto-CX	18	Realização da abertura do programa de TV 123

Matrícula	Projeto	Horas Trabalhadas	Descrição
FV-611	Projeto-CX	26	Realização da abertura do programa de TV 123
FV-465	Projeto-DX	13	Desenvolvimento do website da empresa Y
FV-732	Projeto-DX	9	Desenvolvimento do website da empresa Y
FV-611	Projeto-DX	31	Desenvolvimento do website da empresa Y

Tabela 1.18: Andamento de Projetos.

Assim estruturada, a tabela respeita a Primeira Forma Normal, mas não a Segunda, pois há informações repetidas no campo Descrição. Portanto, será preciso criar mais uma tabela, que podemos chamar Detalhes de Projetos, com as características mostradas a seguir:

Projeto	Descrição
Projeto-AX	Realização de material publicitário para a empresa X
Projeto-AX	Realização de material publicitário para a empresa X
Projeto-BX	Desenvolvimento do website da empresa X
Projeto-BX	Desenvolvimento do website da empresa X
Projeto-BX	Desenvolvimento do website da empresa X
Projeto-CX	Realização da abertura do programa de TV 123
Projeto-CX	Realização da abertura do programa de TV 123
Projeto-CX	Realização da abertura do programa de TV 123
Projeto-DX	Desenvolvimento do website da empresa Y
Projeto-DX	Desenvolvimento do website da empresa Y
Projeto-DX	Desenvolvimento do website da empresa Y

Tabela 1.19: Detalhes de Projetos.

Neste caso, a chave primária é o campo Projeto, cujo conteúdo é um nome que identifica de maneira unívoca cada projeto em andamento.

A Terceira Forma Normal – (3FN)

A Terceira Forma Normal nos impõe outra diretriz a ser seguida: nenhum atributo pode depender de outro que não seja a chave primária. Para entender melhor, vejamos mais um exemplo que usa o sistema de controle de projetos dos tópicos anteriores. Suponhamos que surja a necessidade de acrescentar, para cada projeto, o nome do respectivo Supervisor; provavelmente, seríamos levados a reestruturar a tabela Detalhes de Projetos da seguinte maneira:

Projeto	Descrição	ID_Supervisor	Nome_Supervisor
Projeto-AX	Realização de material publicitário para a empresa X	SV-1504	Frederico
Projeto-AX	Realização de material publicitário para a empresa X	SV-1504	Frederico
Projeto-BX	Desenvolvimento do website da empresa X	SV-1512	Márcio
Projeto-BX	Desenvolvimento do website da empresa X	SV-1512	Márcio
Projeto-BX	Desenvolvimento do website da empresa X	SV-1512	Márcio
Projeto-CX	Realização da abertura do programa de TV 123	SV-2606	Luciane
Projeto-CX	Realização da abertura do programa de TV 123	SV-2606	Luciane
Projeto-CX	Realização da abertura do programa de TV 123	SV-2606	Luciane
Projeto-DX	Desenvolvimento do website da empresa Y	SV-0906	Eduardo
Projeto-DX	Desenvolvimento do website da empresa Y	SV-0906	Eduardo
Projeto-DX	Desenvolvimento do website da empresa Y	SV-0906	Eduardo

Tabela 1.20: Nome do Supervisor.

Com essa nova estrutura, porém, a tabela não respeita nem a Segunda Forma Normal nem a Terceira Forma Normal: primeiramente porque existem dados redundantes nos registros da tabela, e também porque o atributo Nome_Supervisor não depende da chave primária da tabela (que é o campo Projeto), e sim, de um outro código (no campo ID_Supervisor) que não é chave primária.

Para que o nosso banco de dados seja normalizado, isto é, respeite todas as Formas Normais, será preciso criar uma nova tabela, que no nosso exemplo chamamos "Supervisores", que contenha apenas os dados relativos aos supervisores e que será o lado um de um relacionamento um-para-muitos com a tabela Detalhes de Projetos, na qual permanecerá apenas o campo ID_Supervisor que será a chave externa usada para estabelecer o relacionamento com o mesmo campo (a chave primária) da tabela Supervisores.

Resumindo, essa será a estrutura das tabelas Detalhes de Projetos e Supervisores:

Projeto	Descrição	ID_Supervisor
Projeto-AX	Realização de material publicitário para a empresa X	SV-1504
Projeto-AX	Realização de material publicitário para a empresa X	SV-1504
Projeto-BX	Desenvolvimento do website da empresa X	SV-1512
Projeto-BX	Desenvolvimento do website da empresa X	SV-1512
Projeto-BX	Desenvolvimento do website da empresa X	SV-1512
Projeto-CX	Realização da abertura do programa de TV 123	SV-2606
Projeto-CX	Realização da abertura do programa de TV 123	SV-2606
Projeto-CX	Realização da abertura do programa de TV 123	SV-2606
Projeto-DX	Desenvolvimento do website da empresa Y	SV-0906
Projeto-DX	Desenvolvimento do website da empresa Y	SV-0906
Projeto-DX	Desenvolvimento do website da empresa Y	SV-0906

Tabela 1.21: Detalhes de Projetos.

ID_Supervisor	Nome_Supervisor
SV-1504	Frederico
SV-1504	Frederico
SV-1512	Márcio
SV-1512	Márcio
SV-1512	Márcio
SV-2606	Luciane

ID_Supervisor	Nome_Supervisor
SV-2606	Luciane
SV-2606	Luciane
SV-0906	Eduardo
SV-0906	Eduardo
SV-0906	Eduardo

Tabela 1.22: Supervisores.

Depois de rever a estrutura das tabelas do nosso exemplo para adequá-la às três Formas Normais, obteremos um banco de dados articulado em quatro tabelas, como podemos ver no esquema da **Figura 1.15**.

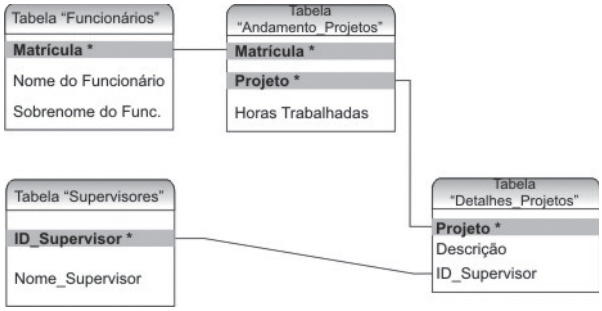


Figura 1.15: O banco de dados usado para o nosso exemplo, respeita agora as três Formas Normais. Veja como cada tabela está relacionada com a(s) outra(s). Os campos com asterisco representam as chaves primárias de cada tabela.

Algumas considerações sobre planejamento

É preciso lembrar que as regras fixadas para as três Formas Normais são cumulativas; isto quer dizer que, para respeitar a Terceira Forma Normal, a primeira e a segunda também devem ser respeitadas. Vale a pena ressaltar que, na teoria desenvolvida por especialistas em bancos de dados, foram definidas outras Formas Normais para a padronização da estrutura das tabelas, contudo, a aplicação das três formas vistas neste capítulo é mais que suficiente para criar bancos de dados normalizados, bem planejados e, principalmente, funcionais.

Na fase de planejamento das tabelas que agruparão os dados do nosso banco de dados, o conhecimento das Formas Normais é muito importante, todavia, é indispensável analisar e definir com cuidado as finalidades do banco de dados e traçar os resultados que se deseja obter. Não se esqueça que um banco de dados deve ser basicamente uma ferramenta para manipular com maior facilidade dados que, na maioria dos casos, são a matéria-prima do nosso trabalho ou atividade.

Capítulo 2

Trabalhando com o SQL (Parte 1)

Introdução

Por incrível que pareça, a linguagem SQL não trazia, originalmente, nenhum comando para a criação de um banco de dados. Isto pode parecer ilógico, mas existe uma explicação: o SQL é uma ferramenta para criar, gerenciar e utilizar tabelas, enquanto o banco de dados é o objeto (software) que as contêm. De fato, não faz parte das finalidades do SQL definir características e funcionamento de um banco de dados, mas apenas manipular dados contidos dentro de tabelas.

Todavia, precisamos de um banco de dados para, em seguida, estruturar as tabelas e inserir os dados a serem armazenados, por isso, todas as implementações do SQL trazem pelo menos um comando para criar um banco de dados vazio e algumas ferramentas para excluir bancos de dados existentes.

Ao criar um banco de dados como MySQL em ambiente Windows, é criada apenas uma pasta vazia, dentro da qual serão armazenados os arquivos gerados utilizando os comandos para criar tabelas.

Em outros SGBDR (Sistemas para o Gerenciamento de Bancos de Dados Relacionais), como o Oracle ou o Microsoft SQL Server, por exemplo, o comando para a criação de um novo banco de dados gera estruturas de dados complexas. Todos os procedimentos e exemplos dos tópicos deste capítulo serão baseados no software MySQL.

Na medida em que digitamos uma linha de comando, a interface do MySQL Control Center destaca na cor azul e em negrito as palavras reconhecidas como palavras-chave do SQL, deixando as demais palavras em estilo normal e na cor cinza. As palavras-chave podem ser digitadas em maiúsculo ou minúsculo, sem problemas, tanto no MySQL como nos demais softwares para o uso desta linguagem.

No decorrer dos exemplos deste livro, usaremos tipos de fonte diferentes para representar as linhas de código para realizarmos operações em bancos de dados com o Structured Query Language (SQL). Veja alguns exemplos:

- Todas as palavras-chave das instruções SQL serão escritas em maiúsculo, como mostrado a seguir:

```
CREATE TABLE
```

• Os elementos, ou parâmetros, especificados para cada comando aparecerão em itálico, como no exemplo a seguir:

```
CREATE TABLE nome_da_tabela
```

• Parâmetros ou comandos opcionais, isto é, instruções adicionais dadas aos comandos, aparecerão em maiúsculo e dentro de colchetes, assim:

```
CREATE TABLE [IF NOT EXISTS] nome_da_tabela
```

• Quando, em uma instrução, podemos utilizar um entre vários parâmetros alternativos, todas as opções serão apresentadas e separadas por barras verticais (|). Caso esses parâmetros sejam opcionais, serão listados entre colchetes, como no exemplo a seguir:

```
DROP TABLE nome_da_tabela [RESTRICT | CASCADE]
```

• Nos casos em que é obrigatório escolher um parâmetro dentro de uma série de opções, a lista das alternativas será exibida dentro de chaves, e cada parâmetro disponível será separado dos outros por barras verticais, como mostrado a seguir:

```
UPDATE {nome_da_tabela | nome_da_view}
```

• Em algumas circunstâncias, é possível que um mesmo elemento possa ser repetido diversas vezes; nesses casos, usaremos a seguinte representação:

```
Col1, col2, ...
```

Quando, no meio de uma linha de comando, encontrarmos parênteses e vírgulas, esses elementos constituem parte integrante da linha de comando e devem ser escritas exatamente como apresentado no livro. Já os colchetes [ ] e as chaves { } não deverão ser digitados (veja o significado desses elementos nos parágrafos anteriores).

As linhas de comando SQL podem ser muito compridas, portanto, é possível que se articulem em mais de uma linha na tela do computador. A quebra de linha pode ocorrer em qualquer lugar da linha em que seja necessário dar um espaço. Jamais devemos quebrar palavras-chave ou nomes de parâmetros e opções.

Na linguagem SQL, toda linha de comando (ou instrução) deve ser encerrada com o caractere ponto-e-vírgula (;). Todavia, muitos ambientes de programação aceitam sua omissão, o que não prejudica o funcionamento do código.

A linguagem SQL não impõe nenhum limite quanto ao tamanho das tabelas criadas, isto é, à quantidade de linhas e colunas. Entretanto, cada ambiente de desenvolvimento apresenta as suas próprias limitações. No caso do ambiente MySQL, os limites são praticamente infinitos, pois as tabelas criadas por esse SGBDR nada mais são que simples arquivos gerados de acordo com as especificações do sistema operacional no qual estamos trabalhando. Portanto, não há nada que impeça de criar uma tabela de tamanho considerável (um ou mais gigabytes, por exemplo), contudo, uma tabela com essas características traria problemas na fase de trabalho, pois nem todo computador seria capaz de gerenciar corretamente tamanha quantidade de dados em uma só tabela. Por isso, deveremos sempre levar em consideração o conceito de base que norteia a criação de um banco de dados: os dados devem ser sempre ordenados em várias tabelas, cada qual contendo dados homogêneos, ou seja, do mesmo tipo.

Criando um novo banco de dados

Vejamos, então, como se cria um novo banco de dados utilizando o MySQL. Siga os passos a seguir:

1. Ative o **MySQL Command Line Client** e, ao ser solicitado, digite a sua senha de acesso.

2. No Prompt do MySQL, digite a seguinte linha de comando:

```
mysql>  CREATE DATABASE Teste;    [Enter]
```

Lembre-se de finalizar sempre as linhas de comando com o caractere ponto-e-vírgula.

O novo banco de dados foi criado e o MySQL nos informa disso exibindo uma mensagem (em inglês) logo depois da instrução que digitamos:

```
mysql>  CREATE DATABASE Teste;
Query OK, 1 row affected (0.08 sec)

mysql>
```

3. Podemos verificar rapidamente a existência do banco de dados recém-criado, bem como a de todos os outros criados anteriormente, utilizando a instrução `SHOW DATABASES` (mostrar bancos de dados); para isso, digite a seguinte linha de comando:

```
mysql>  SHOW DATABASES;    [Enter]
+--------------------+
| Databases          |
+--------------------+
| information _ schema |
| mysql              |
| test               |
| teste              |
+--------------------+
4 rows in set (0.05 sec)
```

Como podemos observar, é exibida a lista dos bancos de dados existentes que, no nosso exemplo, são quatro; os primeiros três são criados na instalação do MySQL (information_schema, mysql e test), o último (teste), foi criado por nós usando o comando `CREATE DATABASE` (criar banco de dados).

O comando `SHOW DATABASES` é muito útil para sabermos os nomes dos bancos de dados já existentes, visto que não podemos criar dois bancos de dados com o mesmo nome. Por exemplo, se tentássemos criar novamente o banco de dados teste, receberíamos a seguinte mensagem de erro:

```
Can't create database 'Teste'. Database exists
```

A mensagem nos informa que não é possível criar um banco de dados com o nome teste, pois ele já existe.

Este tipo de erro pode ser prevenido acrescentando ao comando `CREATE` a clausula `if not exists` (em inglês "se não existe"). Veja um exemplo:

```
mysql>  CREATE DATABASE IF NOT EXISTS Teste;   [Enter]
```

Traduzindo ao pé da letra, seria algo como: "Se não existe, crie o banco de dados Teste".

Caso o banco de dados tenha sido criado com sucesso, será exibida a mensagem já vista no **Passo 2** desses procedimentos. Se já houver um banco de dados com o nome especificado, nada acontecerá e não será exibida qualquer mensagem de erro, devido à cláusula limitativa que especificamos.

Ao definirmos o nome do novo banco de dados, precisamos respeitar algumas limitações:

- O comprimento máximo do nome não pode superar 64 caracteres;
- Podemos usar letras, números, traços, underlines; desaconselha-se o uso de acentos e cedilhas, e é proibido o uso de barras (/ ou \) e pontos (.).

Excluindo um banco de dados

No tópico anterior, vimos como criar um banco de dados com o comando `CREATE DATABASE`. Vejamos agora como excluir um banco de dados existente.

É preciso ressaltar que, ao apagar um banco de dados, todas as suas tabelas e os dados nelas contidos também serão apagados e, portanto, perdidos de maneira irreversível.

Para excluir um banco de dados, usa-se o comando `DROP DATABASE`; para excluir, por exemplo, o banco de dados Teste criado no tópico anterior, faça o seguinte:

1. No prompt do MySQL, digite a linha de comando como mostrada a seguir:

```
mysql>  DROP DATABASE Teste;      [Enter]
```

2. Será exibida a mensagem de confirmação a seguir (o valor entre parênteses é o tempo que o computador levou para realizar a operação):

```
mysql>  DROP DATABASE Teste;
Query OK, 1 row affected (0.05 sec)
```

Se tentarmos apagar um banco de dados que não existe, receberemos uma mensagem de erro. Por exemplo, tente apagar novamen-

te o banco de dados Teste. Como ele já foi excluído e, portanto, não existe mais, o MySQL mostrará esta mensagem:

```
ERROR 1008 (HY000): Can't drop database 'Teste'; database
doesn't exist
```

O que significa que não foi possível apagar o banco de dados Teste, pois ele não existe.

Podemos evitar esse erro usando a cláusula `IF EXISTS` ("se existe, em inglês"), da seguinte maneira:

```
mysql>  DROP DATABASE IF EXISTS Teste;  [Enter]
```

Traduzindo o comando em português, ficaria mais ou menos assim: "Exclua o banco de dados Teste, caso exista."

Selecionando um banco de dados

Como vimos, podemos criar vários bancos de dados, porém, podemos manipular apenas um por vez. Assim, antes de começar, é preciso selecionar qual será o banco de dados que queremos alterar. Isso é feito utilizando o comando `USE` ("usar" em inglês), seguido pelo nome do banco de dados em questão.

Então, façamos o seguinte: vamos criar um novo banco de dados, listar os bancos de dados existentes e, por fim, selecionar o nosso para que possa ser manipulado. Siga os passos a seguir:

1. No prompt do MySQL, digite:

```
mysql>  CREATE DATABASE MeuBD;   [Enter]
```

2. Recebida a mensagem de confirmação, digite:

```
mysql>  SHOW DATABASES;    [Enter]
```

Que resultará na seguinte lista:

```
+--------------------+
| Databases          |
+--------------------+
| information_schema |
| meubd              |
```

```
| mysql              |
| test               |
+--------------------+
4 rows in set (0.05 sec)
```

3. Note que o segundo item da lista é o banco de dados recém-criado; então, digite:

```
mysql>  USE MeuBD;  [Enter]
```

A mensagem `Database changed` nos informa que o banco de dados foi ativado.

O comando `USE` não prevê o uso de argumentos (cláusulas), portando, não é possível, por exemplo, usar `IF EXISTS` para evitar erros. Além disso, é preciso digitar o nome do banco de dados que se deseja ativar exatamente da mesma forma em que foi digitado ao criá-lo. Assim, recomendamos sempre usar o comando `SHOW DATABASES` para visualizar os nomes dos bancos de dados, mais especificamente do banco de dados que se deseja utilizar, para não incorrer em mensagens de erro.

Criando e listando as tabelas

No tópico **Criando um novo banco de dados** aprendemos a usar o comando `CREATE` associado ao objeto `DATABASE` para criar um banco de dados.

O comando `CREATE` pode ser associado também ao objeto `TABLE` (tabela) para criar novas tabelas no banco de dados ativo. De fato, o comando `CREATE TABLE` é um dos mais importantes da linguagem SQL, pois é a partir de uma nova tabela que começaremos a manipulação dos dados.

A sintaxe deste comando é a seguinte:

```
CREATE [GLOBAL TEMPORARY | LOCAL TEMPORARY] TABLE nome _
da _ tabela
[ON COMMIT {PRESERVE ROWS | DELETE ROWS}] (nome _ da _ coluna
tipo _ de _ dados especificações, [nome _ da _ coluna, tipo _
de _ dados especificações2,...]) | [LIKE nome _ da _ tabela] |
[vínculo _ tabela],...n] ]
```

Vista assim, a sintaxe pode parecer muito complexa. Na verdade, é mais simples do que possamos pensar. Vamos nos aprofundar um pouco mais.

Antes de tudo, é importante lembrar que cada dialeto do SQL pode apresentar inúmeras opções e argumentos adicionais para este comando, além das previstas pelo padrão SQL99. Obviamente, nos nossos exemplos veremos como usar o comando `CREATE TABLE` no MySQL, começando pelas opções mais básicas para, em seguida, analisarmos as mais complexas.

Uma tabela deve ter um nome e pelo menos uma coluna, também identificada com um nome próprio e obrigatoriamente diferente do nome da tabela. Visto que as colunas são, na verdade, recipientes de dados, para cada tipo de coluna é preciso definir que tipo de dados conterá (números, textos etc.), especificando uma ou mais características (especificações) que identifiquem aquele determinado tipo de dados. Assim, considerando apenas as opções essenciais, a sintaxe do comando `CREATE TABLE` é a seguinte:

```
CREATE TABLE nome _ da _ tabela (nome _ coluna1 tipo _ de _ dados especificações, [nome _ coluna2 tipo _ de _ dados especificações,...]
```

Com base nas informações que possuímos até o momento, vamos criar uma tabela, que chamaremos de Cadastro, cuja estrutura é composta por duas colunas (Nome e sobrenome).

> **Observação:** lembre-se que para criar uma tabela é indispensável selecionar o banco de dados que a conterá. No tópico **Selecionando um banco de dados**, criamos e selecionamos um banco de dados vazio chamado MeuBD. Se desejar acompanhar os procedimentos descritos a seguir, você deverá estar com um banco de dados criado e ativo.

Então, siga os passos mostrados:

1. No prompt do MySQL, digite esta linha de comando:

```
mysql>  CREATE TABLE Cadastro (Nome CHAR (15), Sobrenome CHAR (20));  [Enter]
```

Vejamos em detalhes o que significa a instrução digitada: primeiramente foi usado o comando CREATE TABLE (criar tabela), seguido pelo nome da tabela a ser criada (no nosso exemplo, Cadastro). Entre parênteses, definimos o nome da primeira coluna (Nome), o tipo de dados (CHAR, ou seja, caracteres de texto) e o tamanho máximo dos dados desta coluna (15); em outras palavras, criamos uma coluna chamada Nome, que conterá um texto de, no máximo, 15 caracteres. Após a vírgula, definimos as características da segunda coluna, que se chama Sobrenome, também conterá textos (CHAR) e seu comprimento máximo será de 20 caracteres. Neste caso, CHAR é uma palavra-chave que identifica o tipo de dados que a coluna armazenará. Configurando uma coluna para armazenar dados desse tipo, não significa que não poderemos digitar números, mas sim que, mesmo que seu conteúdo seja uma série numérica, o banco de dados a tratará sempre como se fosse um texto, ou seja, não será possível, por exemplo, realizar cálculos matemáticos com o conteúdo daquela coluna.

2. Após termos recebido a mensagem que confirma a execução do comando (Query OK, 0 rows affected (0.16 sec)), e de voltar ao prompt do MySQL, digite a instrução a seguir para verificar a existência da tabela criada:

```
mysql>  SHOW TABLES;       [Enter]
```

Será exibida a lista mostrada a seguir:

```
+--------------------+
| Tables _ in _ meuBD  |
+--------------------+
| cadastro           |
+--------------------+
1 row in set (0.01 sec)
```

O comando SHOW não pertence ao padrão SQL99 definido por ANSI/ISO, mas encontra-se disponível em vários dialetos do SQL. No caso do MySQL, pode ser combinado com a respectiva palavra-chave para visualizar uma lista dos bancos de dados existentes (SHOW DATABASES), das tabelas existentes (SHOW TABLES), das colunas de uma tabela etc. Trata-se de um comando muito útil quando precisamos verificar rapidamente as características de um banco de dados

ou de um dos seus elementos antes de digitar a linha de comando que o afetará.

Visualizando a estrutura das tabelas

Podemos também analisar a estrutura de uma tabela de maneira aprofundada usando o comando `DESCRIBE` ("descrever", em inglês), seguido pelo nome da tabela. Vejamos como aplicá-lo à tabela que criamos no tópico anterior:

1. No prompt do MySQL, digite:

```
mysql>  DESCRIBE Cadastro;        [Enter]
```

O resultado será como mostrado a seguir:

```
+-------------+ +-----------+ +--------+ +-------+ +-----------+ +--------+
| Field       | | Type      | | Null   | | Key   | | Default   | | Extra  |
+-------------+ +-----------+ +--------+ +-------+ +-----------+ +--------+
| Nome        | | char(15)  | | YES    | |       | | NULL      | |        |
| Sobrenome   | | char(20)  | | YES    | |       | | NULL      | |        |
+-------------+ +-----------+ +--------+ +-------+ +-----------+ +--------+
2 rows in set (0.04 sec)
```

Assim como o comando `SHOW`, o comando `DESCRIBE` também não faz parte do padrão SQL99, mas é muito difuso entre os dialetos SQL. De fato, sua utilidade é indiscutível, como podemos perceber no exemplo anterior, obtivemos uma descrição detalhada das informações a respeito da nossa tabela. Sua única limitação é que todas essas informações são exibidas em inglês, o que, se não chega a ser um problema, pelo menos dificulta a vida de quem não domina o idioma. Para ajudar a entender, vejamos detalhadamente cada informação que o comando `DESCRIBE` nos retornou:

- Na primeira coluna, são exibidos os campos (Field) da nossa tabela, que são Nome e Sobrenome;
- A segunda coluna nos mostra o tipo de dados (Type) que cada coluna contém. No nosso exemplo, ambas as colunas foram predispostas para armazenar valores alfanuméricos (`CHAR`), porém, a primeira tem limite de 15 caracteres, enquanto a segunda pode conter até 20 caracteres;
- A terceira coluna (Null) nos informa que ambos os campos poderão (Yes) conter valores nulos;
- A coluna Key se refere à chave primária. Na nossa tabela, nenhum dos dois campos foi definido;

- A quinta coluna (Default) traz informações sobre qual é o valor padrão para cada coluna, ou seja, ao inserir um novo registro, qual será o valor que este terá por padrão. Na nossa tabela, ambas as colunas serão vazias (Null);
- A coluna Extra traz informações adicionais de qualquer tipo a respeito de cada coluna da tabela. Como não especificamos nenhuma, encontra-se vazia.

Inserindo dados em uma tabela

A tabela Cadastro, que criamos nos tópicos anteriores, ainda não contém dados, pois definimos a sua estrutura, mas ainda não realizamos a inserção de valores nas colunas. Para inserir dados nesta e em outras tabelas é preciso usar o comando `INSERT` ("inserir", em inglês), cuja sintaxe é basicamente a seguinte:

```
INSERT nome _ da _ tabela VALUES (valor1, valor2,...)
```

Trata-se de uma sintaxe bastante simples: primeiramente digitamos o comando `INSERT` seguido pelo nome da tabela na qual queremos inserir os dados; então, especificamos a cláusula `VALUES` (valores) e, entre parênteses, os valores de cada coluna da tabela, separados por vírgulas. Quando os valores a serem inseridos forem do tipo texto (CHAR) deverão ser especificados entre aspas duplas.

Veja um exemplo para entender melhor:

Vamos inserir na nossa tabela Cadastro o primeiro item, ou seja, a primeira pessoa:

1. Lembre-se que a nossa tabela é constituída por dois campos (Nome e Sobrenome), ambos do tipo `CHAR` (texto); assim, para inserir, respectivamente, o nome e o sobrenome da primeira pessoa, digite o enunciado mostrado a seguir:

```
mysql>  INSERT Cadastro VALUES (Augusto, J. Vésica); [Enter]
```

2. Em instantes, o MySQL nos retorna a mensagem de confirmação para o comando digitado:

```
Query OK, 1 row affected (0.06 sec)
```

Para verificarmos se os dados realmente foram inseridos na tabela, e, o mais importante, se foram inseridos na seqüência correta, é preciso visualizar o conteúdo da tabela que, até o momento, deveria ser de um único registro; para isso, precisamos do comando `SELECT`, do qual falaremos no tópico seguinte.

Visualizando o conteúdo de uma tabela

No tópico anterior, inserimos o primeiro registro na nossa tabela Cadastro, mas o que fazer se desejarmos ver esse conteúdo para verificar se foi digitado corretamente? É para isso que o SQL oferece o comando `SELECT`, considerado um dos comandos fundamentais da Structured Query Language (SLQ). Este comando pertence ao grupo chamado *Data Manipulation Language* (DML) – Linguagem de Manipulação de Dados – e, como o próprio nome deixa imaginar, é usado para selecionar dados de uma tabela.

O comando `SELECT` é, basicamente, a ferramenta principal para consultar informações de um banco de dados, por isso, é comumente chamado de query (que, em inglês, significa "consulta").

Sua sintaxe essencial é muito simples:

```
SELECT dados _ desejados FROM nome _ da _ tabela;
```

Em que dados_desejados é aquilo que se deseja obter da tabela especificada como argumento da cláusula `FROM`; em outras palavras, dados_desejados é o critério com base no qual serão extraídos os dados da tabela.

Para definir esse critério, podemos usar uma palavra inteira, parte dela, ou, ainda, usar os caracteres especiais asterisco (*) e interrogação (?), cujos significados são os seguintes:

- **Asterisco (*):** significa tudo, ou seja, todos os dados. Pode ser combinado com um ou mais caracteres para especificar conjuntos de dados com algo em comum, por exemplo, em geral, se digitarmos o critério A* significa que queremos ver todos os registros cujo conteúdo começa com a letra A;
- **Interrogação (?):** representa um caractere desconhecido. Por exemplo, se definirmos como critério o valor ?????, quer dizer que queremos ver somente os registros que, em determinado campo, contenham valores de cinco caracteres.

Vamos utilizar o comando SELECT para visualizar todo o conteúdo da nossa tabela. Para isso, digite a linha de comando a seguir:

```
mysql>  SELECT * FROM Cadastro;          [Enter]
```

Em poucos instantes, veremos o resultado, que no nosso exemplo será:

```
+---------+-------------+
| Nome    | Sobrenome   |
+---------+-------------+
| Augusto | J. Vésica   |
+---------+-------------+
1 row in set (0.03 sec)
```

Experimente agora inserir mais um item na tabela usando o comando INSERT, (como mostrado no tópico **Inserindo dados em uma tabela**) e, em seguida, visualize o conteúdo da tabela com a ajuda do comando SELECT.

Considerações sobre os tipos de dados para as colunas

É preciso tomar muito cuidado ao definirmos os tipos de dados para as colunas da tabela em que estamos trabalhando. De fato, se planejarmos de maneira errada as características dos dados que cada coluna irá conter, correremos o risco de perder dados sem perceber, ou então, o nosso banco de dados se tornará inviável para se trabalhar.

Para entender melhor do que estamos falando, façamos um teste com a nossa tabela Cadastro: sabemos que dispomos de dois campos (Nome e Sobrenome), e que, para cada um deles, foi definido um limite de caracteres (respectivamente, 15 para o primeiro e 20 para o segundo), mas, o que aconteceria se tentássemos inserir na tabela um sobrenome com, por exemplo, 30 ou mais caracteres? Vejamos:

1. Com o banco de dados MeuBD ainda ativo e a tabela Cadastro ainda aberta, digite no prompt do MySQL o enunciado mostrado a seguir:

```
mysql>  INSERT Cadastro VALUES (Luiz Henrique, Magalhães
Figueiredo de Castro Júnior);          [Enter]
```

Imediatamente, receberemos do MySQL uma mensagem de erro como mostrado a seguir:

```
ERROR 1406 (22001): Data too long for column 'Sobrenome' at
row 1
```

Isso significa que digitamos um valor longo demais (TOO LONG) para a coluna Sobrenome e, portanto, o registro não foi inserido.

É sabido que sobrenomes como o do nosso exemplo não são raros no nosso país, portanto, é evidente que um limite de apenas 20 caracteres para o sobrenome constitui um obstáculo para o nosso trabalho.

Para evitar erros de planejamento na estrutura das nossas tabelas, veremos neste tópico alguns detalhes importantíssimos sobre os tipos de dados que as colunas podem armazenar e as características de todos eles. A seguir, veremos um exemplo prático:

Um campo destinado a conter caracteres em uma tabela SQL pode conter de 1 até 255 caracteres. Quando este campo é definido como CHAR, especificando um comprimento (como na tabela Cadastro do nosso exemplo), estamos definindo um tamanho máximo efetivo e o campo ocupará sempre o mesmo espaço na memória do computador, mesmo que contenha menos caracteres; em outras palavras, se definimos um campo CHAR com comprimento de 80 caracteres, este ocupará sempre 80 bytes na memória, mesmo se os dados nele contidos não usam os 80 caracteres reservados. Por isso, os campos do tipo CHAR são chamados de campos com comprimento fixo. Dispomos também de outro tipo de dado para armazenar seqüências de caracteres, chamado VARCHAR, que aproveita melhor o espaço na memória: de fato, ao especificarmos um comprimento para um campo do tipo VARCHAR, estamos definindo um teto máximo e não um tamanho fixo. Por isso, os campos desse tipo são tidos como campos com comprimento variável. Em palavras mais simples, se definirmos um campo como VARCHAR (80) esse campo não ocupará sempre 80 bytes na memória, mas apenas o espaço correspondente à quantidade efetiva de caracteres nele contidos.

É evidente, então, o quanto é importante definir corretamente o tipo de dado para cada coluna e como isso afeta de maneira incisiva o funcionamento das tabelas do nosso banco de dados.

A seguir, os tipos de dados disponíveis na linguagem SQL e os do MySQL, e suas respectivas descrições.

Tipo de dados do MySQL	Tipo de dados do padrão SQL99	Descrição
bigint		Usado para números inteiros, positivos ou negativos, que vão de -9.223.372.036.854.775.808 a +9.223.372.036.854.775.807.
CHAR(n)	CHARACTER(n)	Usado para seqüências de caracteres de comprimento fixo, a quantidade de caracteres é definida no parâmetro *n* e pode variar de 1 a 255.
DATE	DATE	Usado para datas (no formato ano-mês-dia), dentro do intervalo de 1000-1-1 a 9999-12-31.
datetime	datetime	Usado para valores de data e hora que vão de 1000-1-1 00:00:00 a 9999-12-31 23:59:59.
DECIMAL(p,s)	DECIMAL(p,s)	Usado para números decimais não arredondados, definidos com *p* números antes da vírgula e *s* casas decimais.
DOUBLE(p,s)Double Precision	Double precision	Usado para valores numéricos com vírgula variável, com precisão dupla (até 308 posições).
ENUM("Valor1", "Valor2",...)		Usado para uma seqüência de caracteres que pode ter somente um valor escolhido dentro de uma lista de valores ("Valor1", "Valor2",...). Permite selecionar dentro de 65.535 valores diferentes.
Float	FLOAT(p)	Usado para valores numéricos com vírgula variável (até 38 posições).
INT, INTEGER	INT, INTEGER	Usado para números inteiros, positivos ou negativos, que vão de -2.147.483.548 a +2.147.483.547.

Tipo de dados do MySQL	Tipo de dados do padrão SQL99	Descrição
LONGBLOB, LONGTEXT	Binary large object (BLOB)	Usado para blocos de caracteres binários ou de texto longo (até 4.294.967.295 caracteres).
MEDIUMBLOB, MEDIUMTEXT		Usado para blocos de caracteres binários ou de texto longo (até 16.444.216 caracteres).
mediumint		Usado para números inteiros, positivos ou negativos, que vão de -8.388.608 a +8.388.607.
NUMERIC(p,s)	NUMERIC(p,s)	O mesmo que DECIMAL.
REAL(p,s)	Double precision	O mesmo que DOUBLE PRECISION.
SET("Valor1", "Valor2",...)		Usado para uma seqüência de caracteres que pode ter zero ou mais valores, especificados na lista ("Valor1", "Valor2",...). A lista pode conter até 64 valores diferentes.
smallint	smallint	Usado para números inteiros, positivos ou negativos, que vão de -32.758 a +32.757.
TEXT	Linha 18 vazia	Usado para blocos de texto longo com até 65.536 caracteres.
TIME	TIME	Usado para valores de hora no formato hh:mm:ss.
TIMESTAMP(n)	Timestamp	Usado para valores de data e hora que vão de 1970-01-01 00:00:00 até 2037-12-31 23:59:59. O parâmetro *n* pode ter seu valor configurado em 14, 12, 8 ou 6 e se refere ao comprimento do valor armazenado.
TINYBLOB, TINYTEXT		Usado para valores de BLOB ou TEXT com até 255 caracteres de comprimento.

Tipo de dados do MySQL	Tipo de dados do padrão SQL99	Descrição
`Tinyint`		Usado para números inteiros, positivos ou negativos, que vão de -128 a +127, ou entre 0 e 255, se não especificar o sinal (+/-).
`VARCHAR(n)`	`CHARACTER VARYING(n)`	Usado para seqüências alfanuméricas de comprimento variável, de até 255 caracteres.
`YEAR[(2 | 4)]`		Usado para valores de anos, com dois ou quatro dígitos, no intervalo de (19)70 até (20)69, para o formato de dois dígitos, ou de 1901 até 2155, para o formato de quatro dígitos - .

Tabela 2.1: Tipos de dados disponíveis na linguagem SQL e do MySQL, e suas respectivas descrições.

Para alguns tipos de dados é prevista a indicação de dois parâmetros entre parênteses, o primeiro representa a precisão e o segundo a escala. Por exemplo:

```
DECIMAL(5,2)
```

Que representa um número decimal com precisão 5 e escala 2. A precisão indica a quantidade de valores significativos que serão armazenados no campo, já a escala representa a quantidade de números após a vírgula, isto é, as casas decimais.

Em uma coluna do tipo `DECIMAL` (5,2), os valores possíveis variam de -99,99 e 999,99 (o sinal de menos (-) ocupa uma posição entre as indicadas no parâmetro precisão). Esse tipo de dado é bastante utilizado para armazenar valores em dinheiro, não arredondáveis, enquanto os outros tipos, com vírgula variável (`FLOAT`, `DOUBLE`, `REAL`), são empregados para armazenar valores numéricos potencialmente muito grandes ou muito pequenos adotando a notação científica, que realiza arredondamentos.

Os tipos de dados `BLOB` (*Large Binari Objects*) se referem a arquivos no formato binário, que geralmente contêm imagens ou sons.

O padrão SQL inclui também um tipo de dados chamado *Boolean*, que não existe no MySQL, usado para armazenar os valores lógicos Verdadeiro/Falso. No MySQL isto é feito usando o tipo *tinyint* com parâmetro (1).

Sobre a notação científica (vírgula variável)

A notação científica é usada, geralmente, nos cálculos relacionados à engenharia, à astronomia, à física e a outras ciências para representar, de maneira mais compacta, números muito extensos ou muito pequenos. Essa representação é feita utilizando o esquema:

$$\mathbf{m \cdot 10^{e}}$$

em que *m* é denominado "*mantissa*" e é a parte significativa do número em questão, e **e** é o expoente que representa a ordem de grandeza.

Para transformar um número qualquer para a notação científica padronizada, devemos deslocar a vírgula obedecendo ao princípio de equlíbrio.

Vejamos o exemplo a seguir:

253 756,42

A notação científica padronizada exige que a mantissa esteja entre 1 e 10. Nessa situação, o valor adequado seria 2,5375642 (observe que a seqüência de algarismos é a mesma, somente foi alterada a posição da vírgula). Para o exponente, vale o princípio de equilíbrio: cada casa decimal que diminui o valor da mantissa aumenta o expoente em uma unidade, e vice-versa.

Nesse caso, o expoente é 5.

Observe a transformação passo a passo:

253 756,42 = 25 375,642 · 101 = 2 537,5642 · 102 = 253,75642 · 103 = 25,375642 · 104 = 2,5375642 · 105

Um outro exemplo, com valor menor que 1:

0,0000000475 = 0,000000475 · 10-1 = 0,00000475 · 10-2 = 0,0000475 · 10-3 = 0,000475 · 10-4 = 0,00475 · 10-5 = 0,0475 · 10-6 = 0,475 · 10-7 = 4,75 · 10-8

Desse modo, os exemplos anteriores ficarão, respectivamente, assim:

- 6 · 105
- 3 · 107
- 5 · 1014
- 7 · 1033
- 4 · 10-4
- 1 · 10-8
- 6 · 10-16
- 8 · 10-49

A notação científica é chamada também "representação com vírgula móvel" (ou variável) – "floating point", em inglês – porque a posição da vírgula (casas decimais) muda em função do valor à direita do símbolo **e**. Por exemplo, o número 5,83E+130 representa o número 583 seguido por 128 zeros.

Nos cálculos realizados com números científicos do tipo FLOAT, o MySQL considera as casas decimais até a trigésima oitava posição.

Para os números do tipo DOUBLE PRECISION, as posições decimais consideradas chegam a 308. Obviamente, trata-se de limites muito elevados, que devem ser utilizados apenas quando criamos tabelas nas quais serão realizados cálculos científicos muito complexos e refinados. Nos bancos de dados comuns, usados, por exemplo, para gerenciar dados relativos à movimentação bancária, controle de estoque, vendas etc., é melhor usar o tipo `DECIMAL`, que armazena dados numéricos decimais e sem arredondamentos.

Capítulo 3

Trabalhando com o SQL (Parte 2)

Criando uma tabela mais complexa

Nos tópicos anteriores, criamos uma tabela chamada Cadastro, cuja estrutura ficou bastante simples. Por meio do exemplo, aprendemos a utilizar o comando `CREATE TABLE` para criar uma nova tabela vazia.

Já vimos também que a sintaxe do comando `CREATE TABLE` possui uma quantidade considerável de opções (cláusulas e parâmetros) que permitem criar tabelas extremamente complexas. De fato, os bancos de dados do SQL são usados normalmente para criar e manipular tabelas com estruturas articuladas em inúmeras colunas e com os mais diversos tipos de dados.

No exemplo desse tópico, criaremos um enunciado mais complexo que criará uma tabela de movimentação bancária, que chamaremos Movimentacao.

Siga os passos a seguir:

1. No prompt do MySQL, digite o enunciado mostrado a seguir:

```
mysql> CREATE TABLE Movimentacao (Codigo INT NOT NULL  PRI-
MARY KEY  AUTO _ INCREMENT, Banco CHAR(10), Conta CHAR(10),
Tipo CHAR(3), Numero CHAR(3), Data DATETIME, Valor DECI-
MAL(8,2) NOT NULL, Descricao VARCHAR(50));
[Enter]
```

Note que a linha de comando foi digitada sem caracteres acentuados e substituindo os cedilhas por c comuns, pois a sintaxe do MySQL não permite o uso desses tipos de caracteres.

2. Aguarde a mensagem de confirmação:

```
Query OK, 0 rows affected (0.06 sec)
```

E, em seguida, use o comando `DESCRIBE` para verificar sua estrutura; o enunciado é o seguinte:

```
mysql> DESCRIBE Movimentacao;            [Enter]
```

3. O resultado será igual ao mostrado a seguir:

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| Codigo     | int(11)      | NO   | PRI | NULL    | auto _ increment|
| Banco      | char(10)     | YES  |     | NULL    |                |
| Conta      | char(10)     | YES  |     | NULL    |                |
| Tipo       | char(3)      | YES  |     | NULL    |                |
| Numero     | char(3       | YES  |     | NULL    |                |
| Data       | datetime     | YES  |     | NULL    |                |
| Valor      | decimal(8,2) | NO   |     |         |                |
| Descricao  | varchar(50)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
8 rows in set (0.02 sec)
```

Vejamos em detalhes o que significa o enunciado que formulamos e qual foi o resultado que obtivemos:

- Usando o comando `CREATE TABLE` criamos uma nova tabela, cujo nome é Movimentacao (foi preciso retirar o acento e o cedilha para que o SQL aceitasse esse nome). Até aqui, tudo igual como ocorreu com a criação da tabela Cadastro, vista anteriormente;
- Definimos que essa nova tabela é formada por oito campos (ou colunas), chamados, respectivamente, Codigo, Banco, Conta, Tipo, Numero, Data, Valor e Descricao (aqui também tornou-se necessário retirar acentos e cedilhas);
- O campo Codigo será um número inteiro (INT) com tamanho máximo de 11 caracteres; esse campo foi definido como chave primária (PRI), e não aceitará valores nulos, ou seja, não poderá ser deixado em branco, e será incrementado automaticamente a cada novo registro (`auto _ increment`);
- Os campos Banco, Conta, Tipo, Numero são do tipo `CHAR`, ou seja, poderão receber seqüências de caracteres alfanuméricos. Para cada campo foi definido o comprimento, ou seja, a quantidade de caracteres que poderá armazenar;
- O campo Descricao é do tipo `VARCHAR` e seu comprimento máximo será de 50 caracteres; este campo poderá conter um texto longo, cujo limite foi definido em 50 caracteres alfanuméricos (incluindo os espaços);
- O campo Data, destinado a armazenar a data de cada operação bancária, é do tipo `DATETIME` (data/hora) e não é preciso especificar um tamanho;
- O campo Valor foi definido como `DECIMAL` e os dados entre parênteses indicam a quantidade máxima de caracteres que esse campo irá hospedar (8) e a quantidade de casas decimais a ser apresentada, ou seja, quantos números deverão aparecer depois

da vírgula (neste caso, dois). Para exemplificar, os valores desse campo terão o formato 12345678,00;
• Todos os campos, com exceção de Codigo e Valor, aceitarão valores nulos.

Para definirmos algumas características adicionais da nossa tabela, precisamos utilizar as seguintes palavras-chave e cláusulas:

Palavra-Chave / Cláusula	O que significa
NOT NULL	Um campo com essa característica não admite valores nulos (vazio). Na tabela do nosso exemplo, os campos Codigo e Valor não poderão estar em branco.
PRIMARY KEY	É usada para definir um campo (coluna) como chave primária da tabela, ou seja, um valor do registro que jamais poderá ser repetido na mesma coluna e que servirá para identificar cada entrada da tabela de maneira unívoca. Falamos sobre esse assunto no tópico **Chaves primárias e univocidade dos dados**, no **Capítulo 1** deste livro.
AUTO _ INCREMENT	Dando essa instrução como argumento para um campo de uma tabela, o conteúdo do campo (numérico) será preenchido automaticamente com um valor numérico único todas as vezes que um novo registro é adicionado à tabela. No nosso exemplo, definimos esta característica para o campo Codigo, que é também a chave primária da tabela Movimentacao; trata-se de uma medida preventiva que evita a inserção de valores repetidos nesse campo, o que é proibido por se tratar de uma chave primária.

Tabela 2.1: Palavra-chave.

Agora, vamos tentar inserir o primeiro registro da tabela, informando uma série de valores (um para cada campo). Como já foi visto, para isso é usado o comando INSERT. Portanto, realize os procedimentos descritos a seguir:

1. Digite a seguinte linha de comando, exatamente como é apresentada:

```
mysql> INSERT Movimentacao VALUES (1, "Bradesco", "12444-8",
"Dep", "019", "2007/1/12", 1587.24, "Pgto aluguel sala");
     [Enter]
```

2. Verifique a correta inserção dos dados; caso não haja erros na sintaxe, o MySQL mostrará a mensagem:

```
Query OK, 1 row affected (0.05 sec)
```

Há algumas considerações a serem feitas sobre a inserção de dados em tabelas com campos de tipos diferentes:

- O conteúdo dos campos do tipo `CHAR` ou `VARCHAR`, deve ser especificado entre aspas simples ou duplas;
- Os valores numéricos, sejam do tipo `INT` ou `DECIMAL`, devem ser digitados sem as aspas. O separador de casas decimais em SQL é sempre o ponto, independentemente das configurações regionais do sistema operacional em uso (como vimos, a vírgula em SQL é um elemento de sintaxe usado para separar valores e não casas decimais);
- Os valores do tipo `DATETIME` devem ser informados entre aspas (simples ou duplas) e sempre no formato AAAA/MM/DD (ano com quatro dígitos / mês com dois dígitos / dia com dois dígitos), independentemente das configurações regionais do sistema operacional em uso.

Note no enunciado que formulamos que apenas os valores dos campos Codigo e Valor foram inseridos sem as aspas, pois são os únicos dois campos numéricos da tabela.

3. Para conferir se tudo está de acordo como planejado, utilize o comando `SELECT` para visualizar o conteúdo da nova tabela; para isso, basta digitar a seguinte instrução no prompt do MySQL:

```
mysql> SELECT * FROM Movimentacao;              [Enter]
```

Veja como será o resultado na tela do computador:

```
+--------+ +-----------+ +--------+ +-------+ +--------+ +---------------+ +---------+ +-------------+
| Codigo | | Banco     | | Conta  | | Tipo  | | Numero | | Data          | | Valor   | | Descricao   |
+--------+ +-----------+ +--------+ +-------+ +--------+ +---------------+ +---------+ +-------------+
| 1      | | Bradesco  | | 12444-8 | | Dep   | | 019    | | 2007-1-12     | | 1587.24 | | Pgto aluguel |
|        | |           | |        | |       | |        | | 00:00:00      | |         | |    sala     |
+--------+ +-----------+ +--------+ +-------+ +--------+ +---------------+ +---------+ +-------------+
1 row in set (0.02 sec)
```

Observando os resultados, podemos notar alguns detalhes:

- Os valores do tipo `CHAR` e `VARCHAR`, que digitamos entre aspas, são armazenados sem elas. Na verdade, as aspas são apenas um código que informa ao SQL que se trata de texto;
- A data sofreu uma alteração significativa desde quando a digitamos: as barras (/) foram substituídas por traços (-) e, visto que não especificamos uma hora, pois o tipo de campo `DATETIME` a prevê, o SQL introduziu a hora zero (00:00:00). Os três pares de zeros representam a hora no formato hh:mm:ss, ou seja, horas, minutos e segundos.

Neste ponto, ficou evidente uma escolha incorreta para o tipo de dados do campo Data: de fato, ao inserir a data de certa movimentação da conta bancária não é indispensável saber a hora exata. Mas isto não chega a ser um problema sério, pois podemos alterar a qualquer momento o tipo de dados, usando o comando `ALTER TABLE` ("alterar tabela", em português). Para o nosso exemplo, usaremos esse comando para alterar o tipo de dados do campo Data de `DATETIME` para `DATE` (apenas data, sem a hora). Veja:

No prompt do MySQL, digite essa linha de comando:

```
myslq> ALTER TABLE Movimentacao MODIFY Data DATE;
[Enter]
```

Ao teclarmos **Enter**, a seguinte mensagem é exibida:

```
Query OK, 1 row affected (0.22 sec)
Records: 1  Duplicates: 0  Warnings: 0
```

Isto significa que o nosso pedido deu certo (`Query OK`), que uma linha da tabela foi afetada (`1 row affected`), que o comando levou 0,22 segundos para ser executado (`0.22 sec`), que nossa tabela possui 1 registro (`Records: 1`), nenhum duplicado (`Duplicates: 0`) e nenhum aviso foi gerado (`Warnings: 0`).

Para conferir, vejamos primeiro a estrutura da tabela, mais especificamente a linha que se refere ao campo Data, e verificamos a alteração feita. Então, digite:

```
mysql> DESCRIBE Movimentacao;          [Enter]

+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| Codigo     | int(11)      | NO   | PRI | NULL    | auto _ increment |
| Banco      | char(10)     | YES  |     | NULL    |                |
| Conta      | char(10)     | YES  |     | NULL    |                |
| Tipo       | char(3)      | YES  |     | NULL    |                |
| Numero     | char(3       | YES  |     | NULL    |                |
| Data       | date         | YES  |     | NULL    |                |
| Valor      | decimal(8,2) | NO   |     |         |                |
| Descricao  | varchar(50)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
8 rows in set (0.02 sec)
```

Note como as características do campo Data foram alteradas de acordo com o especificado por meio do comando `ALTER TABLE`.

Mas o que aconteceu no conteúdo do campo Data? Antes apresentava a data e a hora, mesmo que esta última não tivesse sido digitada. E agora? Vamos visualizar o conteúdo da tabela para checar. Digite o enunciado:

```
mysql> SELECT * FROM Movimentacao;                    [Enter]

+--------+ +-----------+ +---------+ +-------+ +--------+ +--------------+ +----------+ +-------------+
| Codigo | | Banco     | | Conta   | | Tipo  | | Numero | | Data         | | Valor    | | Descricao   |
+--------+ +-----------+ +---------+ +-------+ +--------+ +--------------+ +----------+ +-------------+
| 1      | | Bradesco  | | 12444-8 | | Dep   | | 019    | | 2007-1-12    | | 1587.24  | | Pgto aluguel
|        | |           | |         | |       | |        | |              | |          | | sala        |
+--------+ +-----------+ +---------+ +-------+ +--------+ +--------------+ +----------+ +-------------+
1 row in set (0.02 sec)
```

Como podemos observar no exemplo anterior, o conteúdo do campo Data foi alterado, pois não apresenta mais a hora.

O comando `ALTER TABLE` é uma instrução muito poderosa, que no dialeto do MySQL prevê o uso de várias cláusulas que não fazem parte do padrão SQL99 ANSI/ISO; a cláusula `MODIFY`, por exemplo, existe somente no MySQL e em outro SGBDR muito conhecido e usado, o Oracle.

Inserindo novos dados na tabela

Até o momento, criamos uma tabela que contém os dados relativos à movimentação de uma conta bancária. Durante os procedimentos realizados nos exemplos anteriores, inserimos um registro nessa tabela, o único até agora. Então, vamos adicionar novos dados, usando mais uma vez o comando `INSERT`, porém, desta vez, com uma sintaxe um pouco mais articulada e complexa. Como sem-

pre, começaremos com um exemplo prático, portanto, no prompt do MySQL, digite o enunciado apresentado logo a seguir:

```
mysql> INSERT Movimentacao (Banco, Conta, Tipo, Numero,
Data, Valor, Descricao) VALUES ("HSBC", "31254-8", "Saq",
"099", "2007/02/04", -415, "Saque");
[Enter]
```

Em seguida, usamos o comando SELECT para visualizar o conteúdo da tabela e verificar como os dados foram inseridos:

```
mysql> SELECT * FROM Movimentacao
```

... e obtemos o seguinte resultado:

```
+--------+ +-----------+ +--------+ +-------+ +--------+ +--------------+ +---------+ +------------------+
| Código | | Banco     | | Conta  | | Tipo  | | Numero | | Data         | | Valor   | | Descricao        |
+--------+ +-----------+ +--------+ +-------+ +--------+ +--------------+ +---------+ +------------------+
| 1      | | Bradesco  | | 12444-8 | | Dep   | | 019    | | 2007-01-12   | | 1587,24 | | Pgto aluguel sala
| 2      | | HSBC      | | 31254-8 | | Saq   | | 099    | | 2007-02-04   | | -415.00 | | Saque
+--------+ +-----------+ +--------+ +-------+ +--------+ +--------------+ +---------+ +------------------+
2 rows in set (0.03 sec)
```

Veja que toda a tabela está preenchida nos dois registros relativos a duas operações bancárias (obviamente fictícias).

Olhe atentamente para o enunciado do comando INSERT que usamos para inserir os novos dados: em relação aos exemplos anteriores, neste caso usamos uma sintaxe diferente, ou seja:

```
INSERT nome_da_tabela (campo1, campo2, campo3,...) VALUES
(valor_campo1, valor_campo2, valor_campo3,...);
```

Em outras palavras, além de informar o nome da tabela a ser preenchida, especificamos entre parênteses quais campos daquela tabela deveriam receber os dados (separando-os com vírgulas) e, em seguida, o valor para cada um dos campos (também separados por vírgulas). Note que no nosso exemplo não informamos um valor para o campo Codigo, mas mesmo assim ele foi preenchido com o número 2. Isto ocorreu porque este campo foi configurado com a opção AUTO_INCREMENT, assim, uma vez definido o valor para o primeiro registro, todos os demais serão preenchidos automaticamente incrementando de 1 o valor do registro anterior.

Para entendermos melhor o funcionamento do sistema de incremento automático, vamos realizar algumas alterações na nossa tabela: primeiramente excluiremos o segundo registro (aquele que

inserimos por último), e depois inseriremos novos dados com a ajuda do comando INSERT. Então...

1. No prompt do MySQL, digite esta linha de comando:

```
mysql> DELETE FROM Movimentacao WHERE Codigo=2;[Enter]
```

Em outras palavras, estamos pedindo para excluir (delete) da tabela Movimentacao (from Movimentacao) o registro em que o campo Codigo apresenta o valor 2 (where Codigo=2).

2. O MySQL informará que o comando foi bem-sucedido e que uma linha foi afetada. Agora, insira novamente os mesmo dados, usando o enunciado:

```
mysql> INSERT Movimentacao (Banco, Conta, Tipo, Numero,
Data, Valor, Descricao) VALUES ("HSBC", "31254-8", "Saq",
"099", "2007/02/04", -415, "Saque");
[Enter]
```

3. Agora, use o comando SELECT para visualizar o conteúdo da tabela:

```
mysql> SELECT * FROM Movimentacao
+--------+-----------+---------+-------+--------+---------------+----------+-------------------+
| Codigo | Banco     | Conta   | Tipo  | Numero | Data          | Valor    | Descricao         |
+--------+-----------+---------+-------+--------+---------------+----------+-------------------+
| 1      | Bradesco  | 12444-8 | Dep   | 019    | 2007-01-12    | 1587.24  | Pgto aluguel sala |
| 3      | HSBC      | 31254-8 | Saq   | 099    | 2007-02-04    | -415.00  | Saque             |
+--------+-----------+---------+-------+--------+---------------+----------+-------------------+
2 rows in set (0.03 sec)
```

Observando o resultado, percebemos que o código do segundo registro agora é 3 e não mais 2. Mas, por quê?

É simples. O valor de um campo definido como chave primária (PRIMARY KEY) e AUTO_INCREMENT, é obtido incrementando de uma unidade o valor máximo usado para aquele campo no momento da última inserção de dados, mesmo que o registro que usou o valor máximo não exista mais. No nosso exemplo, excluímos o registro cujo código era 2, e depois inserimos outro (com os mesmos dados, mas isso é irrelevante). Poderíamos pensar que o código do novo registro seria 2 (o código do registro anterior mais 1), porém, tratando-se de uma chave primária, o novo código não pode ser igual a um já usado, mesmo que este tenha sido apagado da tabela. Assim, o AUTO_INCREMENT usa como base o último código utilizado (in-

dependentemente se ainda está na tabela ou não) e cria o próximo somando 1. Por isso, o código do nosso novo registro é 3, e não 2.

Em uma tabela, pode haver apenas um campo com a característica `AUTO _ INCREMENT` e deve ser uma chave (externa ou primária).

Configurando chaves primárias compostas

Podem ocorrer situações em que as informações que precisamos gerenciar fazendo uso de uma tabela contenham elementos que, se combinados, identifiquem um registro de maneira unívoca, constituindo assim uma chave primária composta, ou seja, composta por mais de um campo e não por um só. Em casos como este, torna-se desnecessário criar um campo separado para servir de chave primária, basta indicar ao SQL quais são os campos que, considerados em conjunto, formam a chave primária.

Por exemplo, imagine uma tabela na qual armazenamos os projetos em andamento na nossa empresa estruturada da seguinte forma: cada tipo de projeto possui uma sigla predefinida (DE = Desenvolvimento, TR = Treinamento e DI = Divulgação) e, para cada projeto, há um supervisor responsável, identificado por um código numérico. Associando a sigla do projeto com o código do supervisor responsável, é possível identificar cada projeto de maneira unívoca. Além desses dados, nossa tabela conterá também a data de início e de fim de cada projeto.

Para compreender melhor, vamos criar essa tabela. Então, no prompt do MySQL, digite o enunciado a seguir, que usa o comando `CREATE TABLE` para criar a tabela Projetos e algumas cláusulas que definirão sua estrutura:

```
mysql> CREATE TABLE Projetos (Tipo CHAR(2) NOT NULL, Cod _
Supervisor INT NOT NULL, PRIMARY KEY (Tipo, Cod _ Supervi-
sor), Descricao VARCHAR(30), Data _ Inicio DATE, Data _ Fim
DATE);              [Enter]
```

Nossa tabela foi criada. Como podemos notar, neste exemplo usamos a cláusula `PRIMARY KEY` de maneira bem diferente da tabela Movimentacao. Naquele exemplo, a cláusula foi usada para especificar que um determinado campo (Codigo) era a chave primária da tabela. Já neste caso, a cláusula `PRIMARY KEY` é usada não como

atributo de um campo, mas como comando separado, que usa como parâmetros (entre parênteses) os nomes dos campos que, em conjunto, compõem a chave primária da tabela. Para que isso funcione, é imprescindível que os campos das quais a chave primária será composta sejam definidos como `NOT NULL`, ou seja, estejam configurados para não aceitar o valor vazio, pois a chave primária deve existir e não pode ser omissa.

Vamos usar o comando `DESCRIBE` para ver como ficou a estrutura da nossa nova tabela:

```
mysql> DESCRIBE Projetos;                    [Enter]

+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| Tipo          | char(2)     | NO   | PRI |         |                |
| Cod_Supervisor| int(11)     | NO   | PRI |         |                |
| Descricao     | varchar(30) | YES  |     | NULL    |                |
| Data_Inicio   | date        | YES  |     | NULL    |                |
| Data_Inicio   | date        | YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
5 rows in set (0.02 sec)
```

Repare que os primeiros dois campos (Tipo e Cod_Supervisor) apresentam a característica `NO` na coluna Null, o que indica que nenhum dos dois aceita valores nulos; além disso, ambos são definidos como PRI na coluna Key, ou seja, os dois campos formam a chave primária desta tabela.

A cláusula `PRIMARY KEY` pertence ao padrão SQL99 ANSI/ISO e encontra-se disponível em todos os dialetos do SQL no mercado. Todavia, a sintaxe que mostramos nos exemplos anteriores se refere ao seu uso em ambiente MySQL.

Tabelas e índices

No jargão do SQL, o termo chave ou key é sinônimo de índice e se refere a uma ferramenta interna do SQL. Um índice pode ser criado e utilizado para agilizar as operações de seleção dos dados contidos nas tabelas. Neste tópico, veremos como criar e fazer proveito dos índices; para isso, voltaremos a usar a tabela Movimentacao, criada e modificada no decorrer dos dois tópicos iniciais deste capítulo.

A tabela em questão possui um campo do tipo DATE, chamado Data, cuja finalidade é armazenar a data de cada movimentação da conta bancária. Vamos criar um índice neste campo usando mais

uma vez o comando ALTER TABLE, porém, com algumas variações em sua sintaxe.

1. Para criar um índice no campo Data da tabela Movimentacao, digite a linha de comando a seguir:

```
mysql> ALTER TABLE Movimentacao ADD INDEX Data (Data);
[Enter]
```

2. Após executar o comando, visualize a estrutura da tabela utilizando o comando DESCRIBE. O resultado será como mostrado a seguir:

```
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| Codigo     | int(11)     | NO   | PRI | NULL    | auto_increment |
| Banco      | char(10)    | YES  |     | NULL    |                |
| Conta      | char(10)    | YES  |     | NULL    |                |
| Tipo       | char(3)     | YES  |     | NULL    |                |
| Numero     | char(3      | YES  |     | NULL    |                |
| Data       | date        | YES  | MUL | NULL    |                |
| Valor      | decimal(8,2)| NO   |     |         |                |
| Descricao  | varchar(50) | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
8 rows in set (0.01 sec)
```

Note que, em correspondência do campo Data, a coluna Key foi preenchida com o valor MUL, que nos informa que neste campo foi definido um índice. MUL é abreviação de MULTIPLE ("múltiplo", em português) e se refere à característica de um índice em aceitar valores duplicados, como veremos mais adiante neste tópico.

No enunciado anterior, o comando ALTER TABLE foi completado com cláusula ADD INDEX (adicionar índice), cuja sintaxe é a seguinte:

```
ADD INDEX [nome_do_índice] (coluna_da_tabela,...);
```

O parâmetro nome_do_índice não é obrigatório, já coluna_da_tabela é o nome da coluna da tabela (mas pode ser mais de uma, como veremos) na qual se deseja criar o índice e deve ser especificado entre aspas.

É possível também construir um índice em mais colunas; para isso, basta especificar seus nomes entre parênteses como parâmetro da cláusula ADD INDEX, separando os nomes das colunas com vírgulas. Por exemplo, em uma tabela com os campos Nome e Sobrenome

(como a tabela Cadastro, criada no **Capítulo 2**), podemos configurar o índice em ambos os campos, com a seguinte linha de comando:

```
mysql> ALTER TABLE Cadastro ADD INDEX NomeCompleto (Nome,
Sobrenome);  [Enter]
```

Há também uma outra maneira de criar índices em tabelas, usando um comando e não uma cláusula. Trata-se da instrução `CREATE INDEX` ("criar índice", em português), cuja sintaxe é a seguinte:

```
CREATE INDEX nome_do_índice ON nome_da_tabela (coluna_da_tabela,...);
```

Isto significa que, após digitar o comando, deveremos especificar como parâmetros o nome do índice, a palavra-chave `ON` e, em seguida, a tabela em que o índice deve ser criado e, entre parênteses, os nomes das colunas nas quais o índice será construído com os dados separados por vírgulas. Para exemplificar, a mesma operação do enunciado anterior pode ser realizada usando a seguinte linha de comando:

```
mysql> CREATE INDEX NomeCompleto ON Cadastro (Nome, Sobrenome); [Enter]
```

Uma tabela pode conter até 16 índices e um índice não pode ultrapassar 256 caracteres.

Vimos, então, como criar um índice; mas por que fazê-lo?

Os índices não são indispensáveis para o funcionamento de uma tabela, por isso, seu uso é facultativo, porém, eles ajudam muito nas operações de consulta dos dados armazenados.

Quando realizamos uma consulta em SQL (o que é feito usando o comando `SELECT` seguido por um critério especificado após a cláusula `WHERE`, como vimos no tópico **Visualizando o conteúdo de uma tabela** do **Capítulo 2**), o software analisa todas as tabelas do banco de dados, lendo em seqüência todos os registros e todos os campos, para encontrar aqueles que satisfazem o critério de busca especificado.

Se a tabela contém um índice e a seleção prevê uma comparação no campo indexado, o processo de leitura dos dados das tabelas acontece de maneira muito mais rápida, pois o SQL se posiciona

diretamente no primeiro registro que contém um valor que atende ao critério definido. Para exemplificar, suponhamos que desejamos extrair da nossa tabela Movimentacao todas as operações realizadas a partir do dia 15 de abril de 2006, visualizando apenas os campos Codigo, Valor, Data e Descricao. Poderíamos fazer isto com o seguinte enunciado:

```
mysql>  SELECT Codigo, Valor, Data, Descricao FROM Movimen-
tacao WHERE Data > "2006-04-14";          [Enter]
```

Como na nossa tabela o campo Data foi definido como índice, a busca é realizada rapidamente, porque o índice permite ao SQL apontar imediatamente para o primeiro registro do campo Data cujo valor seja maior que 2006-04-14 e extrair todos os registros sucessivos sem ter que ler o conteúdo dos registros anteriores.

Obviamente, este ganho em termos de velocidade de busca é sensível em tabelas com milhares registros, enquanto é imperceptível em tabelas com poucos dados.

Em termos estruturais, um índice é, na verdade, uma pequena tabela composta por dois campos: o primeiro contém um valor gerado pelo próprio SQL, que funciona como um código de identificação para cada registro da tabela indexada; o segundo, contém o valor do campo no qual foi definido o índice. O mecanismo de busca desencadeado pelo comando `SELECT` verifica todos os campos de cada registro do índice e, quando encontra um valor correspondente na tabela, utiliza o primeiro campo para se posicionar no respectivo registro da tabela indexada.

Em função disso, a criação de um índice acrescenta no nosso banco de dados uma tabela, cujo tamanho em bytes é muito pequeno, o que geralmente não compromete o desempenho da base de dados.

É importante considerar também que quando alteramos uma tabela, adicionando ou excluindo registros, os índices que porventura foram definidos naquela tabela ficarão desatualizados e, portanto, deverão ser gerados novamente, o que torna mais complexa e demorada a atualização dos dados da tabela.

Por essas duas razões (aumento do tamanho do banco de dados e demora na atualização), é recomendável criar índices somente quando sua utilização traz ganhos significativos de tempo e produtividade, caso contrário, esse recurso irá ser uma pedra no sapato e não uma mão na roda.

Uma chave primária também é um índice, com a característica de possuir um valor único para cada registro da tabela. Ao contrário, os outros índices podem ter valores duplicados; por exemplo, em um índice num campo Estado de uma tabela de endereços, é possível que valores como DF, GO, RJ, SP se repitam mais de uma vez ao longo da tabela, porém, se referindo a pessoas ou locais diferentes.

Valores predefinidos para os campos (Default)

No decorrer dos capítulos e tópicos anteriores, empregamos várias vezes o comando `DESCRIBE` para obter a visualização da estrutura de uma tabela, ou seja, dos seus campos, tipos de dados, chave primária etc.

Vimos que a coluna Default da representação da estrutura da tabela mostra o valor predefinido para cada campo, ou seja, o valor que o campo tem por padrão antes de ser preenchido. Isto quer dizer que se preenchermos apenas alguns campos de um registro, e deixarmos os outros sem valor, o SQL irá inserir um valor padrão nos campos não preenchidos. Para os campos que contêm textos (tipos `CHAR` e `VARCHAR`) o valor predefinido é `NULL` (nenhum), e para os campos numéricos não há valor predefinido.

Para visualizarmos isto na prática, vamos inserir um novo registro na tabela Movimentacao, porém, preenchendo apenas alguns campos. Então, digite o enunciado mostrado a seguir:

```
mysql>  INSERT Movimentacao (Banco, Conta) VALUES ("Caixa",
"3211-5");            [Enter]
```

Receberemos a seguinte mensagem de erro:

```
ERROR 1364 (HY000): Field 'Valor' doesn't have a default value
```

O MySQL está nos avisando (em inglês, obviamente) que o campo Valor não possui valor predefinido. O que não é especificado claramente na mensagem é que a inclusão do novo registro não foi realizada porque, ao criarmos esta tabela, determinamos que o campo Valor é do tipo `NOT NULL`, ou seja, não pode estar vazio. Assim, por não termos inserido nenhum valor neste campo e por não haver valor predefinido para ele, o SQL não concluiu a operação de adição de registro.

Então, vamos alterar as propriedades do campo Valor da tabela Movimentacao, especificando o valor zero (0) como default (padrão) para ele. Isto é feito com o enunciado mostrado a seguir:

```
mysql> ALTER TABLE Movimentacao MODFY Valor DECIMAL(8,2) NOT
NULL DEFAULT 0;              [Enter]
```

Veja que a cláusula `MODIFY` (modificar) foi usada para alterar as características do campo Valor. No nosso exemplo, mantivemos o tipo de dados, ou seja decimal com oito valores antes da vírgula e duas casas decimais, e a característica not null (não vazio), mas definimos como padrão (`DEFAULT`) o valor zero, sem aspas por se tratar de um valor numérico e não de texto.

Visualizando a estrutura da tabela com a ajuda do comando `DESCRIBE`, a alteração feita fica visível:

```
+------------+--------------+------+-----+---------+----------------+
| Field      | Type         | Null | Key | Default | Extra          |
+------------+--------------+------+-----+---------+----------------+
| Codigo     | int(11)      | NO   | PRI | NULL    | auto _ increment |
| Banco      | char(10)     | YES  |     | NULL    |                |
| Conta      | char(10)     | YES  |     | NULL    |                |
| Tipo       | char(3)      | YES  |     | NULL    |                |
| Numero     | char(3       | YES  |     | NULL    |                |
| Data       | date         | YES  | MUL | NULL    |                |
| Valor      | decimal(8,2) | NO   |     | 0,00    |                |
| Descricao  | varchar(50)  | YES  |     | NULL    |                |
+------------+--------------+------+-----+---------+----------------+
```

Tentemos inserir novamente os dados do exemplo:

```
mysql>  INSERT Movimentacao (Banco, Conta) VALUES ("Caixa",
"3211-5");           [Enter]
```

Agora recebemos a mensagem de confirmação (`Query OK,...`), portanto, tudo correu como deveria. Para confirmar, vamos visualizar o conteúdo da nossa tabela usando o comando `SELECT`:

```
mysql>  SELECT * FROM Movimentacao;
[Enter]
```

Vejamos como ficou preenchida a nossa tabela:

```
+--------+ +-----------+ +--------+ +-------+ +--------+ +-------------+ +---------+ +------------------+
| Codigo | | Banco     | | Conta  | | Tipo  | | Numero | | Data        | | Valor   | | Descricao        |
+--------+ +-----------+ +--------+ +-------+ +--------+ +-------------+ +---------+ +------------------+
| 1      | | Bradesco  | | 12444-8 | | Dep   | | 019    | | 2007-01-12  | | 1587.24 | | Pgto aluguel sala |
| 3      | | HSBC      | | 31254-8 | | Saq   | | 099    | | 2007-02-04  | | -415.00 | | Saque            |
| 4      | | Caixa     | | 3211-5 | | NULL  | | NULL   | | NULL        | |    0.00 | | NULL             |
+--------+ +-----------+ +--------+ +-------+ +--------+ +-------------+ +---------+ +------------------+
3 rows in set (0.03 sec)
```

Como podemos notar, em todos os campos para os quais não definimos o conteúdo, o SQL inseriu o valor predefinido que é, respectivamente, Null para os campos Tipo, Numero, Data e Descricao e 0.00 para o campo Valor, como especificamos anteriormente.

O comando ALTER TABLE prevê o uso de uma cláusula chamada SET DEFAULT, muito mais prática do que cláusula MODIFY (que usamos no exemplo anterior). A sintaxe a ser usada é a seguinte:

```
ALTER TABLE nome _ da _ tabela ALTER [COLUMN] coluna _ da _ ta-
bela {SET DEFAULT valor | DROP DEFAULT};
```

... em que a palavra COLUMN é facultativa, o argumento coluna_da_tabela é obrigatório e deve ser seguido por uma das duas cláusulas SET DEFAULT (para definir o novo valor predefinido) ou DROP DEFAULT (para apagar o valor predefinido já existente). Após a cláusula SET DEFAULT, devemos especificar o valor que desejamos configurar como padrão para a coluna; já a cláusula DROP DEFAULT não necessita de nenhum argumento.

Digamos, por exemplo, que desejamos definir para o campo Numero um novo valor padrão, que agora é NULL, alterando-o para 000. Usando a cláusula SET DEFAULT, o enunciado ficaria assim:

```
mysql>  ALTER TABLE Movimentacao ALTER COLUMN Numero SET
DEFAULT "000";              [Enter]
```

Se desejar, use o comando DESCRIBE para verificar se a estrutura da tabela foi alterada como pedimos.

Modificando as tabelas

As tabelas de um banco de dados podem ser alteradas tanto em relação ao seu conteúdo, adicionando e excluindo registros, quanto no que diz respeito à estrutura, inserindo e retirando colunas, criando e removendo índices, alterando o tipo de dados, e assim por diante.

As operações de inserção e remoção de registros não alteram a estrutura da tabela e fazem parte do conjunto de operações chamado *Data Manipulation Language* (DML) do SQL. Todas as modificações feitas na estrutura são realizadas usando os comandos ALTER TABLE e DROP; o primeiro prevê o uso de inúmeras cláusulas, opções e parâmetros, já o segundo é muito mais simples e direto.

Para a exclusão de uma tabela, o SQL disponibiliza a seguinte sintaxe:

```
DROP TABLE nome_da_tabela RESTRICT | CASCADE;
```

A cláusula `RESTRICT` inibe a execução do comando caso existam relacionamentos entre a tabela que se deseja excluir e outros elementos do banco de dados; a cláusula `CASCADE` faz com que seja excluída não só a tabela especificada, mas também todos os elementos aos quais ela esteja ligada. Não todos os dialetos do SQL respeitam este padrão, por exemplo, o MySQL, que usamos neste livro, não prevê o uso das cláusulas `RESTRICT` ou `CASCADE`, sendo a sintaxe do comando `DROP TABLE` bem mais simples:

```
DROP [TEMPORARY] TABLE [IF EXISTS] nome_da_tabela1 [,
nome_da_tabela2,...];
```

A cláusula `IF EXISTS` é usada para prevenir mensagens de erro derivadas da digitação de nomes de tabela errados ou se tentarmos excluir uma tabela que não existe. A opção `TEMPORARY` limita a exclusão às tabelas temporárias. A variação mais significativa do MySQL em relação a outros dialetos SQL é que, nesse ambiente, é possível excluir de uma só vez várias tabelas, especificando nos nomes das tabelas a serem excluídas um após o outro, separando-os por vírgulas.

As tabelas no MySQL

Nesta versão do MySQL, para cada tabela são criados três arquivos físicos em disco, que se encontram na subpasta com o nome do banco de dados dentro da subpasta Data na pasta especificada na instalação do MySQL.

A nossa tabela Movimentacao, por exemplo, é gravada em disco sob forma dos três arquivos a seguir:

- **Movimentacao.frm** – este arquivo contém informações sobre o formato da tabela;
- **Movimentacao.myd** – é o arquivo no qual se encontram os dados contidos na tabela;
- **Movimentacao.myi** – trata-se do arquivo em que o MySQL armazena os índices criados na tabela.

Quando excluímos uma tabela usando o comando DROP TABLE, todos os arquivos que formam a tabela são apagados do disco rígido do computador.

Se quisermos apagar apenas o índice, mantendo a tabela, dispomos do comando DROP INDEX, que no MySQL apresenta esta sintaxe:

```
DROP INDEX nome _ do _ indice ON nome _ da _ tabela;
```

O mesmo comando está disponível sob forma de cláusula do comando ALTER TABLE e deve ser usado da seguinte maneira:

```
ALTER TABLE nome _ da _ tabela DROP INDEX nome _ do _ indice
```

Usando mais uma vez o exemplo da tabela Movimentacao, na qual criamos um índice para o campo Data, para excluir esse índice poderíamos usar um desses dois enunciados:

```
mysql>  DROP INDEX Data ON Movimentacao;       [Enter]
```

ou

```
mysql>  ALTER TABLE Movimentacao DROP INDEX Data;
[Enter]
```

Em outras palavras, podemos dizer que a palavra-chave DROP serve para excluir um elemento qualquer do banco de dados, que pode ser o próprio banco de dados, uma tabela, um índice etc.

A palavra-chave ADD é o inverso de DROP e é usada como cláusula do comando ALTER TABLE para adicionar algo. Após criarmos uma tabela e inserido registros, pode ser necessário, por exemplo, adicionar um novo campo, do qual esquecemos na hora de criar a tabela. Para adicionar um campo a uma tabela já existente, basta usar o comando ALTER TABLE com a cláusula ADD, seguida pelo nome do campo e o tipo de dados do mesmo. Por exemplo, se desejássemos adicionar o campo Agencia à tabela Movimentacao, o enunciado a ser digitado seria o seguinte:

```
mysql> ALTER TABLE Movimentacao ADD Agencia TEXT;
[Enter]
```

Se, durante o uso da tabela, percebemos que há um ou mais campos desnecessários, podemos apagá-los usando a cláusula DROP. Por exemplo, se fosse preciso excluir da tabela um campo de nome Observacoes da tabela Movimentacao, a linha de comando ficaria assim:

```
mysql>  ALTER TABLE Movimentacao DROP Observacoes;
[Enter]
```

Podemos também alterar o nome de uma tabela. Trata-se de uma ação importante quando, por exemplo, percebemos ter cometido um erro na definição do nome da tabela ao criá-la. Isto é feito usando o comando ALTER TABLE com a cláusula RENAME. Para renomear a tabela Teste para Primeiro_Teste, bastaria usar o enunciado a seguir:

```
mysql> ALTER TABLE Teste RENAME AS Primeiro _ Teste;
[Enter]
```

Tipos de dados especiais

No exemplo do tópico anterior acrescentamos à nossa tabela uma nova coluna do tipo TEXT. Esse tipo de dados pertence a um grupo de tipos chamados BLOB (acrônimo de *Bynary Large Object* – objetos binários de tamanho grande). Normalmente, esse tipo é usado para arquivos gráficos (desenhos ou imagens) ou sons. São considerados BLOBs também os arquivos de texto muito extensos, que podem ser associados a uma coluna da tabela. No caso do MySQL, dispomos dos tipos TINYTEXT, TEXT, MEDIUMTEXT e LONGTEXT para os documentos de texto e TINYBLOB, BLOB, MEDIUMBLOB e LARGEBLOB para objetos (arquivos) que representam imagens, sons, desenhos etc. A divisão é feita de acordo com o tamanho: em inglês, tiny significa "pequeno", medium significa "médio" e long ou large está por "grande"; assim, por exemplo, tinytext ou tinyblob são usados, respectivamente, para textos e objetos de tamanho pequeno.

Mas como quantificar o que é pequeno, médio ou grande em se tratando de arquivos? Existe uma tabela que traz esses valores, apresentada logo a seguir:

Tipo	Tamanho máximo
TinyBlob / TinyText	Até 2^8 caracteres
Blob / Text	Até 2^{16} caracteres
MediumBlob / MediumText	Até 2^{24} caracteres
LargeBlob / LongText	Até 2^{32} caracteres

Tabela 3.1: Representação de tamanho de dados.

Considerando que, por exemplo, 2^{16} é equivalente a 65.536, escolhendo o tipo `TEXT`, temos à disposição um espaço razoavelmente amplo para inserir textos.

Um outro tipo de dados peculiar é `ENUM`, usado para limitar a entrada de dados de um campo a determinadas opções. De volta ao exemplo da tabela Movimentacao, suponhamos que o campo Tipo possa conter apenas uma das seguintes opções: Dep para depósito, Saq para saque, Tra para Transferência e Pgt para pagamento, não poderá ficar vazio nem aceitar qualquer outro valor além dos citados. Podemos fazer isso com o enunciado a seguir:

```
Tipo ENUM("Dep", "Saq", "Tra", "Pgt") NOT NULL;
```

Se inserirmos no campo Tipo um valor diferente dos especificados, o MySQL o desconsiderará e, em seu lugar, inserirá um zero, sem exibir qualquer tipo de mensagem de erro. Assim, uma busca por valores zero no campo Tipo ficaria fácil encontrar todos os registros da tabela em que o campo Tipo apresenta um valor incorreto.

O tipo de dados `ENUM` pertence ao padrão SQL99 ANSI/ISO e, portanto, está disponível em todos os dialetos do SQL.

Capítulo 4

Inserindo e modificando dados

Introdução

Nos capítulos anteriores aprendemos a criar e excluir tabelas em um banco de dados, bem como visualizar e alterar sua estrutura e seu conteúdo, com a ajuda do grupo de comandos chamado *Data Definition Language* (DDL) – Linguagem de Definição de Dados.

Obviamente, uma tabela em si não é de utilidade nenhuma se não contém dados. Para isso, existe um grupo de comandos pelo qual é possível inserir, modificar e excluir dados de uma tabela. Trata-se dos comandos `INSERT`, `UPDATE` e `DELETE`, que, juntamente ao comando `SELECT`, com o qual extraímos do banco de dados as informações desejadas, formam o componente *Data Manipulation Language* (DML) – Linguagem de Manipulação de Dados.

Em exemplos anteriores, já utilizamos os comandos `SELECT` e `INSERT`. Neste capítulo, iremos aprofundar suas sintaxes e seu uso, utilizando também cláusulas exclusivas do dialeto MySQL e que não se encontram no padrão SQL99, que podem ser extremamente úteis para inserir com rapidez grandes quantidades de dados.

Sobre o comando `INSERT`

O comando `INSERT` permite inserir dados de duas maneiras diferentes: a primeira, que usaremos nos nossos exemplos, consiste em especificar uma lista de valores a serem inseridos em cada uma das colunas (campos) da tabela; a segunda, um pouco mais complexa, consiste em inserir em uma tabela um conjunto de dados resultado de uma busca em outras tabelas por meio do comando `SELECT`.

O esquema sintático do comando `INSERT` é basicamente este:

```
INSERT [INTO] [nome_do_bancodedados. [proprietário.]
{nome_da_tabela | nome da view} [(coluna1, coluna2,...)]
{ [DEFAULT] VALUES | VALUES (valor1, valor2,...)] enunciado_SELECT}
```

> **Observação:** para entender o significado da sintaxe deste e de outros comandos, veja o tópico **Introdução**, no **Capítulo 2** deste livro.

Analisando a sintaxe, notamos que:

- A palavra-chave `INTO` é facultativa;

- Opcionalmente, podemos indicar, antes do nome da tabela, o respectivo proprietário e o banco de dados à qual pertence. Esta opção se torna extremamente útil quando temos dois ou mais bancos de dados contendo tabelas com o mesmo nome e queremos ter certeza de que o comando seja aplicado à tabela que desejamos;
- Os parâmetros `coluna1, coluna2,...` referem-se à lista de colunas da tabela que irão receber os dados especificados em seguida. Omitindo esse parâmetro, o SQL agirá em todas as colunas da tabela;
- Podemos especificar os valores a serem inseridos (`VALUES`) ou um enunciado com o comando `SELECT` que retornará valores a serem armazenados;
- Os parâmetros `valor1, valor2,...` referem-se aos valores (digitados pelo usuário) que deverão ser armazenados nos campos da tabela.

Quando especificamos o conteúdo dos campos com a cláusula `VALUES`, seguida pelos valores entre parênteses e separados por vírgulas, o comando `INSERT` permite a inserção de um registro por vez; já se utilizarmos como cláusula um enunciado com o comando `SELECT`, todos os registros extraídos serão adicionados à tabela de uma só vez.

Inserindo novos registros na tabela

Vejamos como o comando `INSERT` funciona, recorrendo a alguns exemplos práticos. Para isso, precisaremos criar uma nova tabela que conterá dados relativos a algumas pessoas. Esses dados serão: um código (que usaremos como chave primária), o nome, o sobrenome e a data de nascimento, portanto, será articulada em quatro colunas. Chamaremos essa tabela de Pessoas.

Então comece:

1. Para criar a tabela, use o enunciado mostrado a seguir:

```
mysql> CREATE TABLE Pessoas (Codigo INT NOT NULL PRIMARY KEY
AUTO _ INCREMENT, Nome VARCHAR(45), Sobrenome(55), DataNasci-
mento DATE);                [Enter]
```

2. Se tudo correu bem, ou seja, se nenhuma mensagem de erro foi retornada, use o comando DESCRIBE para obter a visualização da estrutura da tabela recém-criada, que deve ser como esta:

```
mysql> DESCRIBE pessoas;          [Enter]
+---------------+-------------+------+-----+---------+----------------+
| Field         | Type        | Null | Key | Default | Extra          |
+---------------+-------------+------+-----+---------+----------------+
| Codigo        | int(11)     | NO   | PRI | NULL    | auto_increment |
| Nome          | varchar(45) | YES  |     | NULL    |                |
| Sobrenome     | varchar(55) | YES  |     | NULL    |                |
| DataNascimento | date       | YES  |     | NULL    |                |
+---------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)
```

3. Vamos agora inserir o primeiro registro na nova tabela que, até o momento, é apenas uma caixa vazia. Então, digite:

```
mysql> INSERT Pessoas (Codigo, Nome, Sobrenome, DataNascimento) VALUES (1, "Fernando Henrique", "Dos Reis Couto", "1989/04/05");        [Enter]
```

Desta forma, cada uma das colunas especificadas entre parênteses na primeira parte do enunciado (Nome, Sobrenome etc.) deve necessariamente corresponder a um valor na lista que segue a cláusula VALUES, caso contrário, seria exibida uma mensagem de erro e nenhum dos dados especificados seria inserido.

4. Para verificarmos que os dados foram inseridos corretamente, visualize o conteúdo da tabela, usando o comando:

```
mysql>  SELECT * FROM Pessoas;            [Enter]
+---------+------------------+---------------+----------------+
| Codigo  | Nome             | Sobrenome     | DataNascimento |
+---------+------------------+---------------+----------------+
| 1       | Fernando Henrique | Dos Reis Couto | 1989-04-05     |
+---------+------------------+---------------+----------------+
1 row in set (0.03 sec)
```

5. Como podemos notar, cada coluna foi preenchida com o respectivo dado. Vamos inserir um outro registro, desta vez com uma sintaxe mais simples, sem especificar as colunas, e sim, apenas os dados a serem armazenados. Use o enunciado mostrado a seguir:

```
mysql> INSERT Pessoas VALUES (2, "Fabrizio", "Vésica", "1976/04/15");                    [Enter]
```

6. Visualizando o conteúdo da tabela, obteremos o seguinte:

```
mysql>  SELECT * FROM Pessoas;              [Enter]
+---------+------------------+---------------+----------------+
| Codigo  | Nome             | Sobrenome     | DataNascimento |
+---------+------------------+---------------+----------------+
| 1       | Fernando Henrique | Dos Reis Couto | 1989-04-05     |
| 2       | Fabrizio          | Vésica         | 1976-04-15     |
+---------+------------------+---------------+----------------+
2 rows in set (0.03 sec)
```

Repare como mesmo não tendo informado as colunas a serem preenchidas, a inserção de dados foi bem-sucedida, pois a quantidade de valores especificado corresponde à quantidade de colunas da tabela, por isso, o MySQL não teve dúvidas sobre como preencher a tabela, inserindo os dados nas colunas exatamente na mesma ordem em que foram digitados no enunciado.

Vejamos agora mais uma maneira de usar o comando `INSERT`.

1. Digite o enunciado a seguir, no qual especificaremos apenas três colunas e os respectivos valores:

```
mysql> INSERT Pessoas (Nome, Sobrenome, DataNascimento) VALUES ("Augusto", "J. Vésica", "2001/01/12");
[Enter]
```

2. Os dados serão adicionados com sucesso, como podemos conferir, visualizando mais uma vez o conteúdo da tabela:

```
mysql>  SELECT * FROM Pessoas;              [Enter]
+---------+------------------+---------------+----------------+
| Codigo  | Nome             | Sobrenome     | DataNascimento |
+---------+------------------+---------------+----------------+
| 1       | Fernando Henrique | Dos Reis Couto | 1989-04-05     |
| 2       | Fabrizio          | Vésica         | 1976-04-15     |
| 3       | Augusto           | J. Vésica      | 2001-01-12     |
+---------+------------------+---------------+----------------+
3 rows in set (0.03 sec)
```

Embora não tenhamos informado nenhum valor para o código, este foi preenchido com o valor 3, por se tratar de um campo `AUTO _ INCREMENT`.

Atualizando registros da tabela

É normal que, durante o uso de uma tabela, se torne necessário alterar o conteúdo de um ou mais registros adicionados. Pense-

mos em uma tabela de cadastro contendo os nomes, endereços e números de telefone de algumas pessoas: é bem provável que, ao longo do tempo, alguém mude de endereço ou de telefone, o que torna necessária a atualização dos dados do cadastro. Já no caso de uma tabela de produtos, podem ocorrer variações nos preços ou na quantidade em estoque, dados esses que devem ser atualizados para manter em dia o banco de dados.

Para modificar o conteúdo de uma tabela, usamos o comando `UPDATE`, que faz parte do padrão SQL99 e, portanto, se encontra em todos os dialetos, mesmo que com algumas variações em relação às cláusulas e aos parâmetros previstos.

A sintaxe do comando `UPDATE` no MySQL é a seguinte:

```
UPDATE nome_da_tabela SET coluna = {DEFAULT | expressão},
{...} [WHERE condição];
```

Em sua versão mais simples, isto é, sem o uso da cláusula `WHERE`, o comando `UPDATE` atribui o valor indicado em expressão a todos os registros da coluna especificada no parâmetro coluna. Para exemplificar, veja o enunciado a seguir:

```
UPDATE Pessoas SET DataNascimento = "1946/06/26";
```

Se aplicássemos este comando à tabela Pessoas (que criamos no tópico anterior) para alterar a data de nascimento de uma das pessoas, todos os registros ficariam com o mesmo valor no campo DataNascimento, pois se não se especifica em qual registro da tabela a atualização deverá ser realizada, o comando `UPDATE` definirá o mesmo valor para todos os registros da coluna.

Para resolver este problema, é imprescindível o uso da cláusula `WHERE` (onde), que torna seletiva à aplicação do comando; de fato, com a ajuda desta cláusula, podemos informar ao SQL em qual registro queremos aplicar a alteração do campo DataNascimento, como mostrado no exemplo a seguir, no qual o enunciado altera a data de nascimento da pessoa cujo código é 2. Então apliquemos a seguinte instrução:

```
mysql> UPDATE Pessoas SET DataNascimento = "1946/06/26" WHERE
Codigo = 2;                [Enter]
```

Receberemos a seguinte mensagem de confirmação:

```
Rows matched: 1  Changed: 1  Warnings: 0
```

O que significa dizer que houve uma alteração em uma linha da tabela.

Visualizando novamente o conteúdo da tabela com o comando SELECT, podemos confirmar a alteração do registro referente à pessoa cujo código de identificação é 2:

```
mysql>  SELECT * FROM Pessoas;              [Enter]
+---------+------------------+---------------+----------------+
| Codigo  | Nome             | Sobrenome     | DataNascimento |
+---------+------------------+---------------+----------------+
| 1       | Fernando Henrique | Dos Reis Couto | 1989-04-05     |
| 2       | Fabrizio          | Vésica         | 1946-06-26     |
| 3       | Augusto           | J. Vésica      | 2001-01-12     |
+---------+------------------+---------------+----------------+
3 rows in set (0.03 sec)
```

A cláusula SET pode conter também uma ou mais expressões aritméticas que realizem cálculos com os valores de um determinado campo. Esta característica se demonstra útil em tabelas nas quais seja necessário fazer cálculos; por exemplo, imagine uma tabela de produtos na qual o campo Preco_Unitario deverá conter, ao invés do valor digitado, o mesmo valor, porém, com acréscimo de 10%. O enunciado do comando UPDATE ficaria assim:

```
UPDATE Pessoas SET Ppreco _ Unitario = Preço _ Unitario * 1.1
```

Em que o asterisco representa, neste caso, o símbolo de multiplicação.

Excluindo registros da tabela

A exclusão de registros da tabela é uma operação tão comum quanto a inserção. Para isso, o padrão SQL oferece o comando DELETE, cuja sintaxe no MySQL é bastante simples:

```
DELETE FROM nome _ da _ tabela [WHERE condição];
```

Como se percebe, a condição definida pela cláusula WHERE é opcional, porém, sem ela o comando DELETE apaga todos os registros da tabela, esvaziando-a, mas não apaga a tabela. Em outras palavras, se digitássemos o enunciado:

```
DELETE FROM Pessoas;
```

Todos os registros da tabela Pessoas seriam excluídos, porém, a tabela e sua estrutura permaneceriam intactas.

Se a nossa intenção é excluir apenas um registro da tabela, o uso da cláusula `WHERE` é indispensável para especificar qual registro (ou quais registros) desejamos apagar. A seguir, alguns exemplos:

- Para excluir da tabela Pessoas o registro da pessoa cujo código é 1, o enunciado seria o seguinte:

```
DELETE FROM Pessoas WHERE Codigo = 2;
```

- Se, em uma tabela Cadastro (com nomes, sobrenomes e endereços), quiséssemos excluir todos os registros que no campo Estado possuam o valor `TO`, deveríamos digitar uma linha de comando assim estruturada:

```
DELETE FROM Cadastro WHERE Estado = "TO";
```

Excluindo todos os registros da tabela de uma só vez

O SQL oferece outro comando para exclusão, chamado `TRUNCATE`. A diferença entre `DELETE` e `TRUNCATE` é que o primeiro, como vimos, permite uso seletivo, especificando quais registros desejamos excluir; já o segundo, é usado para esvaziar a tabela de uma só vez. A sua sintaxe é:

```
TRUNCATE TABLE nome _ da _ tabela;
```

Por exemplo, o enunciado...

```
TRUNCATE TABLE Pessoas;
```

... elimina irreversivelmente a tabela Pessoas para, em seguida, criá-la novamente com as mesmas características.

Capítulo 5

Realizando consultas no SQL

Introdução

O comando principal do SQL é, sem dúvida alguma, SELECT, que permite realizar consultas no banco de dados e extrair as informações desejadas. É justamente essa operação de consulta ("query" em inglês) que deu o nome à linguagem (*Structured Query Language*).

A sintaxe do comando SELECT é muito rica em opções já na versão padrão do SQL e, em cada dialeto, apresenta funcionalidades adicionais que a tornam ainda mais versátil e rápida para realizar buscas detalhadas no banco de dados.

A sintaxe do comando SELECT no padrão SQL é a seguinte:

```
SELECT [ALL | DISTINCT] lista _ de _ seleção [INTO nova _ tabela] FROM tabela _ original [,...] [JOIN condição] [WHERE condição] [GROUP BY expressão _ agrupamento] [HAVING condição _ busca] [ORDER BY critério [ASC | DESC] ];
```

Como podemos perceber observando o esquema sintático anterior, o comando SELECT contempla o uso de sete cláusulas, das quais apenas uma (FROM) é obrigatória. Para algumas cláusulas, existem parâmetros específicos expressos por palavras chave (ON, IN e AS). Além disso, é possível utilizar expressões com operadores aritméticos, lógicos, de concatenação, funções matemáticas e estatísticas.

No decorrer deste capítulo veremos o uso deste poderoso comando.

Uso simples do comando SELECT

Na sua forma mais simples, o comando SELECT, como já visto em alguns exemplos dos capítulos anteriores, é usado com a sintaxe apresentada a seguir:

```
SELECT lista _ de _ seleção FROM nome _ da _ tabela;
```

em que lista_de_seleção é uma lista das colunas a serem extraídas da tabela especificada no parâmetro nome_da_tabela da cláusula FROM. Quando se deseja extrair todas as colunas de uma tabela, usamos o caractere asterisco (*), que significa tudo ou todos. Assim, o enunciado:

```
SELECT * FROM Pessoas;
```

retorna todos os registros e todas as colunas da tabela Pessoas.

Se desejarmos visualizar apenas alguns campos da tabela, devemos especificar os nomes das colunas, separando-os por vírgulas. Por exemplo, para visualizar apenas os campos Nome e DataNascimento da tabela Pessoas, será preciso digitar este enunciado:

```
SELECT Nome, DataNascimento FROM Pessoas;
```

É importante ressaltar que a ordem na qual as colunas aparecerão será exatamente a mesma na qual foram especificadas no enunciado. Isto quer dizer que o enunciado:

```
SELECT DataNascimento, Nome FROM Pessoas;
```

produzirá um resultado diferente do anterior, pois, no primeiro caso, obteremos o seguinte:

```
+-------------------+----------------+
| Nome              | DataNascimento |
+-------------------+----------------+
| Fernando Henrique | 1989-04-05     |
| Fabrizio          | 1946-06-26     |
| Augusto           | 2001-01-12     |
+-------------------+----------------+
```

Já no segundo caso o resultado será este:

```
+----------------+-------------------+
| DataNascimento | Nome              |
+----------------+-------------------+
| 1989-04-05     | Fernando Henrique |
| 1946-06-26     | Fabrizio          |
| 2001-01-12     | Augusto           |
+----------------+-------------------+
```

Uso do alias para os campos

Os nomes das colunas apresentados no resultado de uma consulta podem ser diferentes dos rótulos de colunas originais, definidos na fase de criação da tabela na qual a consulta foi realizada. Esses nomes diferentes são chamados de “alias”, e são gerados usando a palavra chave AS no enunciado que faz a consulta. Vejamos como criar e usar os alias, criando uma nova tabela no nosso banco de dados.

A nova tabela, que chamaremos Fornecedores, será articulada em sete colunas: CodCli para o código do cliente, RagSoc para a razão social, Endereco para o endereço, Cidade, CEP, Estado e CNPJ. O enunciado para a criação da tabela é o seguinte:

```
mysql> CREATE TABLE Fornecedores (CodCli INT NOT NULL PRI-
MARY KEY AUTO _ INCREMENT, RagSoc VARCHAR(50), Endereco VAR-
CHAR(50), Cidade VARCHAR(30), CEP CHAR(9), Estado CHAR(2),
CNPJ CHAR(18));
```

O resultado será uma tabela com a seguinte estrutura (que pode ser visualizada usando o comando `DESCRIBE`):

```
mysql> DESCRIBE Fornecedores;              [Enter]
+----------------+--------------+------+-----+---------+----------------+
| Field          | Type         | Null | Key | Default | Extra          |
+----------------+--------------+------+-----+---------+----------------+
| CodCli         | int(11)      | NO   | PRI | NULL    | auto _ increment |
| RagSoc         | varchar(50)  | YES  |     | NULL    |                |
| Endereco       | varchar(50)  | YES  |     | NULL    |                |
| Cidade         | varchar(30)  | YES  |     | NULL    |                |
| CEP            | char(9)      | YES  |     | NULL    |                |
| Estado         | char(2)      | YES  |     | NULL    |                |
| CNPJ           | char(18)     | YES  |     | NULL    |                |
+----------------+--------------+------+-----+---------+----------------+
7 rows in set (0.02 sec)
```

Repare que, por motivos práticos, os nomes de alguns campos foram abreviados e podem não ser compreensíveis para os estranhos ao banco de dados. Além disso, o campo Endereco teve o "c cedilha" retirado do seu nome por causa da restrição do SQL quanto ao uso de caracteres acentuados e cedilhas. Graças ao uso dos alias, podemos fazer com que, no resultado da consulta, os rótulos desses campos apareçam com um nome diferente do nome do campo da tabela.

Para podermos realizar uma consulta nesta tabela, é necessário inserir dados nela. No tópico **Inserindo novos registros na tabela**, do **Capítulo 4**, vimos como inserir novos registros em uma tabela; portanto, use o comando `INSERT` com a sintaxe correta para inserir na tabela Fornecedores os seguintes registros:

CodCli	RagSoc	Endereco	Cidade	CEP	Estado	CNPJ
1	New-Line Imp.Exp. Ltda.	Av. C33 nº 555	Rio de Janeiro	12345-678	RJ	78945612345654
Auto	TopTech Ltda.	Rua 98 nº 211	São Paulo	12345-111	SP	65498732145654
Auto	Teratron Ltda.	Av. 65 nº 444	São Paulo	12345-789	SP	12365478921123
Auto	UltraTech Ltda.	Pça V656	Brasília	12345-888	DF	98745632145687
Auto	FV Suprimentos Ltda.	Rua H21 nº 12	Goiânia	12345-989	GO	85214796369851
Auto	Flash Informática Ltda.	Al. Contorno nº 97	Goiânia	12345-345	GO	96325874121114
Auto	Nayfa Desenvolvimento	Av. Oceano nº 166	Brasília	12345-111	DF	19875221458478

Tabela 5.1: Fornecedores.

Agora que a nossa tabela contém dados, vejamos como fazer uma consulta usando o alias no lugar dos nomes dos campos, digitando o seguinte enunciado:

```
mysql> SELECT CodCli AS "Código do Cliente", RagSoc AS "Razão
Social", Endereco AS "Endereço", Cidade, CEP, Estado AS "UF",
CNPJ FROM Fornecedores;                    [Enter]
```

O resultado será uma tela muito similar à da tabela, porém, no lugar dos nomes dos campos, apresenta os nomes dos alias que definimos. Algo como o exemplo mostrado a seguir:

Código do Cliente	Razão Social	Endereço	Cidade	CEP	UF	CNPJ
1	New-Line Imp. Exp. Ltda.	Av. C33 nº 555	Rio de Janeiro	12345-678	RJ	78945612345654
2	TopTech Ltda.	Rua 98 nº 211	São Paulo	12345-111	SP	65498732145654
3	Teratron Ltda.	Av. 65 nº 444	São Paulo	12345-789	SP	12365478921123
4	UltraTech Ltda.	Pça V656	Brasília	12345-888	DF	98745632145687
5	FV Suprimentos Ltda.	Rua HZ1 nº 12	Goiânia	12345-989	GO	85214796369851
6	Flash Informática Ltda.	Al. Contorno nº 97	Goiânia	12345-345	GO	96325874121114
7	Nayfa Desenvolvimento	Av. Oceano nº 166	Brasília	12345-111	DF	19875221458478

Repare que no enunciado definimos o alias apenas para alguns campos (CodCli, RagSoc, Endereço e Estado), enquanto mantivemos o nome original para os outros. Um alias é, de fato, um texto, e por isso não sofre as limitações de uso de caracteres típicos dos nomes de campos; isto quer dizer que num alias podemos digitar o nome que quisermos, incluindo espaços, acentos, cedilhas etc.

Lembre-se sempre que os nomes definidos como alias são exibidos somente nos resultados das consultas, ou seja, os nomes originais dos campos na tabela não são alterados.

Sobre os qualificadores

A palavra chave `AS` é um qualificador do comando `SELECT`, e não uma cláusula. O comando `SELECT` possibilita também o uso de dois outros qualificadores: `ALL` e `DISTINCT`. O primeiro serve para especificar que a consulta deverá extrair todos os elementos indicados, mas é praticamente inútil, pois, por padrão, o comando `SELECT` já faz isso; o segundo serve para instruir o SQL a ignorar valores repetidos na tabela, retornando apenas valores únicos.

Como sempre, vejamos um exemplo prático. A tabela Fornecedores contém diversos registros, alguns dos quais apresentam o mesmo valor no campo Estado, pois, obviamente, é possível que dois ou mais fornecedores residam no mesmo estado (o mesmo vale para

o campo Cidade). Agora, queremos saber quantos e quais estados estão cadastrados em nossa tabela. Podemos obter esta informação com uma consulta que utiliza os qualificador `DISTINCT` e, para tornar mais claro o resultado, AS, como no exemplo a seguir:

```
mysql> SELECT DISTINCT Estado AS "Estados Cadastrados" FROM
Fornecedores;        [Enter]
```

e eis a seguir o resultado do nosso enunciado:

```
+---------------------+
| Estados Cadastrados |
+---------------------+
| RJ                  |
| SP                  |
| DF                  |
| GO                  |
+---------------------+
```

Veja que, graças ao qualificador `DISTINCT`, cada estado é exibido uma só vez, mesmo que na tabela apareça em vários registros. Repare também no rótulo da coluna, diferente do nome do campo da tabela (Estado).

A cláusula `ORDER BY`

A cláusula `ORDER BY` é usada para listar os valores do resultado da consulta seguindo uma ordem diferente daquela com a qual são apresentados na tabela.

Sua sintaxe é:

```
ORDER BY critério _ de _ classificação [ASC | DESC];
```

O critério_de_classificação é constituído por uma ou várias colunas da tabela de origem, e pode ser especificado usando os nomes dos campos ou seus respectivos alias. As colunas usadas como critério de classificação são chamadas de chaves de classificação no jargão dos bancos de dados.

As palavras chave `ASC e DESC` especificam o tipo de classificação e são, respectivamente, abreviações das palavras em inglês *ascending* e *descending*, ou seja, classificação crescente (do menor

para o maior) ou decrescente (do maior para o menor); o primeiro é tido como padrão quando não especificamos nenhum.

Voltando ao exemplo do tópico anterior, podemos fazer com que a lista dos estados cadastrados seja exibida em ordem alfabética decrescente (Z > A). Então, vamos reformular o enunciado anterior acrescentando a cláusula `ORDER BY`, digitando-o desta forma:

```
mysql> SELECT DISTINCT Estado AS "Estados Cadastrados" FROM
Fornecedores ORDER BY "Estados Cadastrados" DESC;
[Enter]
```

O novo resultado será igual ao mostrado logo a seguir:

```
+--------------------+
| Estados Cadastrados |
+--------------------+
| SP                 |
| RJ                 |
| GO                 |
| DF                 |
+--------------------+
```

Neste exemplo, usamos o alias do campo Estado para definir o critério de classificação dos registros. Se tivéssemos usado o nome do campo, o resultado teria sido idêntico. A única diferença é que o nome do alias deve ser especificado entre aspas (simples ou duplas), já o nome do campo não.

Como já foi dito, a cláusula `ORDER BY` pode se referir a mais de uma coluna da tabela de origem. Quando especificamos mais de uma coluna, a primeira é chamada chave de classificação primária, a segunda chave de classificação secundária, e assim por diante. No exemplo a seguir, definimos um critério de classificação baseado em duas colunas: Estado e Cidade; isto quer dizer que a classificação será feita primeiramente com base nos dados do campo Estado (que usa o alias UF) e depois usando o conteúdo do campo Cidade. Veja:

```
mysql> SELECT Estado AS "UF", Cidade FROM Fornecedores ORDER
BY "UF", Cidade;           [Enter]
```

Você pode estar se perguntando: como é feita a classificação com duas colunas? É simples: suponhamos que na tabela haja três fornecedores no estado de Goiás, porém, em cidades diferentes: um em Goiânia, outro em Anápolis e outro em Catalão. Na classificação

por Estato, os três aparecerão juntos, mas qual deles será mostrado primeiro? Como o segundo critério de classificação é a cidade, veremos primeiro o fornecedor de Anápolis, depois o de Catalão e, por fim, o de Goiânia (em ordem alfabética).

Sobre funções no SQL

O padrão SQL disponibiliza algumas funções, cuja sintaxe é constituída pelo nome da função seguido pelos argumentos, entre parênteses.

Tais funções retornam um determinado valor realizando uma operação específica (definida pela palavra-chave que identifica a própria função) com base em um valor indicado como argumento da função. Não necessariamente todas as funções precisam de argumentos; por exemplo, a função `CURRENT _ TIME`, que retorna a data e a hora atuais, não precisa de nenhum argumento, pois pega essas informações no relógio do sistema.

O padrão SQL oferece muitas funções, classificadas em dois grandes grupos:

- **Funções de agregação:** operam em conjuntos de valores (uma coluna inteira, por exemplo) e retornam um único valor;
- **Funções escalares:** trabalham em um só valor (o argumento) e retornam um valor baseado no mesmo.

As tabelas a seguir mostram a lista das funções do SQL e a finalidade de cada uma delas.

Função	O que faz
AVG (expressão)	Calcula o valor médio de uma coluna, indicada no argumento expressão.
COUNT (expressão)	Conta as linhas encontradas pela expressão.
COUNT (*)	Conta todas as linhas da tabela especificada.
MIN (expressão)	Retorna o valor mínimo encontrado em uma coluna, a qual é especificada no argumento expressão.
MAX (expressão)	Retorna o valor máximo encontrado em uma coluna, a qual é especificada no argumento expressão.
SUM (expressão)	Calcula a somatória de todos os valores da coluna indicada no argumento expressão.

Tabela 5.3: Funções de Agregação.

Função	O que faz
CURRENT_DATE	Retorna a data atual, de acordo com o relógio do sistema.
CURRENT_TIME	Retorna a hora atual, de acordo com o relógio do sistema.
CURRENT_TIMESTAMP	Retorna data e hora atuais, de acordo com o relógio do sistema.
CURRENT_USER	Retorna o nome do usuário atualmente logado no sistema do banco de dados.
SESSION_USER	Retorna a identificação de autorização atual, se diferente do usuário.
SYSTEM_USER	Retorna o nome do usuário ativo no sistema operacional do computador host.

Tabela 5.4: Funções Escalares do Sistema.

Função	O que faz
BIT_LENGTH (expressão)	Retorna o valor de um número inteiro que representa o número de bits contidos em uma expressão.
CHAR_LENGTH (expressão)	Retorna o valor de um número inteiro que representa o número de caracteres contidos em uma expressão.
EXTRACT (parte_data FROM expressão)	Extrai de uma expressão parte de uma data ou hora (YEAR, MONTH, DAY, HOUR, MINUTE, SECOND, TIMEZONE_HOUR ou TIMEZONE_MINUTE).
OCTET_LENGTH (expressão)	Retorna o valor de um número inteiro que representa o número de "octets" contidos em uma expressão. O número de "octets" corresponde ao resultado de BIT_LENGTH, dividido por oito.

Função	O que faz
POSITION (sequência IN seqüência_da_busca)	Retorna um número inteiro que representa a posição inicial de uma seqüência de caracteres ("string") dentro da seqüência na qual é feita a busca.

Tabela 5.5: Funções escalares numéricas.

Função	O que faz
CONCATENATE (expressão | expressão)	Reúne em uma única seqüência de caracteres uma ou mais seqüências, valores de colunas ou variáveis.
CONVERT()	Converte uma seqüência de caracteres em uma representação diferente dentro do mesmo grupo de caracteres.
LOWER()	Converte em letras minúsculas todos os caracteres maiúsculos de uma seqüência de caracteres.
SUBSTRING()	Extrai parte de uma seqüência de caracteres.
TRANSLATE()	Converte uma seqüência de caracteres de um grupo para outro.
TRIM()	Apaga os caracteres iniciais, finais (ou ambos) de uma seqüência de caracteres.
UPPER()	Converte em letras maiúsculas todos os caracteres minúsculos de uma seqüência de caracteres.

Tabela 5.6: Funções Escalares de Texto.

As funções citadas nas tabelas anteriores são apenas as que encontramos no padrão SQL; todavia os demais dialetos do SQL (Oracle, MySQL, Microsoft SQL Server etc.) possuem um número considerável de outras funções (no MySQL são mais de cem), especialmente as do tipo matemático, que proporcionam ao usuário maior flexibilidade para obter valores a partir de cálculos feitos com o conteúdo das tabelas.

Para realizar cálculos matemáticos no SQL, são empregados símbolos, denominados operadores. São usados para representar as operações aritméticas ou lógicas que podem ser executadas. São eles:

Operador	Significado
+	Usado para realizar operações de soma e concatenação
-	Usado para realizar subtrações
*	Usado para realizar multiplicações
/	Usado para realizar divisões
=	Representa "igual a..."
<> ou !=	Ambos representam "diferente de..."
<	Usado em comparações, está por "menor que..."
>	Usado em comparações, está por "maior que..."
<=	Usado em comparações, está por "menor ou igual a..."
>=	Usado em comparações, está por "maior ou igual a..."

Tabela 5.7: Operadores.

A cláusula `GROUP BY`

Em muitos casos, as funções de agregação são usadas junto com a cláusula `GROUP BY` do comando `SELECT`; esta cláusula é usada para agrupar dados (em inglês, *group by* significa "agrupar por").

Quando usamos a cláusula `GROUP BY`, é preciso especificar, tanto no comando `SELECT` quanto na cláusula `GROUP BY`, a mesma coluna, que é aquela com base na qual se deseja agrupar o resultado da consulta.

Vejamos um exemplo de como usar a cláusula `GROUP BY` para obter a média das idades e dos anos de serviço de cada um dos funcionários de uma empresa, agrupando-os por estado.

Observação: neste caso, não iremos criar nenhuma nova tabela por razões práticas, porque seria necessário não só criar a tabela, mas também preenchê-la com uma quantidade consistente de dados; portanto, suponha estarmos trabalhando em uma tabela chamada Funcionarios, que contém dados relativos aos funcionários de uma empresa qualquer, e na qual temos, entre

outros campos, um Estado, outro com a data de nascimento (DataNascimento) e outro ainda com a data em que cada funcionário foi contratado (DataContratacao).

O enunciado ficaria assim:

```
SELECT Estado, ROUND(AVG(YEAR(NOW()) – YEAR(DataContratacao)))
AS "Média do tempo de serviço", ROUND (AVG(YEAR(NOW()) –
YEAR(DataNascimento))) AS "Média das Idades" FROM Funciona-
rios GROUP BY Estado;                                    [Enter]
```

O enunciado anterior reúne muitos conceitos e ferramentas vistas até o momento: o comando `SELECT` usa como argumento a coluna Estado da tabela Funcionários, local em que a consulta deverá ser feita. No decorrer da consulta, é feito um cálculo para descobrir a quanto tempo cada funcionário trabalha na empresa, subtraindo da data atual a data de contratação, calculando a média de todos os resultados obtidos na coluna e arredondando o resultado final. Para realizar este cálculo, foram usadas diversas funções (`ROUND`, `AVG`,...) e um operador (-). O mesmo foi feito para descobrir a idade de cada funcionário, subtraindo da data atual a data de nascimento, calculando a média de todas as idades e arredondando o resultado final. Os dois resultados foram armazenados em dois alias, um chamado Média do tempo de serviço e o outro Média das idades. Enfim, todos os resultados são agrupados por estado, empregando a cláusula `GROUP BY`. A tabela resultante dessa consulta teria essa estrutura:

```
+----------------+---------------------------+------------------+
| Estado         | Média do tempo de serviço | Média das idades |
+----------------+---------------------------+------------------+
| GO             |                         8 |               37 |
| SP             |                        12 |               44 |
| DF             |                         6 |               39 |
| RJ             |                         9 |               29 |
| MG             |                        10 |               32 |
+----------------+---------------------------+------------------+
```

Na mesma tabela, queremos agora realizar a contagem dos funcionários por estado, isto é, queremos saber quantos funcionários a empresa tem em cada estado. Trata-se de uma operação bastante simples, que pode ser realizada com o enunciado mostrado a seguir:

```
SELECT Estado, COUNT (*) AS Funcionarios FROM Funcionarios
GROUP BY Estado;                                    [Enter]
```

Eis o resultado do comando exibido na tela do computador:

```
+------------ +--------------------+
| Estado      |      Funcionarios  |
+------------ +--------------------+
| GO          |      21            |
| SP          |      19            |
| DF          |      14            |
| RJ          |      16            |
| MG          |      9             |
+------------ +--------------------+
```

Há algo que não podemos esquecer: o SQL não permite selecionar em conjunto as colunas que são resultado da aplicação de uma consulta e colunas reais de uma tabela de origem. Se fizermos isto, receberemos uma mensagem de erro assim:

```
Mixing of GROUP columns (MIN(), MAX(), COUNT()...) with no
GROUP columns is illegal if there is no GROUP BY clause
```

É importante saber também que a cláusula `GROUP BY` retorna os seus resultados na mesma ordem em que os dados são apresentados na tabela de origem. Contudo, é possível alterar a ordem de classificação do resultado da consulta recorrendo à cláusula `ORDER BY`, vista no tópico **A cláusula `ORDER BY`** deste capítulo.

Utilizando as duas cláusulas no mesmo enunciado, a sintaxe a ser seguida deve ser igual à mostrada no exemplo a seguir:

```
SELECT Estado, COUNT (*) AS Funcionarios FROM Funciona-
rios GROUP BY Estado ORDER BY Nome;
[Enter]
```

Observe a linha de comando anterior e repare que a cláusula `ORDER BY` foi digitada depois da cláusula `GROUP BY` e, neste caso, classifica o resultado da consulta em ordem crescente de nomes.

Capítulo 6

Outros SGBDR

Introdução

Durante os exemplos práticos deste livro, nos baseamos no ambiente de programação MySQL pois, como vimos, este traz a vantagem indiscutível de ser gratuito e muito válido.

Todavia, como já dissemos, existem outros sistemas de gerenciamento de bancos de dados relacionais que merecem a nossa atenção. Entre eles, o Oracle e o Microsoft SQL Server são os mais conhecidos.

Neste capítulo, veremos as características mais marcantes desses dois poderosos SGBDR, amplamente usados a nível mundial, para a criação e manutenção de bancos de dados extremamente complexos.

O Oracle

O Oracle é um SGBD (Sistema Gerenciador de Banco de Dados) que surgiu no final da década de 1970, quando Larry Ellison vislumbrou uma oportunidade, que outras companhias não haviam percebido, ao encontrar uma descrição de um protótipo funcional de um banco de dados relacional. Nenhuma empresa tinha se empenhado em comercializar essa tecnologia.

Ellison e os co-fundadores da Oracle Corporation, Bob Miner e Ed Oates, perceberam que havia um tremendo potencial de negócios no modelo de banco de dados relacional e tornaram-se, assim, a maior empresa de software empresarial do mundo.

O SGBD da Oracle é líder de mercado. O Oracle 9i foi pioneiro no suporte ao modelo Web. O Oracle 10g, mais recente, se baseia na tecnologia de grid.

Além da base de dados, a Oracle desenvolve uma suíte de desenvolvimento chamada de *Oracle Developer Suite*, utilizada na construção de programas de computador que interagem com a sua base de dados.

A Oracle também criou a linguagem de programação PL/SQL, utilizada no processamento de transações.

O Oracle XE

O Oracle Database Express Edition (Oracle Database XE) é uma edição do banco de dados Oracle gratuita, fácil de instalar e fácil de

gerenciar. Ideal para desenvolvedores estudantes e todos aqueles que pretendem estudar a tecnologia Oracle.

O Oracle Database XE permite que você utilize uma interface intuitiva baseada em navegador Web para as seguintes situações:

- Administrar o banco de dados;
- Criar tabelas, views e outros objetos de banco de dados;
- Importar, exportar e visualizar dados de tabelas;
- Executar consultas no SQL;
- Gerar relatórios.

O Oracle Database Express inclui o Oracle Applications Express release 2.21, que é uma interface gráfica para desenvolvimento de aplicações baseadas na Web. Além do Oracle Applications Express Release 2.21, você poderá usar as linguagens e ferramentas Oracle ou de terceiros para desenvolvimento de diversas aplicações, utilizando o Oracle Database Express Edition.

No Oracle Database Express Edition, encontramos as seguintes ferramentas de linha de comando:

- **SQL Command Line (SQL Plus):** usado para execução de comandos SQL e PL/SQL e execução de scripts;
- **SQL Loader:** usado para carregar dados em um banco de dados;
- **Data Pump:** trata-se de um utilitário de importação e exportação de dados.

Acessando a Database Home Page

O Oracle Database Express Edition possui uma interface baseada em navegadores Web que permite ao usuário administrar o banco de dados, executar scripts, consultas, criar aplicações Web e muito mais. O ponto de partida desta ferramenta é a Database Home Page. É importante ressaltar que, após a instalação, a Database Home Page estará ativa apenas no computador onde o Oracle Database Express Edition foi instalado, ou seja, o computador local (host).

Os ícones da Database Home Page têm as seguintes funções:

- **Administration:** gerenciar as contas de usuários, gerenciar a memória, capacidade de armazenamento e conexões de rede, monitorar as atividades do banco de dados e visualizar as informações de configuração;
- **Object Browser:** visualizar, criar, modificar e apagar objetos de banco de dados. Usar o PL/SQL editor para editar e compilar pa-

cotes, procedures, funções e triggers, com a vantagem de usufruir de um relatório de erros;

- **SQL***:* acesso as seguintes ferramentas SQL:
  - **SQL Commands:** executar comandos SQL e blocos anônimos de PL/SQL. Rodar os scripts e salvar as consultas;
  - **SQL Scripts:** criar, editar, visualizar, executar e apagar arquivos de scripts. Realizar o upload e o download de scripts para o seu sistema de arquivos local;
  - **Query Builder:** com pouco ou nenhum conhecimento SQL, você poderá criar consultas utilizando uma interface gráfica, visualizar resultados formatados desta consultas e salvá-las.
- **Utilities:** carregar e descarregar dados, gerar DDL, visualizar objetos de relatórios e restaurar objetos apagados do banco de dados.

Ao realizar o login no Database Home Page com um usuário diferente do SYSTEM, um outro ícone chamado Application Builder será exibido. Ele permite que você crie aplicações Oracle no Database Express Edition.

Acessando a Database Home Page da sua área de trabalho

Para acessar a Database Home Page da sua área de trabalho (em diferentes sistemas operacionais), execute os seguintes passos:

1. Nos sistemas operacionais:

- **Windows**: clique no botão **Iniciar** > **Programas** (ou **Todos os programas**) > **Oracle Database 10g Express Edition** e selecione **Go To Database Home Page**;
- **Linux:** utilizando o **Gnome,** clique no menu **Applications**. Em seguida; em **Oracle Database 10g Express Edition** e selecione **Go To Database Home Page**;
- **Linux:** utilizando o **KDE**, clique no ícone do **K menu**. Em seguida, **Oracle Database 10g Express Edition** e selecione **Go To Database Home Page**.

2. Quando a página de login aparecer, faça sua autenticação no banco de dados utilizando um usuário e senha válidos. Para autenticar como administrador, utilize o usuário **SYSTEM** e a senha que você definiu durante a instalação do Oracle Database Home Express Edition.

Acessando a Database Home Page em seu navegador – usuário local

Para acessar o Database Home Page do seu navegador do mesmo computador que você instalou o Oracle Database Express Edition, siga os passos a seguir:

1. Na linha de endereço do seu navegador, digite o seguinte endereço:

http://127.0.0.1:port/apex

Em que **port** é o número da porta TCP para requisições HTTP. O valor padrão é 8080. Você pode trocar este valor durante a instalação. Por exemplo, se você instalou o Oracle Database Express Edition com porta padrão, você pode acessar a Database Home Page através da URL:

http://127.0.0.1:8088/apex

2. Quando a página de login aparecer, digite um usuário e senha válidos para acessar o banco de dados. Para autenticar como administrador, utilize o usuário **SYSTEM** e a senha que você definiu durante a instalação do Oracle Database Home Express Edition.

Acessando a Database Home Page em seu navegador – usuário remoto

Antes de tudo, uma consideração: para acessar o banco de dados remotamente, você precisa ativar o acesso remoto. Para acessar o Database Home Page remotamente pelo seu navegador, siga os seguintes passos:

1. Digite na barra de endereço do seu navegador a seguinte URL:

http://host:port/apex

Em que **host** é o nome do servidor ou o endereço IP do computador onde o Oracle Database Express Edition está instalado e **port** é o número da porta TCP que recebe a requisição HTTP. O valor padrão

é 8080. Você pode mudar este valor durante a instalação do Oracle Database Express Edition. Por exemplo, se você instalou o Oracle Database Express Edition em um computador com o nome de myhost.mydomain.com e o fez utilizando a porta padrão, o endereço da Database Home Page será:

http://myhost.mydomain.com:8080/apex

2. Quando a página de login aparecer, use um usuário válido e senha para acessar a página. Para autenticar como administrador, utilize o usuário **SYSTEM** e a senha que você definiu durante a instalação do Oracle Database Home Express Edition.

Buscando ajuda

Você pode acessar o Help on-line sensitivo ao contexto através de uma das seguintes maneiras:

- Para ajuda sobre a página corrente do Database Home Page, clique no ícone **Help** localizado no canto superior direito da página; assim, a janela de ajuda será aberta. Além da ajuda referente à página de origem, você poderá navegar pelos tópicos de ajuda no painel do lado esquerdo da janela;
- Para ajuda sobre um campo individual, posicione o cursor sobre a descrição do campo (rótulo do campo); se a descrição tornar-se vermelha e o cursor mudar para uma mão, há uma ajuda disponível para aquele campo. Clique sobre a descrição para visualizar a ajuda.

Para utilizar a ajuda do Oracle Database Express Edition quando não estiver utilizando a interface gráfica, faça o seguinte:

- **No Windows**: clique no botão **Iniciar** > **Programas** (ou em **Todos os programas**), clique em **Oracle Database 10g Express Edition**, selecione **Get Help** e, em seguida, clique em **Read Online Help**;
- **No Linux, utilizando o Gnome**: no menu **Applications**, clique em **Oracle Database 10g Express Edition** e selecione **Get Help** e, em seguida, clique em **Read Online Help**;
- **No Linux, utilizando o KDE**: no menu **K**, clique em **Oracle Database 10g Express Edition** e selecione **Get Help** e, em seguida, clique em **Read Online Help**.

O Microsoft SQL Server

O MS SQL Server é um Gerenciador de Banco de Dados Relacional feito pela Microsoft. É um banco de dados muito robusto e usado muito em empresas e por grandes sistemas corporativos. Atualmente, encontra-se na versão 2005. Entre os novos recursos, está a integração com o Framework.Net, que possibilita construir rotinas utilizando as linguagens do .Net como VB.Net e C#.

O MS SQL Server tem versões unicamente para plataformas baseadas no sistema operacional Windows, da Microsoft, ao contrário de seus grandes concorrentes, o Oracle e o Postgres, que funcionam em diversas plataformas e sistemas operacionais diferentes.

Suas ferramentas de gerenciamento são nativas, não necessitando que sejam adquiridas separadamente. São elas:

- **MS SQL Enterprise Manager (Console Central):** integra, num único painel, a maioria das funções que um DBA poderá utilizar-se para configurar e gerenciar esse RDBMS;
- **Query Analyzer:** que permite executar consultas e auxilia o gerenciamento, inclusive em Tunning;
- **Profile:** que é uma espécie de Trace, descortinando os comandos que o gerenciador está executando, além de outras ferramentas.

O SQL Server Express

O SQL Server Express é uma plataforma de banco de dados baseada nas tecnologias do SQL Server 2005 e substituiu o MSDE 2000. Ele é um produto gratuito e possui características de rede e segurança que o diferenciam das demais edições do SQL Server 2005.

Neste tópico, falaremos das suas principais características e faremos algumas comparações com o sistema anterior, o MSDE 2000.

O SQL Server Express tem como grande promessa uma superior facilidade de utilização com um processo de instalação simples e robusto, e com uma ferramenta gráfica (SQL Express Manager), que permitirá realizar a administração do servidor de uma forma simples e prática. Esta ferramenta gráfica ainda está em desenvolvimento e será disponibilizada como um download gratuito no site da Microsoft.

O SQL Server Express foi desenvolvido tendo em mente duas utilizações básicas: a primeira, como um servidor de produtos, especialmente como Web Server ou Database Server. O segundo,

como um cliente standalone, em que a aplicação não precise depender de uma rede para obter acesso aos dados.

O Engine

O SQL Server Express usa o mesmo Engine que as demais edições do SQL Server 2005, mas, por ser uma edição, digamos, light, possui algumas limitações. O Engine suporta uma CPU, 1 GB de RAM e banco de dados com até 4 GB. Uma mudança significativa em relação ao MSDE 2000 é que o Engine do SQL Express não possui a limitação de usuários concorrentes, conhecida como *Concurrent Workload Governor*, em que a performance do MSDE 2000 é extremamente prejudicada na medida em que as conexões concorrentes de usuários aumentam. O SQL Server Express pode ser instalado também em máquinas com processadores múltiplos, mas somente um deles será utilizado pelo Engine. Como conseqüência, características como execução de consultas em paralelo não é suportado pelo SQL Server Express.

O limite de 1 GB RAM é apenas para o buffer pool. O buffer pool é usado para o armazenamento de páginas de dados e outras informações. A memória necessária para o gerenciamento de conexões, locks e outros não está incluída neste limite de 1 GB. Sendo assim, o SQL Express pode ser instalado normalmente em máquinas com mais de 1 GB, mas ele nunca usará mais que 1 GB para o buffer pool. AWE ou /3GB não é suportado. O limite de 4 GB para o banco de dados é aplicado apenas para o arquivo de dados. Entretanto, não existe limite para o número de banco de dados que você pode colocar no servidor.

O SQL Server Express suporta instalação side-by-side com outras versões do SQL Server, podendo coexistir na mesma máquina junto com instalações do SQL Server 2000, SQL Server 2005 ou MSDE 2000. Suporta um número máximo de 50 instâncias na mesma máquina, desde que cada instância seja unicamente identificada, ou seja, você pode realizar até 50 instalações do SQL Express desde que cada instância tenha um nome diferente. Por padrão, o SQL Server Express é instalado com uma named instance chamada SQLEXPRESS.

As ferramentas

Ao contrário do que acontecia no MSDE 2000, que não possui ferramenta gráfica, o SQL Server Express possuirá uma ferramen-

ta gráfica chamada SQL Server Express Manager (XM). Esta, ainda em fase de desenvolvimento, permitirá uma fácil administração dos bancos de dados e é distribuída separadamente para download.

O XM suporta conexões para SQL Server Express e outras edições do SQL Server 2005, SQL Server 2000 e MSDE 2000. Uma janela de conexão guiará o usuário através da seleção de uma instância e o método de autenticação a ser utilizado, suportando conexões locais ou remotas. Diversas funcionalidades de gerenciamento de banco de dados estarão disponíveis por meio do menu de contexto (botão direito) e wizards. Entre elas, podemos citar, criar e modificar banco de dados, tabelas, usuários e logins. O Query Editor do XM permitirá o desenvolvimento e execução de instruções T-SQL e scripts.

Algumas das ferramentas inclusas no SQL Server Express são o SQL Computer Manager, SQL Command e o utilitário BCP. O SQL Computer Manager substitui os atuais Service Manager, Client Network Utilities e Server Network Utilities. Ele é usado para iniciar e parar os serviços do SQL Server e para habilitar e desabilitar os protocolos de rede. O SQL Command (**sqlcmd.exe**) é uma versão OLE DB do **osql.exe**; ele não somente tenta manter as funcionalidades do SQL, como também possui suporte aos novos tipos de dados do SQL Server 2005. SQL Computer Manager e SQL Command são os mesmos em todas as edições do SQL Server 2005.

Suporte a rede

Embora o usuário possa habilitar o suporte a protocolos como TCP/IP e Named Pipes, por padrão, o SQL Server Express é instalado apenas com suporte a shared memory. Isto faz com que por default o SQL Server Express funcione apenas localmente na máquina onde foi instalado e não suporte conexões de clientes em uma rede. Para suportar conexões de clientes remotos, é necessário habilitar o suporte a rede no SQL Server Express. Para habilitar o suporte à rede, você tem as seguintes opções:

- Usar o SQL Computer Manager para habilitar os protocolos necessários (normalmente TCP/IP) e iniciar o serviço SQL Browser;
- Usar a parâmetro DISABLENETWORKPROTOCOLS=0 na linha de comando no momento da instalação.

O SQL Browser é um novo serviço do SQL Server 2005 que identifica as portas que as named instances estão usando. Este serviço

vem desativado por default e deve ser ativado para que a comunicação entre um cliente remoto e um servidor SQL Server Express funcione corretamente.

Funções do SQL Server Express

O SQL Server Express suporta a maioria das funcionalidades do SQL Server 2005. A lista a seguir mostra algumas características e componentes suportados.

- Stored Procedures;
- SQL Computer Manager;
- Views;
- Replication (as a subscriber only);
- Triggers;
- Advanced Query Optimizer;
- Cursors;
- SMO / RMO;
- Sqlcmd and osql utilities;
- Integration with Visual Studio 2005;
- Snapshot Isolation Levels;
- Service Broker (as a client only);
- Native XML support, including XQuery and XML Schemas;
- SQL CLR;
- T-SQL language support;
- Multiple Active Result Sets (MARS).

A seguir, relatamos os principais componentes do SQL Server 2005 que não são suportados nesta versão do SQL Express Edition:

- Reporting Services;
- Notification Services;
- Analysis Services;
- SQL Agent;
- Full text search;
- DTS;
- OLAP Services / Data Mining;
- English Query.

A seguir, destacamos algumas características do SQL Server 2005 e do SQL Server 2000 que não são suportadas nesta versão do SQL Express:

Novas características do SQL Server 2005

- SqlMail and SQLiMail;
- Database mirroring;
- Distributed partitioned views;
- Database snapshot;
- Log shipping;
- Mirrored media sets;
- Parallel Create Index;
- Address Windowing Extensions (AWE);
- Indexed views (materialized views);
- Native HTTP.

Características do SQL Server 2000

- Dedicated Administrator Connection;
- Fail-over clustering;
- Online restore;
- VIA protocol support;
- Parallel index operations;
- Parallel DBCC;
- Scale up partitioning;
- Enhanced Read Ahead and Scan;
- Hot-add memory;
- Partitioned .

O SQL Express e o MSDE

O SQL Express Edition é o sucessor do MSDE e está baseado nas características do SQL Server 2005. Este possui características não presentes no MSDE como um robusto e simples setup GUI, suporte a CLR, ferramentas gráficas e integração com o Visual Studio. Entretanto, algumas características foram removidas do SQL Server Express como: suporte a DTS, replication publishing e SQL Agent. A limitação de conexões concorrentes do MSDE foi removida e, ao invés disso, o SQL Server Express usa as limitações de CPU, RAM e banco de dados para se diferenciar das outras edições.

Como podemos observar, o SQL Server Express promete, principalmente no que diz respeito às facilidades de uso e administração. Ele é seguro por default, totalmente free, compatível com outras edi-

ções do SQL Server 2005 e sua integração com o Visual Studio 2005 torna mais simples o desenvolvimento de aplicações que utilizam o SQL Server como repositório de dados.

Entretanto, também pudemos notar que muitas características e componentes existentes hoje no MSDE foram deixados de lado, como, por exemplo, suporte a dois processadores e 2 GB de RAM, SQL Agent e DTS.

O Firebird

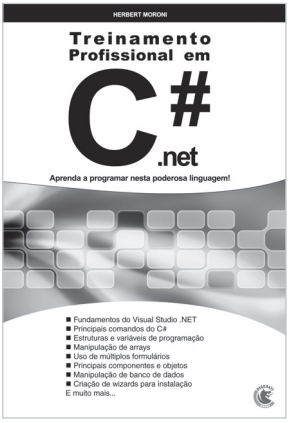
O Firebird, conhecido também com o nome FirebirdSQL, é um Sistema Gerenciador de Bancos de Dados Relacionais multiplataforma, isto é, que roda em Linux, Windows, Mac OS e uma variedade de plataformas baseadas em Unix (como o Linux). A Fundação FirebirdSQL coordena a manutenção e desenvolvimento do Firebird, sendo que os códigos-fonte são disponibilizados sob o CVS em sourceforge.net.

Baseado no código do InterBase da Borland, quando da abertura de seu código na versão 6.0 (que saiu em 25 de julho de 2000), alguns programadores, em associação, assumiram o projeto de identificar e corrigir inúmeros bugs da versão original, surgindo daí o Firebird, e a primeira versão 1.0 se tornou um banco com características próprias, obtendo uma aceitação imediata no círculo de programadores. Quase que totalmente ainda compatível com sua origem, está atualmente em sua versão 1.5, com muitas novidades. A versão 2.0, em fase de produção ainda, deverá trazer muitas inovações. Já está se falando até em uma versão 3.0, que hoje tem o codinome "Vulcan", cujas características já então seriam de um super banco de dados. Seu maior diferencial ainda se baseia na gratuidade; o banco é "free" em todos os sentidos: não há limitações de uso, e seu suporte é amplamente discutido em listas na Internet, o que facilita enormemente a obtenção de ajuda técnica. O produto se mostrou bastante seguro, suportando sistemas com dezenas de usuários simultâneos e bases de dados acima de 2 GB de tamanho.

A seguir, algumas dicas de sites para encontrar informações mais detalhadas, tutoriais e exemplos relacionados ao Firebird:

- http://www.firebase.com.br
- http://www.firebirdsql.org/
- http://www.firebird.com.br
- http://firebird.sourceforge.net/index.php?op=ffoundation&id=main_pt

Conheça outros títulos relacionados



Treinamento Profissional em C# .net

No livro *Treinamento Profissional em C# .net* o leitor aprenderá:

| Fundamentos do Visual Studio .Net;
| Principais comandos do C#;
| Estruturas e variáveis de programação;
| Manipulação de arrays;
| Uso de múltiplos formulários;
| Principais componentes e objetos;
| Manipulação de banco de dados;
| Criação de wizards para instalação.

Além disso, o livro ensina também:

| Passo a passo tudo o que você precisa para criar aplicações Windows e compilar seus programas;
| Como utilizar os comandos básicos, as estruturas de decisão e repetição no C#;
| Como declarar e manipular variáveis e arrays;
| Como montar setups para instalação dos seus aplicativos;
| Tudo o que você precisa saber para construir aplicativos que manipulem diferentes tipos de banco de dados;
E muito mais.

Treinamento Prático em UML

Desenvolva e gerencie seus projetos com essa sensacional ferramenta

Aprenda a trabalhar com UML e cada um de seus componentes:

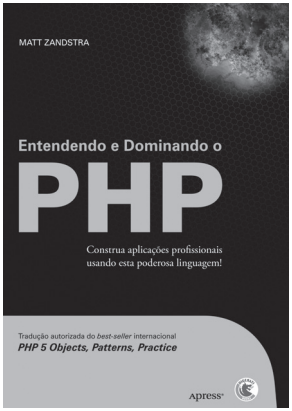
| Principais pacotes e ferramentas da linguagem
| Mecanismos de extensão
| Elementos de modelagem
| Administração de processos
| Elementos para modelar distribuição e concorrência
| Padrões de projeto e colaborações
| Diagramas de atividades
| Refinamento e níveis de abstração de um projeto
| Inferfaces e componentes
| Linguagem de restrições (Object Contraint Language)

Diagramas

Aprenda passo a passo a modelar e representar projetos e situações utilizando:

| Modelagem estruturada
| Modelagem comportamental
| Diagramas de casos de uso
| Diagramas de classes e de objetos
| Diagramas de comportamento
| Diagramas de estados (statechart)

Entre outros...



Entendendo e Dominando o PHP

O *Entendendo e Dominando o PHP* é um livro indicado para todos aqueles que pretendem desenvolver websites usando a linguagem PHP a partir da versão 5.0, que traz como grande inovação o fato de permitir o uso de todo o potencial da orientação a objetos.

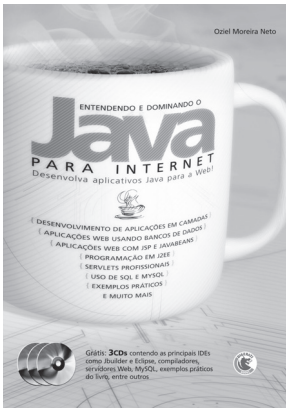
Entre os tópicos abordados no livro estão: orientação a objetos, classes, objetos e métodos; argumentos e tipos de dados; uso de herança; construção de interfaces; manipulação de exceções; interceptadores; definição do projeto do código; programação procedural e orientada a objetos; polimorfismo e encapsulamento; uso da UML para construir diagramas de classes; padrões de projeto; executação e representação de tarefas; documentação com o phpDocumentor; controle de versões com CVS; construção automática com Phing; reusabilidade e muito mais...

Treinamento Avançado em .NET

Orientação a objetos

Conheça todos os truques de programação e orientação a objetos na plataforma .NET:

| Uso do Framework .NET
| Bibliotecas de classes e atributos
| Programação com ActiveX
| Desenvolvimento em XML
| Programação em C#
| VB.NET
| ASP.NET



Entendendo e Dominando o Java para Internet

De alguns anos para cá, o Java se consolidou como a linguagem mais usada por programadores e grandes corporações em todo o mundo. Somado a esse sucesso do Java, vem a crescente demanda por aplicativos que estejam totalmente integrados ao ambiente Web. Essa demanda veio acompanhada de aumento no grau de exigência para o profissional, que precisa ter alto nível de conhecimento, ser extremamente capacitado e possuir experiência e sólidos conhecimentos na linguagem.

Para atender a essas demandas paralelas, *Entendendo e Dominando o Java para Internet* traz explicações e exemplos completos que facilitam o aprendizado dos sofisticados recursos do Java e ensinam a combiná-los para desenvolver aplicativos on-line.

A estrutura do livro foi projetada com base em um método de ensino de eficácia comprovada, desenvolvida em anos de execução de treinamentos intensivos e semi-intensivos em faculdades e empresas, para os mais diversos tipos de alunos, com os mais diferentes níveis de conhecimento em programação.

Com isso, temos certeza de que o *Entendendo e Dominando o Java para Internet* ajudará o leitor em sua busca por informação e especialização.

DESVENDANDO O MySQL

Neste livro você terá oportunidade de aprender a criar poderosas aplicações para banco de dados usando a linguagem SQL, padrão para manipulação de informações em bancos de dados relacionais. Todos os exemplos e comandos apresentados aqui foram testados no MySQL 5.0, o banco de dados livre mais popular do mercado, e que possui recursos que o ajudam a fazer frente a concorrentes de peso como Microsoft, Oracle e IBM.

Atualmente, milhares de sites na Internet são mantidos em MySQL, que se destaca, entre outros fatores, pela velocidade de acesso e a independência de plataforma. Entre os assuntos abordados no livro estão:

- GERENCIAMENTO DE BANCOS DE DADOS
- CRIAÇÃO DE TABELAS
- CONCEITOS BÁSICOS DO SQL
- TABELAS, CONSULTAS E VIEWS
- INSTALAÇÃO DO MYSQL 5.0 NO WINDOWS
- FUNCIONAMENTO DO MYSQL
- USO DE BANCOS DE DADOS RELACIONAIS
- INSERÇÃO DE DADOS EM UMA TABELA
- TIPOS DE DADOS SUPORTADOS PELO SQL
- CRIAÇÃO DE CONSULTAS NO SQL

E MUITO MAIS...

ISBN 978 - 85 - 60480 - 25 - 8
9 788560 480258