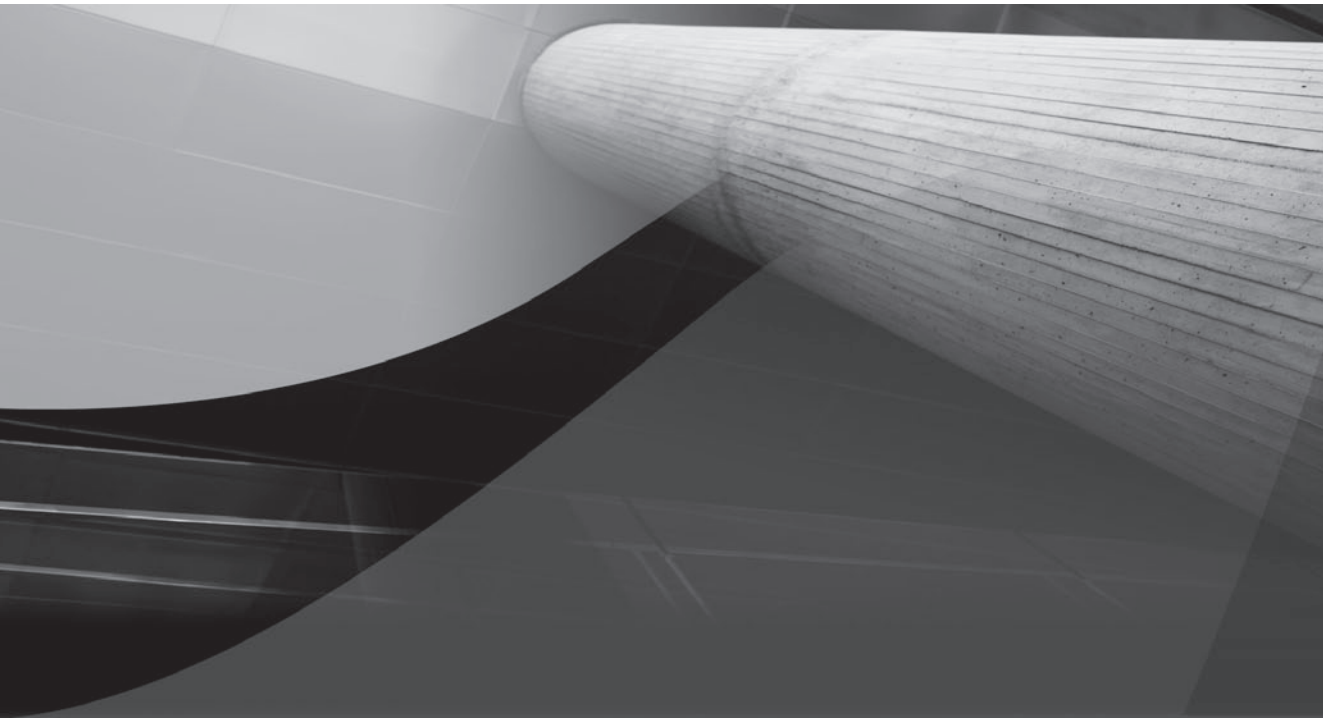


CAPÍTULO 1

Introdução

Neste capítulo, você vai aprender sobre:

- Banco de dados relacionais
- A linguagem SQL (Structured Query Language), usada para acessar um banco de dados
- SQL *Plus, a ferramenta interativa da Oracle para executar instruções SQL
- SQL Developer, uma ferramenta gráfica para desenvolvimento de banco de dados
- PL/SQL, a linguagem de programação procedural da Oracle. O PL/SQL permite desenvolver programas que são armazenados no banco de dados

Vamos começar entendendo o que é um banco de dados relacional.

O QUE É UM BANCO DE DADOS RELACIONAL?

O conceito de banco de dados relacional foi originalmente desenvolvido em 1970 pelo Dr. E. F. Codd. Ele esboçou a teoria dos bancos de dados relacionais em seu artigo intitulado "A Relational Model of Data for Large Shared Data Banks" (Um modelo de dados relacional para grandes bancos de dados compartilhados), publicado na *Comunications of the ACM* (Association for Computing Machinery), vol.13, n°6, junho de 1970.

Os conceitos básicos de um banco de dados relacional são muito fáceis de entender. Um *banco de dados relacional* é uma coleção de informações relacionadas, organizadas em *tabelas*. Cada tabela armazena dados em *linhas*; os dados são organizados em *colunas*. As tabelas são armazenadas em *esquemas* de banco de dados, que são áreas onde os usuários podem armazenar suas próprias tabelas. Um usuário pode conceder *permissões* a outros usuários para que eles possam acessar suas tabelas.

A maioria de nós conhece o armazenamento de dados em tabelas — preços de ações e horários de trem às vezes são organizados em tabelas. A tabela de exemplo usada neste livro registra informações de clientes para uma loja imaginária; a tabela armazena os nomes, sobrenomes, datas de nascimento (dob) e números de telefone dos clientes:

```
nome       sobrenome  dob         telefone
---------- ---------- ----------- ------------
John       Brown      01-JAN-1965 800-555-1211
Cynthia    Green      05-FEV-1968 800-555-1212
Steve      White      16-MAR-1971 800-555-1213
Gail       Black                  800-555-1214
Doreen     Blue       20-MAI-1970
```

Essa tabela poderia ser armazenada de várias formas:

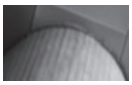
- Um cartão em uma caixa
- Uma página HTML na Web
- Uma tabela em um banco de dados

Um ponto importante a ser lembrado é que as informações que compõem um banco de dados são diferentes do sistema usado para acessar essas informações. O software usado para acessar um banco de dados é conhecido como *sistema de gerenciamento de banco de dados*. O banco de dados Oracle é um desses softwares; outros exemplos incluem o SQL Server, o DB2 e o MySQL.

Evidentemente, todo banco de dados precisa ter algum modo de inserir e extrair dados, de preferência usando uma linguagem comum, entendida por todos os bancos de dados. Os sistemas de gerenciamento de banco de dados implementam uma linguagem padrão conhecida como *Structured Query Language* ou SQL. Dentre outras coisas, a linguagem SQL permite recuperar, adicionar, modificar e excluir informações em um banco de dados.

APRESENTANDO A LINGUAGEM SQL (STRUCTURED QUERY LANGUAGE)

A linguagem SQL (Structured Query Language) é a linguagem padrão projetada para acessar banco de dados relacionais. Pronuncia-se SQL soletrando as letras "S-Q-L".



NOTA

"S-Q-L" é a maneira correta de pronunciar SQL, de acordo com o American National Standards Institute. Contudo, em inglês, a palavra "sequel" é usada com freqüência.

A linguagem SQL é baseada no trabalho pioneiro do Dr. E.F. Codd. Sua primeira implementação foi desenvolvida pela IBM em meados dos anos 1970, dentro de um projeto de pesquisa conhecido como System R. Posteriormente, em 1979, uma empresa então chamada Relational Software Inc. (hoje Oracle Corporation) lançou a primeira implementação comercial da linguagem SQL. Atualmente, a linguagem SQL está totalmente padronizada e é reconhecida pelo American National Standards Institute.

A linguagem SQL usa uma sintaxe simples, fácil de aprender e utilizar. Você vai ver alguns exemplos de sua utilização neste capítulo. Existem cinco tipos de instruções SQL, descritas a seguir:

- **Instruções de consulta** recuperam linhas armazenadas nas tabelas do banco de dados. Você escreve uma consulta usando a instrução SQL `SELECT`.
- **Instruções DML (Data Manipulation Language)** modificam o conteúdo das tabelas. Existem três instruções DML:
  - **`INSERT`** adiciona linhas em uma tabela.
  - **`UPDATE`** altera linhas.
  - **`DELETE`** remove linhas.
- **Instruções DDL (Data Definition Language)** definem as estruturas de dados, como as tabelas, que compõem um banco de dados. Existem cinco tipos básicos de instruções DDL:
  - **`CREATE`** cria uma estrutura de banco de dados. Por exemplo, `CREATE TABLE` é usada para criar uma tabela; outro exemplo é `CREATE USER`, usada para criar um usuário do banco de dados.
  - **`ALTER`** modifica uma estrutura de banco de dados. Por exemplo, `ALTER TABLE` é usada para modificar uma tabela.
  - **`DROP`** remove uma estrutura de banco de dados. Por exemplo, `DROP TABLE` é usada para remover uma tabela.
  - **`RENAME`** muda o nome de uma tabela.
  - **`TRUNCATE`** exclui todas as linhas de uma tabela.

- **Instruções TC (Transaction Control)** registram permanentemente as alterações feitas em linhas ou desfazem essas alterações. Existem três instruções TC:
  - **`COMMIT`** registra permanentemente as alterações feitas em linhas.
  - **`ROLLBACK`** desfaz as alterações feitas em linhas.
  - **`SAVEPOINT`** define um "ponto de salvamento" no qual você pode reverter alterações.
- **Instruções DCL (Data Control Language)** alteram as permissões nas estruturas de banco de dados. Existem duas instruções DCL:
  - **`GRANT`** concede a outro usuário acesso às estruturas de seu banco de dados.
  - **`REVOKE`** impede que outro usuário acesse as estruturas de seu banco de dados.

Existem muitas maneiras de executar instruções SQL e obter resultados do banco de dados, algumas das quais incluem programas escritos usando o Oracle Foms e Reports. As instruções SQL também podem ser incorporadas em programas escritos em outras linguagens, como Pro*C++ da Oracle, que permite adicionar instruções SQL em um programa C++. Você também pode adicionar instruções SQL em um programa Java usando JDBC; para obter mais detalhes, consulte o livro *Oracle9i JDBC Programming* (Oracle Press, 2002).

A Oracle também tem uma ferramenta chamada SQL*Plus, que permite inserir instruções SQL usando o teclado ou executar um script contendo instruções SQL. O SQL*Plus possibilita "conversar" com o banco de dados; você digita instruções SQL e vê os resultados retornados pelo banco de dados. Apresentamos o SQL*Plus a seguir.

USANDO O SQL*PLUS

Se você conhece o banco de dados Oracle, é possível que já esteja familiarizado com o SQL*Plus. Se não estiver, não se preocupe; neste livro você vai aprender a usá-lo.

Nas seções a seguir, você vai aprender a iniciar o SQL*Plus e a executar uma consulta.

Iniciando o SQL*Plus

Se você está usando Windows XP Professional Edition e Oracle Database 11*g*, pode iniciar o SQL*Plus clicando no botão Iniciar e selecionando Programas | Oracle | Application development | SQLPlus.

A Figura 1-1 mostra o SQL*Plus em execução no Windows XP. O SQL*Plus pede um nome de usuário. A Figura 1-1 mostra o usuário `scott` conectando-se no banco de dados (`scott` é um exemplo de usuário contido em muitas versões do banco de dados Oracle; sua senha padrão é `tiger`). A string de host após o caractere `@` informa ao SQL*Plus onde o banco de dados está sendo executado. Se você estiver executando o banco de dados em seu próprio computador, normalmente omitirá a string de host (isto é, você digitará `scott/tiger`) — isso faz com que o SQL*Plus tente conectar a um banco de dados na mesma máquina em que está sendo executado. Se o banco de dados não estiver sendo executado em sua máquina, fale com o administrador do banco de dados (DBA) para obter a string de host. Se o usuário `scott` não existe ou está bloqueado, peça ao DBA um usuário e uma senha alternativos (para os exemplos da primeira parte deste capítulo, você pode utilizar qualquer usuário; não é necessário utilizar o usuário `scott`).

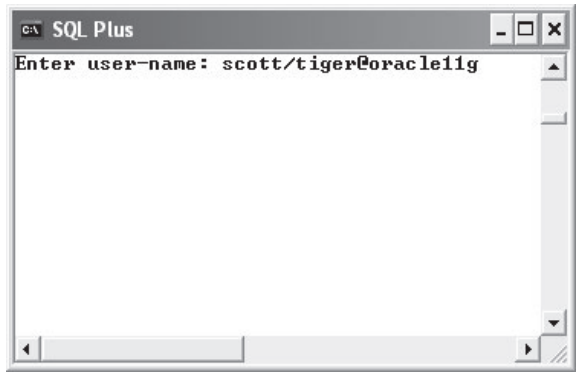
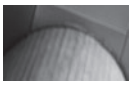


Figura 1-1 *SQL*Plus do Oracle Database 11g em execução no Windows XP.*

Se você esta usando Windows XP e Oracle Database 10*g* ou anterior, pode executar uma versão especial do SQL*Plus para Windows. Você inicia essa versão clicando em Iniciar e selecionando Programas | Oracle | Application Development | SQL Plus. A versão para Windows do SQL*Plus foi descontinuada no Oracle Database 11*g* (isto é, ela não vem com o 11*g*), mas ela ainda se conectará com um banco de dados 11*g*. A Figura 1-2 mostra a versão para Windows do SQL*Plus do Oracle Database 10*g* em execução no Windows XP.



NOTA

*A versão do SQL*Plus do Oracle Database 11g é ligeiramente mais refinada do que a versão para Windows. Na versão 11g, você pode percorrer os comandos executados anteriormente pressionando as teclas de seta para cima e seta para baixo no teclado.*

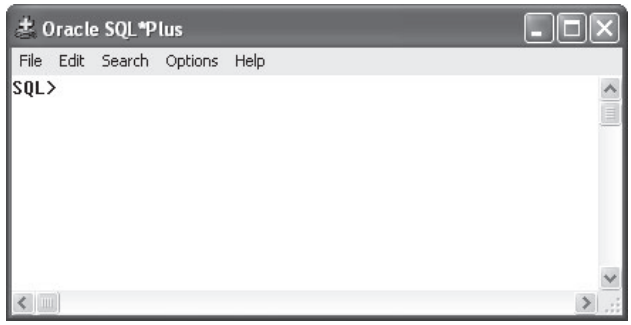


Figura 1-2 *SQL*Plus do Oracle Database 10g em execução no Windows XP.*

Iniciando o SQL*Plus a partir da linha de comando

Você também pode iniciar o SQL*Plus a partir da linha de comando. Para tanto, use o comando `sqlplus`, cuja sintaxe completa é

```
sqlplus [nome_usuário[/senha[@string_host]]]
```

onde

- *nome_usuário* é o nome do usuário do banco de dados
- *senha* é a senha do usuário do banco de dados
- *string_host* é o banco de dados em que você deseja se conectar

Os exemplos a seguir mostram comandos `sqlplus`:

```
sqlplus scott/tiger
sqlplus scott/tiger@orcl
```

Se você estiver usando o SQL*Plus com um sistema operacional Windows, o instalador Oracle adicionará automaticamente o diretório do SQL*Plus na sua variável de ambiente PATH. Se você estiver usando outro sistema operacional que não seja Windows (por exemplo, Unix ou Linux), deverá estar no mesmo diretório que o programa SQL*Plus para executá-lo ou, melhor ainda, deverá adicionar o diretório em sua variável de ambiente PATH. Se precisar de ajuda para fazer isso, fale com seu administrador de sistema.

Por segurança, você pode ocultar a senha quando se conectar ao banco de dados. Por exemplo, você pode digitar:

```
sqlplus scott@orcl
```

Neste momento, o SQL*Plus pedirá para que você digite a senha, que fica oculta durante a digitação. Isso também funciona ao iniciar o SQL*Plus no Windows.

Você também pode digitar apenas

```
sqlplus
```

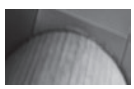
O SQL*Plus pedirá o nome de usuário e a senha. Você pode especificar a string de host adicionando-a no nome de usuário (por exemplo, `scott@orcl`).

Executando uma instrução SELECT usando o SQL*Plus

Quando você estiver conectado ao banco de dados usando o SQL*Plus, execute a seguinte instrução `SELECT` (ela retorna a data atual):

```
SELECT SYSDATE FROM dual;
```

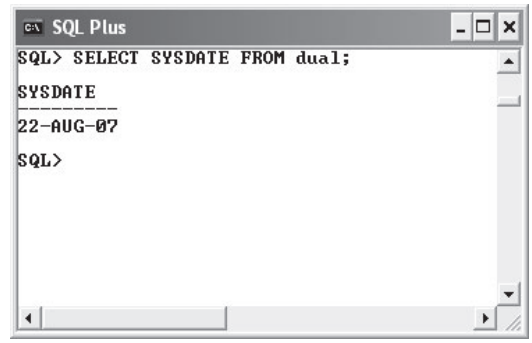
`SYSDATE` é uma função de banco de dados interna que retorna a data atual e `dual` é uma tabela que contém uma única linha. A tabela `dual` é útil quando você precisa que o banco de dados avalie uma expressão (por exemplo, 2 * 15/5) ou quando quer obter a data atual.



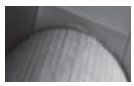
NOTA

*As instruções SQL digitadas diretamente no SQL*Plus são terminadas com um caractere de ponto-e-vírgula (;).*

Esta tela mostra o resultado dessa instrução SELECT no SQL*Plus em execução no Windows. Como você pode ver, a consulta exibe a data atual do banco de dados.



Você pode editar sua ultima instrução SQL no SQL*Plus, digitando EDIT. Isso é útil quando você comete um erro ou deseja fazer uma alteração em sua instrução SQL. No Windows, quando digita EDIT, o aplicativo Bloco de Notas abre; você o usa para editar sua instrução SQL. Quando fecha o Bloco de Notas e salva sua instrução, a nova instrução é passada ao SQL*Plus, onde pode ser novamente executada pela digitação de uma barra normal (/). No Linux ou no Unix, normalmente o editor padrão é definido como o vi ou emacs.

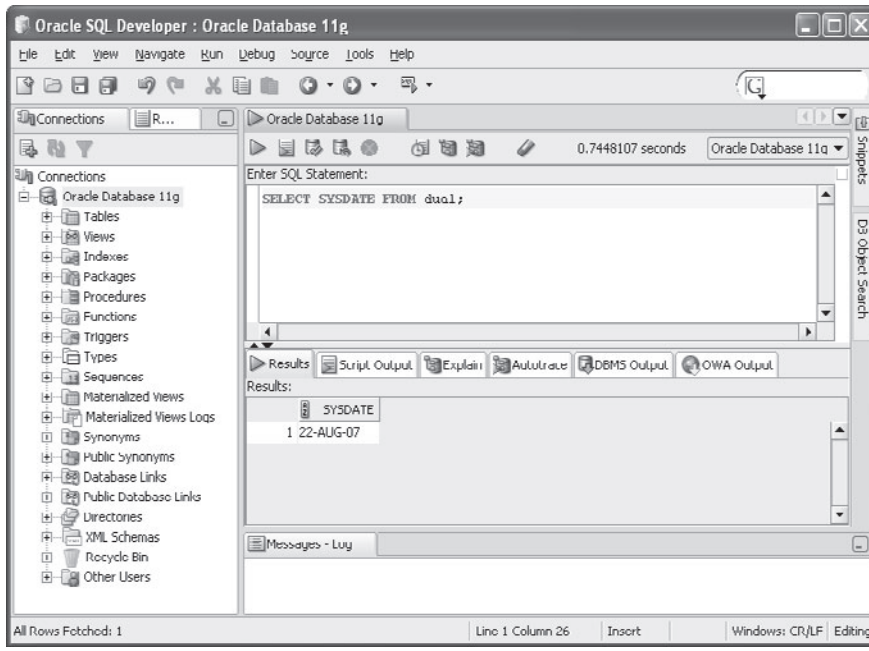


NOTA

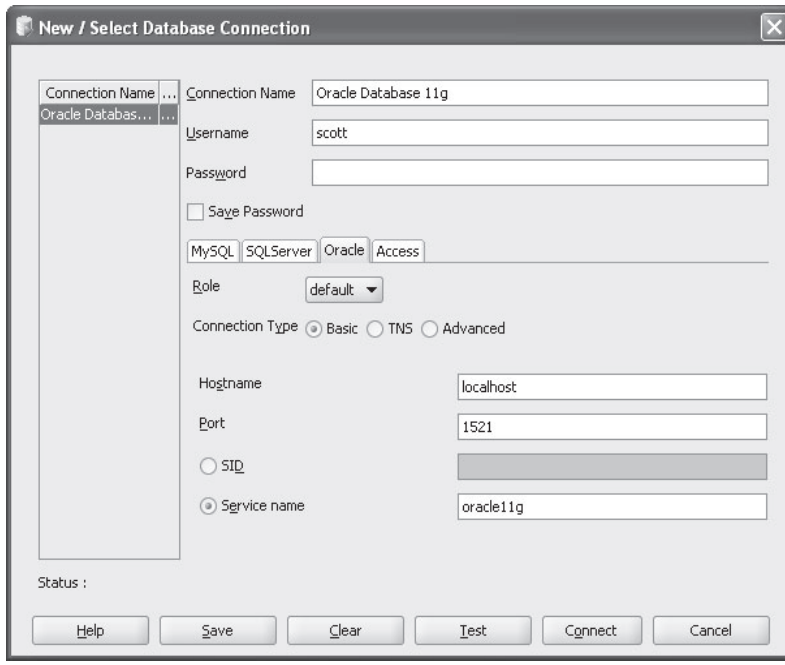
*Você vai aprender mais sobre a edição de instruções SQL com SQL*Plus no Capítulo 3.*

SQL DEVELOPER

Você também pode inserir instruções SQL usando o SQL Developer. O SQL Developer usa uma interface gráfica muito interessante, por meio da qual você pode inserir instruções SQL, examinar tabelas de banco de dados, executar scripts, editar e depurar código PL/SQL e muito mais. Ele pode conectar-se a qualquer banco de dados Oracle (versão 9.2.0.1 e superiores) e é executado no Windows, Linux e Mac OSX. A ilustração a seguir mostra o SQL Developer em execução.

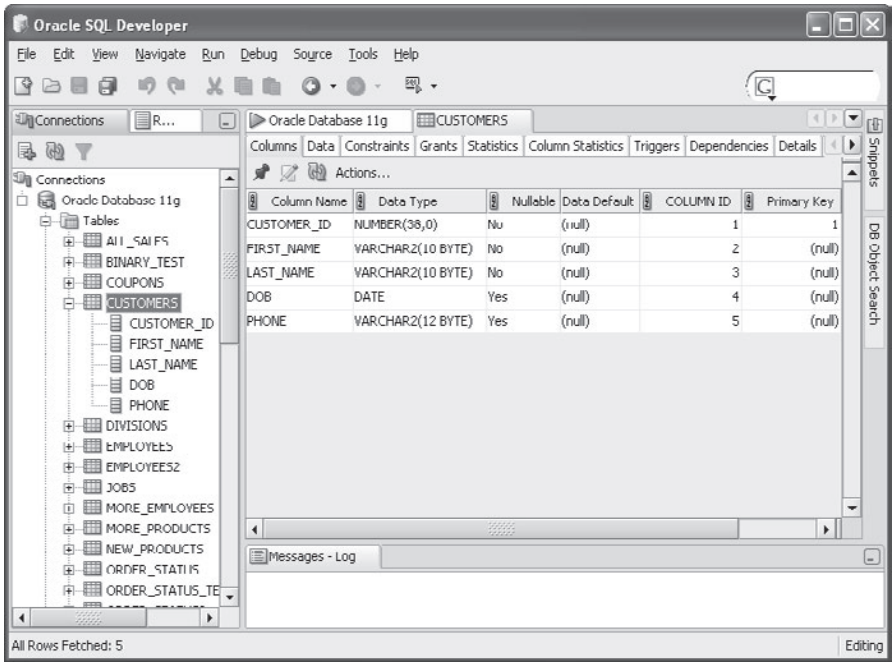


Antes de poder executar o SQL Developer, você precisa ter Java instalado em seu computador. Se estiver usando Windows XP Professional Edition e Oracle Database 11g, inicie o SQL Developer clicando em Iniciar e selecionando Programas | Oracle | Application Development | SQL Developer. O SQL Developer pedirá para que você selecione o executável Java. Navegue até o local onde o instalou e selecione o executável(*). Em seguida, crie uma conexão, clicando o botão direito do mouse em Connections e selecionando New Connection, como mostrado a seguir.

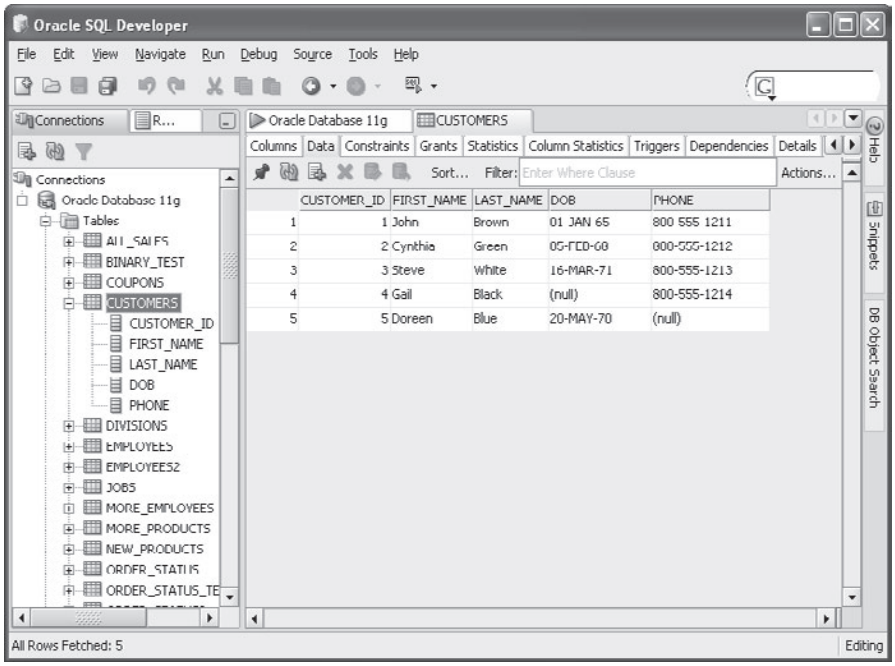


Quando tiver criado e testado uma conexão, você poderá usá-la para conectar-se ao banco de dados e executar consultas, examinar tabelas etc. A imagem a seguir mostra os detalhes de uma tabela de banco de dados chamada `customers`.

* N. de R.T.: Se você não tem o software Java instalado em sua máquina, pode fazer a instalação pelo site http://java.sun.com.



Você também pode exibir os dados armazenados em uma tabela, como mostrado abaixo.



Os detalhes completos sobre o uso do SQL Developer podem ser vistos selecionando Help | Table of Contents na barra de menus do programa.

Na próxima seção, você irá aprender a criar o esquema da loja imaginária usada neste livro.

CRIANDO O ESQUEMA DA LOJA

A loja imaginária vende artigos como livros, vídeos, DVDs e CDs. O banco de dados da loja conterá informações sobre clientes, funcionários, produtos e vendas. O script SQL*Plus para criar o banco de dados é chamado `store_schema.sql` e está localizado no diretório `SQL` onde você extraiu o arquivo Zip deste livro. O script `store_schema.sql` contém as instruções DDL e DML usadas para criar o esquema `store`. Agora você irá aprender a executar o script `store_ schema.sql`.

Executando o script SQL*Plus para criar o esquema da loja

Para criar o esquema `store`, execute os passos a seguir:

1. Inicie o SQL*Plus.
2. Conecte-se no banco de dados como um usuário com privilégios para criar novos usuários, tabelas e packages PL/SQL. Estes scripts podem ser executados utilizando o usuário `system`; esse usuário tem todos os privilégios necessários. Talvez você precise falar com o administrador de seu banco de dados para configurar um usuário com os privilégios necessários (ele também poderá executar o script `store_ schema.sql` para você).
3. Execute o script `store_ schema.sql` dentro do SQL*Plus, usando o comando `@`.

O comando `@` tem a seguinte sintaxe:

```
@ diretório\store_schema.sql
```

onde *`diretório`* é o diretório no qual seu script `store_ schema.sql` está localizado. Por exemplo, se o script está armazenado em `E:\sql_book\SQL`, digite

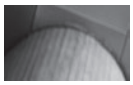
```
@ E:\sql_book\SQL\store_schema.sql
```

Se você tiver colocado o script `store_schema.sql` em um diretório que contém espaços, deve colocar o diretório e o script entre aspas após o comando `@`. Por exemplo:

```
@ "E:\Oracle SQL book\sql_book\SQL\store_schema.sql"
```

Se você estiver usando Unix ou Linux e salvou o script em um diretório chamado `SQL` no sistema de arquivos `tmp`, digite

```
@ /tmp/SQL/store_schema.sql
```

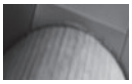


NOTA

O Windows usa caracteres de barra invertida (\) em caminhos de diretórios, já o Unix e o Linux usam caracteres de barra normal (/).

A primeira linha executável no script `store_schema.sql` tenta eliminar o usuário `store`, gerando um erro, porque o usuário ainda não existe. Não se preocupe: a linha está lá para que você não precise eliminar o usuário `store` manualmente, quando recriar o esquema mais adiante no livro.

Quando o script `store_schema.sql` tiver terminado de executar, você estará conectado como o usuário `store`. Se desejar, abra o script `store_schema.sql` usando um editor de textos como o Bloco de Notas do Windows e examine as instruções nele contidas. Não se preocupe com os detalhes das instruções contidas no script — você irá conhecê-los à medida que avançar neste livro.

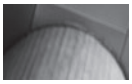


NOTA

*Para finalizar o SQL*Plus, digite* `EXIT`*. Para conectar-se novamente no esquema store no SQL*Plus, digite* `store` *como nome de usuário, com a senha* `store_password`*. Enquanto você está conectado no banco de dados, o SQL*Plus mantém uma sessão aberta. Quando se desconecta do banco de dados, sua sessão é finalizada. Você pode desconectar-se do banco de dados e manter o SQL*Plus em execução digitando* `DISCONNECT`*. Você pode conectar-se novamente digitando* `CONNECT`*.*

Instruções DDL (Data Definition Language) usadas para criar o esquema da loja

Conforme mencionado anteriormente, as instruções DDL (Data Definition Language) são usadas para criar usuários e tabelas, além de muitos outros tipos de estruturas no banco de dados. Nesta seção, você vai ver as instruções DDL usadas para criar o usuário `store` e algumas das tabelas.



NOTA

As instruções SQL que você verá no restante deste capítulo são as mesmas contidas no script `store_schema.sql`*. Você não precisa digitar as instruções, basta executar o script* `store_schema.sql`*.*

As próximas seções descrevem:

- Como criar um usuário de banco de dados
- Os tipos de dados comumente usados em um banco de dados Oracle
- Algumas das tabelas da loja imaginária

Criando um usuário de banco de dados

Para criar um usuário no banco de dados, use a instrução `CREATE USER`. A sintaxe simplificada da instrução `CREATE USER` é:

```
CREATE USER nome_usuário IDENTIFIED BY senha;
```

onde

- *`nome_usuário`* é o nome do usuário
- *`senha`* é a senha do usuário

Por exemplo, a instrução `CREATE USER` a seguir cria o usuário `store` com a senha `store_password`:

```
CREATE USER store IDENTIFIED BY store_password;
```

Se você quiser que o usuário possa trabalhar no banco de dados, ele deverá receber as *permissões* necessárias para realizar esse trabalho. No caso de `store`, esse usuário deve ser capaz de conectar-se no banco de dados (o que exige a permissão `connect`) e criar itens como tabelas de banco de dados (o que exige a permissão `resource`). As permissões são concedidas por um usuário privilegiado (por exemplo, o usuário `system`) usando a instrução `GRANT`.

O exemplo a seguir concede as permissões `connect` e `resource` para `store`:

```
GRANT connect, resource TO store;
```

Uma vez criado o usuário, as tabelas e outros objetos de banco de dados podem ser criados para esse usuário no esquema associado. Muitos exemplos deste livro usam o esquema `store`. Antes de entrarmos nos detalhes das tabelas da loja, você precisa saber a respeito dos tipos comumente usados do banco de dados Oracle.

Os tipos comuns do banco de dados Oracle

Existem muitos tipos que podem ser usados para manipular dados em um banco de dados Oracle. Alguns dos mais usados são mostrados na Tabela 1-1.

O Apêndice apresenta uma relação completa dos tipos de dados. A tabela a seguir ilustra alguns exemplos de como números de tipo `NUMBER` são armazenados no banco de dados.

Formato	Número fornecido	Número armazenado
`NUMBER`	1234.567	1234.567
`NUMBER(6, 2)`	123.4567	123.46
`NUMBER(6, 2)`	12345.67	O número excede a precisão especificada, portanto, é rejeitado pelo banco de dados.

Examinando as tabelas da loja

Nesta seção, você vai aprender como as tabelas do esquema `store` são criadas. Algumas das informações mantidas no esquema `store` incluem:

- Detalhes do cliente
- Tipos de produtos vendidos
- Detalhes do produto
- Um histórico dos produtos adquiridos pelos clientes
- Funcionários da loja
- Nível salarial

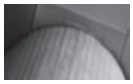
As tabelas a seguir são usadas para conter as informações:

- **`customers`** contém os detalhes dos clientes
- **`product_types`** contém os tipos de produtos vendidos pela loja

- **products** contém os detalhes dos produtos
- **purchases** mostra quais produtos foram adquiridos por quais clientes
- **employees** contém os detalhes dos funcionários
- **salary_grades** contém os detalhes dos níveis salariais

Tabela 1-1 *Tipos de dados comumente usados do Oracle*

Tipo do oracle	Significado
CHAR (*comprimento*)	Armazena strings de comprimento fixo. O parâmetro *comprimento* especifica o comprimento da string. Se uma string de comprimento menor for armazenada, ela será preenchida com espaços no final. Por exemplo, CHAR(2) pode ser usado para armazenar uma string de comprimento fixo de dois caracteres; se "C" for armazenado em CHAR(2), um espaço será adicionado no final; "CA" é armazenado como está, sem preenchimento.
VARCHAR2 (*comprimento*)	Armazena strings de comprimento variável. O parâmetro *comprimento* especifica o comprimento máximo da string. Por exemplo, VARCHAR2(20) pode ser usado para armazenar uma string de até 20 caracteres de comprimento. Nenhum preenchimento é usado no final de uma string menor.
DATE	Armazena data e horas. O tipo DATE armazena o século, todos os quatro dígitos de um ano, o mês, o dia, a hora (no formato de 24 horas), o minuto e o segundo. Ele pode ser usado para armazenar datas e horas entre 1° de janeiro de 4712 a.C. e 31 de dezembro de 4712 d.C.
INTEGER	Armazena valores inteiros. Um valor inteiro não contém um ponto flutuante: trata-se de um número inteiro, como 1, 10 e 115.
NUMBER (*precisão, escala*)	Armazena números de ponto flutuante, mas também pode ser usado para armazenar valores inteiros. A *precisão* é o número máximo de dígitos (à esquerda e à direita de um ponto decimal, se for usado) que podem ser usados para o número. A precisão máxima suportada pelo banco de dados Oracle é 38. A *escala* é o número máximo de dígitos à direita de um ponto decimal (se for usado). Se nem a *precisão* nem a *escala* forem especificadas, qualquer número poderá ser armazenado, até uma precisão de 38 dígitos. Qualquer tentativa de armazenar um número que ultrapasse a *precisão* será rejeitada pelo banco de dados.
BINARY_FLOAT	Lançado no Oracle Database 10*g*, armazena um número de ponto flutuante de 32 bits e precisão simples. Você vai aprender mais sobre BINARY_FLOAT posteriormente, na seção "Os tipos BINARY_FLOAT e BINARY_ DOUBLE".
BINARY_DOUBLE	Introduzindo no Oracle Database 10*g*, armazena um número de ponto flutuante de 64 bits e precisão dupla. Você vai aprender mais sobre BINARY_DOUBLE posteriormente, na seção "Os tipos BINARY_FLOAT e BINARY_ DOUBLE".



NOTA
O script `store_schema.sql` *cria outras tabelas e itens de banco de dados não mencionados na lista anterior. Você vai aprender sobre esses itens em capítulos posteriores.*

Nas seções a seguir, você vai ver os detalhes de algumas das tabelas e as instruções `CREATE TABLE` incluídas no script `store_schema.sql` que as criam.

A tabela customers A tabela `customers` contém os detalhes dos clientes. Nessa tabela estão contidos os seguintes itens:

- Nome
- Sobrenome
- Data de nascimento (dob)
- Número do telefone

Cada um destes itens exige uma coluna na tabela `customers`. A tabela `customers` é criada pelo script `store_schema.sql` usando a seguinte instrução `CREATE TABLE`:

```
CREATE TABLE customers (
  customer_id INTEGER CONSTRAINT customers_pk PRIMARY KEY,
  first_name VARCHAR2(10) NOT NULL,
  last_name VARCHAR2(10) NOT NULL,
  dob DATE,
  phone VARCHAR2(12)
);
```

Como você pode ver, a tabela `customers` contém cinco colunas, uma para cada item da lista anterior, e uma coluna adicional chamada `customer_id`. As colunas são:

- **`customer_id`** Contém um valor numérico único para cada linha da tabela. Cada tabela deve ter uma ou mais colunas que identificam cada linha exclusivamente; esta coluna (ou colunas) é conhecida como *chave primária*. A cláusula `CONSTRAINT` indica que a coluna `customer_id` é a chave primária. Uma cláusula `CONSTRAINT` restringe os valores armazenados em uma coluna. Para a coluna `customer_id`, as palavras-chaves `PRIMARY KEY` indicam que essa coluna deve conter um valor único para cada linha. Você também pode anexar um nome opcional em uma constraint, o qual deve vir imediatamente após a palavra-chave `CONSTRAINT` — por exemplo, `customers_pk`. Você sempre deve nomear suas constraints de chave primária para que, quando ocorrer um erro de restrição, seja fácil identificar onde ele aconteceu.
- **`first_name`** Contém o nome do cliente. A constraint `NOT NULL` é usada nessa coluna — isso significa que você deve fornecer um valor para `first_ name` ao adicionar ou modificar uma linha. Se a constraint `NOT NULL` for omitida, não é preciso fornecer um valor e a coluna poderá permanecer vazia.

- **last_name** Contém o sobrenome do cliente. Essa coluna é NOT NULL e, portanto, um valor deve ser fornecido ao se adicionar ou modificar uma linha.
- **dob** Contém a data nascimento do cliente. Não há uma constraint NOT NULL especificada para essa coluna; portanto, é pressuposto o valor padrão NULL e o valor é opcional ao se adicionar ou modificar uma linha.
- **phone** Contém o número do telefone do cliente. Este é um valor opcional.

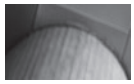
O script store_schema.sql preenche a tabela customers com as seguintes linhas:

```
customer_id first_name last_name  dob       phone
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65 800-555-1211
          2 Cynthia    Green      05-FEB-68 800-555-1212
          3 Steve      White      16-MAR-71 800-555-1213
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
```

Observe que a data de nascimento do cliente n° 4 é nula, assim como o número do telefone do cliente n° 5. Você pode ver as linhas da tabela customers executando a instrução SELECT a seguir com o SQL*Plus:

```
SELECT * FROM customers;
```

O asterisco (*) indica que você deseja recuperar todas as colunas das tabelas customers.



NOTA

Neste livro, as instruções SQL mostradas em **negrito** *são as que você deve digitar e executar se quiser acompanhar os exemplos. As instruções que não estão em negrito são as que você não precisa digitar.*

A tabela product_types A tabela product_types contém os nomes dos tipos de produtos vendidos pela loja. Essa tabela é criada pelo script store_schema.sql com a seguinte instrução CREATE TABLE:

```
CREATE TABLE product_types (
  product_type_id INTEGER CONSTRAINT product_types_pk PRIMARY KEY,
  name VARCHAR2(10) NOT NULL
);
```

A tabela product_types contém as duas colunas a seguir:

- **product_type_id** identifica exclusivamente cada linha da tabela; a coluna product_type_id é a chave primária dessa tabela. Cada linha da tabela product_types deve ter um valor inteiro único para a coluna product_type_id.
- **name** contém o nome do tipo de produto. Essa é uma coluna NOT NULL e, portanto, um valor deve ser fornecido ao se adicionar ou modificar uma linha.

O script `store_schema.sql` preenche a tabela `product_types` com as seguintes linhas:

```
product_type_id name
--------------- ----------
              1 Book
              2 Video
              3 DVD
              4 CD
              5 Magazine
```

A tabela `product_types` contém os tipos de produto da loja. Cada produto vendido pela loja deve ser de um desses tipos. Você pode ver as linhas da tabela `product_types` executando a instrução `SELECT` a seguir com o SQL* Plus:

```
SELECT * FROM product_types;
```

A tabela products A tabela `products` contém os produtos vendidos pela loja. Para cada produto são mantidas as seguintes informações:

- Tipo de produto
- Nome
- Descrição
- Preço

O script `store_schema.sql` cria a tabela products usando a seguinte instrução `CREATE TABLE`:

```
CREATE TABLE products (
  product_id INTEGER CONSTRAINT products_pk PRIMARY KEY,
  product_type_id INTEGER
    CONSTRAINT products_fk_product_types
    REFERENCES product_types(product_type_id),
  name VARCHAR2(30) NOT NULL,
  description VARCHAR2(50),
  price NUMBER(5, 2)
);
```

As colunas dessa tabela são:

- **`product_id`** identifica exclusivamente cada linha da tabela. Essa coluna é a chave primária da tabela.
- **`product_type_id`** associa cada produto a um tipo de produto. Essa coluna é uma referência à coluna `product_type_id` da tabela `product_types`; isso é conhecido como *chave estrangeira,* pois faz referência a uma coluna em outra tabela. A tabela que contém a chave estrangeira (a tabela `products`) é conhecida como tabela *detalhe* ou *filha* e a que é referenciada (a tabela `product_types`) é conhecida como tabela *mestre* ou *pai*. Esse tipo de relação é conhecido como *mestre-detalhe* ou *pai-filho*. Quando adiciona um novo

produto, você o associa a um tipo fornecendo um valor de `product_types.product_type_id` correspondente na coluna `products.product_type_id` (você verá um exemplo posteriormente).

- **`name`** contém o nome do produto, que deve ser especificado, pois a coluna name é `NOT NULL`.
- **`description`** contém uma descrição opcional do produto.
- **`price`** contém um preço opcional para um produto. Essa coluna é definida como `NUMBER (5, 2)` — a precisão é 5 e, portanto, no máximo cinco dígitos podem ser fornecidos para esse número. A escala é de 2, portanto, dois dígitos deste máximo de cinco podem estar à direita do ponto decimal.

A seguir está um subconjunto das linhas armazenadas na tabela `products`:

```
product_id   product_type_id name         description       price
---------- ---------------- ------------ ------------ ----------
         1                1 Modern       A                 19.95
                              Science     description
                                          of modern
                                          science

         2                1 Chemistry    Introduction         30
                                          to Chemistry

         3                2 Supernova    A star            25.99
                                          explodes

         4                2 Tank War     Action movie      13.95
                                          about a
                                          future war
```

A primeira linha da tabela `products` tem o valor `product_type_id` igual a 1, o que significa que o produto é um livro (esse valor de `product_type_id` corresponde ao tipo de produto "book" na tabela `product_types`). O segundo produto também é um livro, mas o terceiro e o quarto produtos são vídeos (seu `product_type_id` é 2, o que corresponde ao tipo de produto "video" na tabela `product_types`). Você pode ver todas as linhas da tabela `products` executando a instrução `SELECT` a seguir com o SQL*Plus.

```
SELECT * FROM products;
```

A tabela purchases A tabela `purchases` contém as compras feitas por um cliente. Para cada compra feita por um cliente, as seguintes informações são mantidas:

- Identificação do produto
- Identificação do cliente
- Número de unidades do produto adquiridas pelo cliente

O script `store_schema.sql` usa a seguinte instrução `CREATE TABLE` para criar a tabela `purchases`:

```
CREATE TABLE purchases (
  product_id INTEGER
    CONSTRAINT purchases_fk_products
    REFERENCES products(product_id),
  customer_id INTEGER
    CONSTRAINT purchases_fk_customers
    REFERENCES customers(customer_id),
  quantity INTEGER NOT NULL,
  CONSTRAINT purchases_pk PRIMARY KEY (product_id, customer_id)
);
```

As colunas dessa tabela são:

- **`product_id`** contém a identificação do produto que foi adquirido. Isso deve corresponder a um valor na coluna `product_id` da tabela `products`.
- **`customer_id`** contém a identificação do cliente que fez a compra. Isso deve corresponder a um valor na coluna `customer_id` da tabela `customers`.
- **`quantity`** contém o número de unidades do produto que foram adquiridas pelo cliente.

A tabela `purchases` tem uma restrição de chave primária chamada `purchases_pk` que abrange duas colunas: `product_id` e `customer_id`. A combinação dos dois valores de coluna deve ser única para cada linha. Quando uma chave primária consiste em várias colunas, ela é conhecida como chave primária *composta*.

A seguir está um subconjunto das linhas armazenadas na tabela `purchases`:

```
product_id customer_id   quantity
---------- ----------- ----------
         1           1          1
         2           1          3
         1           4          1
         2           2          1
         1           3          1
```

Como você pode ver, a combinação dos valores das colunas `product_id` e `customer_id` é única para cada linha. Você pode ver todas as linhas da tabela `purchases` executando a instrução `SELECT` a seguir com o SQL*Plus:

```
SELECT * FROM purchases;
```

A tabela employees A tabela `employees` contém os detalhes dos funcionários. As seguintes informações são mantidas na tabela:

- Identificação do funcionário

- Identificação do gerente do funcionário (se aplicável)
- Nome
- Sobrenome
- Cargo
- Salário

O script `store_schema.sql` usa a seguinte instrução `CREATE TABLE` para criar a tabela `employees`:

```
CREATE TABLE employees (
  employee_id INTEGER CONSTRAINT employees_pk PRIMARY KEY,
  manager_id INTEGER,
  first_name VARCHAR2(10) NOT NULL,
  last_name VARCHAR2(10) NOT NULL,
  title VARCHAR2(20),
  salary NUMBER(6, 0)
);
```

O script `store_schema.sql` preenche a tabela `employees` com as linhas a seguir:

```
employee_id manager_id first_name last_name  title            salary
----------- ---------- ---------- ---------- ------------- ----------
          1            James      Smith      CEO               800000
          2          1 Ron        Johnson    Sales Manager     600000
          3          2 Fred       Hobbs      Salesperson       150000
          4          2 Susan      Jones      Salesperson       500000
```

Como você pode ver, James Smith não tem gerente, pois ele é o diretor executivo da loja.

A tabela salary_grades A tabela `salary_grades` contém os diferentes níveis salariais disponíveis para os funcionários. São mantidas as seguintes informações:

- Identificação do nível salarial
- Limite salarial inferior para o nível
- Limite salarial superior para o nível

O script `store_schema.sql` usa a seguinte instrução `CREATE TABLE` para criar a tabela `salary_grades`:

```
CREATE TABLE salary_grades (
  salary_grade_id INTEGER CONSTRAINT salary_grade_pk PRIMARY KEY,
  low_salary  NUMBER(6, 0),
  high_salary NUMBER(6, 0)
);
```

O script `store_schema.sql` preenche a tabela `salary_grades` com as seguintes linhas:

```
salary_grade_id low_salary high_salary
--------------- ---------- -----------
              1          1      250000
              2     250001      500000
              3     500001      750000
              4     750001      999999
```

ADICIONANDO, MODIFICANDO E REMOVENDO LINHAS

Nesta seção, você vai aprender a adicionar, modificar e remover linhas em tabelas de banco de dados usando as instruções `INSERT`, `UPDATE` e `DELETE`. Você pode tornar suas alterações permanentes no banco de dados usando a instrução `COMMIT` ou desfazê-las com a instrução `ROLLBACK`. Esta seção não aborda todos os detalhes do uso dessas instruções; você vai aprender mais sobre elas no Capítulo 8.

Adicionando uma linha em uma tabela

A instrução `INSERT` é usada para adicionar novas linhas em uma tabela. Em uma instrução `INSERT`, você especifica as seguintes informações:

- A tabela na qual a linha vai ser inserida
- Uma lista de colunas para as quais você quer especificar valores
- Uma lista de valores a serem armazenados nas colunas especificadas

Ao inserir uma linha, você precisa fornecer um valor para a chave primária e para todas as outras colunas que são definidas como `NOT NULL`. Não é necessário especificar valores para as outras colunas, caso você não queira; essas colunas serão configuradas automaticamente como nulas se os seus valores forem omitidos. Você pode identificar quais colunas são definidas como `NOT NULL` usando o comando `DESCRIBE` no SQL*Plus. O exemplo a seguir utiliza o comando `DESCRIBE` na tabela `customers`:

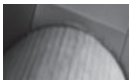
```
SQL> DESCRIBE customers
Name                                      Null?    Type
----------------------------------------- -------- ------------
CUSTOMER_ID                               NOT NULL NUMBER(38)
FIRST_NAME                                NOT NULL VARCHAR2(10)
LAST_NAME                                 NOT NULL VARCHAR2(10)
DOB                                                DATE
PHONE                                              VARCHAR2(12)
```

Como você pode ver, as colunas `customer_id, first_name`, e `last_name` são `NOT NULL`, significando que você deve fornecer um valor para elas. As colunas `dob` e `phone` não exigem um valor; se quiser, você pode omitir os valores e eles serão configurados automaticamente como nulos.

Execute a instrução `INSERT` a seguir, a qual adiciona uma linha na tabela `customers`; observe que a ordem dos valores na lista de `VALUES` corresponde à ordem na qual as colunas são especificadas na lista de colunas:

```
SQL> INSERT INTO customers (
  2    customer_id, first_name, last_name, dob, phone
```

```
  3  ) VALUES (
  4    6, 'Fred', 'Brown', '01-JAN-1970', '800-555-1215'
  5  );

1 row created.
```



NOTA
*O SQL*Plus enumera as linhas automaticamente depois que você pressiona ENTER no final de cada uma.*

No exemplo anterior, o SQL*Plus responde dizendo que uma linha foi criada após a instrução `INSERT` ser executada. Você pode verificar isso executando a seguinte instrução `SELECT`:

```
SELECT *
FROM customers;

CUSTOMER_ID FIRST_NAME LAST_NAME  DOB       PHONE
----------- ---------- ---------- --------- ------------
          1 John       Brown      01-JAN-65 800-555-1211
          2 Cynthia    Green      05-FEB-68 800-555-1212
          3 Steve      White      16-MAR-71 800-555-1213
          4 Gail       Black                800-555-1214
          5 Doreen     Blue       20-MAY-70
          6 Fred       Brown      01-JAN-70 800-555-1215
```

Observe a nova linha que foi adicionada no final da tabela.

Por padrão, o banco de dados Oracle exibe datas no formato `DD-MMM-AA`, onde `DD` é o número do dia, `MMM` são os três primeiros caracteres do mês (em maiúsculas) e `AA` são os dois últimos dígitos do ano. Na verdade, o banco de dados armazena todos os quatros dígitos do ano, mas, por padrão, ele exibe apenas os dois últimos.

Quando uma linha é adicionada na tabela `customers`, um valor exclusivo deve ser fornecido para a coluna `customer_id`. O banco de dados Oracle não permite adicionar uma linha com um valor de chave primária que já exista na tabela; por exemplo, a instrução `INSERT` a seguir causa um erro, pois já existe uma linha com `customer_id` igual a 1:

```
SQL> INSERT INTO customers (
  2    customer_id, first_name, last_name, dob, phone
  3  ) VALUES (
  4    1, 'Lisa', 'Jones', '02-JAN-1971', '800-555-1225'
  5  );

INSERT INTO customers (
*
ERROR at line 1:
ORA-00001: unique constraint (STORE.CUSTOMERS_PK) violated
```

Observe que o nome da constraint é mostrado no erro (`CUSTOMERS_PK`). É por isso que você sempre deve dar nomes às suas constraints de chave primária; caso contrário, o banco de dados Oracle atribuirá à constraint um nome não amigável, gerado pelo sistema (por exemplo, `SYS_C0011277`).

Modificando uma linha existente em uma tabela

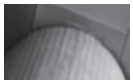
A instrução UPDATE é usada para alterar linhas em uma tabela. Normalmente, ao usar a instrução UPDATE, você especifica as seguintes informações:

- A tabela que contém as linhas a serem alteradas
- Uma cláusula WHERE especificando as linhas a serem alteradas
- Uma lista de nomes de colunas, junto com seus novos valores, especificados com a cláusula SET

Você pode alterar uma ou mais linhas usando a mesma instrução UPDATE. Se mais de uma linha for especificada, a mesma alteração será feita para todas as linhas. O exemplo a seguir atualiza o last_name do cliente n° 2 para Orange:

```
UPDATE customers
SET last_name = 'Orange'
WHERE customer_id = 2;

1 row updated.
```

O SQL*Plus confirma que uma linha foi atualizada.



CUIDADO

Se você se esquecer de adicionar uma cláusula WHERE*, todas as linhas serão atualizadas.*

A consulta a seguir confirma que a atualização funcionou:

```
SELECT *
FROM customers
WHERE customer_id = 2;

CUSTOMER_ID FIRST_NAME LAST_NAME   DOB       PHONE
----------- ---------- ---------- --------- ------------
          2 Cynthia    Orange     05-FEB-68 800-555-1212
```

Removendo uma linha de uma tabela

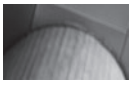
A instrução DELETE é usada para remover linhas de uma tabela. Normalmente, uma cláusula WHERE é utilizada para limitar as linhas que você deseja excluir; caso contrário, *todas* as linhas serão excluídas da tabela. A instrução DELETE a seguir remove o cliente n° 2:

```
DELETE FROM customers
WHERE customer_id = 2;

1 row deleted.
```

Para desfazer as alterações feitas nas linhas, use ROLLBACK:

```
ROLLBACK;

Rollback complete.
```

Execute a instrução ROLLBACK para desfazer todas as alterações que você fez até agora. Assim, seus resultados corresponderão àqueles mostrados nos próximos capítulos.



NOTA
É possível tornar as alterações nas linhas permanentes usando COMMIT. *Você vai aprender a fazer isso no Capítulo 8.*

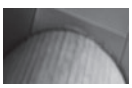
OS TIPOS BINARY_FLOAT E BINARY_DOUBLE

O Oracle Database 10*g* introduziu dois novos tipos de dados: BINARY_FLOAT e BINARY_DOUBLE. BINARY_FLOAT armazena um número de ponto flutuante de 32 bits e precisão simples; BINARY_DOUBLE armazena um número de ponto flutuante de 64 bits e precisão dupla. Esses novos tipos de dados são baseados no padrão IEEE (Institute of Electrical and Electronics Engineers) para aritmética binária de ponto flutuante.

Vantagens de BINARY_FLOAT e BINARY_DOUBLE

BINARY_FLOAT e BINARY_DOUBLE são destinados a complementar o tipo NUMBER existente. Eles oferecem as seguintes vantagens em relação a NUMBER.

- **Necessidade de armazenamento menor** BINARY_FLOAT e BINARY_DOUBLE exigem 5 e 9 bytes de espaço de armazenamento, enquanto NUMBER pode usar até 22 bytes.
- **Maior intervalo de números representados** BINARY_FLOAT e BINARY_DOUBLE suportam números muito maiores e menores do que os que podem ser armazenados em NUMBER.
- **Execução mais rápida de operações** Normalmente, as operações envolvendo BINARY_FLOAT e BINARY_DOUBLE são executadas de forma mais rápida do que as operações com NUMBER. Isso porque geralmente as operações com BINARY_FLOAT e BINARY_DOUBLE são executadas no hardware, enquanto os valores NUMBER devem ser primeiro convertidos usando software, antes que as operações possam ser executadas.
- **Operações fechadas** As operações aritméticas envolvendo BINARY_FLOAT e BINARY_DOUBLE são fechadas, o que significa que é retornado um número ou um valor especial. Por exemplo, se você dividir um BINARY_FLOAT por outro BINARY_FLOAT, um BINARY_FLOAT será retornado.
- **Arredondamento transparente** BINARY_FLOAT e BINARY_DOUBLE usam o sistema binário (base 2) para representar um número, enquanto NUMBER usa o sistema decimal (base 10). A base usada para representar um número afeta o seu arredondamento. Por exemplo, um número decimal de ponto flutuante é arredondado para a casa decimal mais próxima, mas um número binário de ponto flutuante é arredondado para a casa binária mais próxima.

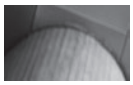


DICA
Se você estiver desenvolvendo um sistema que envolva muitos cálculos numéricos, use BINARY_FLOAT *e* BINARY_DOUBLE *para representar números. Você precisa usar Oracle Database 10g ou superior.*

Usando BINARY_FLOAT e BINARY_DOUBLE em uma tabela

A instrução a seguir cria uma tabela chamada `binary_test` que contém uma coluna `BINARY_FLOAT` e uma coluna `BINARY_DOUBLE`:

```
CREATE TABLE binary_test (
  bin_float BINARY_FLOAT,
  bin_double BINARY_DOUBLE
);
```



NOTA

No diretório SQL do arquivo zip com os códigos de exemplo, você encontrará um script chamado `oracle_10g_examples.sql` *que cria a tabela* `binary_test` *no esquema* `store`*. O script também executa as instruções* `INSERT` *apresentadas nesta seção. Você poderá executar esse script se estiver usando Oracle Database 10g ou superior.*

O exemplo a seguir adiciona uma linha na tabela binary_test:

```
INSERT INTO binary_test (
  bin_float, bin_double
) VALUES (
  39.5f, 15.7d
);
```

Observe que `f` indica que um número é `BINARY_FLOAT` e que `d` indica que um número é `BINARY_DOUBLE`.

Valores especiais

Com `BINARY_FLOAT` e `BINARY_DOUBLE`, você também pode usar os valores especiais mostrados na Tabela 1-2. O exemplo a seguir insere `BINARY_FLOAT_INFINITY` e `BINARY_DOUBLE_INFINITY` na tabela `binary_test`:

```
INSERT INTO binary_test (
  bin_float, bin_double
) VALUES (
  BINARY_FLOAT_INFINITY, BINARY_DOUBLE_INFINITY
);
```

Tabela 1-2 *Valores especiais*

Valor especial	Descrição
`BINARY_FLOAT_NAN`	Valor não-numérico (`NaN--Not a number`) para o tipo `BINARY_FLOAT`
`BINARY_FLOAT_INFINITY`	Infinito (`INF--Infinity`) para o tipo `BINARY_FLOAT`
`BINARY_DOUBLE_NAN`	Valor não-numérico (`NaN--Not a number`) para o tipo `BINARY_DOUBLE`
`BINARY_DOUBLE_INFINITY`	Infinito (`INF--Infinity`) para o tipo `BINARY_DOUBLE`

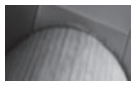
A consulta a seguir recupera as linhas de binary_test:

```
SELECT *
FROM binary_test;

 BIN_FLOAT BIN_DOUBLE
---------- ----------
 3.95E+001  1.57E+001
       Inf        Inf
```

SAINDO DO SQL*PLUS

Para sair do SQL*Plus, use o comando `EXIT`, como mostra o exemplo a seguir:

```
EXIT
```



NOTA

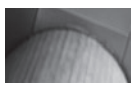
*Quando você sai do SQL*Plus usando este comando, ele automaticamente executa um* `COMMIT`*. Se o SQL*Plus terminar de forma anômala — por exemplo, se o computador em que ele está sendo executado falha — uma instrução* `ROLLBACK` *é executada automaticamente. Você aprenderá mais sobre isso no Capítulo 8.*

INTRODUÇÃO AO PL/SQL DA ORACLE

PL/SQL é a linguagem procedural da Oracle que permite adicionar construções de programação em torno de instruções SQL. Os códigos PL/SQL são usados principalmente para criar procedures e funções em um banco de dados que contenha a lógica do negócio. O código PL/SQL contém construções de programação padrão, como:

- Declarações de variável
- Lógica condicional (if-then-else etc.)
- Loops
- Procedures e funções

A instrução `CREATE PROCEDURE` a seguir cria uma procedure chamada `update_product_price()`, que multiplica o preço de um produto por um fator — a identificação do produto e o fator de reajuste são passados como parâmetros para a procedure. Se o produto especificado não existe, a procedure nada faz; caso contrário, ela atualiza o preço do produto.



NOTA

Não se preocupe com os detalhes do código PL/SQL mostrado na listagem a seguir — você vai aprender tudo sobre PL/SQL no Capítulo 11. Agora, o importante é que você tenha uma idéia do PL/SQL.

```
CREATE PROCEDURE update_product_price (
  p_product_id IN products.product_id%TYPE,
  p_factor     IN NUMBER
) AS
  product_count INTEGER;
```

```
BEGIN
  -- conta o número de produtos com
  -- product_id fornecido (será 1 se o produto existir)
  SELECT COUNT(*)
  INTO product_count
  FROM products
  WHERE product_id = p_product_id;

  -- se o produto existe (isto é, product_count = 1), então
  -- atualiza o preço desse produto
  IF product_count = 1 THEN
    UPDATE products
    SET price = price * p_factor
    WHERE product_id = p_product_id;
    COMMIT;
  END IF;
EXCEPTION
  WHEN OTHERS THEN
    ROLLBACK;
END update_product_price;
/
```

Os erros que ocorrem em códigos PL/SQL são tratados com exceções. No exemplo anterior, o bloco `EXCEPTION` executa uma instrução `ROLLBACK`, caso uma exceção seja lançada no código.

RESUMO

Neste capítulo, você aprendeu que:

- Um banco de dados relacional é uma coleção de informações relacionadas que foram organizadas em estruturas conhecidas como tabelas. Cada tabela contém linhas organizadas em colunas. Essas tabelas são armazenadas no banco de dados, em estruturas conhecidas como esquemas, que são áreas onde os usuários do banco de dados podem armazenar seus objetos (como tabelas e procedures PL/SQL).
- SQL (Structured Query Language) é a linguagem padrão projetada para acessar bancos de dados relacionais.
- O SQL*Plus permite executar instruções SQL e comandos SQL*Plus.
- O SQL Developer é uma ferramenta gráfica para desenvolvimento de bancos de dados.
- Como executar instruções `SELECT`, `INSERT`, `UPDATE` e `DELETE`.
- PL/SQL é a linguagem procedural da Oracle que contém instruções de programação.

No próximo capítulo, você vai aprender mais sobre recuperação de informações de tabelas de banco de dados.