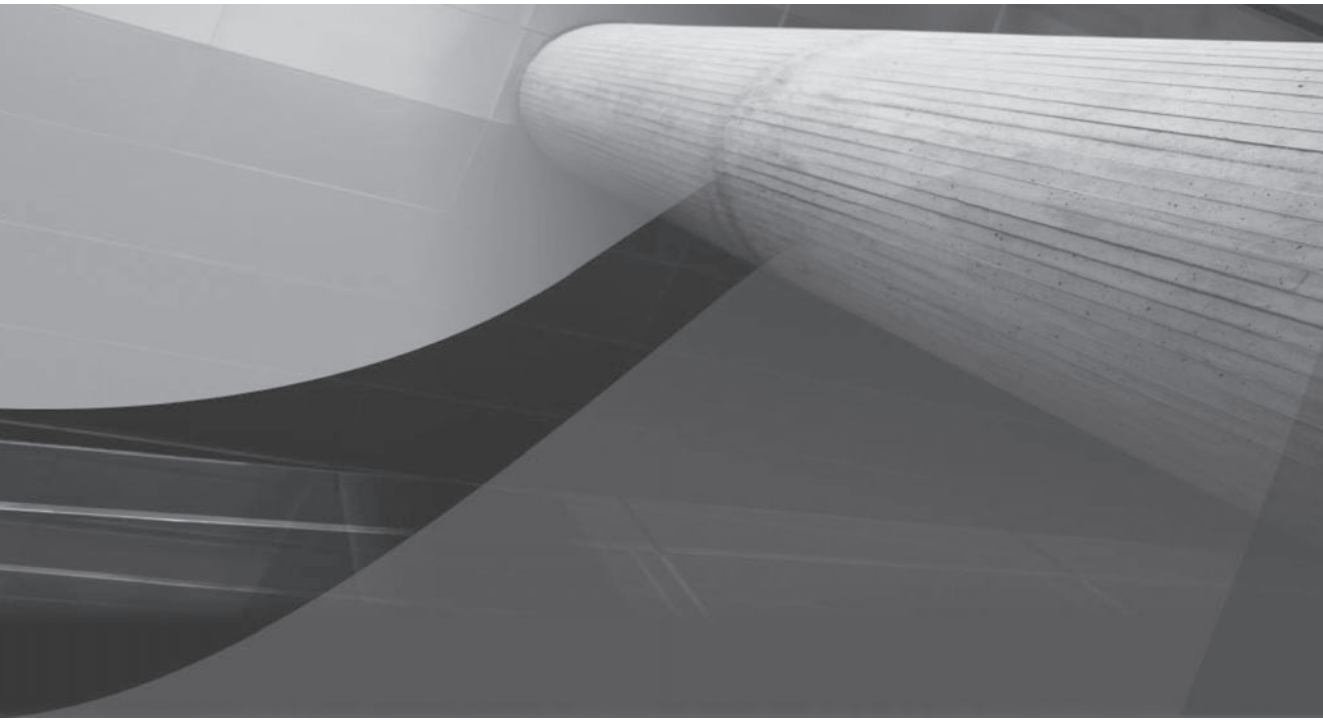
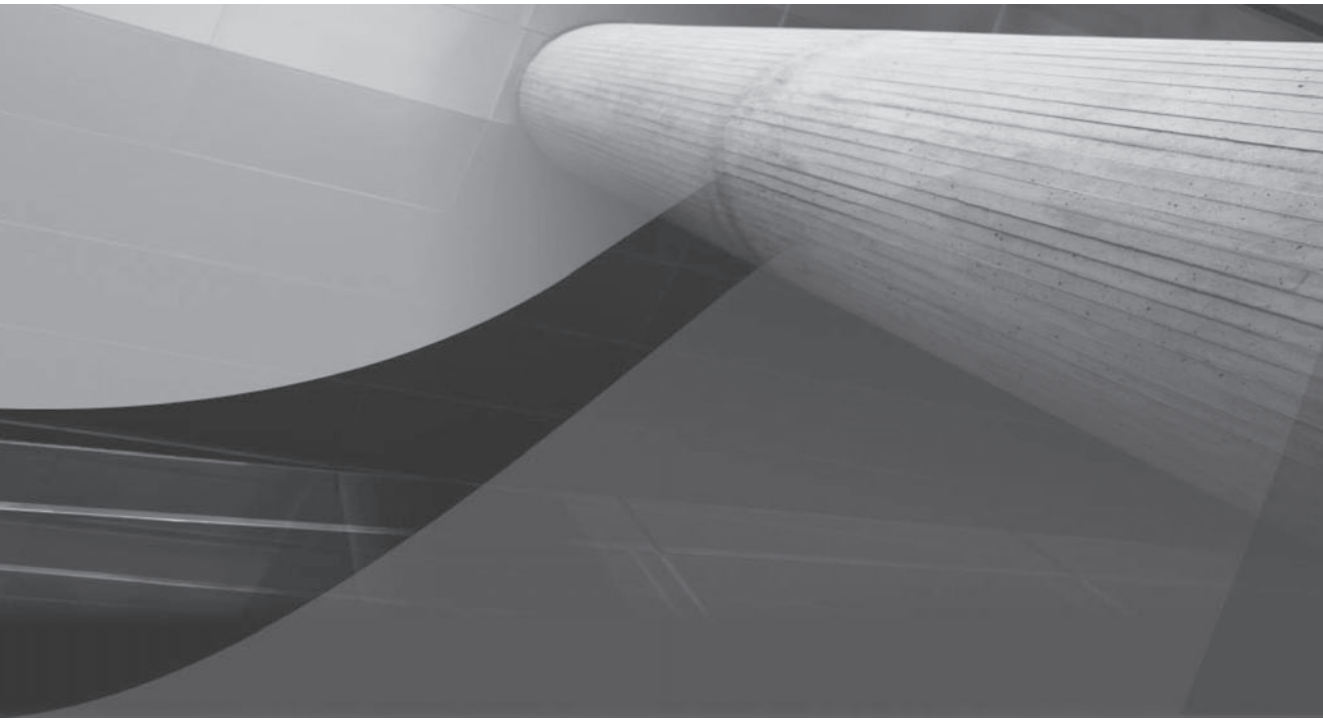


# PARTE I

# Arquitetura de banco de dados



# CAPÍTULO 1

# Apresentando a arquitetura do Oracle

O banco de dados Oracle 11*g* é uma evolução da versão anterior, o Oracle 10*g*, que foi, por sua vez, uma etapa revolucionária em relação ao Oracle9i em termos de recursos de configuração. O Oracle 11*g* continua a tradição de aprimoramento de recursos, tornando o gerenciamento de memória mais automatizado, adicionando vários novos advisors (supervisores) e melhorando a disponibilidade e funcionalidades de failover. A Parte I deste livro aborda os conceitos básicos da arquitetura e da implantação de uma infra-estrutura bem-sucedida do Oracle, com conselhos práticos para uma nova instalação ou atualização de uma versão anterior. Para fornecer uma boa base para o software Oracle 11*g*, abordaremos os problemas de configuração de sistema operacional e hardware de servidor nas seções relevantes.

Na Parte II deste livro, discutiremos sobre várias áreas relevantes para a manutenção e operação diária de um banco de dados Oracle 11*g*. O primeiro capítulo da Parte II relata os requisitos que um DBA precisa antes de montar o CD de instalação no seu servidor. Os capítulos seguintes tratam das maneiras como o DBA pode gerenciar o espaço em disco, o uso da CPU e o ajuste de parâmetros do Oracle para otimizar os recursos do servidor, usando diversas ferramentas à sua disposição para monitorar o desempenho do banco de dados. O gerenciamento de transações é bastante simplificado pelo Automated Undo Management (AUM), um recurso do Oracle Database introduzido no Oracle9*i* e aprimorado no Oracle 10*g* e Oracle 11*g*.

A Parte III deste livro enfoca os aspectos de alta disponibilidade do Oracle 11*g*. Isso inclui o uso do Recovery Manager (RMAN) do Oracle para executar e automatizar backups e recuperações de bancos de dados, juntamente com outros recursos, como o Oracle Data Guard, para fornecer um modo fácil e confiável de recuperar a partir de uma falha de banco de dados. Por último, mas não menos importante, mostraremos como o Real Application Clusters (RAC) do Oracle 11*g* pode, ao mesmo tempo, fornecer escalabilidade extrema e funcionalidades transparentes de failover para um ambiente de banco de dados. Mesmo que você não use os recursos do RAC do Oracle 11*g*, os recursos de standby o tornam quase tão disponível quanto uma solução clusterizada. A possibilidade de ser capaz de alternar facilmente entre o banco de dados standby e o principal, bem como consultar um banco de dados standby físico, proporciona uma solução de alta disponibilidade robusta até que você esteja pronto para implementar um banco de dados em RAC. Na Parte IV deste livro, trataremos sobre várias questões envolvendo o Networked Oracle (Oracle em rede). Discutiremos não apenas o modo como o Oracle Net pode ser configurado em um ambiente de *n* camadas, mas também como gerenciar bancos de dados grandes e distribuídos que poderiam residir em diferentes cidades ou países.

Neste capítulo, abordaremos os conceitos básicos do Oracle Database 11*g*, destacando grande parte dos recursos que discutiremos no restante do livro, bem como os fundamentos da instalação do Oracle 11*g* usando o Oracle Universal Installer (OUI) e o Database Configuration Assistant (DBCA). Falaremos a respeito dos elementos que compõe uma instância do Oracle 11*g*, explorando desde as estruturas de memória às estruturas de disco, parâmetros de inicialização, tabelas, índices e PL/SQL. Cada um desses elementos desempenha um papel importante para tornar o Oracle 11*g* um ambiente seguro, disponível e altamente escalonável.

## VISÃO GERAL DOS BANCOS DE DADOS E INSTÂNCIAS

Embora os termos "banco de dados" e "instância" sejam muitas vezes usados de forma alternada, eles são muito diferentes. Eles são entidades distintas em um ambiente Oracle, como você verá nas seções a seguir.

### Bancos de dados

Um *banco de dados* é uma coleção de dados em um ou mais arquivos no disco de um servidor que coleta e mantém informações relacionadas. O banco de dados consiste em várias estruturas físicas e lógicas, sendo a tabela a sua estrutura lógica mais importante. Uma tabela é composta de linhas e colunas que contêm dados relacionados. No mínimo, um banco de dados deve conter ao menos

tabelas para armazenar informações úteis. A Figura 1-1 mostra uma tabela de exemplo contendo quatro linhas e três colunas. Os dados em cada linha da tabela estão relacionados: cada linha contém informações sobre um funcionário específico da empresa.

Além disso, um banco de dados fornece níveis de segurança para impedir o acesso não autorizado aos dados. O Oracle Database 11*g* fornece muitos mecanismos para manter seguros dados confidenciais. Segurança e controle de acesso no banco de dados Oracle serão abordados mais detalhadamente no Capítulo 9.

Os arquivos que compõem um banco de dados se encaixam em duas categorias abrangentes: arquivos de banco de dados e arquivos não banco de dados. A distinção encontra-se no tipo de dado que está armazenado em cada um deles. Os arquivos de banco de dados contêm dados e metadados; os arquivos não banco de dados contêm parâmetros de inicialização, informações de log e assim por diante. Os arquivos de banco de dados são críticos para a operação contínua do banco de dados momento a momento. Cada uma dessas estruturas físicas de armazenamento será discutida posteriormente, na seção "Estruturas de armazenamento físico do Oracle".

## Instâncias

Os componentes principais de um servidor corporativo típico são um ou mais CPUs, espaço em disco e memória. Enquanto o banco de dados Oracle é armazenado em um disco do servidor, uma instância Oracle existe na memória do servidor. Uma *instância* Oracle é composta de um grande bloco de memória alocado em uma área denominada *System Global Area (SGA),* junto com vários processos em segundo plano que interagem entre a SGA e os arquivos de banco de dados no disco.

Em um Oracle Real Application Cluster (RAC), o mesmo banco de dados será usado por mais de uma instância. Embora as instâncias que compartilham o banco de dados possam estar no mesmo servidor, é mais provável que elas estejam em servidores separados conectados por uma interconexão de alta velocidade e que acessam um banco de dados que reside em um subsistema de disco especializado com RAID habilitado. Para obter mais detalhes sobre como uma instalação RAC é configurada, consulte o Capítulo 10.

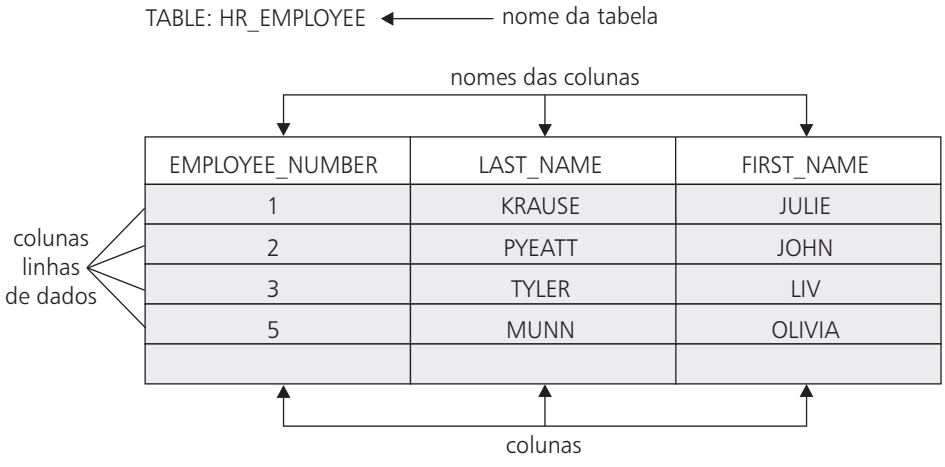



**Figura 1-1** *Tabela de banco de dados de exemplo.*

## ESTRUTURAS DE ARMAZENAMENTO LÓGICO DO ORACLE

Os arquivos de dados em um banco de dados Oracle são agrupados em uma ou mais tablespaces. Dentro de cada tablespace, as estruturas lógicas do banco de dados, como tabelas e índices, são segmentos subdivididos em ainda mais extensões e blocos. Essa subdivisão lógica do armazenamento permite ao Oracle controlar com mais eficiência o uso do espaço em disco. A Figura 1-2 mostra a relação entre as estruturas lógicas de armazenamento em um banco de dados.

### Tablespaces

Um *tablespace* Oracle consiste em um ou mais arquivos de dados; um arquivo de dados pode ser parte de apenas um tablespace. Durante uma instalação do Oracle 11*g* são criados um mínimo de dois tablespaces: o tablespace SYSTEM e o tablespace SYSAUX; uma instalação padrão do Oracle 11*g* cria seis tablespaces (consulte o apêndice "Instalação e configuração" para obter exemplos de instalações). O Oracle 11*g* permite criar um tipo especial de tablespace denominado *bigfile tablespace,* que pode ter até 128TB (terabytes). O uso de bigfiles torna o gerenciamento do tablespace completamente transparente para o DBA; em outras palavras, o DBA pode gerenciar o tablespace como uma unidade sem se preocupar com o tamanho e a estrutura dos arquivos de dados subjacentes.

Usar o Oracle Managed Files (OMF) facilita ainda mais o gerenciamento de arquivos de dados do tablespace. Com o OMF, o DBA especifica um ou mais locais no sistema de arquivos onde os arquivos de dados, arquivos de controle e arquivos de redo log residirão, e o Oracle trata automaticamente a nomeação e o gerenciamento desses arquivos. Discutiremos o OMF com mais detalhes no Capítulo 4.

Se um tablespace for *temporário,* ele em si será permanente; somente os segmentos salvos no tablespace são temporários. Um tablespace temporário pode ser usado para operações de classificação e para tabelas que existam apenas durante a sessão do usuário. Dedicar um tablespace a esses tipos de operações ajuda a reduzir a disputa de E/S entre segmentos temporários e segmentos permanentes armazenados em outro tablespace, como tabelas.

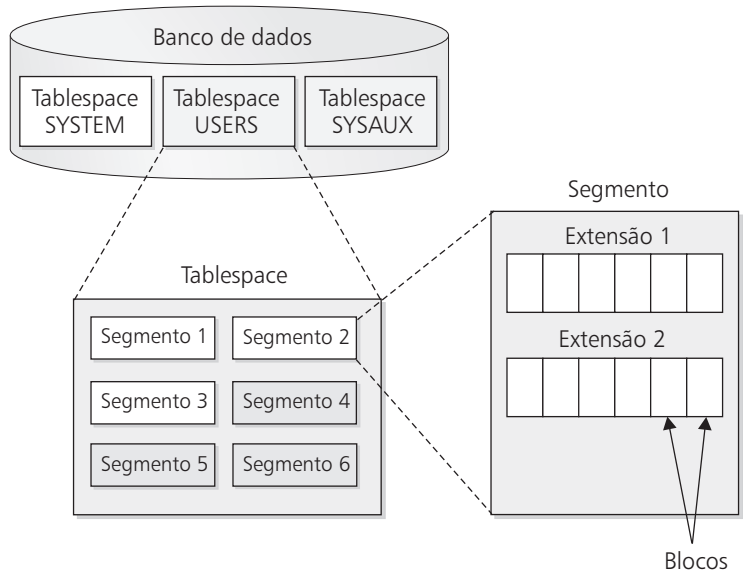



**Figura 1-2** *Estruturas de armazenamento lógico.*

Os tablespaces podem ser *gerenciados por dicionário* ou *gerenciados localmente*. Em um tablespace gerenciado por dicionário, o gerenciamento de extensões é registrado nas tabelas do dicionário de dados. Portanto, mesmo se todas as tabelas da aplicação estiverem no tablespace USERS, o tablespace SYSTEM ainda será acessado para gerenciar operações DML nas tabelas da aplicação. Como todos os usuários e aplicações devem usar o tablespace SYSTEM para o gerenciamento de extensões, isso cria um gargalo potencial para aplicações que fazem uso intenso de gravação. Em um tablespace gerenciado localmente, o Oracle mantém um bitmap em cada arquivo de dados do tablespace para monitorar a disponibilidade de espaço. Apenas quotas são gerenciadas no dicionário de dados, reduzindo muito a disputa para as tabelas de dicionários de dados.

A partir do Oracle9*i*, se o tablespace SYSTEM é gerenciado localmente, todos os outros tablespaces devem ser gerenciados da mesma forma, caso seja necessário executar operações de leitura e gravação neles. Os tablespaces gerenciados por dicionário devem ser somente para leitura em bancos de dados com um tablespace SYSTEM gerenciado localmente.

## Blocos

Um *bloco* de banco de dados é a menor unidade de armazenamento no banco de dados Oracle. O tamanho de um bloco é um número específico de bytes de armazenamento dentro de um determinado tablespace, dentro do banco de dados.

Um bloco é normalmente um múltiplo do tamanho de bloco do sistema operacional para facilitar a E/S de disco eficiente. O tamanho de bloco padrão é especificado pelo parâmetro de inicialização do Oracle DB_BLOCK_SIZE. Outros quatro tamanhos podem ser definidos para outros tablespaces no banco de dados, embora os blocos nos tablespaces SYSTEM, SYSAUX e em qualquer tablespace temporário devam ser do tamanho DB_BLOCK_SIZE.

## Extensões

A *extensão* é o nível seguinte de agrupamento lógico no banco de dados. Uma extensão consiste em um ou mais blocos de banco de dados. Quando um objeto de banco de dados é expandido, o espaço adicionado ao objeto é alocado como uma extensão.

## Segmentos

O próximo nível de agrupamento lógico em um banco de dados é o *segmento*. Um segmento é um grupo de extensões que abrange um objeto de banco de dados tratado pelo Oracle como uma unidade, por exemplo, uma tabela ou índice. Como resultado, essa é normalmente a menor unidade de armazenamento com a qual um usuário final do banco de dados lidará. Quatro tipos de segmentos são encontrados em um banco de dados Oracle: segmentos de dados, segmentos de índices, segmentos temporários e segmentos de rollback.

### *Segmento de dados*

Cada tabela no banco de dados reside em um único *segmento de dados*, consistindo em uma ou mais extensões; o Oracle alocará mais de um segmento para uma tabela se ela for particionada ou clusterizada. As tabelas particionadas e clusterizadas serão discutidas mais adiante neste capítulo. Os segmentos de dados incluem segmentos de LOB (large object) que armazenam os dados do LOB referenciados por uma coluna Localizador de LOB em um segmento de tabela (caso o LOB não esteja armazenado inline na tabela).

### *Segmento de índice*

Cada índice é armazenado no seu próprio *segmento de índice*. Como nas tabelas particionadas, cada partição de um índice é armazenado no seu próprio segmento. Incluídos nesta categoria estão os segmentos de índice de LOB; colunas não LOB de uma tabela, colunas LOB de uma tabela e os índices associados de LOBs podem residir em seus próprios tablespaces para melhorar o desempenho.

### *Segmento temporário*

Quando uma instrução SQL de um usuário precisa de espaço em disco para completar uma operação, como uma operação de classificação que não encontra memória suficiente, o Oracle aloca um *segmento temporário*. Os segmentos temporários têm a mesma duração das instruções SQL.

### *Segmento de rollback*

A partir do Oracle 10*g*, os segmentos de rollback só existem no tablespace SYSTEM, e normalmente o DBA não precisa manter o segmento de rollback SYSTEM. Nas versões anteriores do Oracle, um segmento de rollback era criado para salvar os valores anteriores de uma operação DML do banco de dados, se a transação fosse revertida (rollback), e para manter os dados da imagem do "antes" para fornecer visões consistentes de leitura dos dados da tabela para outros usuários que a estivessem acessando. Os segmentos de rollback eram também usados durante a recuperação do banco de dados para aplicar rollback em transações não encerradas com commit, que estivessem ativas quando a instância do banco de dados travasse ou terminasse inesperadamente.

O Automatic Undo Management (Gerenciamento de Undo Automático) trata a alocação automática e o gerenciamento de segmentos de rollback dentro de um tablespace de undo. Dentro de um tablespace de undo, esses segmentos de undo são estruturados de maneira similar aos segmentos de rollback, exceto que os detalhes de como esses segmentos são gerenciados estão sob o controle do Oracle, em vez de serem gerenciados (freqüentemente de forma ineficiente) pelo DBA. Os segmentos de undo automáticos estão disponíveis desde o Oracle9*i*, mas os segmentos de rollback gerenciados manualmente ainda estão disponíveis no Oracle 10*g*. Entretanto, esta funcionalidade é obsoleta no Oracle 10*g* e não estará mais disponível nas versões futuras. No Oracle 11*g*, o Automatic Undo Management está ativado por padrão; além disso, uma procedure PL/SQL é fornecida para ajudar a dimensionar o tablespace UNDO. O Automatic Undo Management será discutido em detalhes no Capítulo 7.

## ESTRUTURAS LÓGICAS DO BANCO DE DADOS DO ORACLE

Nesta seção, discutiremos os pontos mais importantes de todas as principais estruturas lógicas do banco de dados, iniciando com tabelas e índices. Em seguida, discutiremos os vários tipos de dados que podemos usar para definir as colunas de uma tabela. Quando criamos uma tabela com colunas, podemos impor *constraints* (restrições) sobre as colunas.

Uma das muitas razões pelas quais usamos um sistema de gerenciamento de banco de dados relacional (RDBMS) para gerenciar nossos dados é tirar proveito da segurança e dos recursos de auditoria do banco de dados Oracle. Examinaremos as maneiras como poderemos segregar o acesso ao banco de dados pelo usuário ou pelo objeto que está sendo acessado.

Além disso, mencionaremos muitas outras estruturas lógicas que podem ser definidas tanto pelo DBA quanto pelo usuário, incluindo sinônimos, links a arquivos externos e a outros bancos de dados.

### Tabelas

Uma *tabela* é a unidade básica de armazenamento em um banco de dados Oracle. Sem tabelas, um banco de dados não tem valor algum para uma empresa. Independentemente do tipo de tabela, os seus dados são armazenados em *linhas* e *colunas*, similar ao modo como os dados são armazenados em uma planilha. Mas as semelhanças terminam aí. A robustez de uma tabela de banco de dados devida à confiabilidade, integridade e escalabilidade do ambiente do banco de dados Oracle torna uma planilha uma segunda opção ineficiente ao decidir sobre um local para armazenar informações importantes.

Nesta seção, examinaremos os muitos tipos diferentes de tabelas do banco de dados Oracle e como elas podem atender à maioria das necessidades de armazenamento de dados para uma empresa. Você pode encontrar mais detalhes sobre como escolher entre esses tipos de tabelas para uma aplicação específica e como gerenciá-las no Capítulo 5 e no Capítulo 8.

### *Tabelas relacionais*

Uma tabela *relacional* é o tipo mais comum de tabela em um banco de dados. Uma tabela relacional é uma tabela não ordenada, ou organizada por heap (HOT – Heap Organized Table); em outras palavras, suas linhas não são armazenadas em uma ordem específica. No comando **create table**, você pode especificar a cláusula **organization heap** para definir uma tabela não ordenada, mas como esse é o padrão, a cláusula pode ser omitida.

Cada linha de uma tabela contém uma ou mais colunas; cada coluna tem um tipo de dados e um comprimento. A partir do Oracle versão 8, uma coluna pode conter um tipo de objeto definido pelo usuário, uma tabela aninhada ou um VARRAY. Além disso, uma tabela pode ser definida como uma tabela de objetos. Examinaremos as tabelas de objetos e objetos mais adiante nesta seção.

Os tipos de dados internos do Oracle são apresentados na Tabela 1-1. O Oracle também dá suporte a tipos de dados compatíveis com ANSI; o mapeamento entre os tipos de dados ANSI e os tipos de dados Oracle é fornecido na Tabela 1-2.

**Tabela 1-1** *Tipos de dados internos do Oracle*

| Tipos de dados internos do Oracle | Descrição |
|---|---|
| VARCHAR2 (*tamanho*) [BYTE \| CHAR] | Um conjunto de caracteres de comprimento variável com no máximo 4000 bytes e no mínimo 1 byte. CHAR indica que a semântica de caracteres é usada para dimensionar a string; BYTE indica que a semântica de bytes é usada. |
| NVARCHAR2(*tamanho*) | Um conjunto de caracteres de comprimento variável com no máximo 4000 bytes. |
| NUMBER(p,s) | Um número com uma precisão (*p*) e escala (*s*). A precisão varia de 1 a 38 e a escala pode ser de –84 a 127. |
| LONG | Um dado de caractere de comprimento variável com um comprimento até 2GB ($2^{31}$–1). |
| DATE | Valores de data de 1º de janeiro de 4712 A.C. a 31 de dezembro de 9999 A.D. |
| BINARY_FLOAT | Um número de ponto flutuante de 32 bits. |
| BINARY_DOUBLE | Um número de ponto flutuante de 64 bits. |
| TIMESTAMP (*segundos_fracionários*) | Ano, mês, dia, hora, minuto, segundo e segundos fracionários. O valor de *segundos_fracionários* pode variar de 0 a 9; em outras palavras, até a precisão de um bilionésimo de um segundo. O padrão é 6 (um milionésimo). |
| TIMESTAMP (*segundos_fracionários*) WITH TIME ZONE | Contém um valor de TIMESTAMP além de um valor de deslocamento de fuso horário. O deslocamento de fuso horário pode ser um deslocamento UTC (como "-06:00") ou um nome de região (por exemplo, "US/Central"). |
| TIMESTAMP (*segundos_fracionários*) WITH LOCAL TIME ZONE | Similar a TIMESTAMP WITH TIMEZONE, exceto que (1) os dados são normalizados de acordo com o fuso horário do banco de dados quando são armazenados e (2) durante a recuperação de colunas com esse tipo de dados, o usuário vê os dados no fuso horário da sessão. |
| INTERVAL YEAR (*precisão_do_ano*) TO MONTH | Armazena um período de tempo em anos e meses. O valor de *precisão_do_ano* é o número de dígitos do campo YEAR. |
| INTERVAL DAY (*precisão_do_dia*) TO SECOND (*precisão_dos_segundos_fracionários*) | Armazena um período de tempo como dias, horas, minutos, segundos e segundos fracionários. O valor para *precisão_do_dia* varia de 0 a 9, com um padrão igual a 2. O valor de *precisão_dos_segundos_fracionários* é similar aos segundos fracionários em um valor TIMESTAMP; o intervalo é de 0 a 9, com um padrão igual a 6. |
| RAW(*tamanho*) | Dados binários brutos, com um tamanho máximo de 2000 bytes. |

(*continua*)

**Tabela 1-1** *Tipos de dados internos do Oracle (continuação)*

| Tipos de dados internos do Oracle | Descrição |
|---|---|
| LONG RAW | Dados binários brutos, comprimento variável, até 2GB. |
| ROWID | Uma string em base 64 representando o endereço único de uma linha na sua tabela correspondente. Esse endereço é exclusivo em todo o banco de dados. |
| UROWID [(*tamanho*)] | Uma string de base 64 representando o endereço lógico de uma linha em uma tabela organizada por índices. O *tamanho* máximo é 4000 bytes. |
| CHAR(*tamanho*) [ BYTE \| CHAR ] | Uma string de caractere de comprimento fixo, cujo comprimento corresponde ao argumento *tamanho*. O tamanho mínimo é 1 e o máximo é 2000 bytes. Os parâmetros BYTE e CHAR são semântica de BYTE e CHAR, como em VARCHAR2. |
| NCHAR(*tamanho*) | Uma string de caractere de comprimento fixo de até 2000 bytes; o argumento *tamanho* máximo depende da definição do conjunto de caracteres nacional para o banco de dados. O argumento *tamanho* padrão é igual a 1. |
| CLOB | Um Character Large Object contendo caracteres single-byte ou multibytes; suporta conjuntos de caracteres de largura fixa ou de largura variável. O tamanho máximo é (4GB – 1) * DB_BLOCK_SIZE. |
| NCLOB | Similar ao CLOB, exceto que caracteres Unicode são armazenados tanto de conjuntos de caracteres de largura fixa quanto de largura variável. O tamanho máximo é (4GB – 1) * DB_BLOCK_SIZE. |
| BLOB | Um Binary Large Object; o tamanho máximo é (4GB – 1) * DB_BLOCK_SIZE. |
| BFILE | Um ponteiro para um Large Binary File armazenado fora do banco de dados. Arquivos binários devem ser acessíveis a partir do servidor que executa a instância Oracle. O tamanho máximo é de 4GB. |

**Tabela 1-2** *Tipos de dados Oracle equivalente ao ANSI*

| Tipo de dados ANSI SQL | Tipo de dados Oracle |
|---|---|
| CHARACTER(n)<br>CHAR(n) | CHAR(n) |
| CHARACTER VARYING(n)<br>CHAR VARYING(n) | VARCHAR(n) |
| NATIONAL CHARACTER(n)<br>NATIONAL CHAR(n)<br>NCHAR(n) | NCHAR(n) |
| NATIONAL CHARACTER VARYING(n)<br>NATIONAL CHAR VARYING(n)<br>NCHAR VARYING(n) | NVARCHAR2(n) |
| NUMERIC(p,s)<br>DECIMAL(p,s) | NUMBER(p,s) |
| INTEGER<br>INT<br>SMALLINT | NUMBER(38) |
| FLOAT(b)<br>DOUBLE PRECISION<br>REAL | NUMBER |

### *Tabelas temporárias*

As tabelas temporárias estão disponíveis desde o Oracle8*i*. Elas são temporárias no que diz respeito aos dados que armazenam, não à definição da tabela propriamente dita. O comando **create global temporary table** cria uma tabela desse tipo.

Desde que outros usuários tenham permissões para a acessar a tabela, eles podem executar instruções **select** ou comandos DML (Data Manipulation Language Commands), como **insert**, **update** ou **delete**, em uma tabela temporária. Entretanto, cada usuário vê apenas seus próprios dados na tabela. Quando um usuário trunca uma tabela temporária, apenas os dados que ele inseriu são removidos dela.

Existem dois tipos diferentes de dados em uma tabela temporária: temporários em relação à duração da transação e temporários em relação à duração da sessão. A longevidade dos dados temporários é controlada pela cláusula **on commit**; **on commit delete rows** remove todas as linhas da tabela temporária quando um comando **commit** ou **rollback** é emitido e **on commit preserve rows** mantém as linha na tabela além do limite da transação. Entretanto, quando a sessão do usuário termina, todas as linhas do usuário na tabela temporária são removidas.

Existem algumas outras coisas que é bom lembrar ao usar tabelas temporárias. Embora seja possível criar um índice em uma tabela temporária, as entradas no índice são deletadas juntamente com as linhas de dados, como em uma tabela normal. Além disso, devido à natureza temporária dos dados em uma tabela temporária, nenhuma informação de redo é gerada para operações DML nelas; entretanto, as informações de undo são criadas no tablespace de undo.

### *Tabelas organizadas por índice*

Como você descobrirá mais tarde, na subseção sobre índices, a criação de um índice torna mais eficiente a localização de uma linha específica em uma tabela. Entretanto, isso aumenta um pouco o overhead, porque o banco de dados precisa atualizar as linhas de dados e as entradas de índice para a tabela. E se sua tabela não tiver muitas colunas e o acesso à tabela ocorrer principalmente em uma única coluna? Nesse caso, uma *tabela organizada por índice (IOT, index-organized table)* poderia ser a solução correta. Uma tabela organizada por índice armazena as linhas de uma tabela em um índice de árvore B (B-tree), onde cada nó do índice de árvore B contém uma coluna indexada junto com uma ou mais colunas não indexadas.

A vantagem mais óbvia de uma tabela organizada por índice é que apenas uma estrutura de armazenamento precisa ser atualizada em vez de duas; da mesma maneira, os valores para a chave primária da tabela são armazenados apenas uma vez em uma tabela organizada por índice, versus duas vezes em uma normal.

Entretanto, existem algumas desvantagens ao usar uma tabela organizada por índice. Algumas tabelas, como as de eventos de logging, podem não precisar de uma chave primária ou de quaisquer chaves; uma tabela organizada por índice deve ter uma chave primária. Além disso, as tabelas organizadas por índice não podem ser membro de um cluster. Finalmente, uma tabela organizada por índice não é a melhor solução se houver um grande número de colunas na tabela e grande parte delas for freqüentemente acessada quando suas linhas forem recuperadas.

### *Tabelas de objetos*

Desde o Oracle8, o banco de dados Oracle tem dado suporte a muitos recursos orientados a objetos no banco de dados. Os tipos definidos pelo usuário, junto com quaisquer métodos definidos para esses tipos de objetos, podem permitir uma implementação sem falhas de um projeto de desenvolvimento orientado a objeto (OO) no Oracle.

As tabelas de objetos têm linhas que são objetos propriamente ditos, ou instanciações de definições de tipos. As linhas em uma tabela de objetos podem ser referenciadas pela ID do objeto (OID, object ID), ao contrário de uma chave primária em uma tabela relacional ou normal; entretanto, as tabelas de objetos podem ter ainda chaves primárias e únicas, exatamente como as relacionais.

Vamos supor, por exemplo, que você esteja criando um sistema de RH (Recursos Humanos) a partir do zero, portanto, tem a flexibilidade de projetar o banco de dados a partir de um ponto de vista inteiramente orientado a objeto. A primeira etapa é definir um objeto ou tipo funcionário, criando o tipo:

```
create type PERS_TYP as object
   (Last_Name        varchar2(45),
   First_Name        varchar2(30),
   Middle_Initial    char(1),
   Surname           varchar2(10),
   SSN               varchar2(15));
```

Nesse caso específico, você não está criando um método com o objeto PERS_TYP, mas, por padrão, o Oracle cria um método construtor para o tipo que tem o mesmo nome do tipo (nesse caso, PERS_TYP). Para criar uma tabela de objetos como uma coleção de objetos PERS_TYP, use a sintaxe familiar **create table**, como a seguir:

```
create table pers of pers_typ;
```

Para adicionar uma instância de um objeto à tabela de objetos, você pode especificar o método constructor no comando **insert**:

```
insert into pers
   values(pers_typ('Graber','Martha','E','Ms.','123-45-6789'));
```

Desde o Oracle Database 10*g*, não é necessário usar o método construtor se a tabela for composta de instâncias de um único objeto; aqui está a sintaxe simplificada:

```
insert into pers values('Graber','Martha','E','Ms.','123-45-6789');
```

Referências a instâncias do objeto PERS_TYP podem ser armazenadas em outras tabelas como objetos REF e os dados da tabela PERS podem ser recuperados sem uma referência direta à tabela PERS. Para obter mais exemplos de como usar objetos para implementar um projeto orientado a objeto, consulte o Capítulo 5.

### *Tabelas externas*

As *tabelas externas* foram introduzidas no Oracle9*i*. Em resumo, *elas* permitem que um usuário acesse uma origem de dados, como um arquivo de texto, como se fosse uma tabela no banco de dados. Os metadados para a tabela são armazenados dentro do dicionário de dados Oracle, mas o conteúdo dela é armazenado externamente.

A definição de uma tabela externa contém duas partes. A primeira parte e mais familiar é a definição da tabela do ponto de vista do usuário do banco de dados. Essa definição é parecida com uma definição típica que seria vista em uma instrução **create table**.

A segunda parte é que diferencia uma tabela externa de uma tabela normal. É aí que ocorre o mapeamento entre as colunas do banco de dados e a origem de dados externa – em qual(is) coluna(s) o elemento de dados inicia, qual a largura da coluna e se o formato da coluna externa é caractere ou binário. A sintaxe para o tipo padrão da tabela externa, ORACLE_LOADER, é praticamente idêntica àquela de um arquivo de controle no SQL*Loader. Essa é uma das vantagens das tabelas externas; o usuário só precisa saber como acessar uma tabela de banco de dados padrão para chegar ao arquivo externo.

Há, porém algumas desvantagens no uso de tabelas externas. Nelas, não é possível criar índices nem executar operações de inserção, atualização ou exclusão. Essas desvantagens são menores quando consideramos as vantagens de usá-las para carregar tabelas nativas de bancos de dados, por exemplo, em um ambiente de data warehouse.

### *Tabelas clusterizadas*

Quando duas ou mais tabelas são freqüentemente acessadas juntas (por exemplo, uma tabela de pedidos e uma de detalhes de produtos), a criação de uma *tabela clusterizada* talvez seja uma boa maneira de melhorar o desempenho de consultas que fazem referência a elas. No caso de uma tabela de pedidos com uma de detalhes de produtos associada, as informações do cabeçalho do pedido podem ser armazenadas no mesmo bloco dos registros de detalhes do produto, reduzindo assim a quantidade de E/S (entrada e saída) necessária para recuperar as informações sobre o pedido e o produto.

As tabelas clusterizadas também reduzem a quantidade de espaço necessário para armazenar as colunas que as duas têm em comum, também conhecido como *valor de chave do cluster* (*cluster key value*). O valor de chave do cluster é também armazenado em um índice do *cluster* (cluster index). O índice do cluster opera quase como um tradicional, pois ele aprimora as consultas em relação às tabelas clusterizadas quando acessadas pelo valor da chave do cluster. No nosso exemplo com pedidos e produtos, o número do pedido é armazenado apenas uma vez, em vez de ser repetido para cada linha dos detalhes do produto.

As vantagens da clusterização de uma tabela são reduzidas caso operações **insert**, **update** e **delete** ocorram com freqüência na tabela em relação ao número de instruções **select** realizadas na tabela. Além disso, consultas freqüentes em tabelas individuais no cluster também podem reduzir os benefícios da clusterização das mesmas.

### *Clusters de hash*

Um tipo especial de tabela clusterizada, um *cluster de hash*, opera de modo muito similar a uma tabela clusterizada normal, exceto que em vez de usar um índice clusterizado, um cluster de hash usa um algoritmo de hashing para armazenar e recuperar linhas em uma tabela. A quantidade total estimada de espaço necessário para a tabela é alocada quando ela é criada, de acordo com o número de chaves de hash especificado durante a criação do cluster. No nosso exemplo de entrada de pedidos, vamos supor que o banco de dados Oracle precise espelhar o sistema legado de entrada de dados, que reutiliza números de pedidos periodicamente. Além disso, o número do pedido sempre tem seis dígitos. Poderíamos criar o cluster para os pedidos, como no exemplo a seguir:

```
create cluster order_cluster (order_number number(6))
     size 50
     hash is order_number hashkeys 1000000;

create table cust_order (
     order_number      number(6) primary key,
     order_date        date,
     customer_number   number)
cluster order_cluster(order_number);
```

Os clusters de hash apresentam benefícios de desempenho quando as linhas são selecionadas em uma tabela usando uma comparação de igualdade, como neste exemplo:

```
select order_number, order_date from cust_order
   where order_number = 196811;
```

Em geral, esse tipo de consulta recuperará a linha com apenas uma operação de E/S se o número de **chaves de hash** for suficientemente alto e se a cláusula **hash is**, que contém o algoritmo de hashing, produzir uma chave de hash proporcionalmente distribuída.

### *Clusters de hash classificados*

Os *clusters de hash classificados* foram lançados no Oracle 10*g*. Eles são similares aos clusters de hash normais pelo fato de que o algoritmo de hashing é usado para localizar uma linha em uma

tabela. No entanto, além disso, os clusters de hash classificados permitem que as linhas na tabela sejam classificadas por uma ou mais colunas em ordem crescente. Isso permite que os dados sejam processados de forma mais rápida para aplicações que utilizam o processamento "primeiro a entrar, primeiro a sair" (FIFO, First In, First Out).

Clusters de hash classificados são criados usando a mesma síntaxe das tabelas clusterizadas normais, com a adição do parâmetro posicional SORT após as definições de colunas dentro do cluster. Aqui está um exemplo da criação de uma tabela em um cluster de hash classificado:

```
create table order_detail (
      order_number      number,
      order_timestamp   timestamp sort,
      customer_number   number)
cluster order_detail_cluster (
      order_number,
      order_timestamp);
```

Devido à natureza FIFO de um cluster de hash classificado, quando os pedidos são acessados por **order_number** os pedidos mais antigos são recuperados primeiro com base no valor de **order_timestamp**.

### *Tabelas particionadas*

O *particionamento* de uma tabela (ou índice, como veremos na próxima seção) ajuda a tornar uma grande tabela mais gerenciável. Uma tabela pode ser particionada, ou mesmo subparticionada, em partes menores. Do ponto de vista da aplicação, o particionamento é transparente (isto é, não é necessária uma referência explícita a uma partição específica em qualquer SQL de usuário final). O único efeito que um usuário pode observar é que a consulta à tabela particionada que usa critérios na cláusula **where,** correspondentes ao esquema de particionamento, é bem mais rápida.

Do ponto de vista do DBA, as vantagens do particionamento são muitas. Se uma partição de uma tabela estiver em um volume de disco corrompido, as outras partições da tabela ainda continuarão disponíveis para consultas de usuário enquanto o volume danificado é reparado. Da mesma maneira, os backups de partições podem ocorrer durante dias, uma partição de cada vez, em vez de exigir um único backup da tabela inteira.

As partições dividem-se em três tipos: particionada por faixa de valores (range partitioning), particionada por hash (hash partitioning) ou, desde o Oracle9*i,* particionada por lista (list partitioning). A partir do Oracle 11g, é possível também particionar por relacionamentos pai/filho (parent/child relationships), particionamento gerenciado por aplicação (application-controlled partitioning) e muitas combinações de tipos básicos de partições, incluindo hash de lista (list-hash), lista-lista (list-list), lista-faixa (list-range) e faixa-faixa (range-range). Cada linha em uma tabela particionada pode existir em somente uma partição. A *chave de partição* (partition key) direciona a linha para a partição apropriada; ela pode ser uma composta de até 16 colunas da tabela. Existem algumas constraints sobre os tipos de tabelas que podem ser particionadas; por exemplo, uma tabela que contém uma coluna LONG ou LONG RAW não pode ser particionada. A constraint LONG raramente deve ser um problema; LOBs (CLOBs e BLOBs, character large objects e binary large objects) são muito mais flexíveis e incluem todos os recursos dos tipos de dados LONG e LONG RAW.

**DICA**

*A Oracle Corporation recomenda que o particionamento seja considerado para qualquer tabela que tenha um tamanho maior que 2GB.*

Independentemente do tipo de particionamento em uso, cada membro de uma tabela particionada deve ter os mesmos atributos lógicos, como nomes de colunas, tipos de dados, constraints

e assim por diante. No entanto, os atributos físicos de cada partição podem ser diferentes dependendo do seu tamanho e localização no disco. O principal é que a tabela particionada deve ser logicamente consistente do ponto de vista de uma aplicação ou do usuário.

**Partições por faixa de valores (range partitions)** Uma *partição por faixa de valores* é aquela cuja chave de partição incorre em um determinado intervalo. Por exemplo, as visitas ao site corporativo de comércio eletrônico podem ser atribuídas a uma partição com base na data da visita, com uma partição por trimestre. Um visita ao site em 24 de maio de 2004 será registrada na partição com o nome FY2004Q2, ao passo que uma visita ao site em 2 de dezembro de 2004 será registrada na partição com o nome FY2004Q4.

**Partições por lista (list partitions)** Uma *partição por lista* é aquela cuja chave incorre em grupos de valores distintos. Por exemplo, as vendas por região do país podem criar uma partição para NY, CT, MA e VT, e outra partição para IL, WI, IA e MN. Quaisquer vendas de outros países podem ser atribuídas à sua própria partição quando não houver o código do estado.

**Partições por hash (hash partitions)** Uma *partição por hash* atribui uma linha a uma partição com base em um algoritmo de hashing, especificando a coluna ou colunas usadas nesse algoritmo, mas sem atribuir explicitamente a partição, apenas especificando quantas delas estão disponíveis. O Oracle atribuirá a linha a uma partição e garantirá uma distribuição equilibrada de linhas em cada uma.

As partições por hash são úteis quando não há claramente uma lista ou esquema de particionamento por faixa de valores para os tipos de colunas na tabela, ou quando os tamanhos relativos das partições são alterados com freqüência, exigindo repetidos ajustes manuais para o esquema de particionamento.

**Partições compostas (composite partitions)** Um refinamento ainda maior do processo de particionamento está disponível com as *partições compostas*. Por exemplo, uma tabela pode ser particionada por faixa de valores e dentro de cada faixa, subparticionada por lista ou por hash. Novas combinações no Oracle 11*g* incluem os particionamentos hash de lista, lista-lista, lista-faixa e faixa-faixa.

### *Índices particionados*

Você também pode particionar índices em uma tabela, correspondendo ao esquema de partição da tabela que está sendo indexada *(índices locais)* ou particionados independentemente do esquema de partição da tabela *(índices globais)*. Os índices particionados locais têm a vantagem de aumentar a disponibilidade do índice quando as operações de partição ocorrem; por exemplo, arquivar ou descartar a partição FY2002Q4 e seu índice local não afetará a disponibilidade dos índices para as outras partições da tabela.

## Constraints (Restrições)

Uma *constraint* do Oracle é uma ou mais regras que você pode definir em uma ou mais colunas em uma tabela para ajudar a impor uma regra de negócio. Por exemplo, uma constraint pode impor a regra de negócio que diz que o salário inicial de um funcionário deve ser de no mínimo $25.000,00. Outro exemplo de uma constraint impondo uma regra de negócio é exigir que, quando um novo funcionário seja atribuído a um departamento (embora ele não precise ser atribuído a um departamento específico de imediato), o número do departamento seja validado e exista na tabela DEPT.

Seis tipos de regras de integridade de dados podem ser aplicadas às colunas das tabelas: regra nula, valores de coluna únicos, valores de chave primária, valores de integridade referencial, integridade inline complexa e integridade baseada em trigger (gatilho). Abordaremos cada um deles brevemente nas seções seguintes.

Todas as constraints em uma tabela são definidas quando a tabela é criada ou quando uma coluna é alterada, exceto para triggers, que são definidos de acordo com a operação

DML que você está executando na tabela. As constraints podem ser ativadas ou desativadas na criação ou em qualquer momento no futuro; quando isso acontece (usando as palavras-chave **enable** ou **disable**), os dados existentes na tabela podem ou não precisar ser validados (usando as palavras-chave **validate** ou **novalidate**) em relação à constraint, dependendo das regras de negócio em vigor.

Por exemplo, uma tabela em um banco de dados de um fabricante de carros denominado CAR_INFO, que contém os dados de novos automóveis, precisa de uma nova constraint na coluna AIRBAG_QTY, onde o valor dessa última não deve ser NULL e deve ter um valor de ao menos 1 para todos os veículos novos. No entanto, essa tabela contém dados do modelo para anos anteriores à exigência dos air bags e como resultado, essa coluna é 0 ou NULL. Nesse caso, uma solução seria criar uma constraint na tabela AIRBAG_QTY para impor a nova regra para as novas linhas adicionadas à ela, mas não validar a constraint para linhas existentes.

Eis uma tabela criada com todos os tipos de constraints. Cada uma será examinada nas subseções seguintes.

```
create table CUST_ORDER
   (Order_Number          NUMBER(6)      PRIMARY KEY,
   Order_Date             DATE           NOT NULL,
   Delivery_Date          DATE,
   Warehouse_Number       NUMBER         DEFAULT 12,
   Customer_Number        NUMBER         NOT NULL,
   Order_Line_Item_Qty    NUMBER         CHECK (Order_Line_Item_Qty < 100),
   UPS_Tracking_Number    VARCHAR2(50)   UNIQUE,
   foreign key (Customer_Number) references CUSTOMER(Customer_Number));
```

### *Regra de nulo (null)*

A constraint NOT NULL impede que valores NULL sejam inseridos na coluna Order_Date ou Customer_Number. Isso faz muito sentido do ponto de vista da regra de negócio: cada pedido deve ter uma data e ele só faz sentido se um cliente o fizer.

Observe que um valor NULL em uma coluna não significa que ele seja em branco ou zero; em vez disso, o valor não existe. Um valor NULL não é igual a nada, nem mesmo a outro valor NULL. Esse conceito é importante quando fazemos consultas SQL em colunas que podem ter valores NULL.

### *Valores de coluna únicos*

A constraint de integridade UNIQUE garante que uma coluna ou grupo de colunas (em uma constraint composta) seja única em toda a tabela. No exemplo anterior, a coluna UPS_Tracking_Number não conterá valores duplicados.

Para impor a constraint, o Oracle criará um índice único na coluna UPS_Tracking_Number. Se já houver um índice único válido na coluna, o Oracle usará esse índice para impor a constraint.

Uma coluna com uma constraint UNIQUE também pode ser declarada como NOT NULL. Se a coluna não for declarada com a constraint NOT NULL, qualquer número de linhas poderá conter valores NULL, desde que as linhas restantes tenham valores únicos nessa coluna.

Em uma constraint única composta que permite NULLs em uma ou mais colunas, as colunas que não são NULL determinam se a constraint está sendo satisfeita. A coluna NULL sempre satisfaz a constraint porque um valor NULL não é igual a nada.

### *Valores de chave primária*

A constraint de integridade PRIMARY KEY é o tipo mais comum de constraint encontrado em uma tabela de banco de dados. No máximo, apenas uma constraint de chave primária pode existir em uma tabela. A coluna ou colunas que compõem a chave primária não podem ter valores NULL.

No exemplo anterior, a coluna Order_Number é a chave primária. Um índice único é criado para impor a constraint; se houver um índice único utilizável para a coluna, a constraint de chave primária usará esse índice.

### *Valores de integridade referencial*

A integridade referencial ou constraint FOREIGN KEY é mais complicada que as outras que abordamos até aqui porque ela precisa de outra tabela para restringir que valores podem ser inseridos na coluna com a constraint de integridade referencial.

No exemplo anterior, uma FOREIGN KEY é declarada na coluna Customer_Number; todos os valores inseridos nessa coluna também devem existir na Customer_Number de outra tabela (nesse caso, a tabela CUSTOMER).

Como nas outras constraints que permitem valores NULL, uma coluna com constraint de integridade referencial pode ser NULL sem exigir que a coluna referenciada contenha um valor NULL.

Além do mais, uma constraint FOREIGN KEY pode ser auto-referencial. Em uma tabela EMPLOYEE cuja chave primária é Employee_Number, a coluna Manager_Number pode ser uma FOREIGN KEY declarada em relação à coluna Employee_Number na mesma tabela. Isso permite a criação de uma hierarquia de funcionários e gerentes dentro da própria tabela EMPLOYEE.

Os índices quase sempre devem ser declarados em uma coluna FOREIGN KEY para melhorar o desempenho; a única exceção a essa regra é quando a chave primária ou chave única referenciada na tabela pai nunca é atualizada ou excluída.

### *Integridade inline complexa*

As regras de negócio mais complexas podem ser impostas em termos de coluna usando uma constraint CHECK. No exemplo anterior, a coluna Order_Line_Item_Qty nunca deve exceder a 99.

Uma constraint CHECK pode usar outras colunas da linha que está sendo inserida ou atualizada para avaliá-la. Por exemplo, uma constraint na coluna STATE_CD permitiria valores NULL somente se a coluna COUNTRY_CD não fosse USA. Além disso, a constraint pode usar valores literais e funções internas, como TO_CHAR ou TO_DATE, desde que essas funções operem em literais ou colunas na tabela.

Múltiplas constraints CHECK são permitidas em uma coluna. Todas elas devem ser avaliadas como TRUE para permitir que um valor seja inserido na coluna. Por exemplo, nós poderemos modificar a constraint CHECK anterior para garantir que Order_Line_Item_Qty seja maior que 0 e menor que 100.

### *Integridade baseada em trigger*

Se as regras de negócio forem muito complexas para serem implementadas usando constraints únicas, um *trigger* (gatilho) de banco de dados pode ser criado em uma tabela usando o comando **create trigger** junto com um bloco de código PL/SQL para impor a regra de negócio.

Os triggers são necessários para impor constraints de integridade referencial quando a tabela referenciada existir em um banco de dados diferente. Os triggers também têm muita utilidade fora dos limites de verificação de constraint (auditar o acesso a uma tabela, por exemplo). Os triggers de banco de dados serão discutidos detalhadamente no Capítulo 17.

## Índices

Um *índice* Oracle permite acesso mais rápido às linhas de uma tabela quando um pequeno subconjunto de linhas for recuperado da tabela. Um índice armazena o valor da coluna ou colunas que estão sendo indexadas, junto com o RowID físico da linha que contém o valor indexado, exceto para tabelas organizadas por índice (IOTs, index-organized tables), que usam a chave primária como um RowID lógico. Uma vez que uma correspondência é encontrada no índice, o RowID do

índice aponta para o local exato da linha da tabela: qual arquivo, qual bloco dentro do arquivo e qual linha dentro do bloco.

Os índices são criados em uma única coluna ou em múltiplas colunas. As entradas de índice são armazenadas em uma estrutura de árvore B de modo que percorrê-lo para encontrar o valor da chave da linha exija poucas operações de E/S. Um índice pode servir a um objetivo duplo no caso de um índice único: ele não apenas acelera a procura da linha, mas impõe uma constraint de chave única ou primária na coluna indexada. As entradas dentro de um índice são automaticamente atualizadas sempre que o conteúdo de uma linha da tabela é inserido, atualizado ou excluído. Quando uma tabela é descartada, todos os índices criados na tabela também são automaticamente descartados.

Vários tipos de índices estão disponíveis no Oracle, cada um deles apropriado para um tipo específico de tabela, método de acesso ou ambiente de aplicação. Apresentaremos os destaques e recursos dos tipos de índices mais comuns nas subseções a seguir.

### *Índices únicos*

Um *índice único* é a forma mais comum de índice de árvore B. Ele é usado freqüentemente para impor a constraint de chave primária de uma tabela. Os índices únicos garantem que não existirão valores duplicados na coluna ou colunas que estão sendo indexadas. Um índice único pode ser criado em uma coluna na tabela EMPLOYEE para o Número de Seguridade Social (Social Security Number) porque não podem haver dados duplicados nessa coluna. Entretanto, alguns funcionários podem não ter um Número de Seguridade Social, portanto, esta coluna poderá conter valores NULL.

### *Índices não-únicos*

Um *índice não-único* ajuda a acelerar o acesso a uma tabela sem impor exclusividade. Por exemplo, podemos criar um índice não único na coluna Last_Name da tabela EMPLOYEE para acelerar nossas pesquisas por sobrenome, mas certamente teremos muitas duplicatas para algum determinado sobrenome. Um índice de árvore B não-único é criado em uma coluna por padrão se nenhuma outra palavra-chave for especificada em uma instrução CREATE INDEX.

### *Índices de chave invertida (reverse key indexes)*

Um *índice de chave invertida* é um tipo especial usado normalmente em um ambiente OLTP (online transaction processing). Em um índice de chave invertida, todos os bytes no valor de chave de cada coluna são inversos. A palavra-chave **reverse** especifica um índice de chave invertida no comando **create index**. Aqui está um exemplo da criação de um índice de chave invertida:

```
create index IE_LINE_ITEM_ORDER_NUMBER
        on LINE_ITEM(Order_Number) REVERSE;
```

Se o número de pedido 123459 é colocado, o índice de chave invertida armazena o número do pedido como 954321. As inserções na tabela são distribuídas por todas as chaves de folha do índice, reduzindo a disputa entre diversos processos de gravação que fazem inserções de novas linhas. Um índice de chave invertida também reduz o potencial desses "pontos com grande volume de concorrência de E/S" em um ambiente OLTP se os pedidos forem consultados ou modificados logo após serem feitos.

### *Índices baseados em função (function-based indexes)*

Um *índice baseado em função* é similar a um índice de árvore B padrão, exceto que uma transformação de uma coluna, ou colunas, declarada como uma expressão, é armazenada no índice em vez de ser armazenada nas próprias colunas.

Os índices baseados em função são úteis em casos em que nomes e endereços são armazenados no banco de dados com letras maiúsculas e minúsculas misturadas. Um índice normal em uma coluna que contém o valor "SmiTh" não retornaria um valor caso o critério de pesquisa fosse "Smith". Por outro lado, se o índice armazenasse sobrenomes em letras maiúsculas, todas as pes-

quisas por sobrenomes poderiam usar letras maiúsculas. Aqui está um exemplo da criação de um índice baseado em função na coluna Last_Name da tabela EMPLOYEE:

```
create index up_name on employee(upper(Last_Name));
```

Como resultado, as pesquisas que usam consultas como a seguir utilizarão o índice que acabamos de criar, em vez de pesquisar toda a tabela:

```
select Employee_Number, Last_Name, First_Name, from employee
      where upper(Last_Name) = 'SMITH';
```

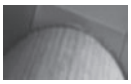
### *Índices de bitmap*

Um *índice de bitmap* tem uma estrutura significativamente diferente de um índice de árvore B no nó de folha do índice. Ele armazena uma string de bits para cada valor possível (a cardinalidade) da coluna que está sendo indexada. O comprimento da string de bits é igual ao número de linhas da tabela que está sendo indexada.

Além de economizar uma quantidade enorme de espaço se comparado aos índices tradicionais, um índice de bitmap pode fornecer melhorias enormes no tempo de resposta, porque o Oracle pode remover rapidamente potenciais linhas de uma consulta que contêm múltiplas cláusulas **where** muito antes de a própria tabela precisar ser acessada. Múltiplos bitmaps podem usar operações **and** e **or** lógicas para determinar que linhas acessar na tabela.

Embora seja possível usar um índice de bitmap em qualquer coluna de uma tabela, ele é mais eficiente quando a coluna que está sendo indexada tem uma baixa *cardinalidade,* ou número de valores distintos. Por exemplo, a coluna Gender na tabela PERS será NULL, M ou F. O índice de bitmap na coluna Gender terá apenas três bitmaps armazenados no índice. Por outro lado, um índice de bitmap na coluna Last_Name terá quase o mesmo número de strings de bitmap que as linhas da tabela. As consultas que procuram um sobrenome específico provavelmente demorarão menos tempo se uma varredura integral de tabela (full table scan) for executada em vez de usar um índice. Nesse caso, um índice de árvore B não único tradicional faz mais sentido.

Uma variação dos índices de bitmap denominada *índices de junção de bitmap* (bitmap join indexes) cria um índice em uma coluna da tabela que freqüentemente sofre junção com uma ou mais tabelas na mesma coluna. Isso proporciona vantagens enormes em um ambiente de data warehouse onde um índice de junção de bitmap é criado em uma tabela fato e em uma ou mais tabelas dimensão, essencialmente fazendo a junção prévia dessas tabelas e economizando recursos de CPU e E/S quando uma junção real é executada.



**NOTA**
*Os índices de bitmap só estão disponíveis na edição Enterprise do Oracle 11g.*

## Visões

As visões permitem que os usuários vejam uma apresentação personalizada dos dados de uma única tabela ou mesmo de uma junção de muitas tabelas. Uma visão também é conhecida como uma *consulta armazenada* – os detalhes das consultas subjacentes à visão são ocultos do usuário. Uma visão normal não armazena dados, apenas a definição, e a consulta subjacente é executada toda vez que a visão é acessada. A extensão de uma visão normal, denominada *visão materializada,* permite que os resultados da consulta sejam armazenados juntamente com a definição da consulta para acelerar o processamento, entre outros benefícios. As visões de objeto, como as visões tradicionais, ocultam os detalhes das junções das tabelas subjacentes e permitem que o desenvolvimento e o processamento orientado a objeto ocorram no banco de dados enquanto as tabelas subjacentes permanecem em um formato relacional.

A seguir, examinaremos os conceitos básicos dos tipos de visões que um DBA, desenvolvedor ou usuário típico do banco de dados irão criar e usar regularmente.

### *Visões regulares*

Uma *visão regular,* ou mais comumente referenciada como uma *visão,* não aloca um armazenamento; apenas sua definição, uma consulta, é armazenada no dicionário de dados. As tabelas na consulta subjacente à visão são denominadas *tabelas base*; cada uma delas em uma visão pode ser adicionalmente definida como uma visão.

As vantagens de uma visão são muitas. Elas ocultam a complexidade dos dados – um analista senior pode definir uma visão composta das tabelas EMPLOYEE, DEPARTMENT e SALARY para que a gerência superior consulte as informações sobre salários de funcionários mais facilmente, por meio de uma instrução **select** executada no que parece ser uma tabela, mas, na verdade, é uma visão que contém uma consulta que junta as tabelas EMPLOYEE, DEPARTMENT e SALARY.

As visões também podem ser usadas para impor segurança. Uma visão da tabela EMPLOYEE denominada EMP_INFO pode conter todas as colunas, exceto a coluna de salários, e ela também pode ser definida como **read only** para impedir atualizações na tabela:

```
create view EMP_INFO as
     select Employee_Number, Last_Name,
            First_Name, Middle_Initial, Surname
from EMPLOYEE
with READ ONLY;
```

Sem a cláusula **read only**, é possível atualizar ou adicionar linhas a uma visão, mesmo a uma que contém múltiplas tabelas. Há algumas construções em uma visão que impedem que ela se torne atualizável, como ter um operador **distinct**, uma função agregada ou uma cláusula **group by**.

Quando o Oracle processa uma consulta que contém uma visão, ele substitui a definição da consulta subjacente na instrução **select** do usuário e processa a consulta resultante como se a visão não existisse. O resultado é que, quando uma visão é utilizada, os benefícios de quaisquer índices existentes nas tabelas base não são perdidos.

### *Visões materializadas*

Em alguns aspectos, uma *visão materializada* é muito similar a uma regular: a sua definição é armazenada no dicionário de dados e a visão oculta do usuário os detalhes da consulta base subjacente. Mas essas são as únicas semelhanças. Uma visão materializada também aloca espaço em um segmento de banco de dados para armazenar o conjunto de resultados da execução da consulta base.

Uma visão materializada pode ser usada para replicar uma cópia somente leitura da tabela para outro banco de dados, com os mesmos dados e definições de coluna da tabela base. Essa é a implementação mais simples de uma visão materializada. Para melhorar o tempo de resposta quando uma visão materializada precisa ser atualizada, um *log de visão materializada* pode ser criado para atualizá-la. Caso contrário, será preciso uma atualização completa quando uma atualização for necessária – os resultados da consulta base devem ser executados na sua totalidade para atualizar a visão materializada. O log de visão materializada facilita as suas atualizações incrementais.

Em um ambiente de data warehouse, as visões materializadas podem armazenar dados agregados de uma consulta **group by rollup** ou **group by cube**. Se os valores dos parâmetros de inicialização apropriados estiverem definidos, como QUERY_REWRITE_ENABLED e a própria consulta permitir reescrita (com a cláusula **query rewrite**), qualquer consulta que parece fazer o mesmo tipo de agregação da visão materializada a utilizará automaticamente em vez de executar a consulta original.

Independentemente do tipo de visão materializada, ela pode ser atualizada de forma automática quando uma transação finalizada com commit ocorrer na tabela base, ou pode ser atualizada sob demanda. As visões materializadas têm muitas semelhanças com os índices, pelo fato de que elas são diretamente ligadas a uma tabela e ocupam espaço, elas devem ser atualizadas quando as tabelas base são atualizadas, suas existências são virtualmente transparentes para o usuário e podem ajudar na otimização das consultas pelo uso de um caminho de acesso alternativo para retornar seus resultados.

O Capítulo 17 fornece mais detalhes sobre como usar visões materializadas em um ambiente distribuído.

### *Visões de objeto*

Os ambientes de desenvolvimento de aplicações orientadas a objeto (OO) estão se tornando cada vez mais predominantes e o banco de dados Oracle 11*g* fornece um suporte completo à implementação de objetos e métodos no banco de dados. Entretanto, uma migração de um ambiente de banco de dados puramente relacional para um puramente orientado a objetos não é uma transição fácil de fazer; poucas empresas têm tempo e recursos para construir um novo sistema a partir de zero. O Oracle 11*g* facilita a transição com visões de objeto. As visões de objeto permitem que as aplicações orientadas a objetos vejam os dados como uma coleção de objetos que tem atributos e métodos, enquanto os sistemas legados ainda podem executar trabalhos em lote na tabela INVENTORY. As visões de objetos podem simular tipos de dados abstratos, identificadores de objetos (OIDs) e referências que um genuíno ambiente de banco de dados orientado a objeto proporcionaria.

Como nas visões regulares, é possível usar triggers **instead of** na definição da visão para permitir operações DML na visão por meio da execução de um bloco de código PL/SQL em vez da instrução DML real fornecida pelo usuário ou aplicação.

## Usuários e esquemas

O acesso ao banco de dados é concedido para uma conta conhecida como *usuário*. Um usuário pode existir no banco de dados sem possuir objetos. No entanto, se o usuário cria e possui objetos no banco de dados, esses objetos são parte do *esquema* que tem o mesmo nome do usuário do banco de dados. Um esquema pode possuir qualquer tipo de objeto no banco de dados: tabelas, índices, seqüências, visões e assim por diante. O proprietário do esquema ou DBA pode conceder acesso a esses objetos para outros usuários do banco de dados. O usuário sempre tem todos os privilégios e controle sobre os objetos do seu esquema.

Quando um usuário é criado pelo DBA (ou por algum outro usuário com o privilégio de sistema **create user**), várias outras características podem ser atribuídas a ele, como quais tablespaces estão disponíveis para a criação de objetos e se a senha é pré-expirada.

Os usuários no banco de dados podem ser autenticados com três métodos: autenticação de banco de dados, autenticação de sistema operacional e autenticação de rede. Com a autenticação de banco de dados, a senha criptografada do usuário é armazenada no banco de dados. Por outro lado, a autenticação de sistema operacional assume que um usuário já autenticado por uma conexão de sistema operacional tenha os mesmo privilégios de um usuário com o mesmo nome ou similar (dependendo do valor do parâmetro de inicialização OS_AUTHENT_PREFIX). A autenticação de rede usa soluções baseadas em PKI (Public Key Infrastructure). Esses métodos de autenticação de rede requerem o Oracle 11*g* Enterprise Edition com a opção Oracle Advanced Security.

## Perfis

Os recursos do banco de dados não são ilimitados; portanto, um DBA deve gerenciar e alocar recursos entre todos os usuários do banco de dados. Alguns exemplos de recursos de banco de dados são tempo de CPU, sessões simultâneas, leituras lógicas e tempo de conexão.

Um *perfil* de banco de dados é um conjunto nomeado de limites de recursos que você pode atribuir a um usuário. Depois que o Oracle é instalado, o perfil DEFAULT passa a existir e é atribuído a qualquer usuário não explicitamente atribuído a um perfil. O DBA pode adi-

cionar novos perfis ou alterar o perfil DEFAULT para se ajustar às necessidades da empresa. Os valores iniciais para o perfil DEFAULT permitem o uso ilimitado de todos os recursos do banco de dados.

## Seqüências

Uma *seqüência* Oracle atribui números seqüenciais, que são únicos, a não ser que a seqüência seja recriada ou redefinida. Ela produz uma série de números únicos em um ambiente multiusuário sem os overheads de bloqueio de disco ou chamadas de E/S especiais, além do que é necessário para carregar a seqüência no shared pool (pool compartilhado).

As seqüências podem gerar números de até 38 dígitos de comprimento; a série de números pode ser ascendente ou descendente, o intervalo pode ser qualquer valor especificado pelo usuário e o Oracle pode armazenar em cache blocos de números de uma seqüência na memória para obter um desempenho ainda mais rápido.

Os números das seqüências são garantidamente únicos, mas não necessariamente seqüenciais. Se um bloco de números for armazenado em cache e a instância for reiniciada ou uma transação que usa um número de uma seqüência sofrer rollback, a próxima chamada para recuperar um número na seqüência não retornará o número que não foi usado na referência original à ela.

## Sinônimos

Um *sinônimo* Oracle é simplesmente um apelido para um objeto de banco de dados, para simplificar as referências aos seus objetos e ocultar os detalhes da origem desses objetos. Os sinônimos podem ser atribuídos a tabelas, visões, visões materializadas, seqüências, procedures, funções e pacotes. Como nas visões, um sinônimo não aloca um espaço no banco de dados além da sua definição no dicionário de dados.

Os sinônimos podem ser públicos ou privados. Um sinônimo privado é definido no esquema de um usuário e está disponível apenas para o usuário. Um sinônimo público é normalmente criado por um DBA e está automaticamente disponível para uso por qualquer usuário do banco de dados.

**DICA**

*Após a criação de um sinônimo público, certifique-se de que os usuários do sinônimo têm os privilégios corretos para o objeto referenciado pelo sinônimo.*

Ao referenciar um objeto de banco de dados, o Oracle primeiro verifica se ele existe no esquema do usuário. Caso não exista, o Oracle verifica se há um sinônimo privado. Se não houver, o Oracle verifica se há um sinônimo público. Em caso negativo, o Oracle retornará um erro.

## PL/SQL

O Oracle PL/SQL é uma extensão da linguagem procedural do Oracle para SQL. O PL/SQL é útil quando o DML padrão e as instruções **select** não podem produzir resultados de uma maneira fácil devido à falta dos elementos procedurais encontrado em uma linguagem de terceira geração tradicional, como C++ e Ada. A partir do Oracle9*i*, o mecanismo de processamento do SQL é compartilhado entre o SQL e o PL/SQL, o que significa que todos os novos recursos adicionados ao SQL estão automaticamente disponíveis para o PL/SQL.

Nas próximas seções, mostraremos algumas vantagens de usar o Oracle PL/SQL.

### *Procedures (procedimentos)/Funções*

As procedures e funções PL/SQL são exemplos de *blocos nomeados* PL/SQL. Um bloco PL/SQL é uma seqüência de instruções PL/SQL tratadas como uma unidade com a finalidade de execução

e contém até três seções: uma seção de declaração de variáveis, uma seção executável e uma seção de exceções. A diferença entre uma procedure e uma função é que uma função retornará um único valor para o programa que a executar, como uma instrução **select** SQL. Uma procedure, por outro lado, não retorna um valor, apenas um código de status. Entretanto, as procedures podem ter uma ou mais variáveis que podem ser definidas e retornadas como parte da sua lista de argumentos.

As procedures e funções têm muitas vantagens em um ambiente de banco de dados. As procedures são compiladas e armazenadas no dicionário de dados uma vez; quando mais de um usuário precisar chamá-la, ela já estará compilada e haverá somente uma cópia da procedure armazenada no shared pool. Além disso, o tráfego da rede é reduzido, mesmo que os recursos procedurais do PL/SQL não sejam usados. Uma chamada PL/SQL consome muito menos largura de banda do que várias instruções SQL **select** e **insert** enviadas separadamente pela rede, sem mencionar o reparsing (reinterpretação) que ocorre para cada instrução enviada pela rede.

### *Pacotes*

Os *pacotes* PL/SQL agrupam funções e procedures relacionadas, juntamente com variáveis e cursores comuns. Os pacotes consistem em duas partes: uma especificação de pacote e um corpo de pacote. Na especificação de pacote, seus métodos e atributos são expostos; a implementação dos métodos e de quaisquer métodos e atributos privados é ocultada no corpo do pacote. Usar um pacote em vez de uma procedure ou função independente permite que a procedure ou função incorporada seja alterada sem invalidar um objeto que referencie os elementos da especificação do pacote, evitando assim a recompilação dos objetos que o referenciam.

### *Triggers*

Os triggers são um tipo especializado de bloco de código PL/SQL ou Java que é executado, ou *disparado,* quando um evento especificado ocorre. Os tipos de eventos podem ser instruções DML em uma tabela ou visão, instruções DDL e mesmo eventos de banco de dados, como inicialização ou shutdown. Um trigger pode ser configurado para ser executado em um evento específico, para um usuário específico, ou como parte de uma estratégia de auditoria.

Os triggers são extremamente úteis em um ambiente distribuído para simular um relacionamento de chave estrangeira entre tabelas que não existem no mesmo banco de dados. Eles também são úteis na implementação de regras de integridade complexas que não podem ser definidas usando os tipos de constraints Oracle predefinidos.

Para obter mais informações sobre como os triggers podem ser usados em um ambiente distribuído robusto, consulte o Capítulo 17.

## Acesso a arquivo externo

Além das tabelas externas, existem diversas outras maneiras como o Oracle pode acessar os arquivos externos:

- No SQL*Plus, seja acessando um script externo que contém outros comandos SQL a serem executados ou enviando a saída a partir de um comando **spool** do SQL*Plus para um arquivo no sistema de arquivos do sistema operacional.
- As informações de texto podem ser lidas ou escritas a partir de uma procedure PL/SQL usando o pacote predefinido UTL_FILE; de maneira similar, as chamadas **dbms_output** dentro de uma procedure PL/SQL podem gerar diagnósticos e mensagens de texto que podem ser capturados por outra aplicação e salvas em um arquivo de texto.
- Os dados externos podem ser referenciados pelo tipo de dados BFILE. Um tipo de dados BFILE é um ponteiro para um arquivo binário externo. Para que os BFILEs possam ser usados em um banco de dados, é necessário criar um *apelido de diretório* com o comando **create directory** que especifica um prefixo contendo o caminho de diretório completo onde o alvo de BFILE será armazenado.

- O DBMS_PIPE pode se comunicar com qualquer linguagem 3GL que o Oracle suporte, como C++, Ada, Java ou COBOL, e trocar informações.
- O UTL_MAIL, um novo pacote no Oracle 10*g*, permite que uma aplicação PL/SQL envie emails sem saber como usar a pilha de protocolos SMTP subjacente.

Ao usar um arquivo externo como uma origem de dados para entrada ou saída, algumas precauções devem ser tomadas. Os tópicos a seguir devem ser cuidadosamente considerados antes de usar uma origem de dados externa:

- Os dados do banco de dados e os dados externos podem estar freqüentemente fora de sincronização quando uma das origens de dados muda sem sincronizar-se com a outra.
- É importante certificar-se de que os backups das duas origens de dados ocorram praticamente ao mesmo tempo para garantir que a recuperação de uma origem de dados manterá as duas origens de dados em sincronia.
- Os arquivos de script podem conter senhas; muitas empresas proíbem a representação em texto simples de qualquer conta de usuário em um arquivo de script. Nesta situação, a validação do sistema operacional pode ser uma boa alternativa para autenticação de usuários.
- Você deve examinar a segurança dos arquivos localizados em um diretório que seja referenciado por cada objeto DIRECTORY. As medidas de segurança extremas nos objetos de banco de dados são penalizadas por uma segurança negligente nos arquivos do sistema operacional referenciado.

## Links de bancos de dados (database links) e bancos de dados remotos

Os links de bancos de dados permitem que um banco de dados Oracle referencie objetos armazenados fora do banco de dados local. O comando **create database link** cria o caminho para um banco de dados remoto que, por sua vez, permite o acesso a objetos nestes. Um link de banco de dados empacota o nome do banco de dados remoto, um método de conexão ao banco de dados remoto e uma combinação de nome de usuário/senha para autenticar a conexão ao banco de dados remoto. De certa maneira, um link de banco de dados é similar a um sinônimo de banco de dados: ele pode ser público ou privado e fornece um modo abreviado e conveniente de acessar outro conjunto de recursos. A principal diferença é que o recurso está fora do banco de dados em vez de estar no mesmo banco de dados e, portanto, requer mais informações para resolver a referência. A outra diferença é que um sinônimo é uma referência a um objeto específico, ao passo que um link de banco de dados é um caminho definido usado para acessar qualquer número de objetos em um banco de dados remoto.

Para que os links funcionem entre os bancos de dados em um ambiente distribuído, o nome do banco de dados global de cada banco de dados no domínio deve ser diferente. Portanto, é importante atribuir corretamente os parâmetros de inicialização DB_NAME e DB_DOMAIN.

Para facilitar ainda mais o uso de links de bancos de dados, você pode atribuir um sinônimo a um link para tornar o acesso de tabela ainda mais transparente; o usuário não sabe se o sinônimo acessa um objeto localmente ou em um banco de dados distribuído. O objeto pode ser movido para um banco de dados remoto diferente ou para um banco de dados local, e o nome do sinônimo pode permanecer o mesmo, tornando o acesso ao objeto transparente para os usuários.

Abordaremos como os links de bancos de dados a bancos de dados remotos são aproveitados em um ambiente distribuído no Capítulo 17.

# ESTRUTURAS DE ARMAZENAMENTO FÍSICO DO ORACLE

O banco de dados Oracle usa diversas estruturas de armazenamento físico no disco para conservar e gerenciar os dados de transações de usuários. Algumas dessas estruturas de armazenamento,

como arquivos de dados, arquivos de redo log e arquivos de redo log arquivados, contêm dados reais de usuários; outras estruturas, como arquivos de controle, mantêm o estado dos objetos do banco de dados e arquivos de rastreamento e alertas baseados em texto contêm informações de log tanto para eventos de rotina como condições de erro no banco de dados. A Figura 1-3 mostra o relacionamento entre essas estruturas físicas e as estruturas de armazenamento lógico que examinamos na seção anterior "Estruturas lógicas do banco de dados Oracle".

## Arquivos de dados

Cada banco de dados Oracle deve conter ao menos um *arquivo de dados*. Um arquivo de dados Oracle corresponde a um arquivo físico do sistema operacional em disco. Cada arquivo de dados em um banco de dados Oracle é membro de somente um tablespace que, entretanto, pode ser composto de muitos arquivos de dados. (Um tablespace BIGFILE é composto de exatamente um arquivo de dados.)

Um arquivo de dados Oracle pode ser expandido automaticamente quando não houver mais espaço, se o DBA o tiver criado com o parâmetro AUTOEXTEND. O DBA também pode limitar a quantidade de expansão para um determinado arquivo de dados usando o parâmetro MAXSIZE. De qualquer maneira, o tamanho do arquivo de dados estará, enfim, limitado pelo volume de disco no qual ele reside.

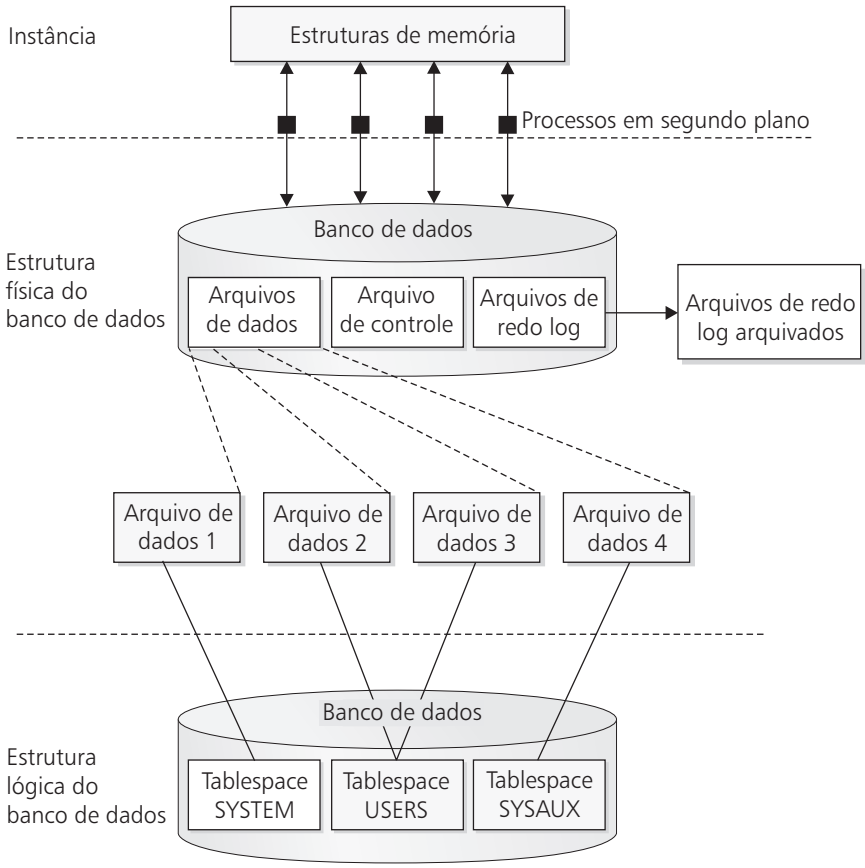
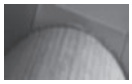



**Figura 1-3** *Estruturas de armazenamento físico do Oracle.*



**DICA**

*O DBA freqüentemente tem de decidir se deve alocar um arquivo de dados que possa ser auto-estendido indefinidamente ou alocar muitos arquivos de dados menores com um limite de quanto cada um deles pode ser estendido. Embora o desempenho de cada solução seja possivelmente muito similar, talvez seja melhor permanecer com mais arquivos de dados com menos de 2 GB cada um. É muito mais fácil mover arquivos relativamente pequenos e, de qualquer maneira, alguns sistemas de arquivos podem limitar o tamanho de um arquivo individual a 2 GB. Além disso, se for preciso mover temporariamente todos os arquivos de dados de um tablespace para outro servidor, geralmente será mais fácil encontrar vários volumes, cada um deles com espaço suficiente para conter um dos arquivos de dados, do que um volume com espaço suficiente para conter um único arquivo que tenha 25 GB.*

O arquivo de dados é o lugar de armazenamento final para todos os dados no banco de dados. Os blocos que são acessados freqüentemente em um arquivo de dados são armazenados em cache na memória; da mesma maneira, os novos blocos de dados não são imediatamente gravados no arquivo de dados mas, em vez disso, são gravados no arquivo de dados dependendo de quando o processo de gravação do banco de dados está ativo. No entanto, antes de uma transação de usuário ser considerada concluída, as alterações da transação são gravadas nos arquivos de redo log.

## Arquivos de redo log

Sempre que os dados são adicionados, removidos ou alterados em uma tabela, índice ou outro objeto Oracle, uma entrada é gravada no *arquivo de redo log* atual. Cada banco de dados Oracle deve ter ao menos dois arquivos de redo log, porque o Oracle reutiliza os arquivos de redo log de maneira circular. Quando um arquivo de redo log é preenchido com entradas, o arquivo de redo log atual é marcado como ACTIVE, caso ele ainda seja necessário para recuperação de instância, ou INACTIVE, caso ele não seja mais necessário para recuperação de instância; o próximo arquivo de log na seqüência é reutilizado do início do arquivo e é marcado como CURRENT.

Idealmente, as informações contidas em um arquivo de redo log nunca são usadas. Entretanto, quando ocorrer uma falha de energia ou alguma outra falha de servidor provocar uma falha na instância Oracle, os blocos de dados novos ou atualizados no cache do buffer do banco de dados talvez ainda não tenham sido gravados nos arquivos. Quando a instância Oracle for reiniciada, as entradas no arquivo de redo log serão aplicadas aos arquivos de banco de dados em uma operação *roll forward* para restaurar o seu estado até o ponto onde a falha ocorreu.

Para conseguir recuperar a perda de um arquivo de redo log dentro de um grupo de redo log, várias cópias de um arquivo de redo log podem existir em diferentes discos físicos. Posteriormente neste capítulo, você verá como os arquivos de redo log, arquivos de log arquivados e arquivos de controle podem ser *multiplexados* para garantir a disponibilidade e a integridade de dados do banco de dados Oracle.

## Arquivos de controle

Cada banco de dados Oracle tem ao menos um *arquivo de controle* que mantém os seus metadados (em outras palavras, os dados sobre a estrutura física do próprio banco de dados). Entre outras coisas, ele contém o nome do banco de dados, quando ele foi criado e os nomes e locais de todos os arquivos de dados e arquivos de redo log. Além disso, o arquivo de controle mantém as informações usadas pelo Recovery Manager (RMAN), como as configurações RMAN persistentes e os tipos de backups que foram executados no banco de dados. O RMAN será discutido em mais detalhes no Capítulo 12. Sempre que qualquer alteração é feita à estrutura do banco de dados, as informações sobre elas são imediatamente refletidas no arquivo de controle.

Como o arquivo de controle é muito importante para a operação do banco de dados, ele também pode ser multiplexado. Entretanto, independentemente da quantidade de cópias do arquivo de controle associada a uma instância, somente um dos arquivos de controle é designado como principal para fins de recuperação dos metadados do banco de dados.

O comando **alter database backup controlfile to trace** é outra maneira de fazer backup do arquivo de controle. Ele produz um script SQL que pode ser usado para recriar o arquivo de controle do banco de dados caso todas as versões binárias multiplexadas do arquivo de controle sejam perdidas devido a uma falha catastrófica.

Esse arquivo de rastreamento também pode ser usado, por exemplo, para recriar um arquivo de controle se o banco de dados precisar ser renomeado ou para alterar vários limites de banco de dados que, do contrário, não poderiam ser alterados sem recriar o banco de dados inteiro.

## Arquivos de log arquivados

Um banco de dados Oracle pode operar em um dos dois modos: modo **archivelog** ou **noarchivelog**. Quando o banco de dados está no modo **noarchivelog**, a reutilização circular dos arquivos de redo log (também conhecidos como arquivos de redo log *online*) significa que as entradas de redo (o conteúdo das transações anteriores) não estão mais disponíveis no caso de uma falha na unidade de disco ou outro tipo de falha relacionada à mídia. A operação no modo **noarchivelog** protege a integridade do banco de dados no caso de falha de uma instância ou uma queda de sistema, porque todas as transações encerradas com commit, mas que ainda não foram gravadas nos arquivos de dados estarão disponíveis nos arquivos de redo log online.

Por outro lado, o modo **archivelog** envia um arquivo de redo log preenchido para um ou mais destinos especificados e pode estar disponível para reconstruir o banco de dados a qualquer momento caso ocorra uma falha de mídia do banco de dados. Por exemplo, se a unidade de disco que contém os arquivos de dados for danificada, o conteúdo do banco de dados pode ser recuperado até o momento anterior ao problema, devido a um backup recente dos arquivos de dados e aos arquivos de redo log que foram gerados desde que ele ocorreu.

O uso de múltiplos destinos de logs arquivados para arquivos de redo log preenchidos é crucial para um dos recursos de alta disponibilidade do Oracle conhecido como Oracle Data Guard, antigamente chamado de Oracle Standby Database. O Oracle Data Guard será abordado abordado no Capítulo 13.

## Arquivos de parâmetro de inicialização

Quando uma instância de banco de dados inicia, a memória para a instância Oracle é alocada e um dos dois tipos de *arquivos de parâmetro de inicialização* é aberto: um arquivo texto denominado init<SID>.ora (conhecido genericamente como init.ora ou PFILE) ou um arquivo de parâmetro de servidor (conhecido como SPFILE). A instância primeiro procura um SPFILE no local padrão do sistema operacional ($ORACLE_HOME/dbs no Unix, por exemplo) como spfile<SID>.ora ou spfile.ora. Se nenhum desses arquivos existir, a instância procura um PFILE com o nome init<SID>.ora. Como alternativa, o comando **startup** pode especificar explicitamente um PFILE para ser usado na inicialização.

Os arquivos de parâmetros de inicialização, independentemente do formato, especificam as localizações para arquivos de rastreamento, arquivos de controle, arquivos de redo log preenchidos e assim por diante. Eles também definem os limites quanto aos tamanhos das várias estruturas na System Global Area (SGA) e também quanto à quantidade de usuários que podem se conectar ao banco de dados simultaneamente.

Até o Oracle9*i*, usar o arquivo init.ora era a única maneira de especificar os parâmetros de inicialização para a instância. Embora seja fácil editá-lo com um editor de texto, ele tem algumas desvantagens. Se um parâmetro dinâmico de sistema for alterado na linha de comando com o comando **alter system**, o DBA deverá lembrar-se de alterar o arquivo init.ora para que o novo valor do parâmetro entre em vigor na próxima vez que a instância for reiniciada.

Um SPFILE torna mais fácil e eficaz o gerenciamento de parâmetros para o DBA. Se um SPFILE estiver em uso para a instância em execução, qualquer comando **alter system** que altere um parâmetro de inicialização poderá alterar o parâmetro de inicialização automaticamente no SPFILE, alterá-lo somente para a instância em execução ou ambos. Nenhuma edição do SPFILE é necessária ou mesmo possível sem corrompê-lo.

Embora não seja possível espelhar um arquivo de parâmetros ou SPFILE, é possível fazer um backup de um SPFILE para um arquivo init.ora e tanto o init.ora como o SPFILE para a instância Oracle devem ser copiados em backup por meio dos comandos de sistema operacional convencionais ou usando o Recovery Manager no caso de um SPFILE.

Quando o DBCA é usado para criar um banco de dados, um SPFILE é criado por padrão.

## Arquivos de log de alerta e de rastreamento

Quando algo dá errado, o Oracle pode e freqüentemente gravar mensagens no *log de alerta* e, no caso de processos em segundo plano ou sessões de usuário, nos arquivos de *log de rastreamento*. O arquivo de log de alerta, localizado no diretório especificado pelo parâmetro de inicialização BACKGROUND_DUMP_DEST, contém mensagens de status de rotina e condições de erro. Quando o banco de dados é inicializado ou desligado, uma mensagem é registrada no log de alerta, junto com uma lista de parâmetros de inicialização que são diferentes dos seus valores padrão. Além disso, todos os comandos **alter database** ou **alter system** emitidos pelo DBA são registrados. Operações envolvendo tablespaces e seus arquivos de dados também são registradas aqui, como adicionar e descartar um tablespace e adicionar um arquivo de dados a um tablespace. As condições de erro, como tablespaces sem espaço, redo logs corrompidos e assim por diante, também são registradas aqui.

Os arquivos de rastreamento para os processos em segundo plano da instância Oracle igualmente estão localizados no BACKGROUND_DUMP_DEST. Por exemplo, os arquivos de rastreamento para PMON e SMON contêm uma entrada para quando ocorrer um erro ou quando o SMON precisar executar uma recuperação de instância; os arquivos de rastreamento para QMON contêm mensagens com informações para quando ele gerar um novo processo. Os arquivos de rastreamento também são criados para sessões ou conexões de usuário individual ao banco de dados. Esses arquivos de rastreamento estão localizados no diretório especificado pelo parâmetro de inicialização USER_DUMP_ DEST. Os arquivos de rastreamento para processos de usuário são criados em duas situações: a primeira é quando algum tipo de erro ocorre em uma sessão de usuário devido a um problema de privilégio, falta de espaço e assim por diante. Na segunda situação, um arquivo de rastreamento pode ser criado explicitamente com o comando **alter session set sql_trace=true**. As informações de rastreamento são geradas para cada instrução SQL que o usuário executa, o que pode ser útil ao ajustar uma instrução SQL do usuário.

O arquivo de log de alerta pode ser excluído ou renomeado a qualquer momento; ele será recriado na próxima vez que uma mensagem de log de alerta for gerada. O DBA costuma configurar um job em lote diariamente (seja por meio de um mecanismo do sistema operacional ou usando o Scheduler do Oracle Enterprise Manager) para renomear e arquivar o log de alerta.

## Arquivos de backup

Os arquivos de backup podem ser gerados de diversas formas, como comandos copy do sistema operacional ou o Oracle Recovery Manager (RMAN). Se o DBA excuta um backup "a frio" (cold backup) (consulte a seção intitulada "Visão geral de backup/recuperação" para obter mais detalhes sobre os tipos de backup), esses arquivos são simplesmente cópias via sistema operacional de arquivos de dados, arquivos de redo log, arquivos de controle, arquivos de redo log arquivados e assim por diante.

Além das cópias de imagem bit a bit dos arquivos de dados (o padrão no RMAN), o RMAN pode gerar backups completos e incrementais dos arquivos de dados, arquivos de controle, arquivos de redo log arquivados e SPFILEs que estão em um formato especial, denominado *conjuntos de backup,* legíveis somente pelo RMAN. Os conjuntos de backup do RMAN são geralmente menores do que os arquivos de dados originais porque o RMAN não faz backup de blocos não utilizados.

## Oracle managed files

O Oracle Managed Files (OMF), introduzido no Oracle versão 9*i*, facilita o trabalho do DBA automatizando a criação e a remoção de arquivos de dados que compõem as estruturas lógicas no banco de dados.

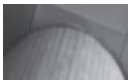
Sem o OMF, um DBA poderia descartar um tablespace e esquecer-se de remover os arquivos subjacentes no sistema operacional. Isso torna ineficiente o uso dos recursos de disco e aumenta desnecessariamente o tempo de backup para os arquivos de dados que não são mais necessários para o banco de dados.

O OMF é bem adequado para bancos de dados pequenos com um número baixo de usuários e sem um DBA dedicado, onde a configuração otimizada de um banco de dados de produção não é necessária.

## Arquivos de senha

Um *arquivo de senha* Oracle é um arquivo dentro da estrutura administrativa ou de diretório do software Oracle usado para autenticar os administradores de sistema Oracle para tarefas como criar um banco de dados ou inicializar ou efetuar shutdown nele. Os privilégios concedidos através desse arquivo são os privilégios SYSDBA e SYSOPER. A autenticação de qualquer outro tipo de usuário é feita dentro do próprio banco de dados; como ele pode estar fora do ar ou não montado, outra forma de autenticação de administrador é necessária nesses casos.

O utilitário de linha de comando Oracle **orapwd** cria um arquivo de senha caso ele ainda não exista ou esteja danificado. Devido aos privilégios extremamente altos concedidos por meio deste arquivo, ele deve ser armazenado em um diretório seguro que não esteja disponível para qualquer pessoa, a não ser para DBAs e administradores do sistema operacional. Após este arquivo ser criado, o parâmetro de inicialização REMOTE_LOGIN_ PASSWORDFILE deve ser definido como EXCLUSIVE para permitir que outros usuários além do SYS usem o arquivo de senha.



**DICA**

*Crie ao menos um usuário além do SYS ou SYSTEM que tenha privilégios de DBA para as tarefas administrativas diárias. Se houver mais de um DBA administrando um banco de dados, cada DBA deve ter sua própria conta com privilégios de DBA.*

Como alternativa, a autenticação para privilégios SYSDBA e SYSOPER pode ser feita com autenticação via Sistema Operacional; nesse caso, um arquivo de senha não precisa ser criado e o parâmetro de inicialização REMOTE_LOGIN_PASSWORDFILE é definido como NONE.

# MULTIPLEXANDO ARQUIVOS DE BANCO DE DADOS

Para minimizar a possibilidade de perda de um arquivo de controle ou um arquivo de redo log, a multiplexação dos arquivos de banco de dados reduz ou elimina os problemas de perda de dados causados por falhas de mídia. A multiplexação pode ser em parte automatizada por meio de uma instância do Automatic Storage Management (ASM), disponível desde o Oracle 10*g*. Para uma empresa com um orçamento menor, os arquivos de controle e os arquivos de redo log podem ser multiplexados manualmente.

## Automatic storage management

O *Automatic Storage Management* é uma solução de multiplexação que automatiza o layout dos arquivos de dados, arquivos de controles e arquivos de redo log por meio da distribuição desses arquivos para todos os discos disponíveis. Quando novos discos são adicionados ao cluster ASM, os arquivos de banco de dados são automaticamente distribuídos para todos os volumes de disco

de forma a otimizar o desempenho. Os recursos de multiplexação de um cluster ASM minimizam a possibilidade de perda de dados e são geralmente mais eficazes do que um esquema manual que coloca arquivos e backups cruciais em diferentes unidades físicas.

## Multiplexação manual

Sem uma solução RAID ou ASM, você ainda pode fornecer algumas salvaguardas para seus arquivos de banco de dados importantes configurando alguns parâmetros de inicialização e fornecendo um local adicional para os arquivos de controle, arquivos de redo log e arquivos de redo log arquivados.

### *Arquivos de controle*

Os arquivos de controle podem ser imediatamente multiplexados quando o banco de dados é criado ou podem ser multiplexados posteriormente com algumas etapas adicionais para copiá-los manualmente para vários destinos. É possível multiplexar até oito cópias de um arquivo de controle.

Multiplexar os arquivos de controle quando o banco de dados for criado, ou posteriormente, não altera o valor do parâmetro de inicialização CONTROL_FILES.

Caso você queira adicionar outro local multiplexado, é necessário editar o arquivo de parâmetro de inicialização e adicionar outro local para o parâmetro CONTROL_FILES. Se estiver usando um SPFILE em vez de um arquivo init.ora, utilize um comando similar ao mostrado a seguir para alterar o parâmetro CONTROL_ FILES:

```
alter system
     set control_files = '/u01/oracle/whse2/ctrlwhse1.ctl,
       /u02/oracle/whse2/ctrlwhse2.ctl,
       /u03/oracle/whse2/ctrlwhse3.ctl'
scope=spfile;
```

Os outros valores possíveis para SCOPE no comando **alter system** são MEMORY e BOTH.

A especificação de um desses valores para SCOPE retornará um erro, porque o parâmetro CONTROL_FILES não pode ser alterado para a instância em execução, somente para sua próxima reinicialização. Portanto, apenas o SPFILE é alterado.

Em ambos os casos, a próxima etapa é efetuar um shutdown no banco de dados. Copie o arquivo de controle para os novos destinos, conforme especificado em CONTROL_FILES e reinicialize o banco de dados. Sempre há possibilidade de verificar os nomes e locais dos arquivos de controle examinando uma das visões do dicionário de dados:

```
select value from v$spparameter where name='control_files';
```

Essa consulta retornará uma linha para cada cópia multiplexada do arquivo de controle. Além disso, a visão V$CONTROLFILE contém uma linha para cada cópia do arquivo de controle junto com seu status.

### *Arquivos de redo log*

Arquivos de redo log são multiplexados convertendo-se um conjunto de arquivos de redo log para um *grupo de arquivos de redo log*. Em uma instalação Oracle padrão, é criado um conjunto de três arquivos de redo log. Como você já aprendeu na seção anterior sobre arquivos de redo log, depois que cada arquivo de log é preenchido, ele começa a preencher o próximo na seqüência. Depois que o terceiro é preenchido, o primeiro é reutilizado. Para alterar o conjunto de três arquivos de log de redo para um grupo, é possível adicionar um ou mais arquivos idênticos associados a cada um dos arquivos de redo log existentes. Depois que os grupos são criados, as entradas de redo log são simultaneamente gravadas para o grupo de arquivos de redo log. Quando o grupo de arquivos de redo log é preenchido, ele começa a gravar as entradas de redo para o próximo grupo.

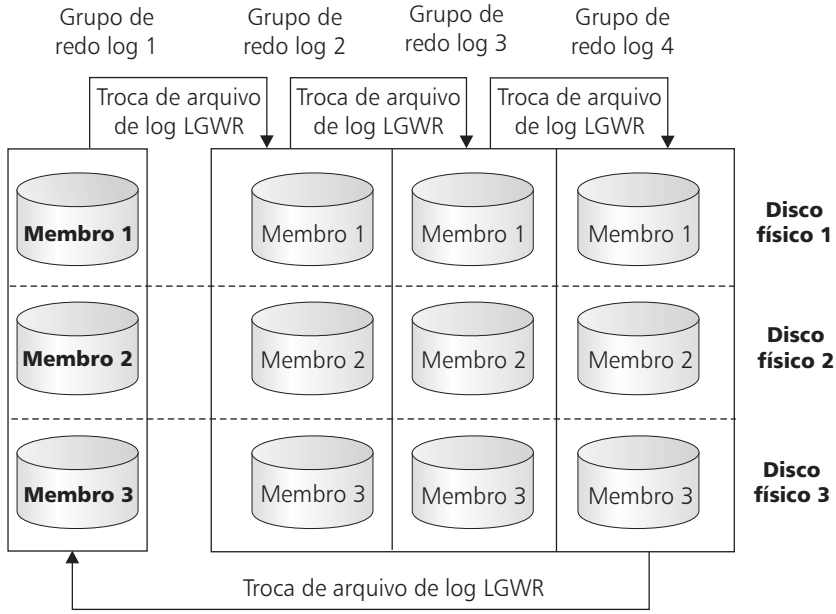



**Figura 1-4** *Multiplexando os arquivos de redo log.*

A Figura 1-4 mostra como um conjunto de quatro arquivos de redo log pode ser multiplexado com quatro grupos, cada grupo contendo três membros.

Adicionar um membro a um grupo de redo log é muito simples. No comando **alter database**, especificamos o nome do novo arquivo e o grupo ao qual adicioná-lo. O novo arquivo é criado com o mesmo tamanho dos outros membros do grupo:

```
alter database
     add logfile member '/u05/oracle/dc2/log_3d.dbf'
     to group 3;
```

Se os arquivos de redo log forem preenchidos mais rapidamente do que possam ser arquivados, uma solução possível é adicionar outro grupo de redo log. Eis um exemplo de como adicionar um quinto grupo de redo log ao conjunto de grupos de redo log da Figura 1-4:

```
alter database
     add logfile group 5
     ('/u02/oracle/dc2/log_3a.dbf',
      '/u03/oracle/dc2/log_3b.dbf',
      '/u04/oracle/dc2/log_3c.dbf') size 250m;
```

Todos os membros de um grupo de redo log devem ter o mesmo tamanho. Entretanto, os tamanhos dos arquivos de log entre os grupos podem ser diferentes. Além disso, os grupos de redo log podem ter número de membros diferente. No exemplo anterior, começamos com quatro grupos de redo log, adicionamos um membro extra ao grupo de redo log 3 (para um total de quatro membros) e adicionamos um quinto grupo de redo log com três membros.