Class Note: Chapter 10

Managing Database Space

(Updated May 17, 2016)

[The "class note" is the typical material I would prepare for my face-to-face class. Since this is an Internet based class, I am sharing the notes with everyone assuming you are in the class.]

This class/chapter is designed for typical functions of an Oracle database administrator (DBA). Databases store information in an organized manner. To store data, an Oracle database uses storage structures. An Oracle database has both logical and physical data storage structures that relate to one another.

- A logical storage structure is a conceptual organization of data, such as a database or a table.
- A physical storage structure is a tangible unit of data storage, such as a file or a data block.

In this chapter, you'll learn about the logical and physical storage structures in an Oracle database, including:

- Table spaces
- Data files
- Control files
- Data, index, temporary, and rollback segments
- Extents
- Data blocks
- Data partitioning for tables and indexes

Chapter Prerequisites

To practice the hands on exercises in this chapter, you need to start SQL*Plus and run the following command script

location`8iStarterKit\Sql\chap10.sql`

Where *location* is the file directory where you expanded the support archive that accompanies this book. For example, after starting SQL*Plus and connecting as SCOTT, you can run this chapter's SQL command script using the SQL*Plus command G, as in the following example (assuming that your chap10.sql file is in C:\temp\8iStarterKit\Sql).

```
SQL> @C:\temp\8istarterKit\Sql\chap10.sq1;
```

Once the script completes successfully, leave the current SQL*Plus session open and use it to perform this chapter's exercises in the order in which they appear.

CAUTION
Carefully read the Warnings screen when running the command script for this chapter. The script drops the PHOTOS tablespace created by a practice exercise later in this chapter. If you are using this script with a database that contains real data in a tablespace named PHOTOS, open and edit the script so that the DROP TABLESPACE command in the script does not conflict with a real tablespace in your database.

10.1. Tablespaces and Data Files

A *tablespace* is a logical organization of data within an Oracle database that corresponds to one or more physical data *files* on disk. Figure 10 1 illustrates the relationship between a tablespace and its data files.

When you create a new database object, such as a table or an index, Oracle stores the database object within the tablespace of your choice; when you do not indicate a specific tablespace for a new database object, Oracle stores the object in your account's default tablespace. See Chapter 9 for more information about setting the default tablespace for a user account.

The physical storage of database objects within a tablespace maps directly to the underlying data files of the tablespace. Figure 10 2 demonstrates how Oracle might store various tables in different tablespaces.

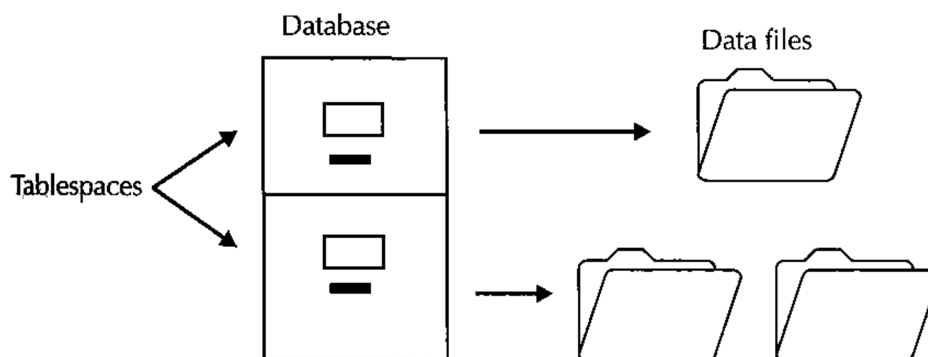


FIGURE 10-1. *Each tablespace in an Oracle database physically stores its data in one or more associated data files*

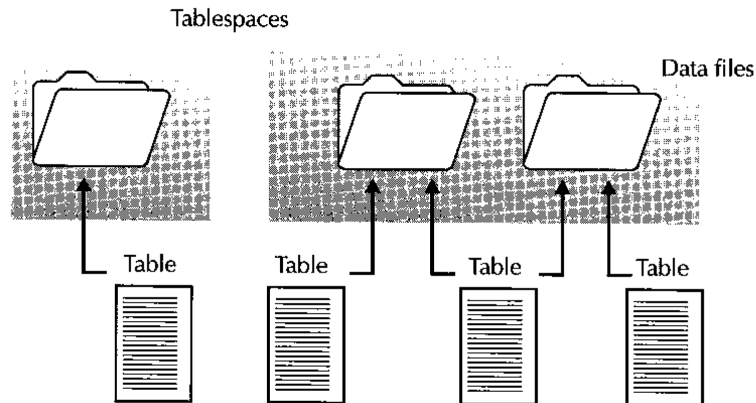


FIGURE 10-2. *Oracle stores objects in a tablespace in one or more data files that comprise the tablespace*

Figure 10-2 shows how Oracle can store data for database objects within tablespaces that have only one data file compared to tablespaces that have multiple data files.

- When a tablespace has only one data file, the tablespace stores the data of all associated objects within the one file.
- When a tablespace has multiple data files, Oracle can store the data for an object within any file of the tablespace. In fact, Oracle might distribute the data of a single object across multiple data files of a tablespace.

10.1.1. The SYSTEM Tablespace

Every Oracle database has at least one tablespace, the SYSTEM tablespace. When you create a new Oracle database, you must indicate the names, sizes, and other characteristics of the data files that make up the physical storage for the SYSTEM tablespace.

Oracle uses the SYSTEM tablespace for several purposes:

- Oracle stores a database's data dictionary in the SYSTEM tablespace. As Chapter 7 states, a database's data dictionary is a set of internal system tables that stores information about the database itself. A database's data dictionary also includes other objects that Oracle uses for internal system processing.
- The SYSTEM tablespace of a database stores the source and compiled code for all PL/SQL programs, such as stored procedures and functions, packages, database triggers, and object type methods. Databases that use PL/SQL extensively should have a sufficiently large SYSTEM tablespace.
- Database objects such as views, object type specifications, synonyms, and sequences are simple definitions that do not store any data. Oracle stores such object definitions in the data dictionary, which is in the SYSTEM tablespace.

10.1.2. Other Tablespaces

Although they are not required, an Oracle database typically has multiple tablespaces that logically and physically organize the storage of data within the database. Most Oracle databases have different tablespaces to separate storage in the following ways:

- Separate application data from the internal data dictionary information in the SYSTEM tablespace
- Separate an application's table data from index data
- Separate the system's transaction rollback data from other types of data
- Separate temporary data used during internal system processing from permanently stored data

For example, suppose you are planning to build an Oracle database to support an accounting and a manufacturing application, and each application uses a different set of database tables. One way to organize the database is to create multiple tablespaces that separate the storage of each application's tables and indexes. Figure 10 3 demonstrates this configuration, as well as providing distinct tablespaces for the system's temporary and rollback data.

By using multiple tablespaces for different sets of application data, you can manage the data for each application independently. For example, you can back up an active application's data frequently and a less active application's data less frequently.

NOTE
Your Windows NT starter database should have the following tablespaces: SYSTEM, USERS, RBS, TEMP, and INDX. Exercise 10.5, later in this chapter, teaches you how to display the names and other information about the tablespaces in your database.

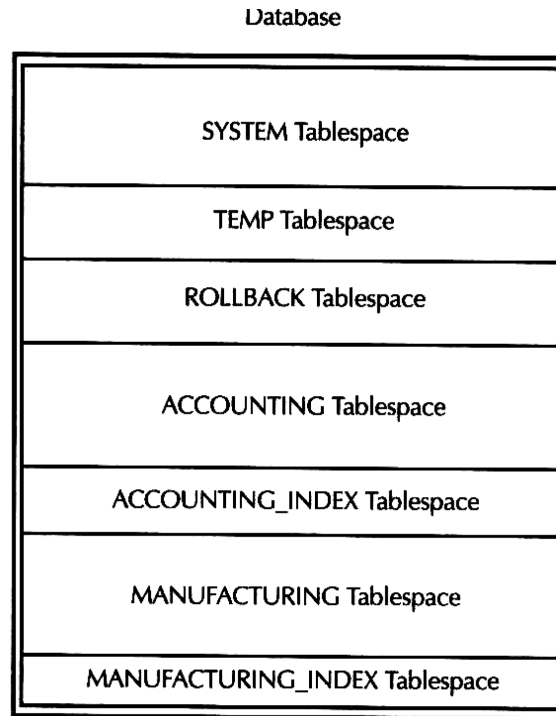


FIGURE 10-3. Using *multiple tablespaces to logically and physically separate the storage of different sets of database information*

10.1.3. Online and Offline Tablespaces

Oracle lets you control the availability of data in a database on a tablespace-by-tablespace basis. That is, a tablespace can either be online or offline.

- The data in an online tablespace is available to applications and databases. Typically, a tablespace remains online so that users can access the information within it.
- The data in an offline tablespace is not available to database users, even when the database is available. An administrator might take a tablespace offline to prevent access to an application's data, because the tablespace is experiencing a problem, or because the tablespace contains historical data that is typically not required by anyone.

NOTE
A database's SYSTEM tablespace must always remain online because information in the data dictionary must be available during normal operation. If you try to take the SYSTEM tablespace offline, Oracle returns an error.

10.1.4. Permanent and Temporary Tablespaces

Most tablespaces in an Oracle database are permanent tablespaces. A permanent tablespace stores information that must persist across individual SQL requests and

transactions. For example, a permanent tablespace is necessary to store table, index, or transaction rollback information.

Oracle also lets you create temporary tablespaces in a database. A temporary tablespace is a large temporary work space that transactions can use to process complicated SQL operations, such as sorted queries, join queries, and index builds. Rather then inefficiently creating and dropping many small temporary space allocations in a permanent tablespace, Oracle can quickly provide temporary work areas for SQL statements by managing entries in a temporary tablespace's sort segment table.

NOTE
To learn more about temporary tablespaces, see the section "Temporary Segments and Temporary Tablespaces" later in this chapter.

When you create a new tablespace, you can create it as either permanent or temporary. You can always change a current tablespace's type to permanent or temporary, if necessary. If you decide to use a temporary tablespace, Oracle will not use the tablespace until you target the tablespace for temporary operations in one or more users' account settings. See Chapter 9 for more information about specifying a user account's temporary tablespace setting.

10.1.5. Dictionary Managed and Locally Managed Tablespaces

By default, Oracle uses a database's data dictionary to manage the allocation and reclamation of space within a tablespace. To boost the performance of an active tablespace that frequently allocates and deallocates space for database objects, you can switch a *dictionary-managed tablespace* to a locally managed tablespace. With a locally managed tablespace, Oracle uses a bitmap that is part of the tablespace to manage the allocation and reclamation of tablespace space with less overhead than is required by a dictionary managed tablespace.

10.1.6. Read Only and Read Write Tablespaces

When you create a new tablespace, it is always a *read-write tablespace*. That is, you can create, alter, and drop database objects within the tablespace, and applications can query, add, modify, and delete information from the database objects within the tablespace. When applications must actively change data in a tablespace, the tablespace must operate as a read write tablespace.

In some cases, a tablespace stores historical data that never changes. When a tablespace's data never changes, you can make the tablespace a *read-only tablespace*. Making a static tablespace read only can protect it from inappropriate data modifications. Making a tablespace read only can also save time when performing database backups. That's because it's not necessary to back up a read only tablespace when you back up the other tablespaces of the database. After you create a new

tablespace and add data to it, you can alter the tablespace and make it a read only tablespace. If necessary, you can always switch a tablespace back to read write mode so that applications can update the objects within the tablespace.

10.1.7. A Tablespace's Data Files

A data file is a physical storage file on disk for a tablespace in an Oracle database. A tablespace can store all of its data in just a single data file, or a tablespace can have multiple data files to collectively store its data. When you create a tablespace, you can create one or more data files for the new tablespace. In general, you create a tablespace with multiple data files on different disks to distribute the disk I/O associated with accessing a tablespace's data. This technique is particularly useful when you explicitly partition database data. For more information about data partitioning, see the "Partitioning Large Tables and Indexes" section, later in this chapter.

NOTE
An Oracle database has an upper limit to the number of data files that it can have, and this limit is set during database creation. When planning a database and its tablespaces, make sure that you do not use too many data files to meet the storage requirements for the system, or else you might reach the upper limit for the number of data files.

After you create a tablespace, you can always add more data files to the tablespace to increase its storage capacity. For example, when a tablespace uses data files that do not grow in size, you can allocate additional storage space for the tablespace by creating one or more data files for the tablespace. The following sections provide more information about data file space usage and sizing options.

10.1.8. Use of Data File Space

When you create a new data file for a tablespace, Oracle preallocates the amount of disk space that you specify for the data file. After you create a new data file, it is like an empty data-bucket the file contains no data, but it is a receptacle that is ready to store database information.

Any time you create a new data storage object, such as a table or index in a tablespace, Oracle designates a certain amount of space from the tablespace's data files to the new object. Allocating data file space to a new database object reduces the remaining amount of available free space in the data file. As applications insert and update data in a data storage object, the preallocated space for the object can eventually become full.

If data consumes all of a data storage object's available storage space, Oracle can automatically allocate additional space from the tablespace's data files for the object.

Allocating more space to a data storage object to extend the storage capacity of the object further reduces the amount of available free space in the tablespace's data files.

10.1.9. Data File Sizes

In general, the size of a data file remains constant. As objects in a tablespace allocate space from the corresponding data files, the tablespace can become full if all of the data files in the tablespace become full. When applications attempt to insert or update data within a tablespace that's full, Oracle returns errors until more storage space becomes available for the tablespace. To increase the storage capacity of a full tablespace, you have a few different options:

- You can add one or more new data files to the tablespace.
- You can manually resize one or more of the existing data files in the tablespace.
- You can configure one or more of the data files in the tablespace to automatically extend when the tablespace becomes full and requires more space.

Each option has certain advantages and disadvantages. For example, the first two options are fine if you are a watchful administrator who frequently monitors the storage capacity of your database's data files. When you notice that a tablespace is running low on free space, you can add more files to the tablespace or increase the size of one of the tablespace's data files. In contrast, the third option allows a tablespace's storage capacity to grow automatically, without manual assistance. If you choose to manually or automatically extend the storage allocation for a data file, do so conservatively fragmented allocations of disk space across a disk drive can decrease database performance.

10.1.10. Data File Corruption

Unfortunately, operating system files are always vulnerable to disk and I/O problems. Such problems can corrupt the integrity of a file. At the expense of system performance, you can configure Oracle to detect and log block level data file corruption.

10.1.11. Online and Offline Data Files

Oracle controls the availability of individual data files of a tablespace. A data file can either be online (available) or offline (not available). Under normal circumstances, a data file is online. When Oracle attempts to read or write a data file and cannot do so because some type of problem prevents this from happening, Oracle automatically takes the data file offline. The encompassing tablespace remains online, because other data files of the tablespace might still be available. You can take a data file offline manually when a known problem exists. Once the problem is fixed (for

example, after a data file recovery), you can bring an offline data file back online manually.

NOTE
The data files of a database's SYSTEM tablespace must always remain online because the data dictionary must always be available during system operation. If Oracle experiences a problem reading or writing a data file in the database's SYSTEM tablespace, the system will not operate correctly until you fix the problem.

10.2. Creating and Managing Tablespaces and Data Files

Now that you have a good understanding of tablespaces and data files, the following practice exercises will teach you how to create and manage tablespaces and their data files.

EXERCISE 10.1: Creating a Tablespace

You create a tablespace in an Oracle database using the SQL command *CREATE TABLESPACE*. For the purposes of this exercise, the abbreviated syntax of the CREATE TABLESPACE command is as follows:

```
 CREATE TABLESPACE tablespace
  DATAFILE
  'filename' [SIZE integer [K/M]] [REUSE]
  [ AUTOEXTEND
  { OFF
  / ON
      [NEXT integer [K/M]]
      [MAXSIZE {UNLIMITED/linteger [K/M]}] } ]
[, ... other data file specifications ... ]
[ONLINE/OFFLINE]
[PERMANENT/TEMPORARY]
```

NOTE
You must have the CREATE TABLESPACE system privilege to create a tablespace.

Notice that when you create a tablespace, you can specify one or more data file specifications for the tablespace, create the tablespace as a permanent or temporary tablespace, and leave the new tablespace in an online or offline state. Assuming that you installed Oracle in the O:\ORACLE directory on your computer, enter the following statement to create a new tablespace called PHOTOS.

```
CREATE TABLESPACE photos
 DATAFILE
  'o:\oracle\oradata\oracle\photos1.dbf' SIZE 100K REUSE
   AUTOEXTEND ON NEXT 100K MAXSIZE 1M,
```

```
 'o:\oracle\oradata\oracle\photos2.dbf' SIZE 100K REUSE
ONLINE
PERMANENT;
```

NOTE
If you installed Oracle in a different directory, modify the file specifications in the previous exercise appropriately.

The previous CREATE TABLESPACE statement creates a new tablespace called PHOTOS with two data files.

- The photos1.dbf file is initially 100K in size, and can automatically extend itself in 100K increments up to 1 MB if the tablespace becomes full and an object requests more space in the tablespace.
- The photos2.dbf file is 100K in size and cannot increase in size unless you manually resize the file or enable automatic extension for the file.
- The REUSE option of a data file specification instructs Oracle to reuse the file, if it exists. The example includes this option in each data file specification so that you can repeat this exercise each time you review the chapter.

EXERCISE 10.2: Modifying the Storage Properties of a Data File

You can alter the properties of a data file at any time using the DATAFILE clause of the SQL command ALTER DATABASE. For the purposes of this exercise, the syntax of this clause in the ALTER DATABASE command is as follows:

```
ALTER DATABASE
 DATAFILE 'filename'
 { RESIZE integer [KIM]
 / AUTOEXTEND
  { OFF
 / ON [NEXT integer [K/M]] [MAXSIZE {UNLIMITED/integer[K/M]}]}}
```

NOTE
You must have the ALTER DATABASE system privilege to use the ALTER DATABASE command.

For example, enter the following ALTER DATABASE statement to adjust the automatic extension properties of the photos1.dbf data file.

```
ALTER DATABASE
 DATAFILE 'o:\oracle\oradata\oracle\photosl.dbf'
  AUTOEXTEND ON NEXT 250K MAXSIZE UNLIMITED;
```

Now enter the following ALTER DATABASE statement to manually resize the photos2.dbf data file.

```
ALTER DATABASE
    DATAFILE 'o:\oracle\oradata\oracle\photos2.dbf' RESIZE 250K;
```

EXERCISE 10.3: Making a Temporary Tablespace

You can switch a permanent tablespace to a temporary tablespace, or vice versa, using the *PERMANENT* and *TEMPORARY* options of the SQL command *ALTER TABLESPACE*.

```
ALTER TABLESPACE tablespace
 [PERMANENT/TEMPORARY]
```

NOTE
You can switch a tablespace from permanent to temporary only if the tablespace does not contain any permanent database objects (such as tables and indexes).

In your starter database, there is a tablespace named TEMP. By default, this tablespace is set as a permanent tablespace rather than a temporary tablespace. In this exercise, enter the following statement to switch the TEMP tablespace to a temporary tablespace.

```
ALTER TABLESPACE temp
    TEMPORARY;
```

TIP
To improve performance, it would also be a good idea to set the temporary tablespace setting for all user accounts to the TEMP tablespace. See Chapter 9 for more information about altering the temporary tablespace setting for a user account using the SQL command ALTER USER.

EXERCISE 10.4: Coalescing Free Space in a Tablespace's Data Files

As you create data storage objects in a tablespace and these objects extend their storage capacity, the free space areas in the tablespace's data files can become fragmented and small. Oracle might not be able to complete subsequent free space allocations for new or existing objects if the free space areas on disk are not large enough. To fix this problem, Oracle can coalesce many small adjacent free space areas into fewer, large free space areas. Figure 10 4 illustrates the concept of coalescing free space.

Oracle periodically coalesces a tablespace's free space automatically as an internal system operation. However, you can coalesce a tablespace manually when you know that this operation is necessary, using the *COALESCE* option of the SQL command *ALTER TABLESPACE*. For example, enter the following statement to coalesce the RBS tablespace's free space.

```
ALTER TABLESPACE rbs
COALESCE;
```

EXERCISE 10.5: Display Properties for All Tablespaces

You can display information about your database's tablespaces by querying the DBA_TABLESPACES data dictionary view. For example, enter the following query to display the name, availability status, and type for each tablespace in your database.

```
SELECT tablespace_name, status, contents
   FROM dba_tablespaces;
```

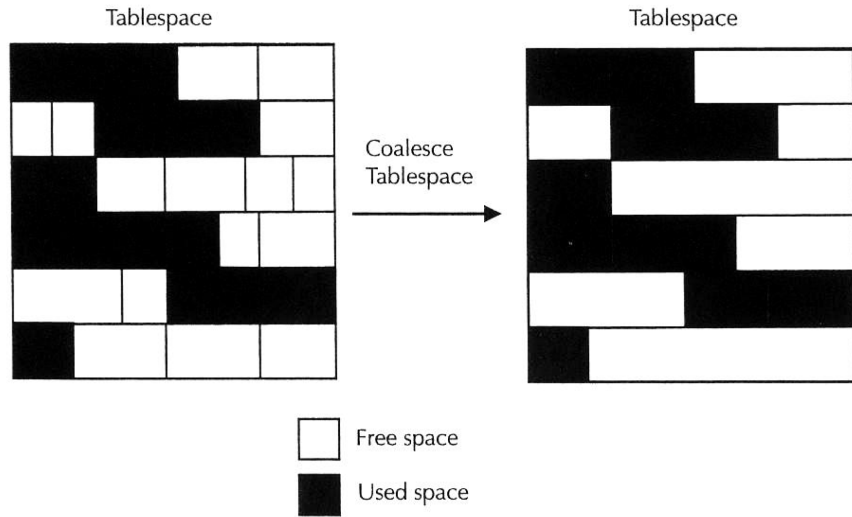


FIGURE 10-4.*Coalescing free space in a tablespace*

The result set should be similar to the following:

```
TABLESPACE_NAME    STATUS       CONTENTS
---------------    ------      ---------
 SYSTEM            ONLINE      PERMANENT
 USERS             ONLINE      PERMANENT
 RBS               ONLINE      PERMANENT
 TEMP              ONLINE      TEMPORARY
 INDX              ONLINE      PERMANENT
 PHOTOS            ONLINE      PERMANENT

6 rows selected.
```

NOTE
The STATUS field in the DBA_TABLESPACES view displays either ONLINE or OFFLINE for all read write tablespaces, and READ ONLY for read only tablespaces.

EXERCISE 10.6: Displaying Information about Data Files

You can display information about a database's data files by querying the DBA_DATA_FILES data dictionary view. For example, enter the following query to display the name, associated tablespace, size (in bytes), and automatic extension capability of each data file in the database.

```
SELECT file_name, tablespace_name, bytes, autoextensible AS auto
    FROM dba_data_files
    ORDER BY tablespace_name, bytes;
```

Assuming that the home directory location for Oracle8i on your computer is O:\ORACLE, the following should be similar to your result set:

```
FILENAME                             TABLESPACE_NAME  BYTES AUTO
--------                             ---------------  ----- ----
0:\ORACLE\ORADATA\ORACLE\INDX01.DBF      INDX        2097152 YES
0:\ORACLE\ORADATA\ORACLE\RBSOI.DBF       RBS        26214400 YES
0:\ORACLE\ORADATA\ORACLE\SYSTEMOI.DBF    SYSTEM    146800640 YES
O:\ORACLE\ORADATA\ORACLE\TEMPOI.DBF      TEMP        2097152 YES
0:\ORACLE\ORADATA\ORACLE\PHOTOSI.DBF     PHOTOS       102400 YES
0:\ORACLE\ORADATA\ORACLE\PHOTOS2.DBF     PHOTOS       256000  NO
0:\ORACLE\ORADATA\ORACLE\USERSOI.DBF     USERS       8388608 YES
 7 rows selected.
```

EXERCISE 10.7: Controlling Tablespace Availability

You can control the access to a tablespace using the *ONLINE* and *OFFLINE* options of the SQL command *ALTER TABLESPACE*.

```
ALTER TABLESPACE tablespace
 {ONLINE/OFFLINE [NORMAL/TEMPORARY/IMMEDIATE/FOR RECOVER]}
```

When taking an online tablespace offline, you can indicate several options:

- The default NORMAL option indicates that you want to take the target tablespace offline under normal conditions. You can use the NORMAL option only if all data files of the tablespace are currently online and available without any I/O problems.
- The TEMPORARY option should be your first choice if you want to take a tablespace off line under abnormal conditions. For example, you might use the TEMPORARY option when one or more data files of the tablespace are off line due to I/O problems. In this case, a recovery operation will be necessary before you can bring the tablespace online again.
- The IMMEDIATE option should be your last resort when taking a tablespace offline under abnormal conditions.
- The FOR RECOVER option explicitly indicates that not all data files in the tablespace are available and that you understand that a recovery operation will be necessary to bring the tablespace online. Do not use this option under normal circumstances.

For example, enter the following query, which targets a table stored in the USERS tablespace.

```
SELECT id, description FROM parts;
```

Because the PARTS table is stored in the USERS tablespace, which is online at the moment, the result set for the query should be as follows:

```
 ID     DESCRIPTION
---    ------------
  1     Fax Machine
  2     Copy Machine
  3     Laptop PC
  4     Desktop PC
  5     Scanner
```

Next, enter the following ALTER TABLESPACE statement to take the USERS tablespace offline.

```
ALTER TABLESPACE users
  OFFLINE NORMAL;
```

Now, reenter the previous query, shown here:

```
SELECT id, description FROM parts;
```

You should see results similar to the following:

```
ERROR at line 1:
ORA 00376: file 2 cannot be read at this time
ORA 01110: data file 2: 'O:\ORACLE\ORADATA\ORACLE\USERSO1.DBF'
```

This error is an indication that the tablespace (or the tablespace's data files) that stores the PARTS table is offline. To bring the USERS tablespace back online, enter the following statement.

```
ALTER TABLESPACE users
   ONLINE;
```

Now, reenter the previous query:

```
SELECT id, description FROM parts;
```

You should again see the following results:

```
 ID    DESCRIPTION
---   ------------
1     Fax Machine
2     Copy Machine
3     Laptop PC
```

```
4     Desktop PC
5     Scanner
```

10.3. Control Files

Every Oracle database has a *control* file. A database's control file contains information about the physical structure of the database. For example, a database's control file includes the name of the database, as well as the names and locations of all files associated with the database. Oracle also uses a database's control file to keep track of internal system information to log the current physical state of the system, including information about tablespaces, data files, and system backups. A database's control file also logs information about database backups that you make with the Recovery Manager utility (see Chapter 11).

When you create a new database, Oracle creates the database's control file. Subsequently, Oracle updates the database's control file automatically with internal information that it needs to record. Additionally, every time you change a physical attribute of a database, Oracle updates the information in the database's control file. For example, when you create a new tablespace with one or more data files, or add a data file to an existing tablespace, Oracle updates the database's control file to log information about the new data files.

10.3.1. Mirrored Control Files

An Oracle database cannot function properly without its control file. To ensure database availability in the event of an isolated disk failure, Oracle lets you mirror a database's control file to multiple locations. When you mirror a database's control file to multiple locations, Oracle updates every copy of the control file at the same time. If one copy of the control file should become inaccessible due to a disk failure, other copies of the control file remain available and permit database processing to continue without interruption.

EXERCISE 10.8: Displaying Your Database's Control File Copies

You can display the names of all control file copies for your database by entering the following query, which targets the V$PARAMETER data dictionary view.

```
SELECT name, value FROM v$parameter
   WHERE name = 'control files';
```

Assuming that the home directory location for Oracle8i on your computer is O:\ORACLE, the following should be your result set:

```
NAME               VALUE
------          ---------------------------------------
control_files   O:\Oracle\oradata\oracle\contro101.ct1,
                O:\Oracle\oradata\oracle\contro102.ct1
```

Notice that that default configuration for the starter database on Oracle8i for Windows NT is to create two copies of the control file in the same location. Exercise 11.8 in the next chapter teaches you how to mirror the database's control file to different locations.

10.4. Segments, Extents, and Data Blocks

Just as Oracle preallocates data files to serve as the physical storage for tablespaces in a database, Oracle preallocates segments of data blocks as the physical storage for database objects such as tables, indexes, data clusters, and other data storage objects. Oracle allocates groups of contiguous *data blocks* for a database object as *extents*. A *segment* is the collection of all the extents dedicated to a database object. Figure 10-5 demonstrates the relationship between a table and its data segment, extents, and data blocks.

When you create a new data storage object, such as a table or an index, you can indicate the tablespace in which the corresponding segment should be created. Oracle then allocates data blocks from one or more of the data files in use by the target tablespace.

10.4.1. Types of Segments in an Oracle Database

An Oracle database can contain many different types of segments, including data segments, index segments, LOB segments, overflow segments, rollback segments, end temporary segments.

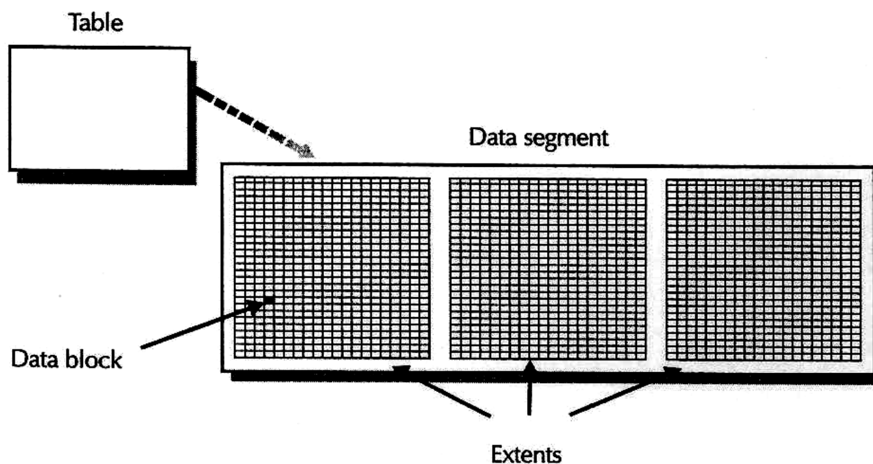


FIGURE 10-5. *Dedicating physical data storage for a table as extents (groups of contiguous data blocks) in a table's data segment*

- A table stores its data in a *data segment*. If the table has multiple partitions, the table stores its data in a corresponding number of data segments. Additionally, a data cluster stores the data of all tables in the cluster within the cluster's data segment.

- An index stores its data in an *index segment*. If the index has multiple partitions, the index stores its data in a corresponding number of index segments.
- If a table has a column that uses a large object (LOB) datatype, such as CLOB, BLOB, or NCLOB, the table can store corresponding LOB values in a *LOB segment* that is separate from the data segment that holds other field values in the table.
- An index organized table is a special type of table that stores the rows of a table within an index segment. An index organized table can also have an *overflow segment to* store rows that do not fit into the original index segment. See Chapter 12 for more information about index organized tables.
- A *rollback segment* is a special type of segment that Oracle uses to store transaction rollback information. When a user rolls back a transaction, Oracle applies information in a rollback segment to "undo" the transaction's operations.
- Oracle creates and uses a *temporary segment* when processing certain types of SQL statements. A temporary segment is a temporary work space on disk that Oracle can use to store intermediate data during SQL statement processing.

10.4.2. Data Blocks

A data block is the unit of disk access for an Oracle database. When you work with a database, Oracle stores and retrieves data on disk using data blocks. For example, when you query a table, Oracle reads into the server's memory all of the data blocks that contain rows in the query's result set.

When you create a database, you can specify the block size that the database will use. A database's block size must be equal to or a multiple of the server's operating system block size. For example, if the server's operating system block size is 512 bytes, the database block size on such a server could be 512 bytes, 1024 bytes, 2048 bytes, and so on.

10.4.2.1. Data Block Allocation

When you create a new data storage object, such as a table, index, or rollback segment, Oracle allocates one or more extents for the object's segment. An extent is a set of contiguous data blocks in a data file of the tablespace that stores the object's segment. If all data blocks in a segment's existing extents are full, Oracle allocates a new extent (set of blocks) for the segment the next time a transaction requests the storage of some new data.

10.4.2.2. Data Block Availability, Free Lists, and Free List Groups

Every data and index segment in an Oracle database has one or more data block free lists. A free list is a catalog of data blocks that are available to hold new data for the corresponding table, cluster, or index. Figure 10 6 shows how data blocks

can go on and off a table's free lists as transactions insert, update, and delete information from a table.

In Figure 10-6, you can see how a data block might go on and off a table's free list. When a transaction wants to insert a new row into the table, all data blocks on the free list are candidates for receiving the data for the new row. As transactions insert more and more rows into the table, data blocks eventually become full (or nearly full), such that Oracle removes them from the table's free list. A block can return to the table's free list after a transaction deletes rows from the table, freeing up space in the block.

NOTE
See Exercise 10.13 later in this chapter, which explains how to control when Oracle puts data blocks on a segment's free lists.

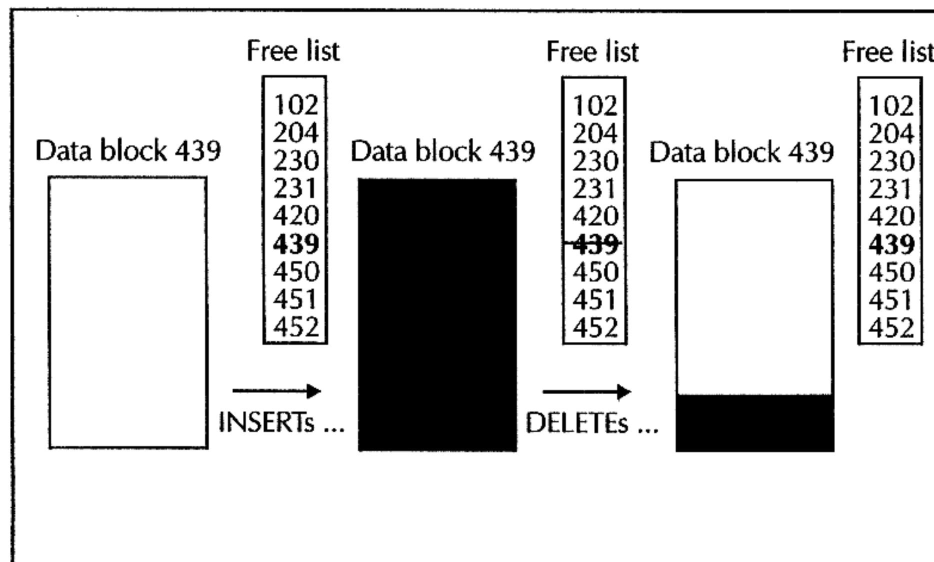


FIGURE 10-6. *A data block going on and off an object's free list, depending on the space available in the block*

When you create a table, cluster, or index, you can indicate the number of free list groups to create for the corresponding segment. A *free list group* is a group of one or more data block free lists for a data storage object's segment. By default, Oracle creates one free list group with one free list per data or index segment. If you are using Oracle with the Parallel Server option, it's possible to reduce contention among servers for free list lookups and improve system performance by creating multiple free list groups with multiple free lists.

10.4.2.3. Row Chaining and Data Block Size

When you insert a new row into a table, Oracle puts the new row into a data block that's on the table's free list. Optimally, Oracle puts all of a row's data into one

data block, assuming that the row can fit within the space of one data block. This way, when you request a row in a table, Oracle has to read only one data block from disk into memory to retrieve all of a row's data.

If a row's length is greater than the data block size, Oracle *chains* the row among two or more data blocks. Figure 10-7 illustrates row chaining.

Row chaining, while unavoidable in this situation, is not desirable. That's because Oracle must read multiple data blocks from disk into memory to access a row's data. More disk I/Os always slows system performance. Therefore, row chaining should be avoided if at all possible.

Typically, the default block size for an installation of Oracle is the optimal setting for most databases. However, databases with certain characteristics can benefit from block sizes that are larger than the default. For example, when many tables in a database will have rows that exceed the default block size, you can reduce the amount of row chaining in the database by creating the database with a larger block size. Oracle can also create a row chain when you update a row in a table or an index. This type of row chaining happens when both of the following occur:

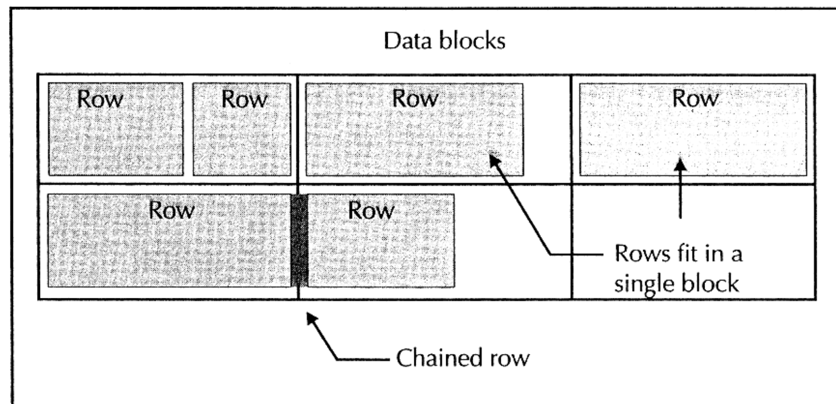


FIGURE 10-7. *Storing pieces of a row in a chain that spans multiple data blocks*

- You update the row so that it is longer than the original row, and
- The data block that holds the row does not have enough empty space to accommodate the update

When you expect that updates to the rows in a data or index segment will increase row sizes, you can prevent row chaining by reserving extra data block space for updates. See Exercise 10.13 later in this chapter, which explains how to control space usage within data blocks.

10.4.2.4. Transaction Entries

When a transaction updates a data block of an object, Oracle allocates a transaction (:entry in the data block. A transaction entry is a small amount of space in the header of a data block that Oracle uses to hold internal processing information until the ;'transaction commits or rolls back. Later in this chapter you'll learn how to fine tune the way Oracle allocates transaction entries within the data blocks of a table, data ',cluster, or index.

10.5. Managing Storage Settings for Tables and Indexes

With great detail, Oracle lets you control the storage characteristics for all types of, segments in a database. Now that you have a general understanding of segments, extents, and data blocks, the practice exercises in this section explain some of the most common tasks that you should consider when creating the tables (data segments) and indexes (index segments) in your database.

NOTE
Subsequent sections of this chapter discuss specialized types of segments in an Oracle database, including rollback segments, LOB segments, and temporary segments.

Exercise 10.9: Display the Segments in Your Schema

You can display information about all of the segments in a database by querying the DBA_ SEGMENTS data dictionary view. For example, enter the following query to display selected information about the segments in your current schema, PRACTICE10.

```
SELECT segment_name, segment_type, tablespace_name, extents, blocks
 FROM dba_segments
 WHERE owner = 'PRACTICE10';
```

The result set is as follows:

```
SEGMENT_NAME      SEGMENT_TYPE TABLESPACE_NAME  EXTENTS  BLOCKS
---------     ---------- ------------ ------ ------
PARTS              TABLE      USERS      1      5
SALESREPS          TABLE      USERS      1      10
P_ID               INDEX      USERS      1      5
PAR_DESCRIPTION    INDEX      USERS      1      5
S_ID               INDEX      USERS      1      5
```

Notice in the result set that there are several segments that correspond to the tables and indexes in your schema, and that each individual segment consists of a single extent.

NOTE
To display information about the segments in your current schema, you could also query the USER_SEGMENTS data dictionary view.

EXERCISE 10.10: Creating a Table in a Specific Tablespace

When you create a new data storage object, such as a table, data cluster, or index, you can use the *TABLESPACE* parameter of the corresponding CREATE command to explicitly indicate which tablespace the object's segment should be created within. Provided that you have the necessary quota in the tablespace and the necessary privileges to create the object, Oracle completes the request. If you omit a tablespace specification when creating a new table, data cluster, or index, Oracle creates the object's segment in your user account's default tablespace.

For example, enter the following CREATE TABLE statement. Use the TABLESPACE parameter to create the ITEMS table in the USERS tablespace.

```
CREATE TABLE items (
 o_id INTEGER,
 id INTEGER,
 p_id INTEGER NOT NULL,
 quantity INTEGER DEFAULT 1 NOT NULL
)
TABLESPACE users;
```

EXERCISE 10.11: Controlling the Number and Size of Extents for a Table

When you create a new table, cluster, or index, you can use the STORAGE clause of the corresponding CREATE command to control several different storage settings related to the allocation of extents for the object's segment. For the purposes of this introductory exercise, an abbreviated form of the STORAGE clause for most CREATE commands in SQL is as follows:

```
STORAGE  (
[INITIAL integer [K/M]]
[NEXT [K/M]]
[MINEXTENTS integer]
[MAXEXTENTS {integer / UNLIMITED}]
[PCTINCREASE integer]
```

- Use the *MINEXTENTS* parameter of the STORAGE clause to determine the number of extents to allocate when creating the segment. Oracle must allocate at least one extent when creating a new segment for a table, cluster, or index.
- Use the *MAXEXTENTS* parameter of the STORAGE clause to limit the maximum number of extents that Oracle can ever allocate for the segment.
- You can control the size of the segment's extents. Use the *INITIAL* parameter of the STORAGE clause to set the size of the segment's initial extent. Use the *NEXT* parameter to control the size of subsequent extents. And use the *PCTINCREASE* parameter to specify a growth factor to apply before allocating subsequent extents for the segment.

For example, suppose you know that your order entry application will need to process a lot of orders every day. In this case, it would be wise to create the ITEMS table with enough storage space to accommodate the expected business. In view of this, let's drop the ITEMS table created in Exercise 10.10 and then recreate it, this time specifying specific storage parameters for the extents of the new table.

```
DROP TABLE items;

CREATE TABLE items (
 O_id INTEGER,
 id INTEGER,
 p_id INTEGER NOT NULL,
 quantity INTEGER DEFAULT 1 NOT NULL
 )
TABLESPACE users
STORAGE (
 INITIAL 100K
 NEXT 100K
 MINEXTENTS 1
 MAXEXTENTS 10
 PCTINCREASE 50
);
```

When Oracle creates the data segment for the ITEMS table, the server allocates one initial extent, 100K in size, for the segment. When this initial extent fills, Oracle allocates the next extent, 100K in size, and updates the segment's next extent size to 150K-that is, the current setting of NEXT (100K) increased by PCTINCREASE (50 percent). When a third extent is necessary, Oracle allocates the next extent, 150K in size, and updates the segment's NEXT storage parameter to 225K (that is, 150K increased by 50 percent). Extent allocation for the data segment continues in this manner until Oracle allocates the tenth extent, which is the limit for the number of extents for the segment. Of course, you can alter an object's storage settings, for example, to increase the maximum number of extents for the object.

NOTE
You can also set a segment's MAXEXTENTS storage parameter to UNLIMITED so that the segment can allocate an unlimited number of extents within the encompassing tablespace.

EXERCISE 10. 12: Altering Extent Settings for a Table

You can alter many extent settings for a table, cluster, or index using the STORAGE clause of the corresponding ALTER command, as follows.

```
STORAGE (
 [NEXT [KIM]]
 [MAXEXTENTS {integer/UNLIMITED}]
 [PCTINCREASE integer]
```

Notice that when you alter extent settings for a segment, it does not make sense to alter the segment's INITIAL or MINEXTENTS storage parameters, because the segment already exists. Furthermore, you cannot specify the TABLESPACE parameter in an ALTER command to "move" the segment from its current tablespace to another- if you want to move a data or index segment to another tablespace, you must drop and recreate the associated table, cluster, or index.

In this practice exercise, assume that you have incorrectly set the NEXT and PCT INCREASE storage parameters for the new ITEMS table. Enter the following ALTER TABLE statement to adjust these parameters.

```
ALTER TABLE items
 STORAGE (
  NEXT 200K
  MAXEXTENTS UNLIMITED
  PCTINCREASE 0
);
```

EXERCISE 10.13: Setting a Table's Data Block Settings

This practice exercise teaches you how to control the use of space within the data locks of the table, index, or cluster using the PCTUSED, PCTFREE, INITTRANS, MAXTRANS, FREELIST GROUPS, and FREELISTS parameters of corresponding CREATE commands.

```
[PCTUSED integer] [PCTFREE integer]
[INITRANS integer] [MAXTRANS integer]
[STORAGE (
 ... other storage parameters ...
[FREELIST GROUPS integer] [FREELISTS integer]   ) ]
```

Use the PCTFREE and PCTUSED parameters to control when data blocks go on and off a table's, index's, or cluster's free lists. You can set a maximum threshold using the PCTFREE parameter to control how much data block space to reserve for future updates to rows. When a data block becomes PCTFREE full, Oracle removes the block from the corresponding segment's free lists. When you expect that only occasional updates will increase the size of rows, set PCTFREE to a low value (perhaps 5 or 10) so that Oracle fills more space of each data block. However, if you expect frequent updates that will increase the size of rows, set PCTFREE to a high value (perhaps 20 or 30) so that Oracle reserves more block space for the updates to existing rows; otherwise, row chains are likely to occur.

You can set a minimum block threshold using the PCTUSED parameter to control when a data block is put back on the corresponding segment's free list. For example, the default PCTUSED for all segments is set to 40 percent. Therefore, when transactions delete rows from a data block so that it becomes only 39 percent full, Oracle puts the block back on the corresponding segment's free lists. When you expect only occasional deletes, set PCTUSED to a high value (perhaps

60) so that data blocks can pop on the free list when the occasional delete happens. However, if you expect frequent deletes, set PCTUSED to a low value (perhaps 40) so that Oracle does not constantly incur the overhead of moving blocks on and off the table's free lists.

NOTE
The sum of the settings for PCTFREE and PCTUSED cannot be greater than 100.

Use the INITRANS and MAXTRANS parameters to tune how Oracle allocates transaction entries within the data blocks of a table, data cluster, or index. The INITRANS parameter determines how much data block header space to preallocate for transaction entries. When you expect many concurrent transactions to touch a data block, you can preallocate more space for associated transaction entries and avoid the overhead of dynamically allocating this space when necessary. The *MAXTRANS* parameter limits the number of transactions that can concurrently use a data block. When you expect many transactions to concurrently access a small table (or index), set the table's INITRANS and MAXTRANS parameters to high values (perhaps 5 and 10, respectively) when creating the table. A higher INITRANS setting preallocates more block space for transaction entries, and a higher MAXTRANS parameter allows many transactions to concurrently access the table's data blocks. With large tables, it's less likely that several transactions will access the same data blocks simultaneously. Consequently, the setting for a large table's INITRANS and MAXTRANS parameters can be correspondingly low (perhaps 2 and 5, respectively). With such settings, less space will be reserved for transaction entries and more space in the table's data blocks will be available for data.
Use the FREELIST GROUPS parameter of the STORAGE clause to control the number of free list groups for the segment. Use the FREELISTS parameter to set the number of free lists per group.

To illustrate the use of the PCTUSED, PCTFREE, INITTRANS, MAXTRANS, FREELIST GROUPS, and FREELISTS parameters, let's drop and recreate the ITEMS table once more. This time, let's recreate the ITEMS table, setting all possible storage related parameters, including the storage parameters related to data blocks.

```
DROP TABLE items;

CREATE TABLE items (
 o_id INTEGER,
 id INTEGER,
 p_id INTEGER NOT NULL,
 quantity INTEGER DEFAULT 1 NOT NULL
)
TABLESPACE users
PCTUSED 40 PCTFREE 10
INITRANS 2 MAXTRANS 5
```

```
STORAGE (
 INITIAL 100K
 NEXT 200K
 MINEXTENTS 1
MAXEXTENTS UNLIMITED
PCTINCREASE 0
FREELIST GROUPS 1
FREELISTS 3
) ;
```

The latest version of the ITEMS table has a single free list group with three data block free lists. The data blocks of the new table save 10 percent of the block for row updates, and do not put blocks back on the segment's free lists until the block drops to less than 40 percent used. The data blocks of the new table also preallocate enough space for two transaction entries and allow up to five concurrent transaction entries per block.

EXERCISE 10.14: Setting Defaults for Object Storage

In most cases, storage settings for objects are optional specifications that you can use to control how Oracle stores data for each object. Oracle always has defaults for storage settings. For example, Chapter 9 explains how to set a user's default and temporary tablespaces. When the user creates a new database object and does not explicitly indicate a tablespace for the object, Oracle stores the new object in the user's default tablespace. A user's temporary tablespace is where Oracle allocates temporary work space for the user's SQL statements, whenever necessary.

When you create a new table, data cluster, or index in a tablespace, and choose not to specify extent settings for the new object, the object's segment assumes the default extent storage settings of the tablespace. Using the DEFAULT STORAGE clause of the SQL commands CREATE TABLESPACE or ALTER TABLESPACE, you can specify default extent storage settings for a tablespace.

```
DEFAULT STORAGE
[INITIAL integer [KIM]]
[NEXT integer [KIM]]
[MINEXTENTS integer]
[MAXEXTENTS {integer/UNLIMITED}]
[PCTINCREASE integer] )
```

For example, enter the following statement to alter the default extent storage settings for the PHOTOS tablespace.

```
ALTER TABLESPACE photos
DEFAULT STORAGE (
INITIAL 100K
NEXT 100K
MINEXTENTS 3
```

```
MAXEXTENTS 100
PCTINCREASE 0 ) ;
```

At this point, if you (or another user) create a table, index, data cluster, rollback segment, or temporary segment in the PHOTOS tablespace and do not specify a particular extent storage parameter for the new object, Oracle uses the default extent settings of the encompassing PHOTOS tablespace.

10.6. Rollback Segments

Transactions can complete either with a commit or a rollback. Typically, a transaction ends with a *commit*, which permanently records the transaction's changes to the database. A rollback undoes all effects of the transaction, as though the transaction never occurred. To provide for transaction rollback, Oracle must keep track of the data that a transaction changes until the transaction commits or rolls back.

Oracle uses a special type of segment called a *rollback segment* (sometimes called an *undo* segment) to record rollback data for a transaction. Should you choose to roll back a transaction, Oracle reads the necessary data from a rollback segment to rebuild the data as it existed before the transaction changed it. Figure 10-8 illustrates how Oracle uses rollback segments to undo the effects of a transaction that is rolled back.

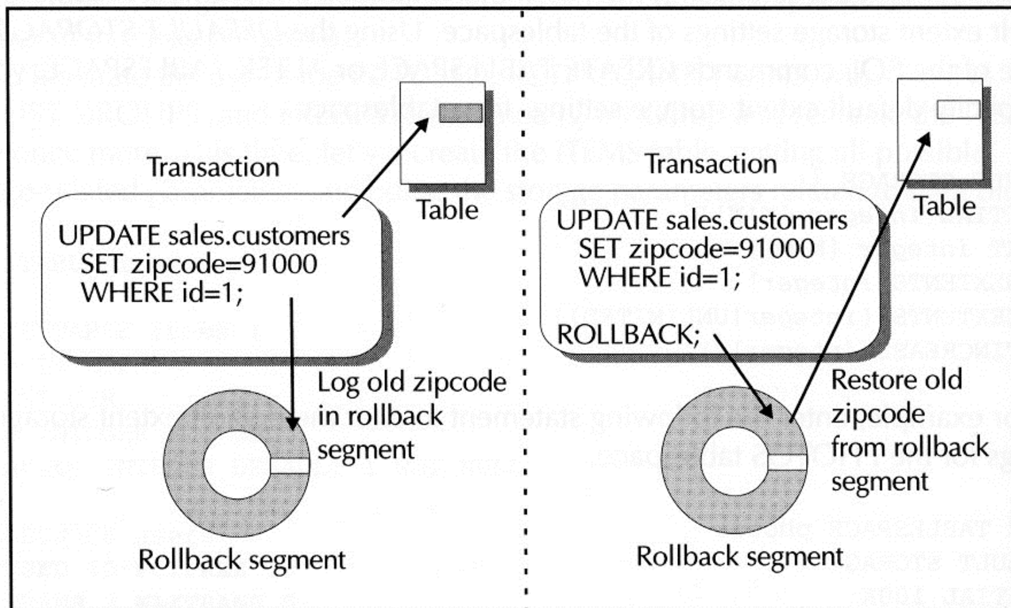


FIGURE 10-8. *Rollback segments keep track of the data that transactions change and facilitate transaction rollback*

10.6.1. How Oracle Writes to a Rollback Segment

Oracle writes information to rollback segments differently than it does with other types of segments. Figure 10 9 shows how Oracle writes information to the multiple extents in a rollback segment.

As Figure 10 9 shows, a rollback segment is a circle of extents. As transactions write to the *current extent* of a rollback segment and eventually fill it with rollback information, the transactions then wrap to the next extent of the segment to continue recording rollback information. If a transaction is so long that it wraps across all extents of a rollback segment and the current extent becomes full, Oracle must allocate an additional extent so that the system does not overwrite earlier rollback information that would be necessary to roll back the long transaction. Oracle can shrink back rollback segments to an optimal number of extents after they grow larger than their original size and once the extra extents are no longer needed. Later in this chapter you'll learn more about specific storage parameters that you can set to control extent allocation and deallocation for rollback segments.

10.6.2. The SYSTEM Rollback Segment

Every Oracle database has at least one rollback segment the SYSTEM rollback segment. When Oracle creates a new database, it automatically creates the SYSTEM rollback segment in the database's SYSTEM tablespace.

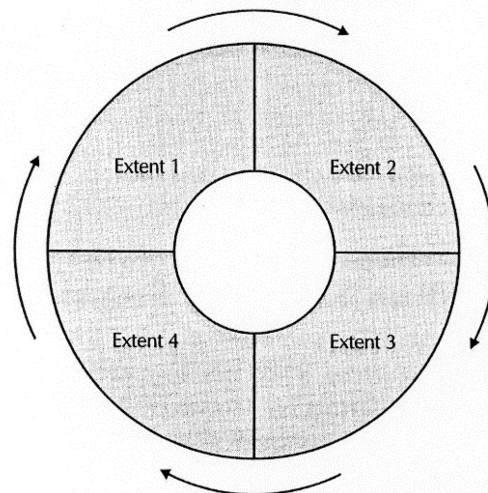


FIGURE 10-9. *Oracle writes rollback information circularly to the extents of a rollback segment*

A database's SYSTEM rollback segment alone cannot adequately support a production database system. After you create a new database, you should create additional rollback segments to support the planned transaction loads of the database.

NOTE

The default starter database for Oracle8i Enterprise Edition for Windows NT contains 24 rollback segments (named RB1 through RB24) in addition to the SYSTEM rollback segment.

10.6.3. Rollback Segments and Transactions

Typically, an Oracle database has multiple rollback segments that reside in a tablespace specifically set aside for rollback segment data. You can create any number of rollback segments for a database. Each rollback segment can be a different size and can have different storage attributes. You'll learn more about segment storage, including rollback segment storage, later in this chapter.

When you start a new transaction, Oracle automatically assigns it to an available rollback segment in the database. Oracle assigns transactions to rollback segments on a round robin basis to distribute the load across all available rollback segments.

Once Oracle assigns a transaction to a rollback segment, the segment records the changed data for all of the transaction. Multiple transactions can share a single rollback segment, but a transaction never uses more than one rollback segment.

NOTE

See Exercise 10.18, later in this chapter, which explains how a transaction can choose a specific rollback segment when necessary.

10.6.4. Online and Offline Rollback Segments

Just like tablespaces and data files, a rollback segment is available if it is online and unavailable if it is offline. Typically, a rollback segment is online so that transactions can use it to record rollback information. However, some administrative operations require that you first take rollback segments offline. For example, to take offline a tablespace that contains rollback segments, you must first take all rollback segments in the tablespace offline. After you bring the tablespace back online, you can then bring the rollback segments back online, as well.

10.6.5. Public and Private Rollback Segments

A rollback segment in a database can be either public or private. A *public rollback segment* is a rollback segment that Oracle automatically acquires access to and brings online for normal database operations. On the other hand, an Oracle instance acquires a *private rollback segment* only if its server parameter file explicitly lists the name of the private rollback segment. Private rollback segments are useful when you are using Oracle's Parallel Server option and want the various servers for the same database to acquire a mutually exclusive set of the database's rollback segments. Unless you use Oracle with the Parallel Server option, it's much easier to create and use public rollback segments.

10.6.6. Deferred Rollback Segments

When a disk problem forces Oracle to take one or more data files offline, it's typical to take associated tablespaces offline so that users do not notice file access errors when using applications. However, when you take a tablespace offline and Oracle cannot access all of the tablespace's data files, Oracle might create a deferred rollback segment in the SYSTEM tablespace. A deferred rollback segment contains transaction rollback information that Oracle could not apply to the damaged offline tablespace. Oracle keeps track of this information so that when you recover the damaged tablespace and bring it back online, Oracle can roll back the transactions that affected the tablespace and make its data consistent with the other data in the database.

10.6.7. The Other Functions of Rollback Segments

Oracle also uses rollback segments to provide read consistent sets of data for concurrent transactions in a multiuser database system and to help during database recovery. To learn more about the function of rollback segments in database recovery, see Chapter 11.

10.6.8. Creating and Managing Rollback Segments

Now that you understand the fundamental concepts related to rollback segments the next few practice exercises will teach you the basics of creating, managing, and using rollback segments in an Oracle database.

EXERCISE 10.15: Creating a Rollback Segment

To create a rollback segment, you use the SQL command *CREATE ROLLBACK SEGMENT*, which has the following syntax.

```
CREATE [PUBLIC] ROLLBACK SEGMENT segment
[TABLESPACE tablespace]
[STORAGE  (
 [INITIAL integer [K/M]]
 [NEXT integer [K/M]]
 [MINEXTENTS integer]
 [MAXEXTENTS integer]
 [OPTIMAL {integer [KIM] / NULL}] )]
```

Notice that when you create a rollback segment, you can specify tablespace and extent storage settings, as you can for tables and indexes, using the same TABLESPACE parameter and most of the same parameters in the STORAGE clause. However, because Oracle writes information to the extents of a rollback segment in a circular fashion, and because Oracle can increase and reduce the number of extents in a rollback segment as necessary, there are some special rules for specifying the storage parameters of a rollback segment:

- At all times, a rollback segment can have no fewer than two extents (that is, MINEXTENTS must be equal to or greater than 2).
- Considering that rollback segments can grow, shrink, grow, shrink, and so on, the PCTINCREASE parameter is not available for rollback segments-by default, the growth factor for a rollback segment's extents is set to 0.
- You can set an *optimal size* for a rollback segment using the *OPTIMAL* parameter of the STORAGE clause. If a rollback segment grows larger than its optimal size, Oracle eventually deallocates one or more extents from the segment to shrink it back to its optimal size.
- If you omit a tablespace specification when creating a rollback segment, Oracle creates the segment in the SYSTEM tablespace.

For example, enter the following CREATE ROLLBACK SEGMENT statement to create a public rollback segment and specify its storage settings.

```
CREATE PUBLIC ROLLBACK SEGMENT rb100
TABLESPACE rbs
STORAGE (
INITIAL 100K
NEXT 200K
MINEXTENTS 2
MAXEXTENTS UNLIMITED
OPTIMAL 1024K );
```

NOTE

When you create a public rollback segment, Oracle records the owner of the segment in the data dictionary as PUBLIC. When you create a private rollback segment, the owner of the segment is your account.

Oracle creates the new RB 100 rollback segment in the RBS tablespace with two extents: the first extent is 100K, the second extent is 200K. Furthermore, the segment can grow to an unlimited number of extents (although this limit is bound by the operating system limit of the host server), and Oracle will try to keep the size of the segment at 1024K.

EXERCISE 10.16: Controlling a Rollback Segment's Availability

To control the availability of a rollback segment, you must use the ONLINE and OFFLINE options of the SQL command ALTER ROLLBACK SEGMENT, which has the following syntax:

```
ALTER ROLLBACK SEGMENT segment
{ONLINE/OFFLINE}
```

For example, after you create a new rollback segment, as in the previous exercise, the rollback segment is left offline. To make the new RB100 rollback segment available to transactions, enter the following statement to bring the segment online.

```
ALTER ROLLBACK SEGMENT rb100
ONLINE;
```

At this point, the rollback segment is available for recording transaction rollback data.

EXERCISE10.17: Altering a Rollback Segment's Storage Settings

You can alter the extent storage settings for a rollback segment using the *STORAGE* clause of the SQL command *ALTER ROLLBACK SEGMENT*.

```
ALTER ROLLBACK SEGMENT segment
STORAGE
 [NEXT integer [KIM]]
 [MINEXTENTS integer]
 [MAXEXTENTS integer]
 [OPTIMAL {integer [KIM]INULL]] )
```

For example, enter the following statement to alter the OPTIMAL size of the RB100 rollback segment.

```
ALTER ROLLBACK SEGMENT rb100
STORAGE (OPTIMAL 512K);
```

EXERCISE 10. 18: Targeting a Specific Rollback Segment

An application can explicitly target a rollback segment at the very beginning of a new transaction with the *USING ROLLBACK SEGMENT* parameter of the SQL command *SET TRANSACTION*. For example, before starting a large batch operation, you can target a sufficiently large rollback segment in the database. By doing so, you can avoid assigning the large transaction to a small rollback segment and forcing Oracle to allocate additional space for the segment; the end result is that you reduce the overhead necessary to record rollback data for the transaction and improve the performance of the operation.

For example, enter the following statement to start a new transaction that targets the RB100 rollback segment.

```
 -- End current transaction with ROLLBACK
ROLLBACK;
SET TRANSACTION USE ROLLBACK SEGMENT rb100;
```

EXERCISE 10.19: Displaying Rollback Segment Information

To display information about rollback segments, you can query the DBA_ROLLBACK_SEGMENTS data dictionary view. For example, enter the following query to display the names and storage settings for all rollback segments in your starter database.

```
SELECT segment_name AS segment,
tablespace_name AS "TABLESPACE",
initial_extent AS "INITIAL",
next_extent AS "NEXT",
min_extents AS "MIN",
max_extents AS "MAX",
status
FROM dba_rollback_segs;
```

The result set is as follows:

```
SEGMENT     TABLESPACE   INITIAL   NEXT    MIN  MAX    STATUS
------- ---------- -------  ----   --- ---   ------
SYSTEM    SYSTEM   51200    51200  2  121   ONLINE
RB_TEMP   SYSTEM   102400   102400 10 1024  OFFLINE
RB1       RBS      102400   256000 2  121   ONLINE
RB2       RBS      102400   256000 2  121   ONLINE
RB3       RBS      102400   256000 2  121   ONLINE
RB4       RBS      102400   256000 2  121   ONLINE
RB5       RBS      102400   256000 2  121   ONLINE
RB6       RBS      102400   256000 2  121   ONLINE
RB7       RBS      102400   256000 2  121   ONLINE
RB8       RBS      102400   256000 2  121   ONLINE
RB9       RBS      102400   256000 2  121   ONLINE
RB10      RBS      102400   256000 2  121   ONLINE
RB11      RBS      102400   256000 2  121   ONLINE
RB12      RBS      102400   256000 2  121   ONLINE
RB13      RBS      102400   256000 2  121   ONLINE
RB14      RBS      102400   256000 2  121   ONLINE
RB15      RBS      102400   256000 2  121   ONLINE
RB16      RBS      102400   256000 2  121   OFFLINE
RB17      RBS      102400   256000 2  121   OFFLINE
RB18      RBS      102400   256000 2  121   OFFLINE
RB19      RBS      102400   256000 2  121   OFFLINE
RB20      RBS      102400   256000 2  121   OFFLINE
RB21      RBS      102400   256000 2  121   OFFLINE
RB22      RBS      102400   256000 2  121   OFFLINE
RB23      RBS      102400   256000 2  121   OFFLINE
RB24      RBS      102400   256000 2  121   OFFLINE
RB100     RBS      102400   204800 2 2.147E+09 ONLINE

27 rows selected.
```

Unfortunately, the DBA_ROLLBACK_SEGS view does not contain information about a rollback segment's optimal size or corresponding statistics about the number of extent allocations and deal locations-for this information, you must join data from two other data dictionary views, V$ROLLNAME and V$ROLLSTAT. For example, enter the following query:

```
SELECT name, extents, optsize, shrinks, extends, aveshrink
FROM v$rollname, v$rollstat
WHERE v$rollname.usn = v$rollstat.usn;
```

Your result set should be similar to the following:

```
NAME       EXTENT   OPTSIZE   SHRINKS   EXTENDS  AVEERSHRINK
----     ------ ------ ------- ------- ---------
SYSTEM    8              0        0        0
RB1       3     512000   0        0        0
RB2       3     512000   0        0        0
RB3       3     512000   0        0        0
RB4       3     512000   0        0        0
RB5       3     512000   0        0        0
RB6       3     512000   0        0        0
RB7       3     512000   0        0        0
RB8       3     512000   0        0        0
RB9       3     512000   0        0        0
RB10      3     512000   0        0        0
RB11      3     512000   0        0        0
RB12      3     512000   0        0        0
RB13      3     512000   0        0        0
RB14      3     512000   0        0        0
RB15      2     512000   0        0        0
RB16      2     512000   0        0        0
RB17      2     512000   0        0        0
RB18      2     512000   0        0        0
RB19      2     512000   0        0        0
RB20      2     512000   0        0        0
RB21      2     512000   0        0        0
RB22      2     512000   0        0        0
RB23      2     512000   0        0        0
RB24      2     512000   0        0        0
RB100     2     524288   0        0        0
26 Rows selected
```

The result set of the most recent query provides you with the information that you need to tune a rollback segment's optimal size. When you notice that rollback segments are extending and shrinking frequently, try, increasing the optimal size of the rollback segment to reduce dynamic extent allocations and deallocations that can slow overall server performance.

10.7. Temporary Segments and Temporary Tablespaces

SQL statements often require temporary work areas. For example, when you create an index for a large table, Oracle typically must allocate some temporary system space so that it can sort all of the index entries before building the index's segment. When processing a SQL statement that requires temporary work space, Oracle allocates small temporary segments from a tablespace in the database. When the statement completes, Oracle releases the segments back to the tablespace so that other objects can use the space. Thus the term "temporary segment."

To optimize temporary segment allocation, you should create one or more temporary tablespaces in your database. (See the section "Permanent and Temporary Tablespaces" earlier in this chapter.) You can think of a temporary tablespace as one large temporary segment that all transactions can use for temporary workspace. A temporary tablespace can more efficiently provide temporary workspace to transactions

because Oracle inserts and deletes simple table entries in the temporary tablespace's segment table, rather than physically allocating and deallocating segments on demand.

10.8. Unique Data Storage for Multimedia Data

When a table in a database includes a column that uses a LOB datatype (for example, CLOB, BLOB, or NCLOB) or a BFILE datatype, Oracle stores only a small locator inline with each row in the table. A locator is a pointer to the location of the actual LOB or BFILE data for the row. For CLOB, BLOB, and NCLOB columns, a LOB locator points to a storage location inside the database; for a BFILE column, a BFILE locator points to an external file managed by the server's operating system. Figure 10-10 illustrates the concept of locators for LOB and BFILE columns.

Notice in Figure 10-10 that a LOB column can have storage characteristics independent of those of the encompassing table. This makes it easy to address the large disk requirements typically associated with LOBS. In this example, the table stores all non LOB and non BFILE data for each row together in one tablespace, a LOB column's data in another tablespace, and a BFILE column's data in the server's file system. By doing so, you can distribute the storage of primary table data and related multimedia data to different physical locations (for example, different disk drives) to reduce disk contention and improve overall system performance.

More About LOB Locators
To provide for efficient access to LOB data, Oracle stores a CLOB or BLOB column's pointers within a corresponding B tree index. By doing so, Oracle can quickly access specifically requested chunks (pieces) of individual LOBS in a column.

EXERCISE 10.20: Creating a Table with a LOB Column

When you create a table that has a column that uses the BLOB, CLOB, or NCLOB datatype, you can store the LOB column's data in a LOB segment separate from the table's data segment by specifying the *LOB* clause of the SQL command *CREATE TABLE*. An abbreviated form of the LOB clause is as follows:

```
LOB (column [, column] ... )
 { [segment]
/ STORE AS [segment] (
 [TABLESPACE tablespace]
 [{ENABLE/DISABLE} STORAGE IN ROW]
[STORAGE ( ... ) ]
[CHUNK integer]
[PCT VERSION integer]
[INDEX
{ index
/ [index] (
[INITRANS integer] [MAXTRANS integer]
[TABLESPACE tablespace]
[STORAGE ( ... )] ) } ] }
```

The following list briefly describes the parameters of the LOB clause:

- Indicate a list of one or more CLOB, BLOB, or NCLOB columns to which the LOB clause should apply. If you indicate only one column, you can name the LOB segment that Oracle creates for storing the column; however, if you list more than one column, you cannot name the LOB segment.
- Use the *TABLESPACE* parameter of the LOB clause to specify the tablespace that will hold the LOB segment.
- Use the *ENABLE STORAGE IN ROW* option to have Oracle store a LOB value less than 4,000 bytes with the other data in the same row; when a LOB value is greater than 4,000 bytes, Oracle stores the LOB value in the LOB segment. Use the *DISABLE STORAGE IN ROW* option to always store LOB values in the LOB segment, regardless of each LOB value's length.
- Use a STORAGE clause to specify several storage parameters that determine the size and number of extents that comprise the segment for a nonclustered table's LOB segment. See Exercise 10.11, earlier in this chapter, for more information about the STORAGE clause and its parameters.
- Use the *CHUNK* parameter to specify the storage allocation unit for a LOB segment. Specify an integer for the CHUNK parameter. The storage allocation unit for a LOB segment is the result of the CHUNK parameter setting multiplied by the database's data block size. For example, when a LOB segment's CHUNK parameter is 10 and the database's data block size is 2K, the storage allocation unit for the LOB segment is 20K. The maximum value of a LOB segment's storage allocation unit is 32K. Additionally, ensure that the extent sizes for a LOB segment (INITIAL and NEXT) are greater than or equal to the storage allocation unit for the segment.
- When a transaction modifies a LOB or part of a LOB, Oracle creates a new version of the LOB's data blocks and leaves the older version of the LOB intact, to support consistent reads of prior versions of the LOB. Use the *PCTVERSION* parameter to control the overall percentage of used LOB data blocks in a LOB segment that are available for versioning of old LOB data. PCTVERSION is a threshold of LOB segment storage space that must be reached before Oracle overwrites older versions of LOB data with newer versions. The default value is 10. When updates to LOBS are infrequent, set PCTVERSION to 5 or lower to minimize the amount of disk space required to store LOBS.
- Use the INDEX clause to specify the extent storage parameters for the index that Oracle automatically creates for a LOB segment. If you indicate only one column for the LOB clause, you can name the index that Oracle creates for the LOB segment; however, if you list more than one column, you cannot name the index.

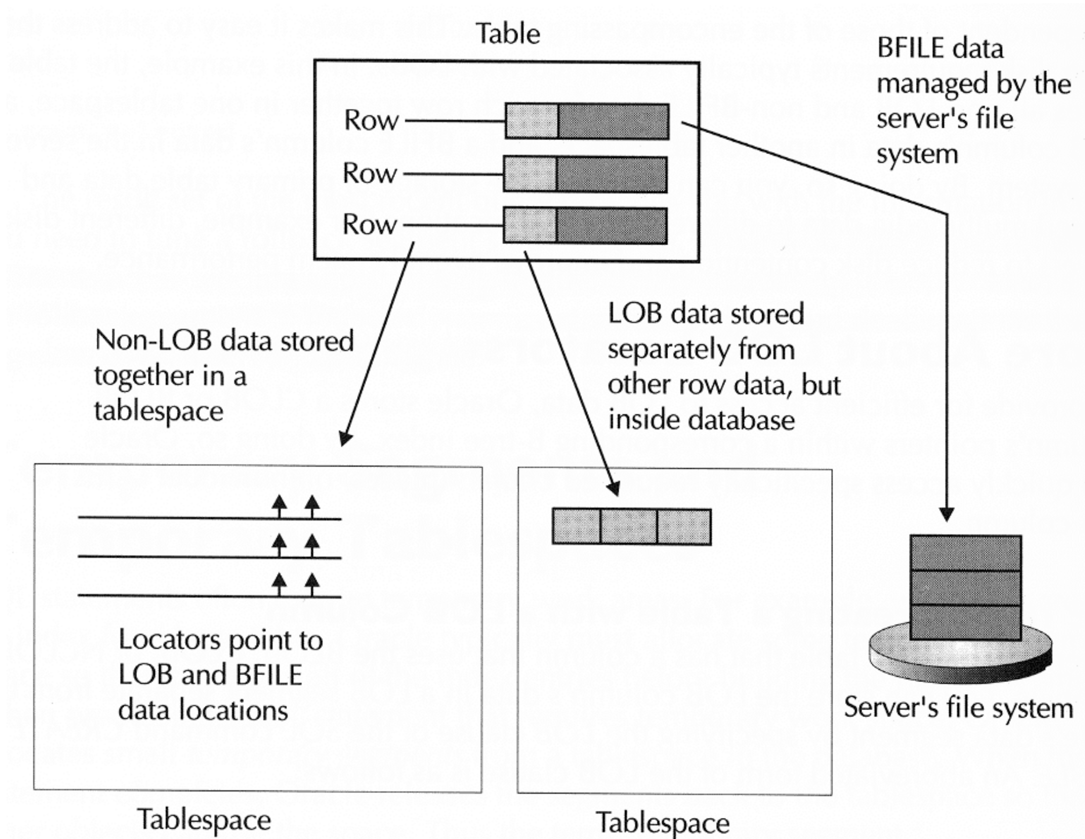


FIGURE 10-10. *Distributing the storage of primary table data and related multimedia data to different physical locations to reduce disk contention*

To demonstrate the use of the LOB clause in a CREATE TABLE statement, enter the following statement to create a CUSTOMERS table that can store a photograph for each customer record.

```
CREATE TABLE customers (
  id INTEGER,
  lastname VARCHAR2(100) CONSTRAINT lastname NOT NULL,
  firstname VARCHAR2(50) CONSTRAINT firstname NOT NULL,
  companyname VARCHAR2(100),
  street VARCHAR2(100),
  city VARCHAR2(100),
  state VARCHAR2(50),
  zipcode VARCHAR2(50),
  phone VARCHAR2(30),
  fax VARCHAR2(30),
  email VARCHAR2(100),
  s_id INTEGER CONSTRAINT salesrep NOT NULL,
  photo BLOB
)
--storage parameters for data segment
PCTFREE 10 PCTUSED 70
```

```
INITRANS 2 MAXTRANS 5
TABLESPACE users
STORAGE (
    INITIAL 100K NEXT 100K
    MINEXTENTS 1 MAXEXTENTS 10
    PCTINCREASE 0
    FREELISTS 3
)
--storage parameters for LOB segment
LOB (photo) STORE AS cust_photo (
TABLESPACE photos
ENABLE STORAGE IN ROW
STORAGE (
    INITIAL 100K NEXT 100K
    MINEXTENTS 1 MAXEXTENTS 10
    PCTINCREASE 0
    FREELISTS 3
)
 CHUNK 5
 PCTVERSION 5
 --storage parameters for index of LOB column
 INDEX photos (
  TABLESPACE indx
  STORAGE (INITIAL 50K NEXT 50K)
  )
)
;
```

Notice the following points about the CREATE TABLE statement in this exercise:

- The statement specifies storage parameters for the table's data segment using the familiar PCTFREE, PCTUSED, TABLESPACE, INITRANS, MAXTRANS, and STORAGE clause parameters.
- The statement specifies storage settings for the table's LOB segment using the LOB clause.
- The statement specifies storage settings for the index of the LOB segment.

10.9. Partitioning Large Tables and Indexes

Large tables (and indexes) can create or magnify several problems in production database systems because of their size and storage characteristics. For example, consider the following scenarios:

- A table becomes so large that associated management operations take longer to complete than the time window that is available.
- A query requires Oracle to complete a full table scan of a very large table. Application and system performance suffers while Oracle reads the numerous data blocks for the corresponding table.

- A mission critical application depends primarily on a single large table. The table becomes unavailable when just a single data block in the table is inaccessible due to a disk failure.
- An administrator must recover the entire tablespace that contains the table before the table and corresponding mission critical application can be brought back online.

To help reduce the types of problems that large tables and indexes can create, along with other problems, Oracle8i supports partitioned tables and partitioned indexes.

10.9.1. Partitioned Tables

Oracle lets you divide the storage of a table into smaller units of disk storage called *partitions*, as Figure 10 11 shows.

Oracle's built in partitioning features supports the *horizontal partitioning* of a table that is, each partition of a table contains the same logical attributes, including the same set of columns with the same datatypes and the same integrity constraints. However, each table partition is an individual data segment that can have different physical attributes. For example, Oracle can store each partition of a table in separate tablespaces, and the data segment of each partition can have different extent and data block storage settings.

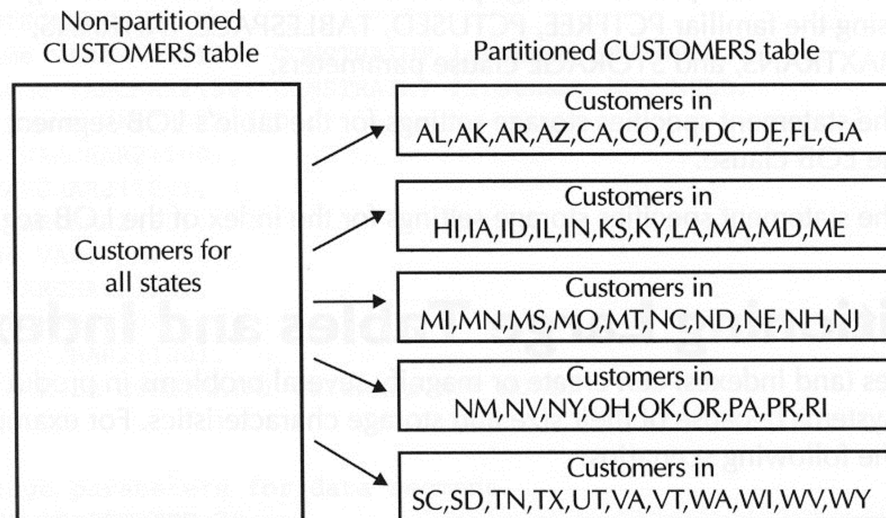


FIGURE 10-11.*Oracle lets you horizontally partition a table by storing the table's rows in two or more segments*

10.9.1.1. A Table's Partition Key

Every partitioned table has a partition key. A table's partition key is a column or ordered set of columns that characterize the horizontal partitioning of table rows. When you create a new table, you specify the table's partition key and configure

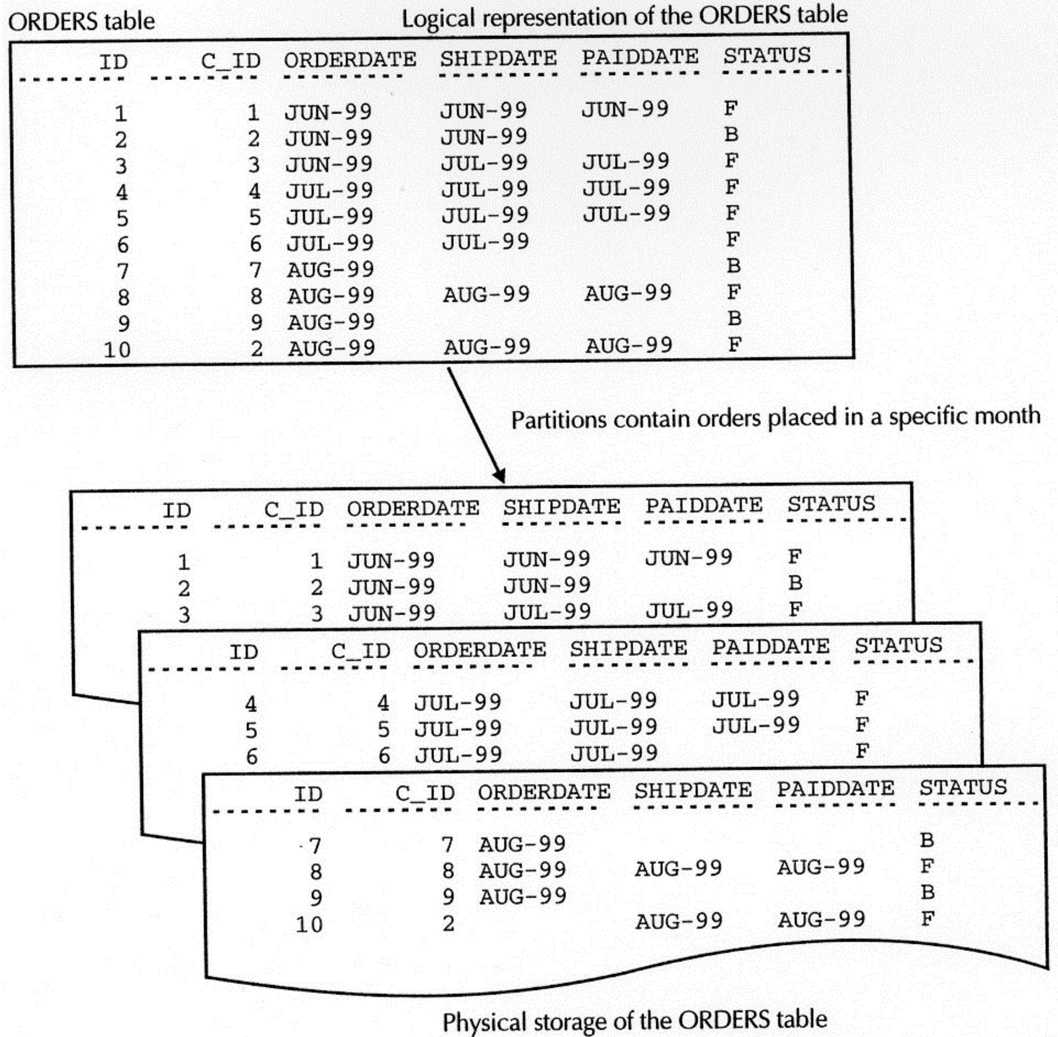
the table to place rows into available partitions using range partitioning, hash partitioning, or a composite form of range and hash partitioning. The next few sections explain each type of option for table partitioning.

10.9.1.2. Range Partitioning

One way to horizontally partition the data of a table is range partitioning. Each partition in the table stores rows according to the partition key values of a row. For example, Figure 10 12 demonstrates how you might partition the rows in the ORDERS table according to the ORDERDATE of each row.

> **NOTE**
> *To prevent the overhead associated with migrating rows among table partitions, applications should never update data in a table's partition key.*

When you create a range partitioned table, you configure a noninclusive upper bound, or partition bound, for each partition in the table. Each partition, except for the first, has an implicit lower value, which is the upper bound of the previous partition. Therefore, it's important to declare table partitions with ranges that ascend in value. For example, in Figure 10 12, each partition of the range partitioned ORDERS table stores a month's worth of sales order records.



ID	C_ID	ORDERDATE	SHIPDATE	PAIDDATE	STATUS
1	1	JUN-99	JUN-99	JUN-99	F
2	2	JUN-99	JUN-99		B
3	3	JUN-99	JUL-99	JUL-99	F
4	4	JUL-99	JUL-99	JUL-99	F
5	5	JUL-99	JUL-99	JUL-99	F
6	6	JUL-99	JUL-99		F
7	7	AUG-99			B
8	8	AUG-99	AUG-99	AUG-99	F
9	9	AUG-99			B
10	2	AUG-99	AUG-99	AUG-99	F

ID	C_ID	ORDERDATE	SHIPDATE	PAIDDATE	STATUS
1	1	JUN-99	JUN-99	JUN-99	F
2	2	JUN-99	JUN-99		B
3	3	JUN-99	JUL-99	JUL-99	F

ID	C_ID	ORDERDATE	SHIPDATE	PAIDDATE	STATUS
4	4	JUL-99	JUL-99	JUL-99	F
5	5	JUL-99	JUL-99	JUL-99	F
6	6	JUL-99	JUL-99		F

ID	C_ID	ORDERDATE	SHIPDATE	PAIDDATE	STATUS
7	7	AUG-99			B
8	8	AUG-99	AUG-99	AUG-99	F
9	9	AUG-99			B
10	2		AUG-99	AUG-99	F

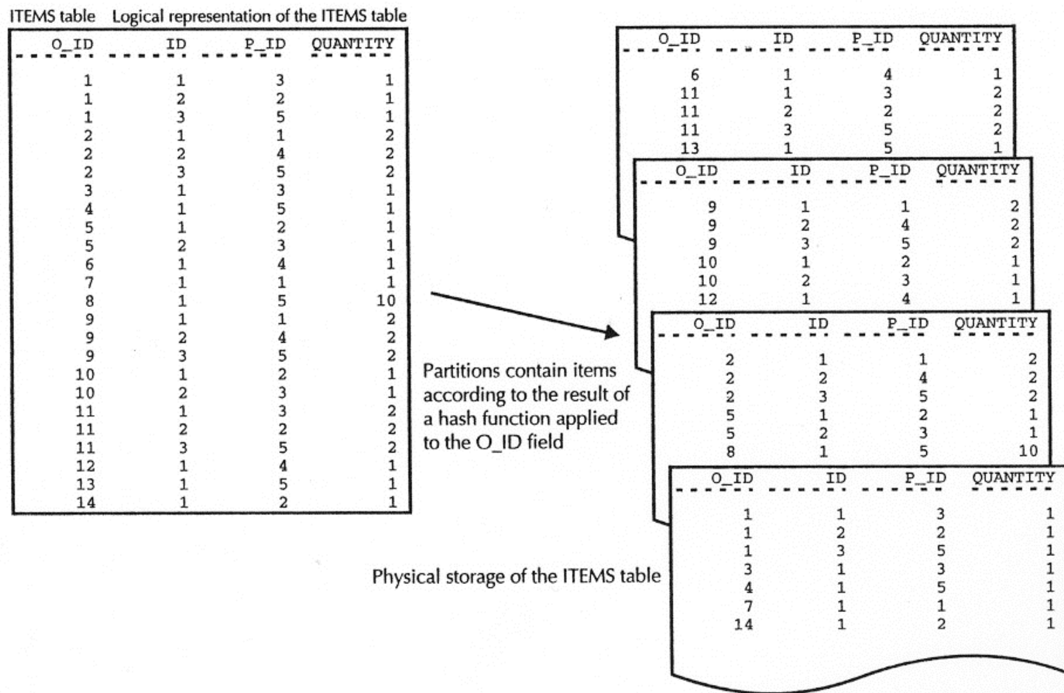
FIGURE 10-12. *With range partitioning, Oracle p/aces a row in a partition according to the partition key of the row*

Range partitioning is very useful to manage a large historical table in a data warehouse or decision support system (DSS) database that must keep a revolving set of records that correspond to a particular time interval. For example, you might want to keep the most recent year's sales order records in the ORDERS table of a schema in a DSS. In this case, you could add a new partition every month to the ORDERS table in the DSS to hold the most recent month's records from the production OLTP system. At the same time, you could drop the oldest partition in the ORDERS table to delete the oldest records in the table. Exercise 10.23, later in this chapter, uses this example to teach you how to use range partitioning to maintain a historical table.

10.9.1.3. Hash Partitioning

Another way to partition the data of a table is *hash partitioning*. To place a row in a hash partitioned table, Oracle8i first applies a hash function to the partition key of the row and then uses the resulting hash value to determine which partition will hold the row. For example, Figure 10 13 shows a version of the ITEMS table that uses hash partitioning.

All things considered, hash partitioning is a good choice when your primary goal is to improve the response time of SQL statements that target the table. That's because hash partitioning typically distributes the rows of the table uniformly among all available partitions, which you can create on different physical disk drives to reduce contention for disk access during full table scans. Hash partitioning is also useful when the data of a large table is not suitable for range partitioning - for example, when the data is not evenly spread among ranges of partition key values, or when historical management of table data is not called for.



O_ID	ID	P_ID	QUANTITY
1	1	3	1
1	2	2	1
1	3	5	1
2	1	1	2
2	2	4	2
2	3	5	2
3	1	3	1
4	1	5	1
5	1	2	1
5	2	3	1
6	1	4	1
7	1	1	1
8	1	5	10
9	1	1	2
9	2	4	2
9	3	5	2
10	1	2	1
10	2	3	1
11	1	3	2
11	2	2	2
11	3	5	2
12	1	4	1
13	1	5	1
14	1	2	1

O_ID	ID	P_ID	QUANTITY
6	1	4	1
11	1	3	2
11	2	2	2
11	3	5	2
13	1	5	1

O_ID	ID	P_ID	QUANTITY
9	1	1	2
9	2	4	2
9	3	5	2
10	1	2	1
10	2	3	1
12	1	4	1

O_ID	ID	P_ID	QUANTITY
2	1	1	2
2	2	4	2
2	3	5	2
5	1	2	1
5	2	3	1
8	1	5	10

O_ID	ID	P_ID	QUANTITY
1	1	3	1
1	2	2	1
1	3	5	1
3	1	3	1
4	1	5	1
7	1	1	1
14	1	2	1

FIGURE 10-13. *With hash partitioning, Oracle places a row in a partition according to the result of applying a hash function to the row's partition key*

10.9.1.4. Composite Partitioning

When you require the management benefits of range partitioning but would also like to receive the performance benefits of hash partitioning, you can combine the two forms of partitioning in the same table. When you use *composite partitioning* with a table, Oracle first partitions the rows of the table into ranges, and then stripes the rows within a partition into subpartitions using hash partitioning.

10.10. Creating and Managing Partitioned Tables

Now that you have a basic understanding of range, hash, and composite partitioning for tables, the next few practice exercises teach you how to get started by creating range- and hash-partitioned tables.

EXERCISE 10.21: Creating a Range Partitioned Table

This exercise and the following two teach you how to manage a fictitious ORDERS historical table in a DSS database with range partitioning. To create a range-partitioned table, you must use the *PARTITION BY RANGE* clause of the SQL and *CREATE TABLE*. An abbreviated form of the syntax of the PARTITION BY RANGE clause is as follows:

```
CREATE TABLE table
(... columns and constraints ... )
PARTITION BY RANGE (column [, column] ... )
PARTITION [partition]
    VALUES LESS THAN ({values list/MAXVALUE})
    [PCTFREE integer] [PCTUSED integer]
    [INITRANS integer] [MAXTRANS integer]
    [TABLESPACE tablespace]
    [STORAGE ( ... )]
[, ...] )
```

THE PARTITION BY RANGE clause lets you specify one or more PARTITION clauses to specify the characteristics of each table partition. For each successive partition , you indicate the partition bound using the VALUES LESS THAN parameter - each value in the partition bound must be a literal or a TO_DATE( ) or RPAD( ) function with a constant argument. You can also indicate storage settings for each partition using familiar storage settings such as the PCTFREE and TABLESPACE Iris and a STORAGE clause.

NOTE
You can also indicate the partition bound of the last partition in a range partitioned table using the MAXVALUE *keyword rather than a values list. In this case, the last partition in the table stores all rows with a partition key value greater than the upper bound of the next to last partition. Additionally, when a table's partition key accepts nulls, Oracle sorts rows with null partition keys greater than all other values except MAXVALUE.*

Enter the following CREATE TABLE statement to create a range partitioned version of the ORDERS table.

```
CREATE TABLE orders (
id INTEGER,
c_id INTEGER NOT NULL,
orderdate DATE DEFAULT SYSDATE NOT NULL,
shipdate DATE,
```

```
paiddate DATE,
status CHAR(1) DEFAULT 'F' )
PARTITION BY RANGE (orderdate)
( PARTITION jun1999
    VALUES LESS THAN (TO_DATE('07 01 1999', 'MM DD  YYYY'))
    TABLESPACE users,
  PARTITION jul1999
     VALUES LESS THAN (TO_DATE('08 01 1999', 'MM DD YYYY'))
TABLESPACE users,
PARTITION aug1999
VALUES LESS THAN (TO_DATE('09 01 1999', 'MM DD YYYY')) TABLESPACE
users);
```

This example statement creates the ORDERS table with three partitions to hold sales order records for the months of June, July, and August of 1999.

EXERCISE 10.22: Understanding How Oracle Places Rows into a Range Partitioned Table

To see how Oracle places the rows of a table into the various partitions of the range partitioned ORDERS table, let's insert some rows with ORDERDATE values that correspond to the different partition ranges in the table. Enter the following statements:

```
INSERT INTO orders
 VALUES (1,1,'18-JUN-99','18 JUN-99','30-JUN,99', 'F');
INSERT INTO orders
 VALUES (2,2,'23-JUL-99',NULL,NULL,'B');
INSERT INTO orders
 VALUES (3,3,'08-AUG-99','08-AUG-99','08 AUG-99','F');
```

The PARTITION parameter of the SQL command SELECT lets you target rows in a specific partition. For example, enter the following query to see only the rows in the JUL1999 partition of the ORDERS table.

```
SELECT *
FROM orders
PARTITION (jul1999);
```

The result set is as follows:

```
ID  C_ID ORDERDATE SHIPDATE PAIDDATE  STATUS
--  ---- --------- -------- --------  -------
2     2  23-JUL-99                      B
```

Notice that Oracle correctly placed the record with an ORDERDATE of 23-JUL-1999 into the JUL1999 partition. If you wish, you can modify the preceding query to display the records in the other partitions of the ORDERS table.

Now try to insert another sales order into the ORDERS table with the following INSERT statement.

```
INSERT INTO orders
 VALUES (4,4,'04-SEP-99','04-SEP-99','04-SEP-99','F');
```

In this example, Oracle returns an error, because the ORDERDATE of the new record is higher in value that the partition bound of the last partition in the table.

```
ORA-14400: inserted partition key is beyond highest legal
partition key
```

You cannot store the new record until you create a new partition for the table with a higher partition bound.

EXERCISE 10.23: Managing a Range Partitioned Historical Table

This final exercise concerning range partitioning teaches you how to roll in and roll out historical data in the range partitioned ORDERS table of a fictitious DSS database. In this example, you will see how to add a new partition to the ORDERS table to accept records for the month of September, 1999, and then drop the partition that contains the oldest sales records.

First, let's add a new partition to the ORDERS table to accept records for the month of September, 1999. To add a new partition to a range partitioned table, you use the *ADD PARTITION* clause of the SQL command *ALTER TABLE*.

```
ADD PARTITION [partition]
VALUES LESS THAN ({values list/MAXVALUE})
[PCTFREE integer] [PCTUSED integer]
[INITRANS integer] [MAXTRANS integer]
[TABLESPACE tablespace]
[STORAGE ( ... )]
```

For example, enter the following statement to add the new SEP1999 partition to the ORDERS table.

```
ALTER TABLE orders
ADD PARTITION sep1999
    VALUES LESS THAN (TO_DATE('10-01-1999', 'MM-DD-YYYY'))
     TABLESPACE users;
```

Now, retry the last INSERT statement of the previous exercise, and the ORDERS table will accept the new row.

```
INSERT INTO orders
VALUES (4,4,'04-SEP-99','04-SEP-99','04-SEP-99','F');
```

Next, let's see how to drop the partition that contains the oldest sales records of the ORDERS table. To drop a partition from a table, you use the *DROP PARTITION* parameter of the SQL command *ALTER TABLE*. For example, enter

the following statement to drop the JUN1999 partition and all records in the partition from the ORDERS table.

```
ALTER TABLE orders
DROP PARTITION jun1999;
```

The examples in the previous three exercises explain just a few of the many different management operations available for range partitioned tables. See your Oracle documentation for more information about other management operations related to range partitioned tables, including how to convert a nonpartitioned table to a partitioned table (and vice versa), how to split or merge partitions in the middle of a range partitioned table, and how to delete all rows in an individual partition of a table.

EXERCISE 10.24: Creating a Hash Partitioned Table

In this exercise, let's create yet another version of the ITEMS table-this time, a hash partitioned version. To create a hash partitioned table, you use the *PARTITION BY HASH* clause of the SQL command *CREATE TABLE*.

```
PARTITION BY HASH (column [, column] ... )
 { PARTITIONS integer STORE IN (tablespace [, tablespace] ... )
 / ( PARTITION [partition]
  [PCTFREE integer] [PCTUSED integer]
  [INITRANS integer] [MAXTRANS integer]
  [TABLESPACE tablespace]
  [STORAGE ( ... )]
    [,...]) )
```

Notice that the PARTITION BY HASH clause lets you specify the partitions of a hash-partitioned table using two different techniques:

- Use the PARTITIONS clause to specify a number of partitions and a list of a corresponding number of tablespaces for the partitions. In this case, all partitions have the same extent storage parameters.
- Specify a list of partitions using PARTITION clauses when you want to specify names and individual extent storage settings for each partition in the table.

TIP

To obtain the best distribution of rows among table partitions, create a hash partitioned table with an even number of partitions (for example, 2, 4, 6, ...).

Enter the following series of statements to drop the existing ITEMS table, and then create a new hash partitioned version of the table.

```
DROP TABLE items;
```

```
CREATE TABLE items (
 o_id INTEGER,
 id INTEGER,
 p_id INTEGER NOT NULL,
 quantity INTEGER DEFAULT 1 NOT NULL )
PARTITION BY HASH (o_id)
( PARTITION iteml TABLESPACE users,
PARTITION item2 TABLESPACE users,
PARTITION item3 TABLESPACE users,
PARTITION item4 TABLESPACE users );
```

EXERCISE 10.25: Confirming the Distribution of Rows in a Hash-Partitioned Table

In this exercise, let's insert some rows into the hash partitioned ITEMS table and then confirm the distribution of rows. Enter the following INSERT statements to insert line items into the ITEMS table that correspond to the four sales orders in the ORDERS table.

```
INSERT INTO items VALUES (1,1,3,1);
INSERT INTO items VALUES (1,2,2,1);
INSERT INTO items VALUES (1,3,5,1);
INSERT INTO items VALUES (2,1,1,2);
INSERT INTO items VALUES (2,2,4,2);
INSERT INTO items VALUES (2,3,5,2);
INSERT INTO items VALUES (3,1,3,1);
INSERT INTO items VALUES (4,1,5,1);
```

Now enter the following queries to reveal how Oracle8i's hash function distributes the rows among the available partitions.

```
SELECT * FROM items PARTITION (iteml);
SELECT * FROM items PARTITION (item2);
SELECT * FROM items PARTITION (item3);
SELECT * FROM items PARTITION (item4);
```

When you examine the result sets for these queries, you should notice that some partitions do not have any rows (yet), while others have rows for various line items according to their O_ID value. If you insert more and more rows into the ITEMS table, the number of rows in each partition will eventually even out.

10.10.1. Partitioned Indexes

Chapter 7 of this book introduced the advantage of creating one or more indexes for the columns in a table-using indexes, you can reduce the amount of disk I/O necessary to locate a specific row or set of rows in the corresponding table, which translates to faster response times for SQL statements that target the table. If you create an index for a large table, you should also know that Oracle8i supports the horizontal partitioning of an index using range, hash, or composite partitioning. Just as with tables, each partition of an index has the same logical attributes (index columns), but can have different physical characteristics (tablespace placement and

storage settings). An index's partition key and the type of partitioning option that you choose for the index determines which partition Oracle stores index entries within. An index's partition key must include one or more of the columns that define the index.

10.10.2. Partitioned Index Options

Should you always create partitioned indexes for partitioned tables? What columns should a partitioned index incorporate? How should you structure the partitions in an index? These are good questions to answer before deciding how to index the partitioned tables in your database.

First things first. It doesn't always make sense to partition an index of a partitioned table. In general, you should consider a partitioned index when the index itself is large enough to justify partitioning, or when you frequently manage partitions of a table and you want the index to be available for SQL statements that happen during administrative tasks.

Choosing the key columns for a partitioned index is no different from choosing the key columns for a nonpartitioned index-index the columns in a table that SQL statements use within a WHERE clause's search criteria. You'll learn more about indexes and general guidelines for indexing in Chapter 12 of this book.

Once you decide to create a partitioned index for a table, you must then decide how to structure the index's partitions. Considering the plethora of options that Oracle8i offers for indexes and partitioning, determining the best option for an index can get confusing. The following sections introduce you to the fundamental options available and provide you with some basic considerations for each option.

10.10.3. GLOBAL INDEXES

A nonpartitioned index of a partitioned table is called a global index. As the name implies, a partition in a global index contains index entries that correspond to rows that originate from two or more partitions of a partitioned table. Figure 10-14 illustrates a nonpartitioned global index of a partitioned table.

You can also create a global partitioned index. *A global partitioned index* has partitions that do not match the partitions in the associated table. For example, to consolidate the number of partitions in an index of a range partitioned table, the index entries in each partition of the global index might correspond to the rows in three successive partitions in the associated table.

In general, global indexes are useful for improving the response times of SQL statements that target rows in many partitions of a partitioned table. However, the administration and availability of a global index can be affected when you modify the associated table. For example, if you load new data into a single partition of a

.table that has a global index, you might need to rebuild the entire global index to synchronize it with the new table data.

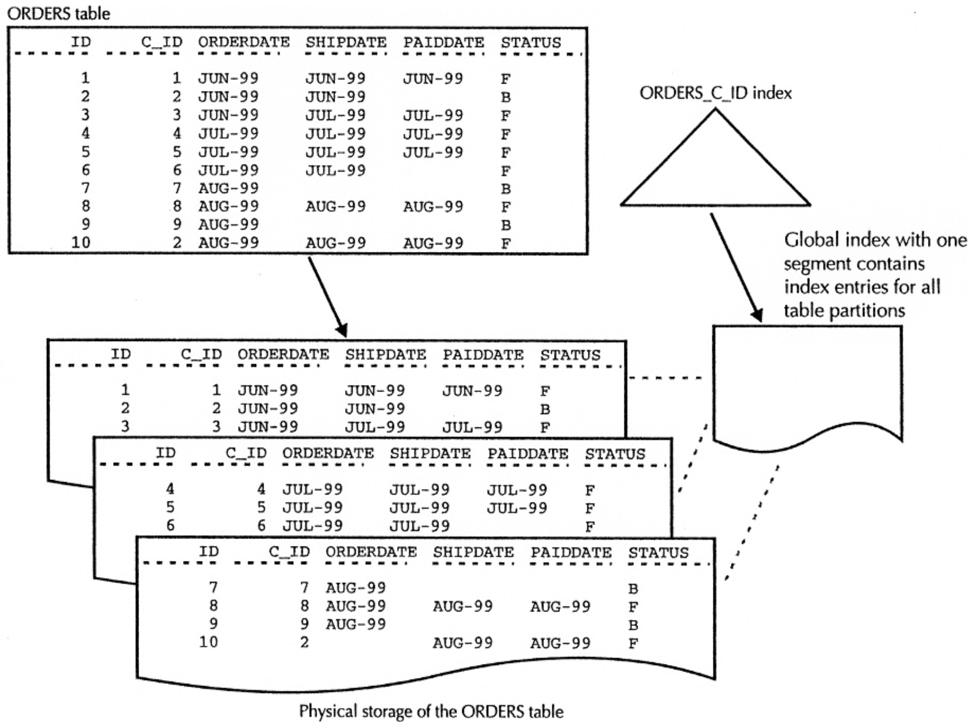


FIGURE 10-14.*A nonpartitioned index of a partitioned table*

10.10.4. EQUI PARTITIONED OBJECTS AND LOCAL INDEXES

Two or more database objects are equi partitioned if they have identical logical partitioning attributes. For example, a range partitioned table and an associated rangepartitioned index would be equi partitioned if they have the same number of partitions, the same data ranges for each partition, and the same columns defining their respective partition keys.

When you equi partition an index with its table, all keys in a particular index partition refer only to the rows in the corresponding table partition thus, a partitioned index that is equi partitioned with its table is a *local partitioned index*. Oracle can use the local index partitions to generate excellent query plans. Additionally, an administrator affects the availability of only one index partition when performing a maintenance operation on a table partition. Figure 10 15 illustrates an equi partitioned table and a local index.

Equi partitioned tables and local indexes can provide benefits for administration and data availability because an administrative operation that affects a specific table partition will affect the corresponding index partition only. For example, if you perform a bulk data load into the range partitioned historical table in a DSS database, and the load updates data in just one table partition, you only have to rebuild the corresponding index partition. All other partitions of the index remain

available and valid, because the data load does not affect the rows that correspond to the index entries in other partitions of the index.

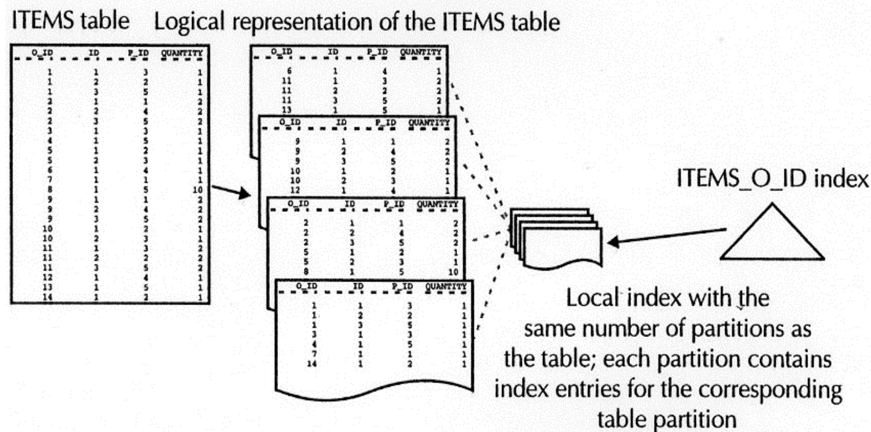


FIGURE 10-15. *Each partition of a local index contains index entries that correspond to rows in the same partition of its table*

10.11. Creating and Managing Partitioned Indexes

The following two practice exercises teach you how to create normal (B tree) indexes for the partitioned ORDERS and ITEMS tables created in Exercises 10.21 and 10.24.

EXERCISE 10.26: Creating Global Indexes

To create a global index for a table, you use various forms of the SQL command *CREATE INDEX*. For example, to create a nonpartitioned global index for a table, you create the index without any special syntax. Enter the following statement to create a nonpartitioned global index for the C_ID column of the range partitioned ORDERS table.

```
CREATE INDEX orders_c_id ON orders (c_id);
```

To create a range partitioned global index for a table, you use the GLOBAL keyword followed by the *PARTITION BY RANGE* clause of the SQL command *CREATE INDEX*. The syntax of the PARTITION BY RANGE clause in the CREATE INDEX command is identical to the same clause in the CREATE TABLE command, listed in Exercise 10.21. For example, enter the following statement to create a partitioned global index of the ORDERDATE column in the ORDERS table.

```
CREATE INDEX orders_orderdate ON orders (orderdate)
 GLOBAL
 PARTITION BY RANGE (orderdate)
 ( PARTITION q11999
  VALUES LESS THAN (TO_DATE('04-01-1999', 'MM-DD-YYYY'))
```

```
  TABLESPACE users,
  PARTITION q21999
  VALUES LESS THAN (TO_DATE('07-01-1999', 'MM-DD-YYYY'))
  TABLESPACE users,
  PARTITION q31999
  VALUES LESS THAN (TO_DATE('10-01-1999', 'MM DD YYYY'))
  TABLESPACE users,
  PARTITION q41999
  VALUES LESS THAN (MAXVALUE)
TABLESPACE users );
```

NOTE
The last partition in a range-partitioned global index to must use the MAXVALUE keyword.

In the preceding example, notice that each partition in the range partitioned global index of the ORDERS table contains index entries that correspond to sales records for each fiscal quarter, rather than to each month, as with the associated range partitioned ORDERS table. This query would be appropriate if queries frequently analyze rows in the ORDERS table by quarter.

EXERCISE 10.27: Creating Local, Equi Partitioned Indexes

Creating a local, equi partitioned index of a partitioned table is very simple all you need to do is include the LOCAL keyword when creating the index with the SQL command CREATE INDEX. For example, enter the following statement to create a local, equi partitioned index for the hash partitioned ITEMS table.

```
CREATE INDEX items_o_id ON items (o_id) LOCAL;
```

In this example, Oracle creates the ITEMS_0_ID index with four partitions (ITEM1, ITEM2, ITEM3, ITEM4) in the USERS tablespace-refer to the CREATE TABLE statement in Exercise 10.24 if you do not remember the specific partitions in the hash partitioned ITEMS table.

Chapter Summary

This chapter has explained the logical and physical database storage structures, including databases, tablespaces, data files, control files, segments, extents, and data blocks.

- Tablespaces are logical storage divisions within an Oracle database. You create and manage tablespaces using the SQL commands CREATE TABLESPACE and ALTER TABLESPACE.
- Each tablespace has one or more data files to physically store its data. You can specify the names and properties of a tablespace's data files when you create the tablespace with the CREATE TABLESPACE command. You can subsequently add data files to a tablespace, or change the storage characteristics of a data file using the ALTER DATABASE and ALTER TABLESPACE commands.
- A segment is the collection of data blocks for a data storage object, such as a table, data cluster, index, or rollback segment. An extent is a set of contiguous data blocks allocated to an object's segment. A data block is the unit of physical disk access for an Oracle database.
- When you create a database object, such as a table, you can explicitly specify storage parameters that determine how Oracle allocates extents for the object's segment and storage parameters that determine how Oracle uses the space within the data blocks of the object's segment.
- Oracle uses a rollback segment to record rollback data for a transaction. If you choose to roll back a transaction, Oracle reads the necessary data from a rollback segment to "undo" the effects of the transaction. You create and manage rollback segments with the SQL commands CREATE ROLLBACK SEGMENT and ALTER ROLLBACK SEGMENT.
- A database's control file keeps track of internal system information about the physical structure of the database.

Additionally, you learned how to partition the data of large tables and indexes to improve the performance and manageability associated with these objects.