Using the New Tablespace Management Options in Oracle8i

Michael R. Ault
DBMentors International, Inc.

Introduction

Many new tablespace management options have been added in Oracle8i. Locally managed tablespaces, transportable tablespaces and special options on read-only tablespaces are the major additions. This article will show by example these new tablespace options and how they are used.

Locally Managed Tablespaces

As their name implies, locally managed tablespaces are not managed by the data dictionary but are instead managed at the local tablespace level. This local management is accomplished through the use of space bit-maps that track extent usage. I also suggest that it is a good practice for all extents in a locally managed tablespace to be of a uniform size.

This leads to two basic types of tablespace, the dictionary managed tablespace which is synonymous with what we all consider to be a normal tablespace, and, a locally managed tablespace.

A tablespace that manages its own extents maintains a bitmap in each datafile to keep track of the free or used status of blocks in that datafile.

Each bit in the bitmap corresponds to a block or a group of blocks. When an extent is allocated or freed for reuse, Oracle changes the bitmap values to show the new status of the blocks. These changes do not generate rollback information because they do not update tables in the data dictionary (except for special cases such as tablespace quota information).

Locally-managed tablespaces have the following advantages over dictionary-managed tablespaces:

- Local management of extents avoids recursive space management operations, which can occur in dictionary-managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in a rollback segment or data dictionary table.
- Local management of extents automatically tracks adjacent free space, eliminating the need to coalesce free extents.

The sizes of extents that are managed locally can be determined automatically by the system. Alternatively, all extents can have the same size in a locally-managed tablespace. See "Extents Managed Locally" for more information.

The LOCAL option of the EXTENT MANAGEMENT clause specifies this method of space management in various CREATE commands:

For the SYSTEM tablespace, you can specify EXTENT MANGEMENT LOCAL in the CREATE DATABASE command. If the SYSTEM tablespace is locally managed, other tablespaces in the database can be dictionary-managed but you must create all rollback segments in locally-managed tablespaces.

For a permanent tablespace other than SYSTEM, you can specify EXTENT MANGEMENT LOCAL in the CREATE TABLESPACE command.

For a temporary tablespace, you can specify EXTENT MANGEMENT LOCAL in the CREATE TEMPORARY TABLESPACE command

Say we want to create a data tablespace for the accounts receivable (AR) application in our accounting package. The database for our AR application is called 'ORACTP' short for Oracle Accounting Production database. First let's look at the command to create a dictionary managed tablespace with an initial 500 megabyte datafile with autoextention to a maximum size of 1 gigabyte (1024 megabytes). The tables in our AR application will hold a fairly large amount of data so just in case our developers forget to size their tables (they wouldn't do that would they?) let's size the default storage to INITIAL 10M NEXT 1M PCTINCREASE 10.

```
CREATE TABLESPACE ar DATAFILE '\ORACLE1\ORACTP\data\oractp_ar01.dbf' SIZE 500M

AUTOEXTEND ON NEXT 200M MAXSIZE 1024M

DEFAULT STORAGE (INITIAL 10M NEXT 1M PCTINCREASE 1)

PERMANENT

ONLINE

LOGGING;
```

I've included the PERMANENT, ONLINE and LOGGING clauses for illustration only, they are the default if nothing is specified. Why did I specify a PCTINCREASE value of 1? If PCTINCREASE is set to zero the SMON process will not coalesce free space. If PCTINCREASE is not specified, it will default to 50%, therefore specifying it at a low value is suggested.

Since for this example we want the extents to be locally managed due to the level of dynamic allocation which will happen, the CREATE TABLESPACE clause will change in that we no longer specify the DEFAULT STORAGE clause and instead we use the EXTENT MANAGEMENT clause with the LOCAL clause. If we want to ensure that uniform extents are generated then we can specify the UNIFORM clause as well. Let's shoot for 2 megabyte LOCAL UNIFORM extent management.

```
CREATE TABLESPACE ar DATAFILE '\ORACLE1\ORACTP\data\oractp_ar01.dbf' SIZE 500M

AUTOEXTEND ON NEXT 200M MAXSIZE 1024M

LOCAL UNIFORM SIZE 2M

PERMANENT

ONLINE

LOGGING;
```

Why Use Locally Managed Tabblespaces?

Why should locally managed tablespaces be used? There are several good reasons:

- Improves concurrence of space operations
- Space is allocated and deallocated by changing the bit values (0 to 1 for allocation, 1 to 0 for deallocation).
- Eliminates recursion during space management operations
- Supports temporary tablespace management in standby databases
- Reduces user reliance on the data dictionary
- Necessary information is stored in segment headers and bit map blocks.

- Creating a Temporary Tablespace

Temporary Tablespaces

If you wish to improve the concurrence of multiple sort operations, reduce their overhead, or avoid Oracle space management operations altogether, you can create temporary tablespaces.

Within a temporary tablespace, all sort operations for a given instance and tablespace share a single sort segment. Sort segments exist in every instance that performs sort operations within a given tablespace. You cannot store permanent objects in a temporary tablespace. You can view the allocation and deallocation of space in a temporary tablespace sort segment via the V$SORT_SEGMENT table.

To identify a tablespace as temporary during tablespace creation, issue the following statement:

```
CREATE TABLESPACE tablespace TEMPORARY;
```

To identify a tablespace as temporary when it exists, issue the following statement:

```
ALTER TABLESPACE tablespace TEMPORARY;
```

Unless a tablespace is altered by an alter command the tablespace status (TEMPORARY or PERMANENT) is permanent. This means you can take temporary tablespaces offline and returning temporary tablespaces online does not affect their temporary status.

You can manage space for sort operations more efficiently by designating temporary tablespaces exclusively for sorts. Doing so effectively eliminates serialization of space management operations involved in the allocation and deallocation of sort space.

All operations that use sorts--including joins, index builds, ordering (ORDER BY), the computation of aggregates (GROUP BY), and the ANALYZE command for collecting optimizer statistics--benefit from temporary tablespaces. The performance gains are significant in Oracle Parallel Server environments.

Sort Segments

A temporary tablespace can be used only for sort segments. A temporary tablespace is not the same as a tablespace that a user designates for temporary segments, which can be any tablespace available to the user. No permanent schema objects can reside in a temporary tablespace.

Sort segments are used when a segment is shared by multiple sort operations. One sort segment exists for every instance that performs a sort operation in a given tablespace.

Temporary tablespaces provide performance improvements when you have multiple sorts that are too large to fit into memory. The sort segment of a given temporary tablespace is created at the time of the first sort operation. The sort segment expands by allocating extents until the segment size is equal to or greater than the total storage demands of all of the active sorts running on that instance.

Temporary Datafiles

Temporary datafiles differ from permanent datafiles in that they do not appear in the DBA_DATA_FILES view. Instead, they appear in the DBA_TEMP_FILES view, which is similar to DBA_DATA_FILES view except that it contains information about temporary datafiles. In SQL, files belonging to temporary tablespaces are also identified as TEMPFILES, rather than DATAFILES.

Creating a Locally Managed Temporary Tablespace

If you wish to allocate space that can contain schema objects for the duration of a session in the database, you can create a locally managed temporary tablespace.

You must have the CREATE TABLESPACE system privilege to create a locally managed temporary tablespace.

The following statement creates a temporary tablespace in which each extent is 16M. The default database block size is 4K; each bit in the map represents one extent, thus each bit maps 4,000 blocks.

```
CREATE TEMPORARY TABLESPACE tbs_1 TEMPFILE 'file_1.f'
   BITMAP ALLOCATION UNIFORM SIZE 16M;
```

Locally managed temporary tablespaces have temporary datafiles (tempfiles), which are similar to ordinary datafiles except that:

- Tempfiles are always set to NOLOGGING mode.
- You cannot make a tempfile read-only.
- You cannot rename a tempfile.
- You cannot create a tempfile with the ALTER DATABASE command.
- Media recovery does not recognize tempfiles.
    - BACKUP CONTROLFILE does not generate any information for tempfiles.
    - CREATE CONTROLFILE cannot specify any information about tempfiles.

Altering a Locally Managed Temporary Tablespace

You can alter or add a datafile (or temporary file) to a locally managed temporary tablespace.

The following statement adds files to a locally managed temporary tablespace:

```
ALTER TABLESPACE tbs_1
   ADD TEMPFILE 'file_1.f';
```

The following statements take offline and bring online temporary files:

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' OFFLINE;
```

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' ONLINE;
```

The following statement resizes temporary file TEMP_FILE_1.F to 12K:

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' RESIZE 12K;
```

The following statement drops a temporary file:

```
ALTER DATABASE TEMPFILE 'temp_file_1.f' DROP;
```

Use of the DBMS_SPACE_ADMIN Package to Administer Locally Managed Tablespaces

The DBMS_SPACE_ADMIN package is used to provide management functionality for locally managed tablespaces. The package runs with SYS privileges allowing any user who has the privilege to execute the package to have the ability to manipulate the locally managed tablespace bitmaps.

The DBMS_SPACE_ADMIN package contains a number of useful procedures, we will examine them shortly but first we must lay a little ground work.

DBMS_SPACE_ADMIN Constants

The DBMS_SPACE_ADMIN package defines the constants shown in Table 1.

Constant	Description
SEGMENT_VERIFY_EXTENTS	Verifies that the space owned by the segment is appropriately reflected in the tablespace bitmap as being used. Equivalent to one (1).
SEGMENT_VERIFY_EXTENTS_GLOBAL	Verifies that the space owned by the segment is appropriately reflected in the tablespace bitmap as being used and that no other segment in the tablespace claims any of this space. Equivalent to two (2).
SEGMENT_MARK_CORRUPT	Marks a temporary segment as being corrupt facilitating is elimination (not reclamation) from the dictionary. Equivalent to three (3).
SEGMENT_MARK_VALID	Marks a corrupt temporary segment as being valid. Used when corruption has been resolved and the segment should be reclaimed. Equivalent to four (4).
SEGMENT_DUMP_EXTENT_MAP	Causes a dump of the extent map for the specific segment. Equivalent to five (5).
TABLESPACE_VERIFY_BITMAP	Reconciles the tablespace's bitmap with the extent maps of the segments in the tablespace to verify consistency. Equivalent to six (6),
TABLESPACE_EXTENT_MAKE_FREE	Makes the specified extent of free space available in the tablespace bitmaps. Equivalent to seven (7).
TABLESPACE_MAKE_EXTENT_USED	Makes the specified extent of free space marked as used in the tablespace bitmaps. Equivalent to eight (8).

Table 1 DBMS_SPACE_ADMIN Constants

The constants shown in table 1 are used throughout the rest of the DBMS_SPACE_ADMIN procedures for default values and should be used to specify non-default values where needed.

The DBMS_SPACE_ADMIN SEGMENT_VERIFY Procedure

The purpose of the SEGMENT_VERIFY procedure is to verify that the extent map of the segment is consistent with the segments bitmap. The SEGMENT_VERIFY procedure has the following input variables:

- Tablespace_name VARCHAR2
- Header_relative_file POSITIVE
- Header_block POSITIVE
- Verify_option POSITIVE with a default of SEGMENT_VERIFY_EXTENTS

Where the variables have the following definitions:

- Tablespace_name is the name of the locally managed tablespace

- Header_relative_file is the relative file number of the segment from DBA_SEGMENTS where the value is for the table, cluster or index to be verifed.
- Header_block is the block number of the segment header from DBA_SEGMENTS where the value is for the table, cluster or index to be verifed.
- Verify_option is the action to perform, either the default SEGMENT_VERIFY_EXTENTS or SEGEMENT_VERIFY_EXTENTS_GLOBALLY.

An example of the use of the SEGMENT_VERIFY procedure would be:

```
SQL> select segment_name,relative_fno,header_block
  2  from dba_segments
  3  where segment_name='TEST1'
  4* and tablespace_name='TEST_EM'


SEGMENT_NAME   RELATIVE_FNO HEADER_BLOCK
-------------- ------------ ------------
TEST1                     8           17
1 row selected.


SQL> execute dbms_space_admin.segment_verify('TEST_EM',8,17);
PL/SQL procedure successfully completed.
```

In the above example no trace files where generated, had there been an error or problem a trace file would have been generated that contained the anomalies output as data block address range (dba-range), bitmap-block, bitmap-block range, anomaly information for all anomalies found.

The DBMS_SPACE_ADMIN SEGMENT_CORRUPT Procedure

The segment_corrupt procedure marks the specified segment corrupt or valid so that the proper error recovery can be performed. The procedure has the following input variables:

- Tablespace_name VARCHAR2
- Header_relative_file POSITIVE
- Header_block POSITIVE
- Corrupt_option POSITIVE with a default of SEGMENT_MARK_CORRUPT

Where the varibles tablespace_name, header_relative_file and header_block have the same definitions as for the segment_verify procedure. The values for a given object are found in the DBA_SEGMENTS view. The corrupt_option variable tells the procedure to set the segment as corrupt (the default) or as SEGMENT_MARK_VALID or valid. So if we determined that the segment TEST1 (a table in tablespace TEST_EM) was corrupt we could mark it as so with:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('TEST_EM',8,17);
```

Then, once we fixed the corruption we could set it back to valid with:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.SEGMENT_CORRUPT('TEST_EM',8,17, SEGMENT_MARK_VALID);
```

Using the DBSM_SPACE_ADMIN SEGMENT_DROP_CORRUPT Procedure

The procedure SEGMENT_DROP_CORRUPT drops the specified segment if it has been set to corrupt by the SEGMENT_CORRUPT procedure. In addition to being marked as corrupt, the segment itself must be marked as temporary by issuing a "DROP segment_type segment_name" command against the segment. The SEGMENT_DROP_CORRUPT has the following input variables:

- Tablespace_name VARCHAR2
- Header_relative_file POSITIVE
- Header_block POSITIVE

Where the varibles tablespace_name, header_relative_file and header_block have the same definitions as for the segment_verify procedure. An example showing what happens when the various requirements aren't met follow.

```
-- The object isn't marked as corrupt or dropped:
--
SQL> execute dbms_space_admin.segment_drop_corrupt('TEST_EM',8,17);
begin dbms_space_admin.segment_drop_corrupt('TEST_EM',8,17); end;

*
ERROR at line 1:
ORA-03211: The segment does not exist or is not in a valid state
ORA-06512: at "SYS.DBMS_SPACE_ADMIN", line 0
ORA-06512: at line 1
-- The object isn't dropped (in a temporary state)
--
SQL> execute dbms_space_admin.segment_corrupt('TEST_EM',8,17);
PL/SQL procedure successfully completed.

SQL> execute dbms_space_admin.segment_drop_corrupt('TEST_EM',8,17);
begin dbms_space_admin.segment_drop_corrupt('TEST_EM',8,17); end;
*
ERROR at line 1:
ORA-03211: The segment does not exist or is not in a valid state
ORA-06512: at "SYS.DBMS_SPACE_ADMIN", line 0
ORA-06512: at line 1
-- All criteria are met
```

```
--
SQL> drop table test1;


Table dropped.


SQL> execute dbms_space_admin.segment_drop_corrupt('TEST_EM',8,17);
PL/SQL procedure successfully completed.
```

Using the DBMS_SPACE_ADMIN SEGMENT_DUMP Procedure

The SEGMENT_DUMP procedure generates a dump of the specified segments bitmap into the USER_DUMP_DEST in a file named for the process identifier of the session executing the procedure. On UNIX this value is shown a decimal and the file name is created in decimal, on NT the value is shown in decimal but the file name is created in hexadecimal so be aware this occurs and be prepared. The SEGMENT_DUMP procedure has four input variables:

- Tablespace_name VARCHAR2
- Header_relative_file POSITIVE
- Header_block POSITIVE
- dump_option POSITIVE with a default of SEGMENT_DUMP_EXTENT_MAP

Where the varibles tablespace_name, header_relative_file and header_block have the same definitions as for the segment_verify procedure. The dump_option currently has only one option the default. Here is an example of its use:

```
SQL> execute dbms_space_admin.dump_segment('TEST_EM',8,33);
```

The resulting trace file looks like:

```
Dump file c:\oracle1\ortest1\admin\udump\ORA00317.TRC
Fri May 07 23:23:30 1999
ORACLE V8.1.3.0.0 - Beta vsnsta=1
vsnsql=d vsnxtr=3
Windows NT V4.0, OS V5.101, CPU type 586
Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta
With the Partitioning and Objects options
PL/SQL Release 8.1.3.0.0 - Beta
Windows NT V4.0, OS V5.101, CPU type 586
Instance name: test
Redo thread mounted by this instance: 1
Oracle process number: 11
Windows thread id: 317, image: oracle.exe
*** 1999.05.07.23.23.30.187
```

```
*** SESSION ID:(11.12) 1999.05.07.23.23.29.937
Segment Dump: segment dba = 11 : 0x02000021
Segment Type - Data/Index Segment
  Extent Control Header
-----------------------------------------------------------------
Extent Header:: spare1: 0  space2: 0  #extents: 2   #blocks: 31
                last map  0x00000000  #maps: 0     offset: 2080
  Highwater::  0x02000038  ext#: 1      blk#: 7    ext size: 16
  #blocks in seg. hdr's freelists: 0
  #blocks below: 22
  mapblk  0x00000000  offset: 1
                  Unlocked
Map Header::next0x00000000 #extents:2 obj#:5747 flag:0x40000000
Extent Map
-----------------------------------------------------------------
   0x02000022  length: 15
   0x02000031  length: 16

  nfl = 1, nfb = 1 typ = 1 nxf = 0
  SEG LST:: flg: UNUSED lhd: 0x00000000 ltl: 0x00000000
```

Now what possible good this is to you I don't really know but I'm sure Oracle support will be asking for these so be prepared.

Using the DBMS_SPACE_ADMIN TABLESPACE_VERIFY Procedure

The tablespace_verify procedure verifies that the bitmaps and the extent maps for objects in a tablespace are synchronized. The tablespace_verify procedure has two input variables:

- Tablespace_name VARCHAR2
- Verify_option POSITIVE with a default value of TABLESPACE_VERIFY_BITMAP

Where the tablespace_name variable has the same definition as for the segment_verify procedure. The verify_option variable is the verification method you want the procedure to use, currently you have one choice, TABLESPACE_VERIFY_BITMAP the default. An example use of the procedure would be:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_VERIFY('TEST_EM');
PL/SQL procedure sucessfully completed
```

The output is placed into the USER_DUMP_DEST location as a trace file. An example output would be:

```
Dump file c:\oracle1\ortest1\admin\udump\ORA00317.TRC
```

```
Fri May 07 23:23:30 1999

ORACLE V8.1.3.0.0 - Beta vsnsta=1

vsnsql=d vsnxtr=3

Windows NT V4.0, OS V5.101, CPU type 586

Oracle8 Enterprise Edition Release 8.1.3.0.0 - Beta

With the Partitioning and Objects options

PL/SQL Release 8.1.3.0.0 - Beta

Windows NT V4.0, OS V5.101, CPU type 586

Instance name: test


*** 1999.05.07.23.43.53.390

BitMap entry partially used with no Extent Map entry

Range RelFno 8: BeginBlock: 0 EndBlock: 4194303
```

Use of the DBMS_SPACE_ADMIN TABLESPACE_FIX_BITMAPS Procedure

The tablespace_fix_bitmaps procedure marks the appropriate DBA range as free or used in the bitmap. The tablespace_fix_bitmaps procedure has the following inputs:

- Tablespace_name VARCHAR2
- Dbarange_relative_file POSITIVE
- Dbarange_begin_block POSITIVE
- Dbarange_end_block POSITIVE
- Fix_option POSITIVE

Where the variable definitions are:

- Tablespace_name is the name of the tablespace in which the bitmap needs to be fixed.
- Dbarange_relative_file is the relative file number of the file that contains the bitmap of interest
- Dbarange_begin_block is the block number of the beginning of the extent to fix
- Dbarange_end_block is the block number of the ending of the extent to fix.
- Fix_option is one of TABLESPACE_EXTENT_MAKE_FREE or TABLESPACE_EXTENT_MAKE_USED with no default value so one or the other must be specified.

For example the blocks 33 through 64 in the TEST_EM tablespace contain the table TEST2 which is a copy of the DBA_TABLES view as a table. We want to mark this as used space:

```
SQL>DBMS_SPACE_ADMIN.TABLESPACE_FIX_BITMAPS('TEST-EM',8,33,64,

DBMS_SPACE-ADMIN.TABLESPACE_EXTENT_MAKE_USED);


PL/SQL procedure sucessfully completed.
```

Using the DBMS_SPACE_ADMIN TABLESPACE_REBUILD_BITMAPS Procedure

The TABLESPACE_REBUILD_BITMAPS procedure rebuilds the bitmaps for the specified tablespace. As of version 8.1.5 it will only do all the bitmaps for the entire tablespace. The TABLESPACE_REBUILD_BITMAPS procedure has three input variables:

- Tablespace_name VARCHAR2
- Bitmap_relative_file POSITIVE with a default of NULL
- Bitmap_block POSITIVE with a default of NULL

Where the input variables are defined as:

- Tablespace_name is the name of the tablespace that requires its bitmaps to be rebuilt
- Bitmap_relative_file is the relative file number of the file in which the bitmap that needs rebuilt resides, however, this is not functional as of 8.1.5 so don't specify it or an error will result.
- Bitmap_block is the block number for the bitmap, however, this is not functional as of 8.1.5 so don't specify it or an error will result.

An example would be if we wanted to rebuild the TEST_EM tablespace bitmaps:

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_BITMAPS('TEST_EM');

PL/SQL procedure successfully completed.
```

Using the DBMS_SPACE_ADMIN TABLESPACE_REBUILD_QUOTAS Procedure

The TABLESPACE_REBUILD_QUOTAS procedure rebuilds the quotas for a specified tablespace. The procedure has a single input value, the tablespace_name as a VARCHAR2. An example of the use of tablespace_rebuild_quotas follows.

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_REBUILD_QUOTAS('TEST_EM');

PL/SQL procedure successfully completed.
```

Using the DBMS_SPACE_ADMIN TABLESPACE_MIGRATE_FROM_LOCAL Procedure

The procedure TABLESPACE_MIGRATE_FROM_LOCAL is used to migrate a locally managed tablespace to a system or directory managed tablespace. The SYSTEM tablespace and temporary tablespaces, if they have been created as a locally managed tablespaces, cannot be migrated to directory managed tablespaces as of release 8.1.5 although this functionality may be supported in future releases. The TABLESPACE_MIGRATE_FROM_LOCAL procedure has one input variable, tablespace_name as a VARCHAR2 which corresponds to the name of the tablespace to migrate.

The tablespace being migrated must be online and in READ WRITE mode during the execution of the TABLESPACE_MIGRATE_FROM_LOCAL procedure.

```
SQL> EXECUTE DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL('TEST_EM');

PL/SQL procedure successfully completed.
```

When to Use The DBMS_SPACE_ADMIN Package

I would suggest using the procedures in the DBMS_SPACE_ADMIN package against locally managed tablespaces that undergo high activity, especially if there are inadvertent power fluctuations that result in database crashes. Of course the procedures in DBMS_SPACE_SADMIN should also be run anytime you suspect corruption of the bitmaps or need to migrate from a locally to a directory managed tablespace. It may even be advisable to verify all locally managed tablespace bitmaps on a periodic, scheduled basis and perform periodic rebuilds as well.

The periodic running of DBMS_SPACE_ADMIN procedures such as TABLESPACE_VERIFY, TABLESPACE_REBUILD_BITMAPS and TABLESPACE_REBUILD_QUOTAS should be accomplished by encapsulating the calls to these packages inside PL/SQL begin-end blocks and using the DBMS_JOB package. The rebuilds should be scheduled during times with the tablespaces (and database) are quiescent if possible.

Making a Tablespace Read-Only

One of the new features with later versions of Oracle7 and all versions of Oracle8 is the read-only tablespace. A read-only tablespace, as its name inplies, allows read-only access to its information. This is beneficial in several ways, since we are only reading data no redo or rollback is required for read-only tablespaces, read-only tablespaces only need to be backed up once after they are made read-only, then you can remove them from the backup plan, read-only tablespaces can be left offline until just before they are needed (in Oracle8i) for example, on removable media such as CD-ROM or PD CD-ROM and then brought online only when needed and finally, read-only tablespaces do not participate in checkpoints.

Normally you would have two related tablespaces, a data and an index tablespace that you would want to make read-only. I would suggest dropping and rebuilding the indexes prior to making the index tablespace read-only as well as correcting any space management problems such as chaining or excessive fragmentation on the data tablespace.

Of course, read-only tablespaces cannot be updated unless they are made readable again through use of the ALTER TABLESPACE command.

The general procedure for making a tablespace read only would be:

1. Create the tablespace as a normal, permanent tablespace.
2. Populate the tablespace as you would a normal tablespace.
3. Once all data has been added to the tables in the tablespace, alter the tablespace to read-only.
4. Backup the tablespace using normal system backups.

Once a tablespace has been made read-only you can transfer it to a media such as CD-ROM and then do a rename command on its datafile(s) using the ALTER DATABASE command to relocate them to the CD-ROM. Script 2 demonstrates this process.

```
SQL> alter tablespace graphics_data read only;

Tablespace altered.


SQL> alter tablespace graphics_data offline;
```

```
Tablespace altered.


... Use operating system commands to copy file to new location ...


SQL> alter database rename file 'D:\ORANT8\DATABASE\ORTEST1_GRAPHICS_DATA01.DBF' to
  2* 'H:\ORACLE5\ORTEST1\DATA\ORTEST1_GRAPHICS_DATA01.DBF'
SQL> /
Database altered.


... Here, just to be sure, I rename the actual file I copied from to .old


SQL> alter tablespace graphics_data online;
Tablespace altered.
```

Script 1: Example of Moving a Tablespace Datafile That Has Been Made Read-Only

In the code in script 1 notice how I renamed the file to a location on the H: drive, this happens to be a Panasonic PD CD-ROM drive, in the next example I will show how to use the new initialization parameter READ_ONLY_OPEN_DELAYED.

Use of READ_ONLY_OPEN_DELAYED With Read-Only Tablespaces

In this next section we will look at the new initialization parameter that deals specifically with read-only tablespaces, READ_ONLY_OPEN_DELAYED. What this parameter tells Oracle is to allow starting of the database even if some or all of the read-only tablespaces are offline or un-available (for example, someone was using the CD-ROM to listen to Bruce Springsteen and forgot to slip the CD with your read-only tablespace back in before startup).

I am afraid you will have to take a few things on faith since it is rather hard in print to show actual activities. What I did to test this was first issue the following commands to demonstrate that the read-only tablespace GRAPHICS_DATA was online and had active data (albeit read-only data):

```
SQL> select table_name from dba_tables where tablespace_name='GRAPHICS_DATA';
TABLE_NAME
------------------------------
GRAPHICS_TABLE
INTERNAL_GRAPHICS
BASIC_LOB_TABLE
3 rows selected.


SQL> desc graphics_dba.internal_graphics
 Name                           Null?    Type
 ------------------------------ -------- ----
 GRAPHIC_ID                              NUMBER
```

```
 GRAPHIC_DESC                              VARCHAR2(30)
 GRAPHIC_BLOB                              BLOB
 GRAPHIC_TYPE                              VARCHAR2(10)

SQL> select graphic_id,graphic_desc from graphics_dba.internal_graphics where rownum<10;

GRAPHIC_ID GRAPHIC_DESC
---------- ------------------------------
         1 April book of days woodcut
         2 August book of days woodcut
         3 Benzene Molecule Graphic
         4 c20conto.gif
         5 cover11b.gif
         6 December book of days woodcut
         7 February book of days woodcut
         8 harris-c.gif
         9 HIV Virus Image
9 rows selected.
```

I then added the initialization parameter READ_ONLY_OPEN_DELAYED=TRUE (it defaults to FALSE) to the initialization parameter file and shutdown the database. Once the database was shutdown I opened the PD CD-ROM CD drawer with the PD CD-ROM disk that holds the GRAPHICS_DATA tablespace datafile. I then restarted the database.

No error was generated even with the PD CD-ROM drawer open thus making the GRAPHICS-DATA datafile unavailable. I then issued the following commands to see what could be expected for attempting to access the offline tablespace:

```
SQL> select graphic_id,graphic_desc from graphics_dba.internal_graphics where rownum<10;
select graphic_id,graphic_desc from graphics_dba.internal_graphics
                                                 *
ERROR at line 1:
ORA-01157: cannot identify data file 4 - file not found
ORA-01110: data file 4: 'H:\ORACLE5\ORTEST1\DATA\ORTEST1_GRAPHICS_DATA01.DBF'
```

After verifying that the datafile was in fact not available I simply reloaded the PD CD-ROM cartridge and reissued the command, all with the database online and active:

```
SQL> r
  1* select graphic_id,graphic_desc from graphics_dba.internal_graphics where rownum<10

GRAPHIC_ID GRAPHIC_DESC
```

```
---------- ------------------------------
         1 April book of days woodcut
         2 August book of days woodcut
         3 Benzene Molecule Graphic
         4 c20conto.gif
         5 cover11b.gif
         6 December book of days woodcut
         7 February book of days woodcut
         8 harris-c.gif
         9 HIV Virus Image
```

As you can see, the database acts like nothing was ever wrong. This allows use of a database to continue even if something has happened to make your read-only tablespaces temporarily unavailable. One caution, always access read-only tablespaces at least once before shutdown when using the READ_ONLY_OPEN_DELAYED parameter set to TRUE. I found by experience that if your read-only tablespace is not accessed and the database is shutdown and restarted several times your control file may loose track of the read-only tablespace and will report it as corrupted when you finally try to access it.

Using Transportable Tablespaces

A new feature with Oracle8i is the ability to move a tablespace and its datafiles from one database to another. A transportable set of tablespace sis one that is self contained. For example, a set of data and index tablespaces where all internal references between tables and indexes are resolved within that set of tablespaces. However, before you get too excited there are some limitations:

1. The source and target database must be on the same platform (not the same machine, but the same platform (i.e. SUN Version 5.6).
2. The source and destination database must have the same blocksize.
3. The source and destination database must have the same character set.
4. There is no way to change the owner of the tablespace objects, so the owner(s) must have users set up in both databases.
5. The tablespaces to be moved cannot containn bitmap indexes, these must be dropped before themove and rebuilt afterwards.
6. There is no way to rename the tablespace in transit, it must not already exist in the target database.
7. Tablespaces containing nested tables and VARRAYs cannot be transported.
8. Both source and target must be have compatibility at least set to 8.1 with the source being at an equal or lower version than the target database. The actual versions do not matter as long as the compatibility setpoints are from equal or lower on the source to equal or higher on the target database.

To check if the set of tablespaces is self-contained Oracle has provided a stored package called DBMS_TTS that contains a package called TRANSPORT_SET_CHECK. This procedure is fed the

names of the tablespaces in your transport set and it verifies the set is self-contained. For example to check if AP_DATA and AP_INDEX tablespaces are a self-contained set the command would be:

```
EXECUTE DBMS_TTS.TRANSPORT_SET_CHECK('AP_DATA,AP_INDEX', TRUE);
```

The TRUE entry in the TRANSPORT_SET_CHECK call corresponds to whether or not you wish to verify for constraints. Constraints have to be internally consistant as well as tables, indexes and clusters. If there are violations in the set of tablespaces that prohibit their being a self-contained set, they will be written in human readable form into the TRANSPORT_SET_VIOLATIONS table.

Once all of the tablespaces in your transport set are verified to be a self-contained set, you set them all to be read-only using the alter tablespace command. Once all are set to read-only a special type of export is performed that creates the data dictionary information for use in the target database. For our example tablespaces the command to create the data dictionary export including all triggers, constraints and grants would be:

```
Exp transport_tablespace=y tablespaces=ap_data,ap_index triggers=y constraints=y grants=y

File=tts.dmp
```

Obviously if you set grants, triggers or constraints to n (no) then they are ignored and not exported. The default settings for the grants, triggers and constraints options is y (yes).

The next step is to copy the tablespace datafiles and the export file to the target database platform. After the tablespace datafiles are copied the original tablespaces can be altered back to READ WRITE mode if desired.

The final step is to "plug in" the tablespaces to the target Oracle8i database.

The final step consists of:

1. Put the target tablespace datafiles into the OFA structure of the target database so the database can find them.
2. Plug in the tablespaces by adding their metadata via an import into the target database from the export file we created before:

```
Imp transport_tablespace=y
datafiles=(d:\oracle2\ap_db\data\ap_data01.dbf,e:\oracle3\ap_db\data\ap_index.dbf) file=tss_dmp
tablespaces=(ap_data,ap_index) owners=(ap_dba);
```

The datafiles parameter must be specified to tell the target database where to find the transported tabelspace datafiles, the tablespaces and owners parameters are optional. Once the import completes, verify the import log for errors and then you can use the alter tablespace command to make the tablespaces READ WRITE again (if desired, they are still in READ ONLY after the import completes.) As with other export and import operations a long list of parameters can be placed into a parameter file and referenced with the PARFILE parameter for ease of use.

Some things to watch are that ROWIDs may no longer be unique after a transportable set of tablespaces are plugged in as ROWIDs are not regenerated. REF values are not checked when consistancy is verified so there may be dangling REF values if the REF targets are not self-contained within the transported set of tablespaces. BFILE values will be invalid unless the external files they refer to are also moved with the tablespace. Triggers, if exported, are exported without validity checks so they may generate an error on import if they are not self-contained in the transport set. Snapshot and replication data is not transported.

As long as you operate within the above guidelines transportable tablespaces will make moving data between databases faster and easier than ever before as it is much faster to do the data dictionary export and copy the files than to do any data extraction and reloading. Another strong point is that you carry the indexes with the transport set so they do not have to be rebuilt (however, beware of duplicate rowid's causing erroneous results.)

Summary

Oracle Corporation has provided DBAs with many new features and tools for managing tablespaces. DBAs also have many new tablespace features that provide expanded functionality allowing more flexible use of tablespaces. The new features include:

Locally Managed Tablespaces

Temporary Tablespaces

Read-Only tablespaces with deferred status checking

Transportable Tablespaces

Through proper use of these new features DBAs should be able to optimize tablespace usage in a manner never available before.

(Parts of the above paper were excerpted from the book: "Oracle8i Administration and Management", Michael R. Ault, John Wiley and Sons Publishers).

Michael Ault can be contacted through the web page http://www.dbmentors.com or via email at mikerault@earthlink.net.