module4-tablespaces_and_datafiles.htm; updated May 22, 2012; Figures shown in these notes are from Oracle document D11321GC11, Production 1.1 D34286, and Oracle Database Concepts 11g Release 2 (11.2).

# Module 4 – Tablespaces and Datafiles

## Objectives

These notes cover the creation and management of tablespaces and their associated datafiles. You will learn how to create both locally managed and dictionary managed tablespaces.

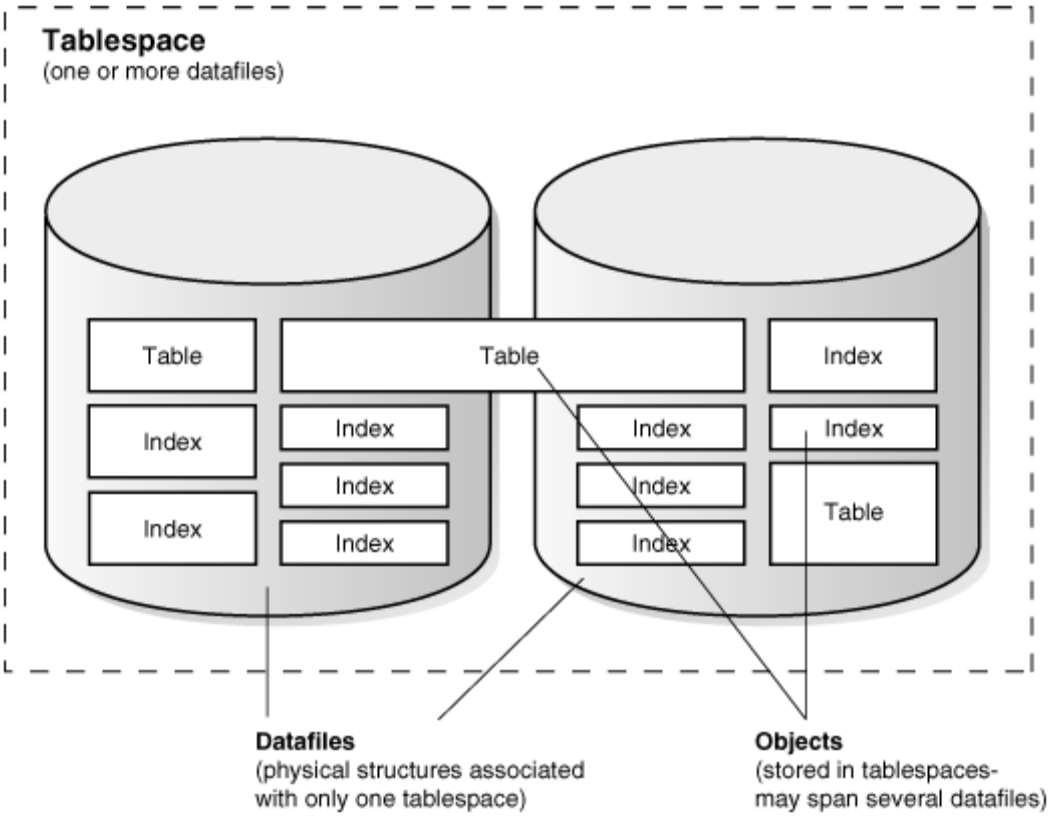
## Tablespaces vs. Datafiles

An Oracle database is comprised of **tablespaces**.

Tablespaces **logically organize** data that are **physically stored** in datafiles.

- A tablespace belongs to only one database, and has at least one datafile that is used to store data for the associated tablespace.
- Because disk drives have a finite size, a tablespace can span disk drives when datafiles from more than one disk drive are assigned to a tablespace. This enables systems to be very, very large.
- Datafiles are always assigned to only one tablespace and, therefore, to only one database.

As is shown in the figure below, a **tablespace** can span datafiles. The term "tablespaces" is misleading because a tablespace can store tables, but can also store many other database objects such as indexes, views, sequences, etc.




# Tablespace Types

There are three types of tablespaces: (1) permanent, (2) undo, and (3) temporary.

- **Permanent** – These tablespaces store objects in segments that are permanent – that persist beyond the duration of a session or transaction.

- **Undo** – These tablespaces store segments that may be retained beyond a transaction, but are basically used to:

    - Provide read consistency for SELECT statements that access tables that have rows that are in the process of being modified.

    - Provide the ability to rollback a transaction that fails to commit.

- **Temporary** – This tablespace stores segments that are transient and only exist for the duration of a session or a transaction. Mostly, a temporary tablespace stores rows for sort and join operations.

## How Many Tablespaces Are Needed for a Database?

Up to Oracle 9i, the smallest Oracle database could have just a **single** tablespace with a single disk drive. This tablespace would be the **SYSTEM** tablespace because it is the one tablespace that is required in every Oracle database.

Beginning with Oracle 10g, the smallest Oracle database is two tablespaces:

- **SYSTEM** – stores the data dictionary.
- **SYSAUX** – stores data for auxiliary applications (covered in more detail later in these notes).

In reality, a typical production database has numerous tablespaces. These include **SYSTEM** and **NON-SYSTEM** tablespaces.

**SYSTEM** – a tablespace that is always used to store **SYSTEM** data that includes data about tables, indexes, sequences, and other objects – this metadata comprises the data dictionary.

- Every Oracle database has to have a **SYSTEM** tablespace—it is the first tablespace created when a database is created.
- Accessing it requires a higher level of privilege.
- You cannot rename or drop a **SYSTEM** tablespace.
- You cannot take a **SYSTEM** tablespace offline.

- The **SYSTEM** tablespace could store user data, but this is not normally done—a good rule to follow is to never allow the storage of user segments in the **SYSTEM** tablespace.

- This tablespace always has a **SYSTEM Undo segment**.

All other tablespaces are referred to as **Non-SYSTEM**. A different tablespace is used to store organizational data in tables accessed by application programs, and still a different one for undo information storage, and so on. There are several reasons for having more than one tablespace:

- Flexibility in database administration.

- Separate data by backup requirements.

- Separate dynamic and static data to enable database tuning.

- Control space allocation for both applications and system users.

- Reduce contention for input/output path access (to/from memory/disk).

# CREATE TABLESPACE Command

To create a tablespace you must have the CREATE TABLESPACE privilege.

The full **CREATE TABLESPACE** (and **CREATE TEMPORARY TABLESPACE**) command syntax is shown here.

```
CREATE TABLESPACE tablespace
  [DATAFILE clause]
  [MINIMUM EXTENT integer[K|M]]
  [BLOCKSIZE integer [K]]
  [LOGGING|NOLOGGING]
  [DEFAULT storage_clause ]
  [ONLINE|OFFLINE]
  [PERMANENT|TEMPORARY]
  [extent_management_clause]
  [segment_management_clause]
```

As you can see, almost all of the clauses are optional. The clauses are defined as follows:

- **TABLESPACE**: This clause specifies the tablespace name.
- **DATAFILE**: This clause names the one or more datafiles that will comprise the tablespace and includes the full path, example:

```
DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf' SIZE
10M
```

- **MINIMUM EXTENT**: Every used extent for the tablespace will be a multiple of this integer value. Use either T, G, M or K to specify terabytes, gigabytes, megabytes, or kilobytes.

- **BLOCKSIZE**: This specifies a nonstandard block size – this clause can only be used if the DB_CACHE_SIZE parameter is used and at least one DB_nK_CACHE_SIZE parameter is set and the integer value for BLOCSIZE must correspond with one of the DB_nK_CACHE_SIZE parameter settings.

- **LOGGING**: This is the default – all tables, indexes, and partitions within a tablespace have modifications written to Online Redo Logs.

- **NOLOGGING**: This option is the opposite of LOGGING and is used most often when large direct loads of clean data are done during database creation for systems that are being ported from another file system or DBMS to Oracle.

- **DEFAULT storage_clause**: This specifies default parameters for objects created inside the tablespace. Individual storage clauses can be used when objects are created to override the specified DEFAULT.

- **OFFLINE**: This parameter causes a tablespace to be unavailable after creation.

- **PERMANENT**: A permanent tablespace can hold permanent database objects.

- **TEMPORARY**: A temporary tablespace can hold temporary database objects, e.g., segments created during sorts as a result of ORDER BY clauses or JOIN views of multiple tables. A temporary tablespace cannot be specified for EXTENT MANAGEMENT LOCAL or have the BLOCKSIZE clause specified.

- **extent_management_clause**: This clause specifies how the extents of the tablespace are managed and is covered in detail later in these notes.

- **segment_management_clause**: This specifies how Oracle will track used and free space in segments in a tablespace that is using free lists or bitmap objects.

- **datafile_clause:** filename [SIZE integer [K|M] [REUSE]

  [ AUTOEXTEND ON | OFF ]

  filename: includes the path and filename and file size. .

  REUSE: specified to reuse an existing file.

- **NEXT**: Specifies the size of the next extent.

- **MAXSIZE**: Specifies the maximum disk space allocated to the tablespace. Usually set in megabytes, e.g., 400M or specified as UNLIMITED.

# Tablespace Space Management

Tablespaces can be either **Locally Managed** to **Dictionary Managed**. Dictionary managed tablespaces have been ***deprecated*** with Oracle 11g; however, you may encounter them when working at a site that is using Oracle 10g.

When you create a tablespace, if you do not specify extent management, the default is locally managed.

## Locally Managed

The extents allocated to a locally managed tablespace are managed through the use of **bitmaps**.

- Each bit corresponds to a block or group of blocks (an extent).

- The bitmap value (on or off) corresponds to whether or not an extent is allocated or free for reuse.

## Locally Managed Tablespaces

- Reduced contention on data dictionary tables
- No undo generated when space allocation or deallocation occurs
- No coalescing required

```
CREATE TABLESPACE userdata
DATAFILE '/u01/oradata/userdata01.dbf' SIZE 500M
EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

- Local management is the default for the **SYSTEM** tablespace beginning with Oracle 10g.
- If the SYSTEM tablespace is locally managed, the other tablespaces in the database must also be either locally managed or read-only.
- Local management reduces contention for the SYSTEM tablespace because space allocation and deallocation operations for other tablespaces do not need to use data dictionary tables.
- The **LOCAL** option is the default so it is normally not specified.
- With the **LOCAL** option, you cannot specify any **DEFAULT STORAGE**, **MINIMUM EXTENT**, or **TEMPORARY** clauses.
- **UNIFORM** – a specification of **UNIFORM** means that the tablespace is managed in uniform extents of the **SIZE** specified.
  - use UNIFORM to enable exact control over unused space and when you can predict the space that needs to be allocated for an object or objects.
  - Use **K**, **M**, **G**, **T**, etc to specify the extent size in kilobytes, megabytes, gigabytes, terabytes, etc. The default is **1M**; however, you can specify the extent size with the SIZE clause of the UNIFORM clause.

- **AUTOALLOCATE** – a specification of **AUTOALLOCATE** instead of **UNIFORM**, then the tablespace is system managed and you cannot specify extent sizes.
  - **AUTOALLOCATE** is the default.
    - this simplifies disk space allocation because the database automatically selects the appropriate extent size.
    - this does waste some space but simplifies management of tablespace.
  - Tablespaces with AUTOALLOCATE are allocated minimum extent sizes of 64K – dictionary-managed tablespaces have a minimum extent size of two database blocks.

**Advantages of Local Management**: Basically all of these advantages lead to improved system performance in terms of response time, particularly the elimination of the need to coalesce free extents.

- Local management avoids recursive space management operations. This can occur in dictionary managed tablespaces if consuming or releasing space in an extent results in another operation that consumes or releases space in an undo segment or data dictionary table.
- Because locally managed tablespaces do not record free space in data dictionary tables, they reduce contention on these tables.
- 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.
- Changes to the extent bitmaps do not generate undo information because they do not update tables in the data dictionary (except for special cases such as tablespace quota information).

Example CREATE TABLESPACE command – this creates a locally managed **Inventory** tablespace with AUTOALLOCATE management of extents.

```
CREATE TABLESPACE inventory

    DATAFILE '/u02/student/dbockstd/oradata/USER350invent01.dbf'
SIZE 50M

    EXTENT MANAGEMENT LOCAL AUTOALLOCATE;
```

Example CREATE TABLESPACE command – this creates a locally managed **Inventory** tablespace with UNIFORM management of extents with extent sizes of 128K.

```
CREATE TABLESPACE inventory

    DATAFILE '/u02/student/dbockstd/oradata/USER350invent01.dbf'
SIZE 50M

    EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K;
```

## Possible Errors

You cannot specify the following clauses when you explicitly specify EXTENT MANAGEMENT LOCAL:

- DEFAULT storage clause
- MINIMUM EXTENT
- TEMPORARY

### Segment Space Management in Locally Managed Tablespaces

Use the SEGMENT SPACE MANAGEMENT clause to specify how free and used space within a segment is to be managed. Once established, you cannot alter the segment space management method for a tablespace.

**MANUAL**: This setting uses free lists to manage free space within segments.

- Free lists are lists of data blocks that have space available for inserting rows.
- You must specify and tune the PCTUSED, FREELISTS, and FREELIST GROUPS storage parameters.
- MANUAL is the default.

**AUTO**: This uses bitmaps to manage free space within segments.

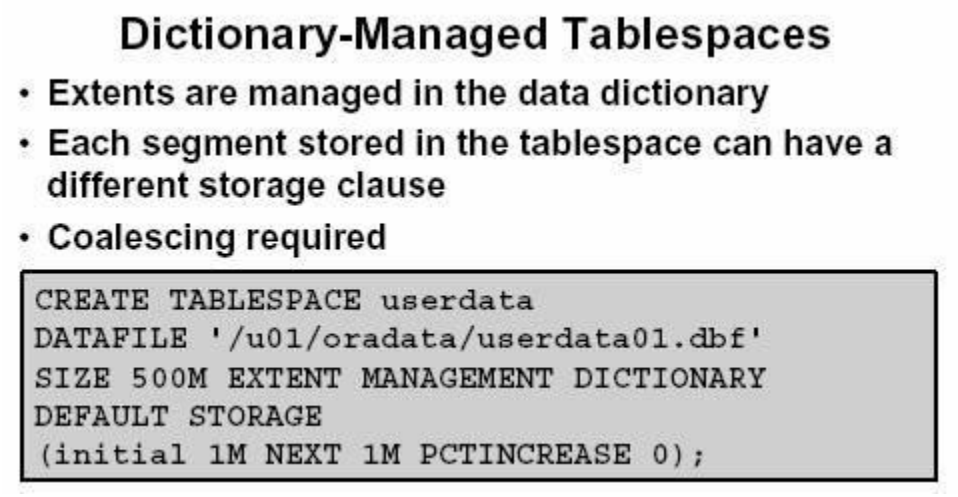
- A bitmap describes the status of each data block within a segment with regard to the data block's ability to have additional rows inserted.
- Bitmaps allow Oracle to manage free space automatically.
- Specify automatic segment-space management only for permanent, locally managed tablespaces.
- Automatic generally delivers better space utilization than manual, and it is self-tuning.

Example CREATE TABLESPACE command – this creates a locally managed **Inventory** tablespace with AUTO segment space management.

```
CREATE TABLESPACE inventory

    DATAFILE '/u02/student/dbockstd/oradata/USER350invent01.dbf'
SIZE 50M

    EXTENT MANAGEMENT LOCAL

    SEGMENT SPACE MANAGEMENT AUTO;
```

## Dictionary Managed

With this approach the data dictionary contains tables that store information that is used to manage extent allocation and deallocation manually.




The **DEFAULT STORAGE** clause enables you to customize the allocation of extents. This provides increased flexibility, but less efficiency than locally managed tablespaces.

Example – this example creates a tablespace using all DEFAULT STORAGE clauses.

```
CREATE TABLESPACE inventory

  DATAFILE '/u02/student/dbockstd/oradata/USER350invent01.dbf'
SIZE 50M

  EXTENT MANAGEMENT DICTIONARY

  DEFAULT STORAGE (

    INITIAL 50K

    NEXT 50K

    MINEXTENTS 2

    MAXEXTENTS 50
```

```
PCTINCREASE 0);
```

- The tablespace will be stored in a single, 50M datafile.
- The EXTENT MANAGEMENT DICTIONARY clause specifies the management.
- All segments created in the tablespace will inherit the default storage parameters unless their storage parameters are specified explicitly to override the default.

The storage parameters specify the following:

- **INITIAL** – size in bytes of the first extent in a segment.
- **NEXT** – size in bytes of second and subsequent segment extents.
- **PCTINCREASE** – percent by which each extent after the second extent grows.
    - **SMON** periodically coalesces free space in a dictionary-managed tablespace, but only if the **PCTINCREASE** setting is NOT zero.
    - Use **ALTER TABLESPACE <tablespacename> COALESCE** to manually coalesce adjacent free extents.
- **MINEXTENTS** – number of extents allocated at a minimum to each segment upon creation of a segment.
- **MAXEXTENTS** – number of extents allocated at a maximum to a segment – you can specify UNLIMITED.

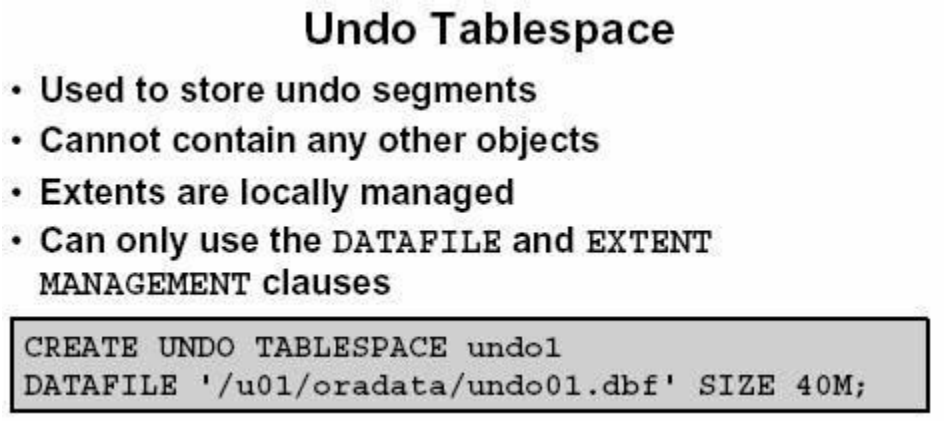
# SYSAUX Tablespace

The **SYSAUX** tablespace stores data for auxiliary applications such as the LogMiner, Workspace Manager, Oracle Data Mining, Oracle Streams, and many other Oracle tools.

- This tablespace is automatically created if you use the Database Creation Assistant software to build an Oracle database.
- Like the **SYSTEM** tablespace, **SYSAUX** requires a higher level of security and it cannot be dropped or renamed.
- Do not allow user objects to be stored in **SYSAUX**. This tablespace should only store system specific objects.
- This is a permanent tablespace.

## UNDO Tablespace

The **Undo tablespace** is used for automatic undo management. Note the required use of the **UNDO** clause within the **CREATE** command shown in the figure here.




More than one UNDO tablespace can exist, but only one can be active at a time.

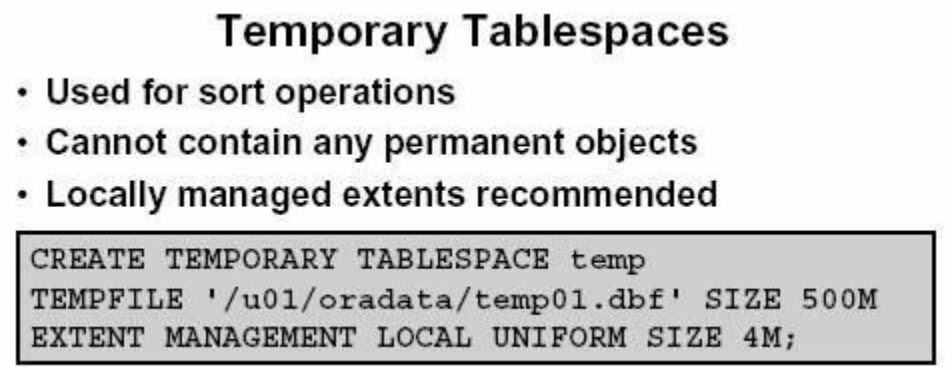
A later set of notes will cover UNDO management in detail.

# TEMPORARY Tablespace

A TEMPORARY tablespace is used to manage space for sort operations. Sort operations generate segments, sometimes large segments or lots of them depending on the sort required to satisfy the specification in a **SELECT** statement's **WHERE** clause.

Sort operations are also generated by **SELECT** statements that join rows from within tables and between tables.

Note the use of the **TEMPFILE** instead of a **DATAFILE** specification for a temporary tablespace in the figure shown below.




- Tempfiles are also in a **NOLOGGING** mode.
- Tempfiles cannot be made read only or be renamed.
- Tempfiles are required for read-only databases.

- Tempfiles are not recovered during database recovery operations.
- The **UNIFORM SIZE** parameter needs to be a multiple of the **SORT_AREA_SIZE** to optimize sort performance.
- The **AUTOALLOCATE** clause is not allowed for temporary tablespaces.
- The default extent **SIZE** parameter is **1M**.

## Default Temporary Tablespace

Each database needs to have a specified **default temporary tablespace**. If one is not specified, then any user account created without specifying a **TEMPORARY TABLESPACE clause** is assigned a temporary tablespace in the **SYSTEM** tablespace!

This should raise a red flag as you don't want system users to execute SELECT commands that cause sort operations to take place within the SYSTEM tablespace.

If a default temporary tablespace is not specified at the time a database is created, a DBA can create one by altering the database.

```
ALTER DATABASE DEFAULT TEMPORARY TABLESPACE temp;
```

After this, new system user accounts are automatically allocated **temp** as their temporary tablespace. If you ALTER DATABASE to assign a new default temporary tablespace, all system users are automatically reassigned to the new default tablespace for temporary operations.

Limitations:

- A default temporary tablespace cannot be dropped unless a replacement is created. This is usually only done if you were moving the tablespace from one disk drive to another.

- You cannot take a default temporary tablespace offline – this is done only for system maintenance or to restrict access to a tablespace temporarily. None of these activities apply to default temporary tablespaces.

- You cannot alter a default temporary tablespace to make it permanent.

## Temporary Tablespace Groups

You can have more than one temporary tablespace online and active. Oracle supports this through the use of ***temporary tablespace groups*** – this is a synonym for a list of temporary tablespaces.

- A single user can have more than one temporary tablespace in use by assigning the temporary tablespace group as the default to the user instead of a single temporary tablespace.

- Example: Suppose two temporary tablespaces named TEMP01 and TEMP02 have been created. This code assigns the tablespaces to a group named TEMPGRP.

```
SQL> ALTER TABLESPACE temp01 TABLESPACE GROUP tempgrp;

Tablespace altered.

SQL> ALTER TABLESPACE temp02 TABLESPACE GROUP tempgrp;

Tablespace altered.
```

- Example continued: This code changes the database's default temporary tablespace to TEMPGRP – you use the same command that would be used to assign a temporary

tablespace as the default because temporary tablespace groups are treated logically the same as an individual temporary tablespace.

```
SQL> ALTER DATABASE DEFAULT TEMPORARY TABLESPACE tempgrp;

Database altered.
```

- To drop a tablespace group, first drop all of its members. Drop a member by assigning the temporary tablespace to a group with an empty string.

```
SQL> ALTER TABLESPACE temp01 TABLESPACE GROUP '';

Tablespace altered.
```

- To assign a temporary tablespace group to a user, the CREATE USER SQL command is the same as for an individual tablespace. In this example user350 is assigned the temporary tablespace TEMPGRP.

```
SQL> CREATE USER user350 IDENTIFIED BY secret_password
  2   DEFAULT TABLESPACE users
  3     TEMPORARY TABLESPACE tempgrp;
```

# USERS, DATA and INDEXES Tablespaces

Most Oracle databases will have a **USERS** permanent tablespace.

- This tablespace is used to store objects created by individual users of the database.

- At SIUE we use the USERS tablespace as a storage location for tables, indexes, views, and other objects created by students.

- All students share the same USERS tablespace.

Many Oracle databases will have one or more **DATA** tablespaces.

- A DATA tablespace is also permanent and is used to store application data tables such as ORDER ENTRY or INVENTORY MANAGEMENT applications.

- For large applications, it is often a practice to create a special DATA tablespace to store data for the application. In this case the tablespace may be named whatever name is appropriate to describe the objects stored in the tablespace accurately.

Oracle databases having a DATA (or more than one DATA) tablespace will also have an accompanying INDEXES tablespace.

- The purpose of separating tables from their associated indexes is to improve I/O efficiency.

- The DATA and INDEXES tablespaces will typically be placed on different disk drives thereby providing an I/O path for each so that as tables are updated, the indexes can also be updated simultaneously.

# Bigfile Tablespaces

A ***Bigfile tablespace*** is best used with a server that uses a RAID storage device with disk stripping – a single datafile is allocated and it can be up to **8EB** (exabytes, a million terabytes) in size.

Why are these important?

- The maximum number of datafiles in an Oracle database is limited (usually to **64K** files) – think big here—think about a database for the internal revenue service.
    - A Bigfile tablespace with **8K** blocks can contain a **32 terabyte** datafile.
    - A Bigfile tablespace with **32K** blocks can contain a **128 terabyte** datafile.
    - These sizes enhance the storage capacity of an Oracle database.
    - These sizes can also reduce the number of datafiles to be managed.

- Bigfile tablespaces can only be locally managed with automatic segment space management except for locally managed undo tablespaces, temporary tablespaces, and the SYSTEM tablespace.
- If a Bigfile tablespace is used for automatic undo or temporary segments, the segment space management must be set to MANUAL.
- Bigfile tablespaces save space in the SGA and control file because fewer datafiles need to be tracked.
- ALTER TABLESPACE commands on a Bigfile tablespace do not reference a datafile because only one datafile is associated with each Bigfile tablespace.

Example – this example creates a Bigfile tablespace named **Graph01** (to store data that is graphical in nature and that consumes a lot of space). Note use of the **BIGFILE** keyword.

```
CREATE BIGFILE TABLESPACE graph01

 DATAFILE '/u03/student/dbockstd/oradata/USER350graph01.dbf'
SIZE 10g;
```

- Example continued: This resizes the Bigfile tablespace to increase the capacity from 10 gigabytes to 40 gigabytes.

```
SQL> ALTER TABLESPACE graph01 40g;

Tablespace altered.
```

- Example continued: This sets the AUTOEXTEND option on to enable the tablespace to extend in size 10 gigabytes at a time.

```
SQL> ALTER TABLESPACE graph01 AUTOEXTEND ON NEXT 10g;

Tablespace altered.
```

Notice in the above two examples that there was no need to refer to the datafile by name since the Bigfile tablespace has only a single datafile.

## Read Only Tablespaces

A tablespace may be made **read only**. One purpose for this action is to enable system maintenance that involves dropping tables and associated indexes stored in the tablespace. This can be accomplished while a tablespace is in read only mode because the **DROP** command affects only information in the Data Dictionary which is in the SYSTEM tablespace, and the SYSTEM tablespace is not read only.

The command to make a tablespace read only is:

```
ALTER TABLESPACE tablespace_name READ ONLY;
```

This also causes an automatic checkpoint of the tablespace.

If the tablespace being modified is locally managed, the segments that are associated with the dropped tables and index are changed to temporary segments so that the bitmap is not updated.

To change a tablespace from read only to read/write, all datafiles for the tablespace must be online.

```
ALTER TABLESPACE tablespace_name READ WRITE;
```

Another reason for making a tablespace read only is to support the movement of the data to read only media such as CD-ROM. This type of change would probably be permanent. This approach is sometimes used for the storage of large quantities of static data that doesn't change. This also eliminates the need to perform system backups of the read only tablespaces. To move the datafiles to a read only media, first alter the tablespaces as read only, then rename the datafiles to the new location by using the **ALTER TABLESPACE RENAME DATAFILE** option..

# Offline Tablespaces

Most tablespaces are **online** all of the time; however, a DBA can take a tablespace **offline**. This enables part of the database to be available – the tablespaces that are online – while enabling maintenance on the offline tablespace. Typical activities include:

- Offline tablespace backup – a tablespace can be backed up while online, but offline backup is faster.

- Recover an individual tablespace or datafile.

- Move a datafile without closing the database.

You cannot use SQL to reference offline tablespaces – this simply generates a system error. Additionally, the action of taking a tablespace offline/online is always recorded in the data dictionary and control file(s). Tablespaces that are offline when you shutdown a database are offline when the database is again opened.

The commands to take a tablespace offline and online are simple ALTER TABLESPACE commands. These also take the associated datafiles offline.

```
ALTER TABLESPACE application_data OFFLINE;

ALTER TABLESPACE application_data ONLINE;
```

The full syntax is:

```
ALTER TABLESPACE tablespace

{ONLINE |OFFLINE [NORMAL|TEMPORARY|IMMEDIATE|FOR RECOVER]}
```

**NORMAL**: All data blocks for all datafiles that form the tablespace are written from the SGA to the datafiles. A tablespace that is offline NORMAL does not require any type of recovery when it is brought back online.

**TEMPORARY**: A checkpoint is performed for all datafiles in the tablespace. Any offline files may require media recovery.

**IMMEDIATE**: A checkpoint is **NOT** performed. Media recovery on the tablespace is required before it is brought back online to synchronize the database objects.

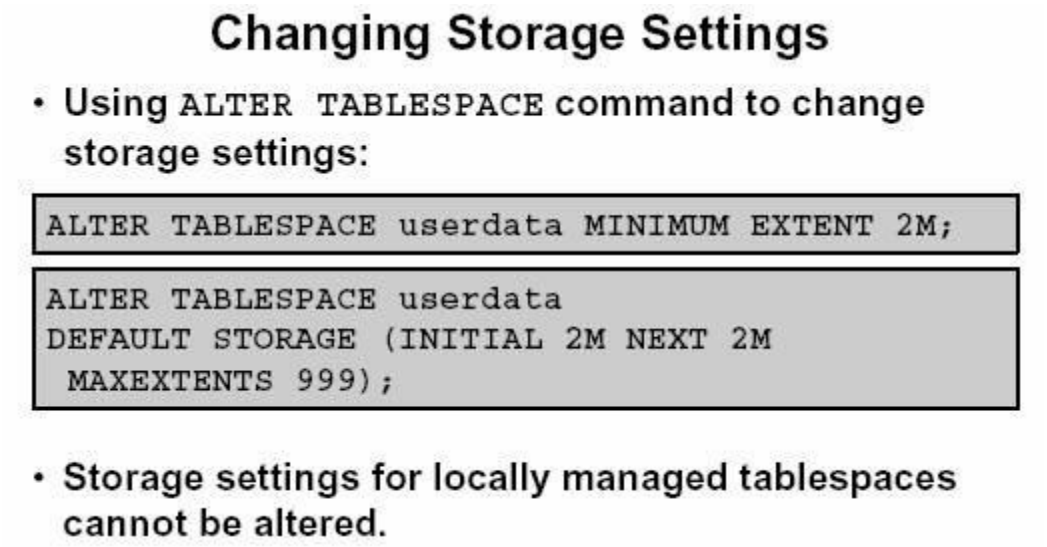
**FOR RECOVER**: Used to place a tablespace in offline status to enable point-in-time recovery.

**Errors and Restrictions**:

- If **DBWn** fails to write to a datafile after several attempts, Oracle will automatically take the associated tablespace offline – the DBA will then recover the datafile.
- The SYSTEM tablespace cannot be taken offline.
- Tablespaces with active undo segments or temporary segments.

# Tablespace Storage Settings

Any of the storage settings for **Dictionary-Managed** tablespaces can be modified with the **ALTER TABLESPACE** command. This only alters the default settings for future segment allocations.




# Tablespace Sizing

Normally over time tablespaces need to have additional space allocated. This can be accomplished by setting the **AUTOEXTEND** option to enable a tablespace to increase automatically in size.

- This can be dangerous if a "runaway" process or application generates data and consumes all available storage space.
- An advantage is that applications will not ABEND because a tablespace runs out of storage capacity.
- This can be accomplished when the tablespace is initially created or by using the **ALTER TABLESPACE** command at a later time.

```
CREATE TABLESPACE application_data

  DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf'
SIZE 200M

  AUTOEXTEND ON NEXT 48K MAXSIZE 500M;
```

This query uses the **DBA_DATA_FILES** view to determine if **AUTOEXTEND** is enabled for selected tablespaces in the SIUE Oracle database.

```
SELECT tablespace_name, autoextensible

FROM dba_data_files;
```

```
TABLESPACE_NAME                AUT

------------------------------ ---

SYSTEM                         YES

DRSYS                          NO

DATA                           NO

DATA_INDEX                     NO

USERS                          NO

CONSTANT_GROW_INDEXES          NO

UNDO1                          NO

CONSTANT_GROW_TABLES           NO

DEPENDENCY_INDEXES             NO

DEPENDENCY_TABLES              NO

OEM_REPOSITORY                 YES
```

- Manually use the **ALTER DATABASE** command to resize a datafile.

```
ALTER DATABASE
```

```
  DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf'
  AUTOEXTEND ON MAXSIZE 600M;
```

This command looks similar to the above command, but this one **resizes** a datafile while the above command sets the **maxsize** of the datafile.

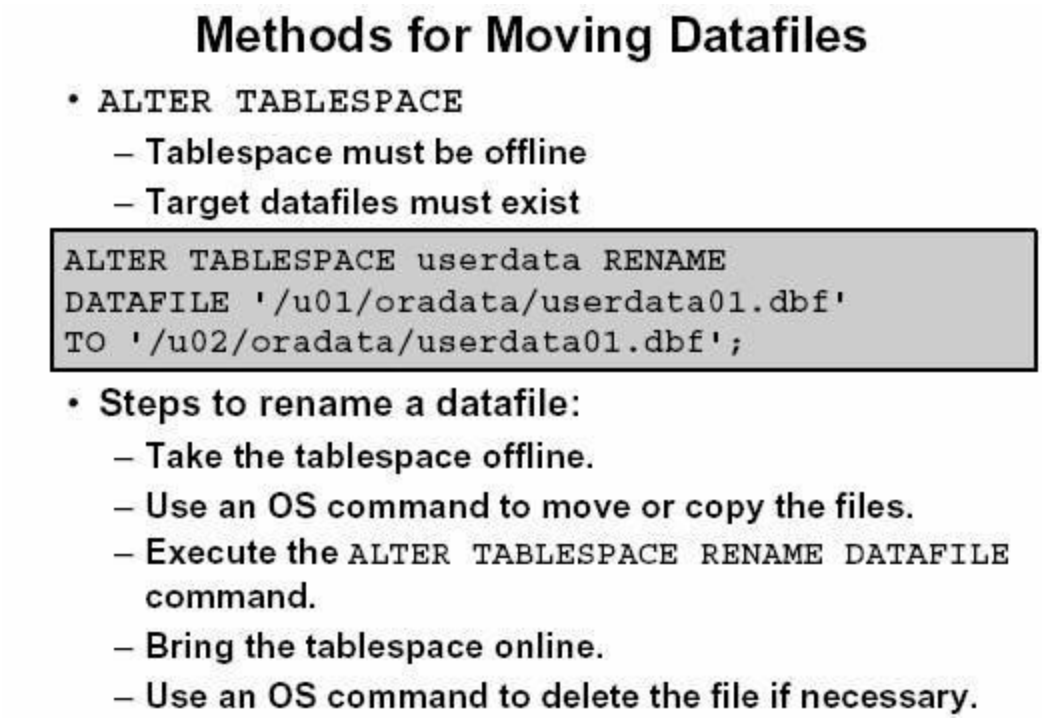
```
ALTER DATABASE
  DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf'
  RESIZE 600M;
```

- Add a new datafile to a tablespace with the **ALTER TABLESPACE** command.

```
ALTER TABLESPACE application_data
  ADD DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf'
  SIZE 200M;
```

# Moving/Relocating Tablespaces/Datafiles

The ALTER TABLESPACE command can be used to move datafiles by renaming them. This cannot be used if the tablespace is the SYSTEM or contains active undo or temporary segments.




The **ALTER DATABASE** command can also be used with the **RENAME** option. This is the method that must be used to move the SYSTEM tablespace because it cannot be taken offline. The steps are:

1. Shut down the database.
2. Use an operating system command to move the files.
3. Mount the database.
4. Execute the **ALTER DATABASE RENAME FILE** command.

```
ALTER DATABASE RENAME
  FILE '/u01/student/dbockstd/oradata/USER350data01.dbf'
    TO '/u02/student/dbockstd/oradata/USER350data01.dbf'
  SIZE 200M;
```

5. Open the database.

## Dropping Tablespaces

Occasionally tablespaces are dropped due to database reorganization. A tablespace that contains data cannot be dropped unless the **INCLUDING CONTENTS** clause is added to the **DROP** command. Since tablespaces will almost always contain data, this clause is almost always used.

A DBA cannot drop the SYSTEM tablespace or any tablespace with active segments. Normally you should take a tablespace offline to ensure no active transactions are being processed. An example command set is:

```
ALTER TABLESPACE application_data OFFLINE;
```

```
DROP TABLESPACE application_data
  INCLUDING CONTENTS AND DATAFILES
  CASCADE CONSTRAINTS;
```

The **AND DATAFILES** clause causes the datafiles to also be deleted. Otherwise, the tablespace is removed from the database as a logical unit, and the datafiles must be deleted with operating system commands.

The **CASCADE CONSTRAINTS** clause drops all referential integrity constraints where objects in one tablespace are constrained/related to objects in another tablespace.

# Non-Standard Block Sizes

It may be advantageous to create a tablespace with a nonstandard block size in order to import data efficiently from another database. This also enables transporting tablespaces with unlike block sizes between databases.

- A block size is nonstandard if it differs from the size specified by the **DB_BLOCK_SIZE** initialization parameter.
- The **BLOCKSIZE** clause of the **CREATE TABLESPACE** statement is used to specify nonstandard block sizes.
- In order for this to work, you must have already set **DB_CACHE_SIZE** and at least one **DB_nK_CACHE_SIZE** initialization parameter values to correspond to the nonstandard block size to be used.
- The **DB_nK_CACHE_SIZE** initialization parameters that can be used are:
  - **DB_2K_CACHE_SIZE**
  - **DB_4K_CACHE_SIZE**
  - **DB_8K_CACHE_SIZE**
  - **DB_16K_CACHE_SIZE**
  - **DB_32_CACHE_SIZE**
- Note that the **DB_nK_CACHE_SIZE** parameter corresponding to the standard block size cannot be used – it will be invalid – instead use the **DB_CACHE_SIZE** parameter for the standard block size.

Example – these parameters specify a standard block size of **8K** with a cache for standard block size buffers of **12M**. The **2K** and **16K** caches will be configured with cache buffers of **8M** each.

```
DB_BLOCK_SIZE=8192

DB_CACHE_SIZE=12M

DB_2K_CACHE_SIZE=8M

DB_16K_CACHE_SIZE=8M
```

Example – this creates a tablespace with a blocksize of **2K** (assume the standard block size for the database was 8K).

```
CREATE TABLESPACE inventory

  DATAFILE '/u01/student/dbockstd/oradata/USER350data01.dbf'

      SIZE 50M

  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 128K

  BLOCKSIZE 2K;
```

# Managing Tablespaces with Oracle Managed Files

As you learned earlier, when you use an OMF approach, the **DB_CREATE_FILE_DEST** parameter in the parameter file specifies that datafiles are to be created and defines their location. The **DATAFILE** clause to name files is not used because filenames are automatically generated by the Oracle Server, for example, **ora_tbs1_2xfh990x.dbf**.

You can also use the **ALTER SYSTEM** command to dynamically set this parameter in the SPFILE parameter file.

```
ALTER SYSTEM SET
  DB_CREATE_FILE_DEST = '/u02/student/dbockstd/oradata';
```

Additional tablespaces are specified with the **CREATE TABLESPACE** command shown here that specifies not the datafile name, but the datafile size. You can also add datafiles with the **ALTER TABLESPACE** command.

```
CREATE TABLESPACE application_data DATAFILE SIZE 100M;
```

```
ALTER TABLESPACE application_data ADD DATAFILE;
```

Setting the **DB_CREATE_ONLINE_LOG_DEST_n** parameter prevents log files and control files from being located with datafiles – this will reduce I/O contention.

When OMF tablespaces are dropped, their associated datafiles are also deleted at the operating system level.

# Tablespace Information in the Data Dictionary

The following data dictionary views can be queried to display information about tablespaces.

- Tablespaces: DBA_TABLESPACES, V$TABLESPACE
- Datafiles: DBA_DATA_FILES, V$_DATAFILE
- Temp files: DBA_TEMP_FILES, V$TEMPFILE

You should examine these views in order to familiarize yourself with the information stored in them.

This is an example query that will display free and used space for each tablespace in a database.

```
SELECT /* + RULE */  df.tablespace_name "Tablespace",

       df.bytes / (1024 * 1024) "Size (MB)",

       SUM(fs.bytes) / (1024 * 1024) "Free (MB)",

       Nvl(Round(SUM(fs.bytes) * 100 / df.bytes),1) "% Free",

       Round((df.bytes - SUM(fs.bytes)) * 100 / df.bytes) "%
Used"

  FROM dba_free_space fs,

       (SELECT tablespace_name,SUM(bytes) bytes

          FROM dba_data_files

         GROUP BY tablespace_name) df

 WHERE fs.tablespace_name (+)  = df.tablespace_name

 GROUP BY df.tablespace_name,df.bytes

UNION ALL

SELECT /* + RULE */ df.tablespace_name tspace,

       fs.bytes / (1024 * 1024),

       SUM(df.bytes_free) / (1024 * 1024),

       Nvl(Round((SUM(fs.bytes) - df.bytes_used) * 100 /
fs.bytes), 1),
```

```
       Round((SUM(fs.bytes) - df.bytes_free) * 100 / fs.bytes)
  FROM dba_temp_files fs,
       (SELECT tablespace_name,bytes_free,bytes_used
          FROM v$temp_space_header
         GROUP BY tablespace_name,bytes_free,bytes_used) df
 WHERE fs.tablespace_name (+)  = df.tablespace_name
 GROUP BY
df.tablespace_name,fs.bytes,df.bytes_free,df.bytes_used
 ORDER BY 4 DESC;
```

This shows output for the **DBORCL** database located on the **SOBORA2** server.

```
Tablespace         Size (MB)  Free (MB)     % Free     % Used
---------------- ---------- ---------- ---------- ----------
UNDOTBS1                435    420.875         97          3
USERS                  22.5      .9375          4         96
SYSAUX                  340     10.875          3         97
SYSTEM                  480     6.6875          1         99
TEMP                     28          0          0        100
```

---

1.

```
SELECT tablespace_name,SUM(bytes) bytes
          FROM dba_data_files
        GROUP BY tablespace_name
```

```
SELECT /* + RULE */  df.tablespace_name "Tablespace",
       df.bytes / (1024 * 1024) "Size (MB)",
       SUM(fs.bytes) / (1024 * 1024) "Free (MB)",
       Nvl(Round(SUM(fs.bytes) * 100 / df.bytes),1) "% Free",
       Round((df.bytes - SUM(fs.bytes)) * 100 / df.bytes) "% Used"
  FROM dba_free_space fs,
       (SELECT tablespace_name,SUM(bytes) bytes
          FROM dba_data_files
         GROUP BY tablespace_name) df
 WHERE fs.tablespace_name (+)  = df.tablespace_name
 GROUP BY df.tablespace_name,df.bytes
UNION ALL
SELECT /* + RULE */ df.tablespace_name tspace,
       fs.bytes / (1024 * 1024),
       SUM(df.bytes_free) / (1024 * 1024),
       Nvl(Round((SUM(fs.bytes) - df.bytes_used) * 100 / fs.bytes), 1),
       Round((SUM(fs.bytes) - df.bytes_free) * 100 / fs.bytes)
  FROM dba_temp_files fs,
       (SELECT tablespace_name,bytes_free,bytes_used
          FROM v$temp_space_header
         GROUP BY tablespace_name,bytes_free,bytes_used) df
 WHERE fs.tablespace_name (+)  = df.tablespace_name
 GROUP BY df.tablespace_name,fs.bytes,df.bytes_free,df.bytes_used
 ORDER BY 4 DESC;
```

END OF NOTES