
ORACLE
SOFTWARE POWERS THE INTERNET™
ORACLE

Frequently Asked Questions

A Smörgasbord of Common Questions and Problems Received by World Wide Support—And How to Resolve Them

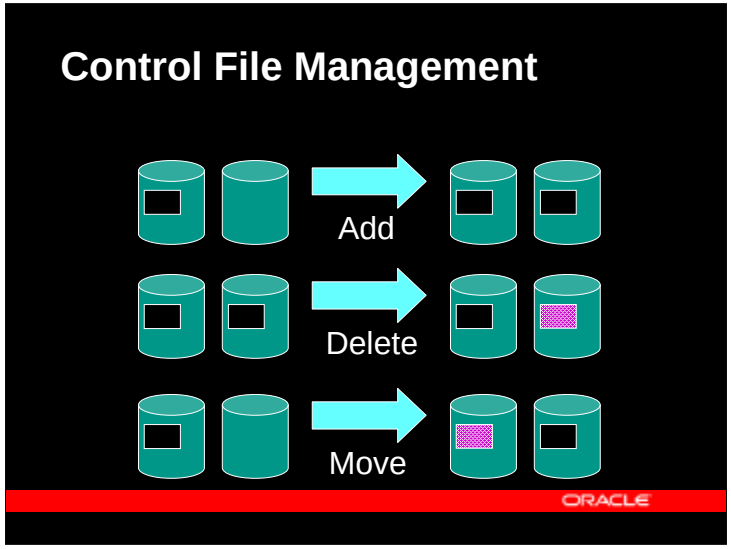
Prepared by David Austin, david.austin@oracle.com

ORACLE

Overview

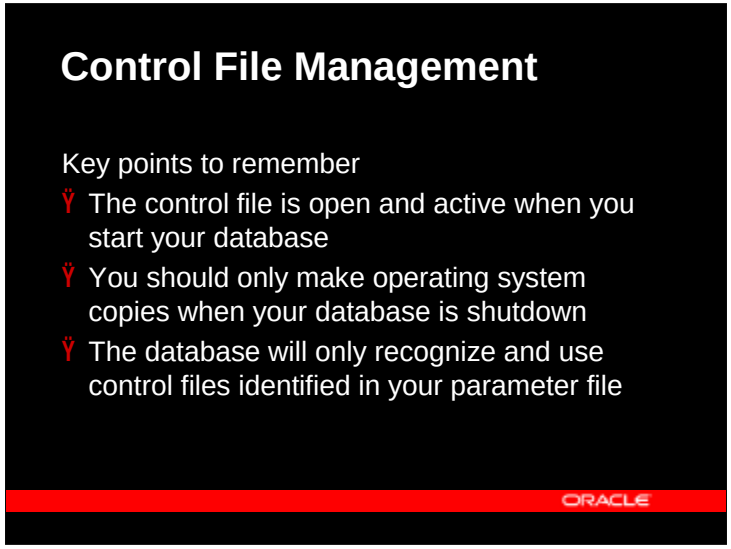
- Variety of topics in no particular order
- Feel free to add your own comments and observations
- Questions happily accepted, even on unrelated topics—but answers, correct or otherwise, are not guaranteed

ORACLE



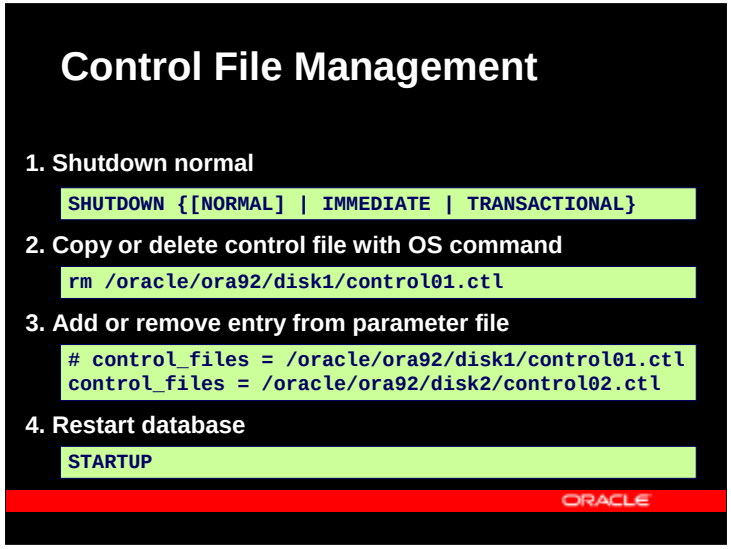
There are various reasons why you may need to manipulate your database control files:

- Add a new one if you have only one
- Delete one that has become damaged
- Replace a lost or damaged copy
- Move the file to a faster, more reliable, or less busy disk system



The control file is an integral part of your database and is modified almost continuously while the database is active. You must use one of the supported Oracle tools to make copies or backups of the control file when your database is mounted or open. Before making an operating system backup, you have to shut down your database.

Remember that your database will only recognize the control files you identify in your initialization parameter file, so you have to restart the database if you wish to change the control file designations.



Step 1

Shut down normally using one of the options listed. Do not attempt to create a new, or move an existing, control file if your database terminates abnormally or if you shut it down with the `ABORT` option.

Step 2

To create an additional control file, use the operating system (OS) copy command

To drop an unwanted file, use the appropriate OS command. Note that if you delete the reference to a copy of the control file from your parameter file, the file will not be used again but will be retained on disk.

To move a control file, first make a copy of the file, just as you do to make an additional copy, and then remove the original file.

Step 3

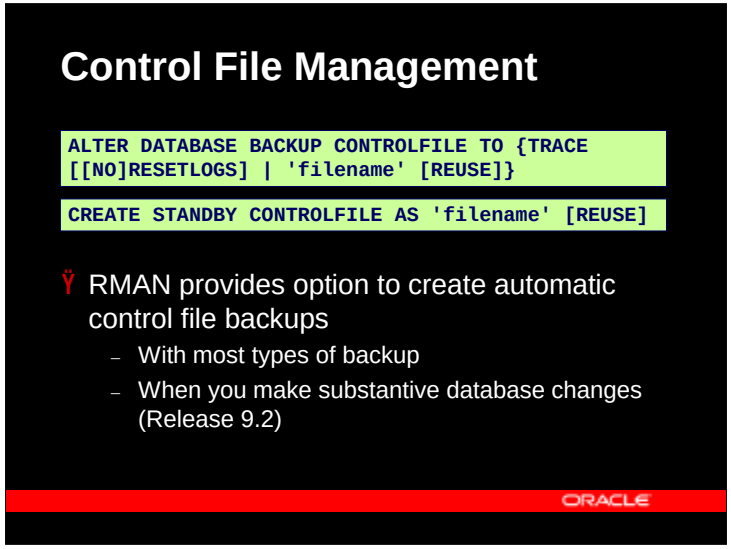
If you are using a text parameter file, you need to edit the `control_files` parameter(s) before restarting your database. If you are using a system parameter file (SPFILE), you will need to start an instance with a temporary text parameter file that includes only the valid control files. Using this instance, you can change the SPFILE entry with a command such as

```
ALTER SYSTEM SET
CONTROL_FILES = '/oracle/ora92/disk1/control02.ctl'
SCOPE = SPFILE
```

Alternatively, execute this command before you perform the shutdown in Step 1. In this case, you can restart your instance with your SPFILE because the required change will already have been made.

Step 4

Ensure your database restart uses a parameter file containing the correct entry (or entries) for the `CONTROL_FILES` parameter. If a deleted copy, or the original name and location of a moved copy, is referenced, the startup will fail with a missing control

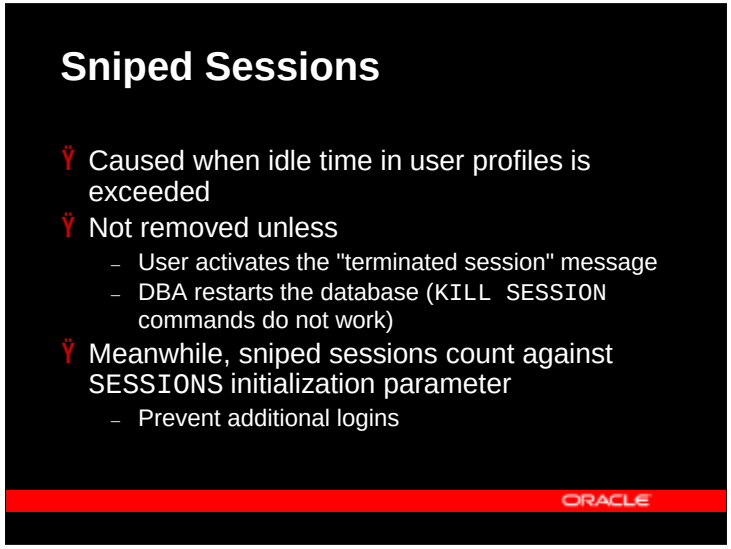


To create a script version of the `CREATE CONTROLFILE` command using your current configuration, use the `ALTER DATABASE BACKUP CONTROLFILE TO TRACE` command. By default, the script will contain a `NORESETLOGS` version of the database start command. You can include the `RESETLOGS` option to override this.

To complete an online, user-managed backup, create a binary copy of the control file with the `ALTER DATABASE BACKUP CONTROLFILE TO` *`filename`* command, providing the full path name of the operating system file in single quotes. You must include the `REUSE` key word if the file already exists to permit Oracle to overwrite it.

To create a control file for a standby database, use the `CREATE STANDBY CONTROLFILE` as shown. As with the binary copy command, you must include `REUSE` if the file already exists.

Most common types of database, tablespace, and data file backups made with RMAN include a copy of the control file automatically if you set the CONTROLFILE AUTOBACKUP configuration option. You can also manually add control file backups as you make individual backups. Beginning with Oracle9*i* Release 2, the CONTROLFILE AUTOBACKUP option also causes a backup of the control file to be created when you use DDL commands to alter your database structure, for example, by adding a tablespace or a new data file to an existing tablespace.

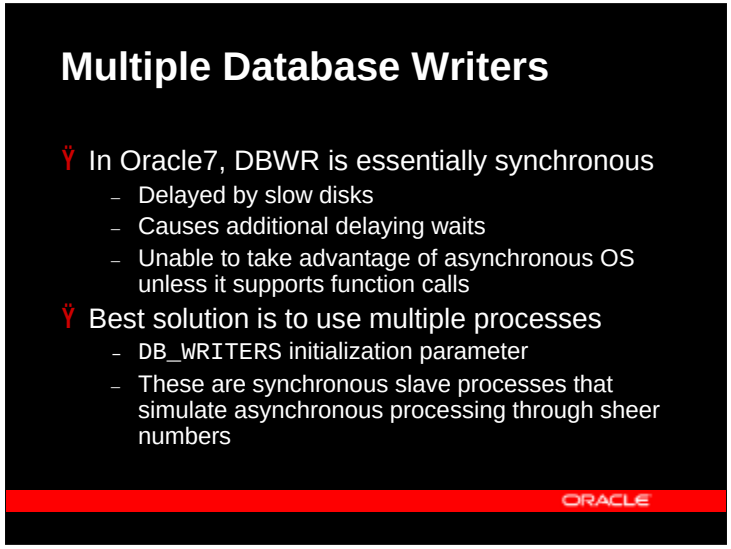


You may notice sessions with a status of `SNIPED` in `V$SESSION`. These are sessions that have timed out due to values for the initialization parameters that control user profiles. In particular, your `resource_limit` is set to `TRUE` and your `idle_time` parameter is non-zero. When a session is idle beyond the limit you have set, it attempts to inform the user that the session is terminated. However, it will not do so until the user makes a further attempt to use the session (by entering a command or just pressing the `return` key). Until such a time, the session remains stuck with the sniped status. If the users whose sessions caused the status are no longer connected, due to a network timeout, for example, they will not be able to transmit a message to the server and hence terminate their sessions.

Should a sufficiently high number of processes attain a sniped status, they may prevent other users from logging on. This is because they are included in the total number of active sessions which are limited by the value, derived or explicit, of the `SESSIONS` initialization parameter. There is no command available within Oracle to terminate sniped sessions, other than to terminate the problem instance.

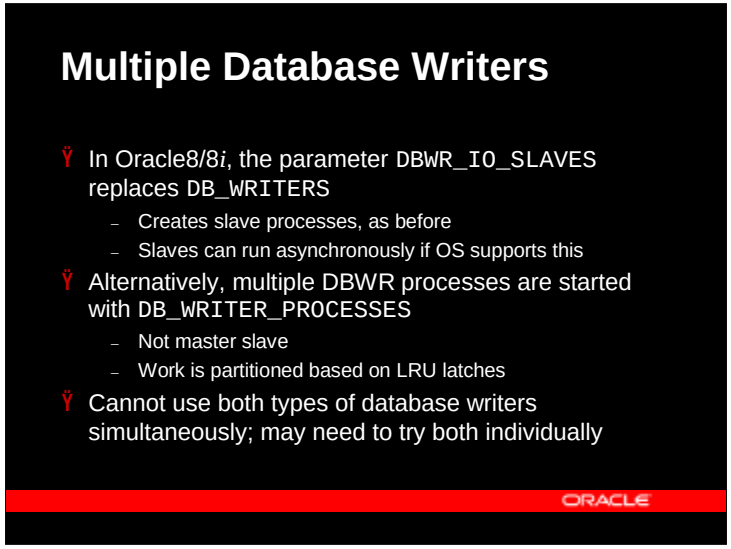
If you do not need to terminate idle sessions, the simplest solution is to reset the related initialization parameters, mentioned previously, to avoid idle time terminations. However, if this is a requirement, for security reasons, for example, then you may need to use the UNIX script shown above to remove sniped sessions from your system. This procedure has been tested against Solaris, AIX, Tru64, and HPUX and is discussed further in Note 1061189.6.

Note: Do not use this script except for dedicated server processes. Occasionally, you may see shared server shadow processes with a sniped status—these cannot be stopped safely except through instance shutdown.



Being only a single, non-threaded, process, the Oracle7 DBWR performs all its activity sequentially, that is, in synchronous mode. This includes identifying the buffers to write, obtaining the required buffer cache latches, sending the write request to the OS, waiting for an acknowledgment, then releasing the latches. If any type of slowdown occurs, for example, conflict for a buffer latch or a busy disk controller or drive, DBWR must wait. If it has already acquired latches, these will continue to be unavailable for other processes, such as a user server process that needs to move a block into one of the buffers being cleaned. Consequently, any delay in the buffer writing activity may cause other work to be queued.

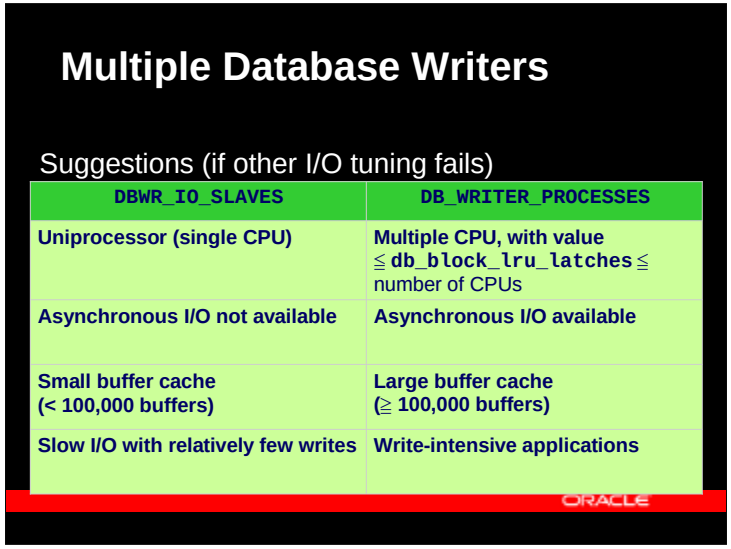
If your OS does not support an asynchronous I/O method that DBWR can use, and if you identify slow I/O by DBWR as a cause of slow processing overall, you can run multiple DBWR processes. You do this by increasing the value of the `DB_WRITERS` initialization parameter in Oracle7. This causes your instance to start that number of special DBWR processes that are controlled by the normal DBWR process. The normal DBWR is still responsible for managing the buffer cache writes, but it parcels out the actual write requests to these special processes so they can each perform disk writes in parallel.



With Oracle8, and continuing into Oracle8*i* and Oracle9*i*, the master/slave option provided with Oracle7 is still an option for you, but the parameter name you need to adjust is changed from `DB_WRITERS` to `DBWR_IO_SLAVES`. In this version, these slave processes are able to take advantage of OS asynchronous operations.

You have another option available in the Oracle8/8*i* releases: multiple DBWR processes that each perform all of the duties of a normal DBWR process. In other words, each process builds its own list of buffers, obtains the latches, performs the writes, and cleans up the latches after the write is acknowledged. The main benefit of this approach is that each process only ties up a subset of the resources, such as buffer latches, and should be able to release them sooner because it has fewer blocks to write. Consequently, the impact of DBWR activity on other processes is minimized. To use multiple, independent DBWR processes, set the initialization parameter `DB_WRITER_PROCESSES` to the number you want to start.

Note that you cannot use both types of DBWR processes concurrently and, on the next page, you will find some guidelines as to when each approach might be useful.

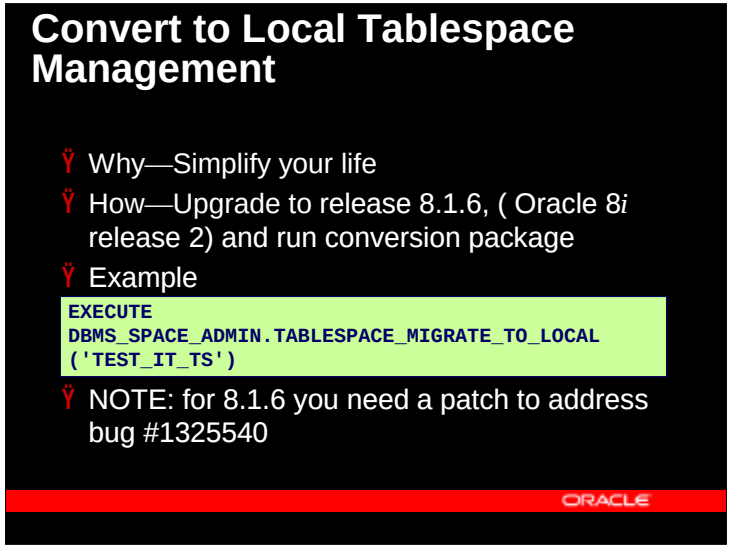


Using multiple DBWRs or slave writers creates additional overhead so only use them if other methods to improve I/O have failed. The table contains guidelines that generally hold true for when one type of processing works better, but these are not absolute. You should try both approaches, but not simultaneously, and measure the results to determine which is better, if either, for you.

Note that you are very unlikely to improve performance with `DB_WRITER_PROCESSES` if your database server only has a single CPU. This is because the processes divide their work based on the number of latches available on the buffer cache. With a single CPU, these latches themselves become available in a synchronous fashion, so the system has to wait for each DBWR to finish its work before another can start on its set of buffers—this reproduces the same behavior as a single DBWR with the additional overhead of running the extra processes.

Similarly, if the OS has to queue the write requests from each DBWR process because it does not support asynchronous I/O, multiple `DB_WRITER_PROCESSES` can, potentially, acquire latches on almost all of the buffer cache while waiting their turns to write blocks. This can degrade performance more than a single process that only tries to write a subset of the buffers at a time.

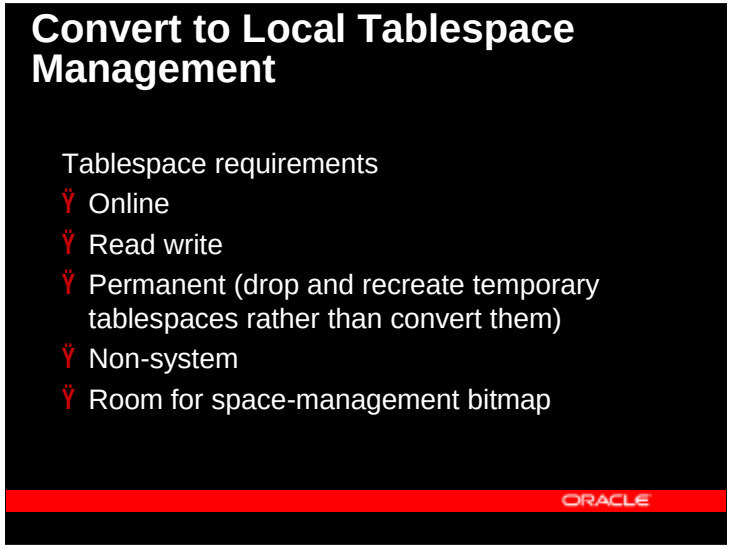
The buffer cache size, 100,000, included in the table, is only a guideline and does not take into account such factors as multiple caches (`KEEP`, `RECYCLE`, or `DB_nK_CACHE_SIZE`) nor the relative ratio of writes to other database activity, also mentioned in the table.



Beginning with release 8.1.5, you can create locally managed tablespaces. A locally managed tablespace is one where the space management information (basis for DBA_EXTENTS and DBA_FREE_SPACE) is stored in bitmaps, located in the datafiles, rather than in the data dictionary. In that release, however, you could not migrate existing tablespaces from dictionary to local management. Typically, existing databases that were migrated from Oracle8 still use the existing Dictionary managed space management model. One of the main reasons for this restriction was that, in 8.1.5, the space management code depends on the bitmap control block and the bitmap blocks themselves are at the start of the file. Thus migration is not possible because the first blocks of any file in an existing tablespace are occupied by existing disk objects. With release 8.1.6, (Oracle 8i release 2) the restriction is lifted and you may migrate existing tablespaces from dictionary managed to locally managed. The bitmap header is created in an available free extent, just like any other segment.

Due to some problems, related to Bug #1325540, Oracle8*i*, version 8.1.6, a number of Oracle customers have been reluctant to migrate to locally managed tablespaces. However, this is the direction in which Oracle is moving, as evidenced by the option to create a locally-managed database (SYSTEM and all other tablespaces), introduced in Oracle9*i*, release 2.

Customers who are using locally managed tablespaces, having applied the patch if they are using release 8.1.6, report excellent results. Free space management is no longer a problem for them unless they are short of disk space anyway, in which case, neither dictionary nor local management can help.



Before converting a tablespace from dictionary to local management with the `DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL` procedure, ensure your tablespace meets the requirements listed. If you create a new tablespace with local space management, you only need to ensure you have disk space for the related data file(s), of course.

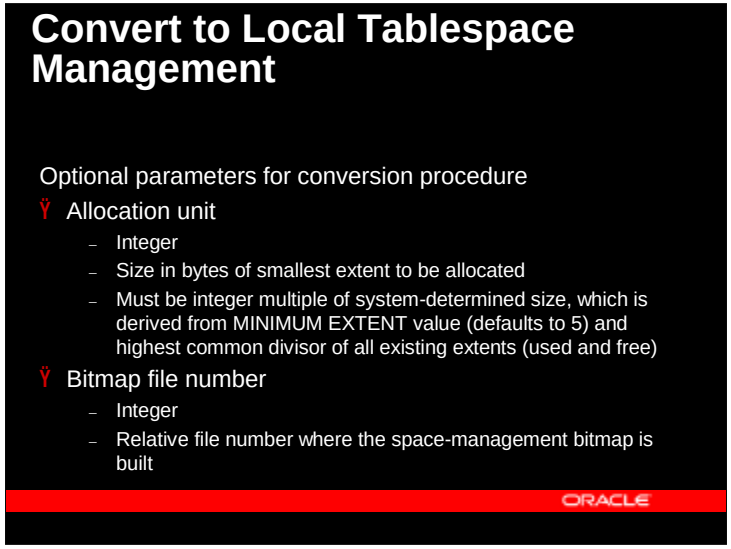
If you should encounter any problems, once you have converted a tablespace to local management, you can migrate it back to dictionary management with the procedure `DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_FROM_LOCAL`.

As noted above, if you wish to convert a temporary tablespace to local management, you cannot use the `DBMS_SPACE_ADMIN` package. However, you should be able to create a new temporary tablespace defined with local management. For example:

```
SQL> CREATE TEMPORARY TABLESPACE local_temp
  2  TEMPFILE 'file_lt1.f'
  3  EXTENT MANAGEMENT LOCAL UNIFORM SIZE 16M;
```

You can then reassign users from the dictionary managed temporary tablespace to your new tablespace, `local_temp`, in this example, and drop the original tablespace. If you are using Oracle9*i* and the original tablespace was defined as the default tablespace, you only need to issue the appropriate ALTER DATABASE command to reassign the affected users. For example:

```
SQL> ALTER DATABASE
  2  DEFAULT TEMPORARY TABLESPACE local_temp;
```

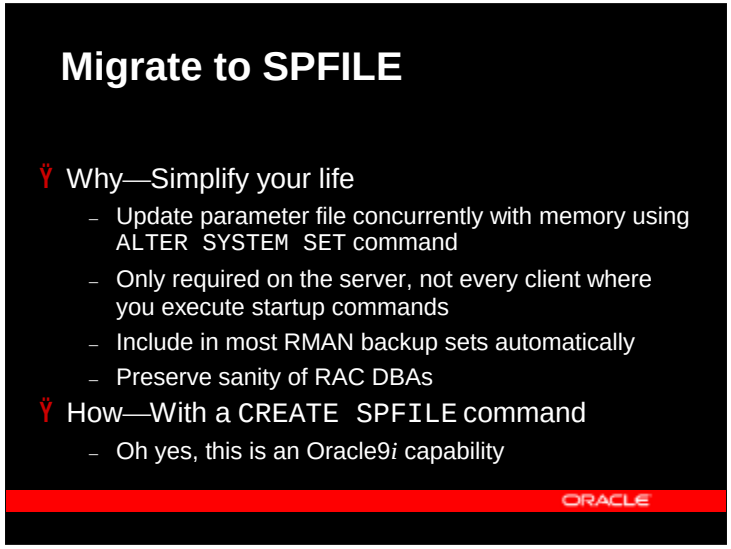


In the earlier example, the only value provided in argument list for the DBMS_SPACE_ADMIN.TABLESPACE_MIGRATE_TO_LOCAL procedure was the tablespace name, `test_it_ts`. This is a required parameter which you must include when you invoke the procedure. You may also provide values for two optional parameters, `unit_size` and `rfno`.

The `unit_size` parameter, also referred to as the "allocation unit," sets the size of the smallest possible chunk of space that can be allocated) in the tablespace. If you don't provide this value in your arguments, it defaults to the highest common divisor of all extents (used or free) for the tablespace, except the last free extent in each file. This number is further trimmed based on the `MINIMUM EXTENT` value for the tablespace (5 if `MINIMUM EXTENT` is not specified).

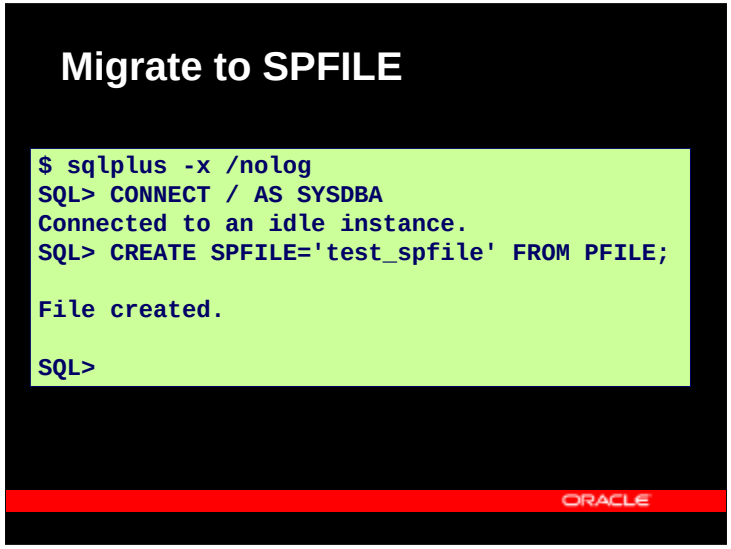
Note that if you specify the allocation unit, it has to be a factor of the UNIT size calculated by the system. Rather than attempting to compute a valid value, most customers choose to take the default value. If this results in unacceptable space usage, you can convert the tablespace back to dictionary management or build a new, locally managed tablespace and move the original contents into it.

The `rfno` parameter, which stands for the "relative file number," identifies the data file where the space management bitmaps will be created. This file must be must be part of the tablespace you are migrating and have sufficient room for the initial bitmap. The size of the bitmap is determined by the allocation unit value discussed previously. If you do not specify a value for the relative file number, then the system will choose a datafile in which to place the initial bitmap blocks.



You can think of a server parameter file (SPFILE) as a repository for your initialization parameters that is maintained on the machine where your Oracle database server executes. It is, by design, a server-side initialization parameter file. Initialization parameters stored in a server parameter file are persistent, in that any changes you make to the parameters while an instance is running may be defined to persist across instance shutdown and startup. This eliminates the need for you to manually update the initialization parameters with your new values after you make changes with an ALTER SYSTEM command.

You can find an example of the CREATE SPFILE command, used to build an SPFILE from an existing text-based parameter file, on the next page.



You can find descriptions of the key components of this session on the following pages.

Migrate to SPFILE

```
$ sqlplus -x /nolog
SQL> CONNECT / AS SYSDBA
Connected to an idle instance.
SQL> CREATE SPFILE='test_spfile' FROM PFILE;

File created.

SQL>
```

FYI
-x suppresses the SQL*Plus headings
/nolog avoids login prompts

ORACLE

Migrate to SPFILE

```
$ sqlplus -x /nolog
SQL> CONNECT / AS SYSDBA
Connected to an idle instance.
SQL> CREATE SPFILE='test_spfile' FROM PFILE;

File created.

SQL>
```

Can be SYSDBA or SYSOPER connection

ORACLE

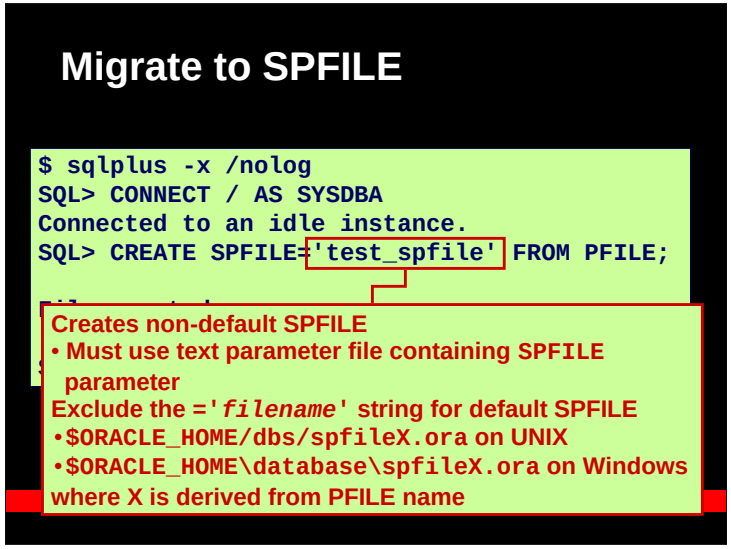
Migrate to SPFILE

```
$ sqlplus -x /nolog
SQL> CONNECT / AS SYSDBA
Connected to an idle instance.
SQL> CREATE SPFILE='test_spfile' FROM PFILE;

File created.

SQL>
```

Instance can be closed, mounted, or open



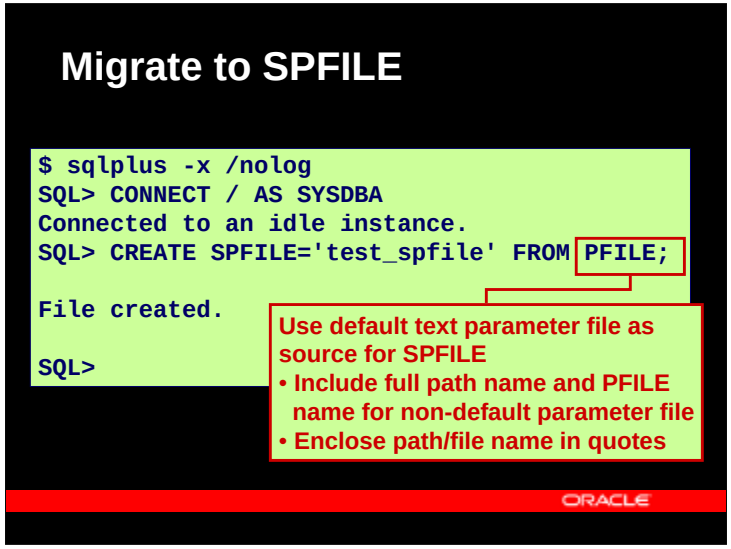
The command will create a default SPFILE if you do not include a string with the path or file name. The default names and locations differ by platform, with the values for UNIX and Windows shown in the graphic. If you want to use your own file definition, you must enclose the string in single quotes.

You can specify an SPFILE file in a text parameter file and this is the only way to start the instance with an SPFILE in a non-default location. For example, suppose you created an SPFILE as follows:

```
CREATE SPFILE='/database/startup/test_spfile' FROM PFILE;
```

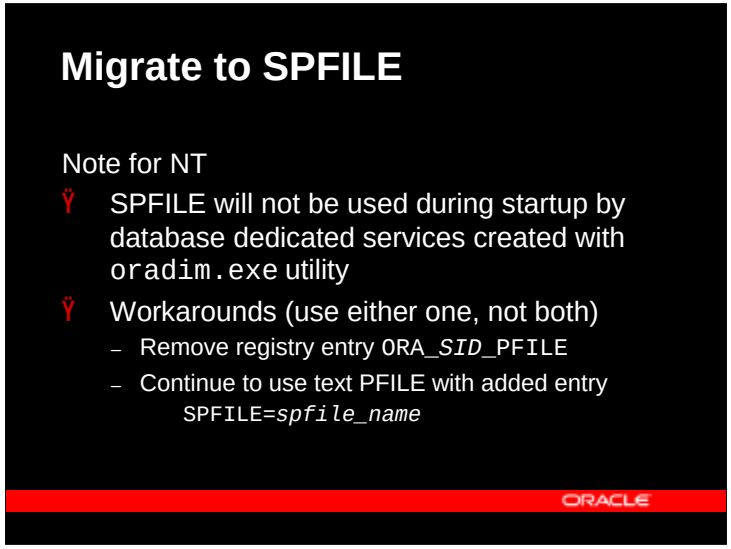
To use this SPFILE, you need to add this entry to the parameter file you use to start the instance:

```
SPFILE = /database/startup/ test_spfile
```



Your CREATE SPFILE command will create the SPFILE from the database's default text-based parameter file if you enter it as shown. If you wish to use a different parameter file as the source for your SPFILE, you need to include the file definition, using the same format as the `SPFILE` file string. Here is an example that creates a default SPFILE on a UNIX server from a text version in a personal directory on the server:

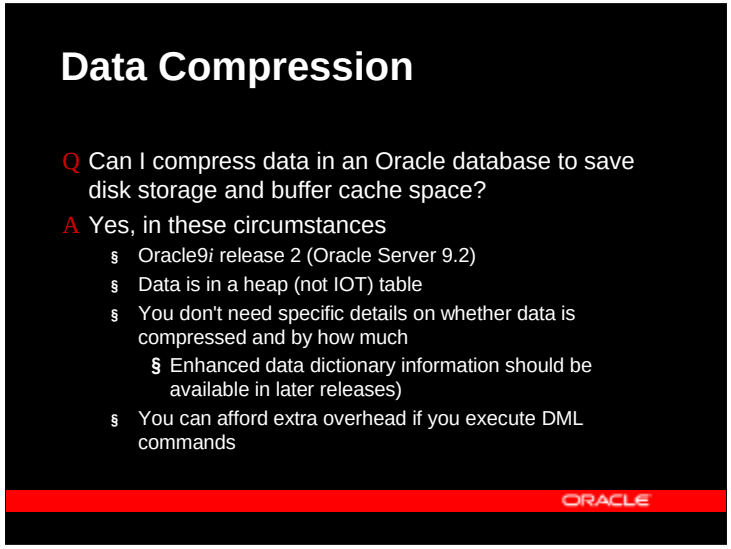
```
CREATE SPFILE FROM PFILE='/users/davida/orafiles/test_for_spfile.ora'
```



If you are running Oracle on NT, each of your databases requires a dedicated service. This service is created with the `oradim.exe` program, executed manually or automatically while running the Database Configuration Assistant to build a database. In Oracle9*i*, version 9.0.1, `oradim.exe` does not support SPFILEs. However, you can still use an SPFILE for a database by employing either one, but not both, of these workarounds:

•Remove the pfile entry, `ORA_<SID>_PFILE`, from the registry.

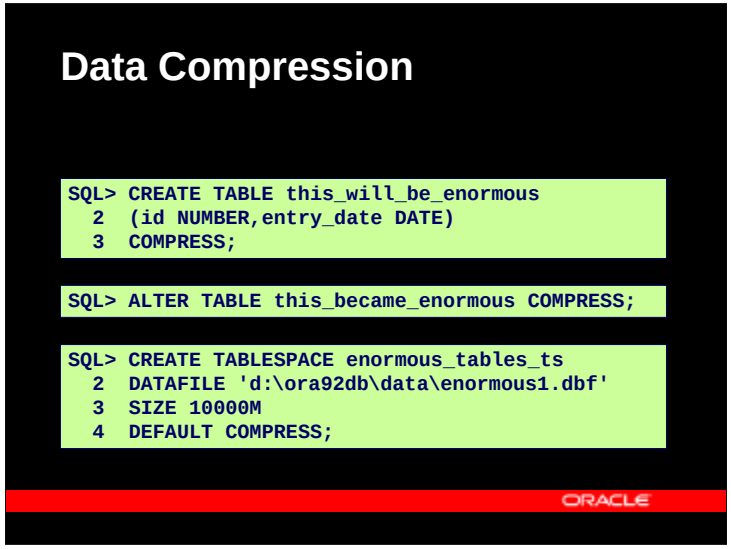
•Add an entry in your text-based parameter file to identify the required SPFILE, as described earlier to employ a non-default SPFILE.



The option for you to compress data in a database table is a new feature introduced in Oracle Server 9.2. The benefits of data compression include reduction in disk and buffer cache usage. This is very useful for read only operations because fewer blocks have to be transferred between disk and memory.

If you employ compression, expect more CPU cycles to be consumed when the data must be displayed in uncompressed form, for example, to satisfy a query. Also, if you execute DML against compressed data, you may experience some DML concurrency loss, depending on the nature of the original data. You are therefore advised to utilize this feature in a read only or DSS database.

Note that you can only compress heap tables—this feature does not work for index-organized tables or for external tables. Also, in this initial release (Release 9.2), you cannot determine the exact amount of space compressed in the segment nor whether or not a segment is compressed. Oracle Corporation expects to provide data dictionary views to see this information, plus other enhancements to this feature in upcoming releases.

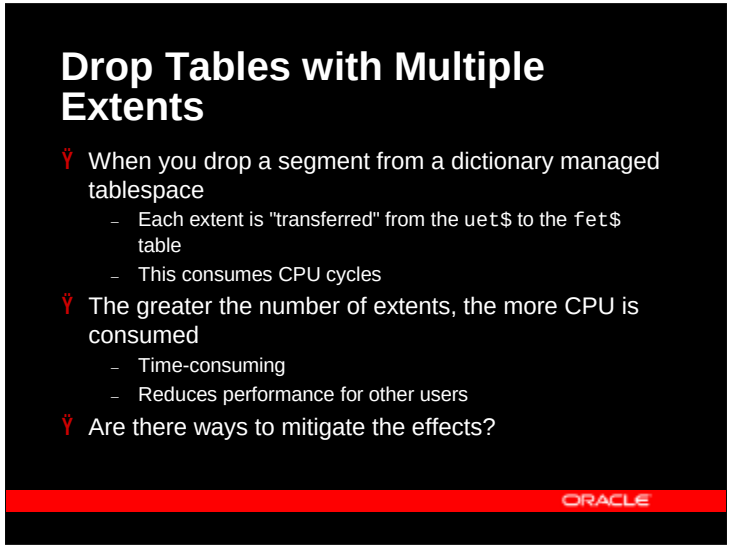


The examples show you three different ways to compress the data in a table.

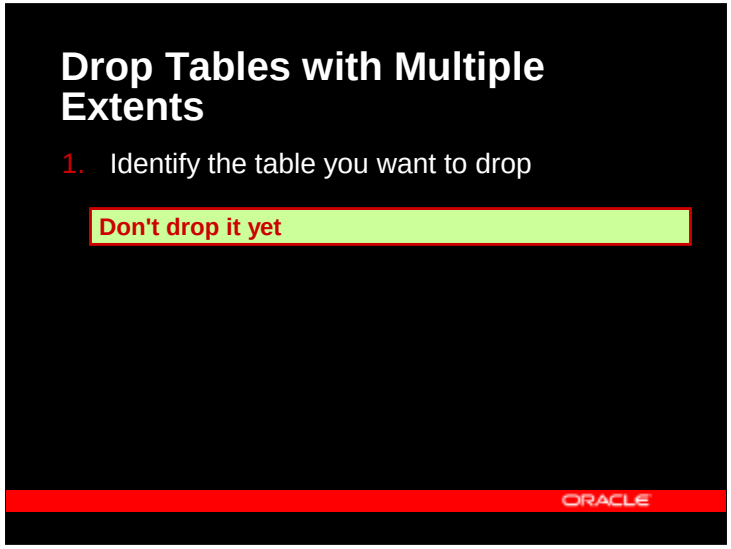
First, you can create the table with the COMPRESS keyword which causes all data to be compressed as you add it to the table. The first example shows the creation of a simple table with this characteristic.

The second example shows how you can modify an existing table to compress its current, and any future, records.

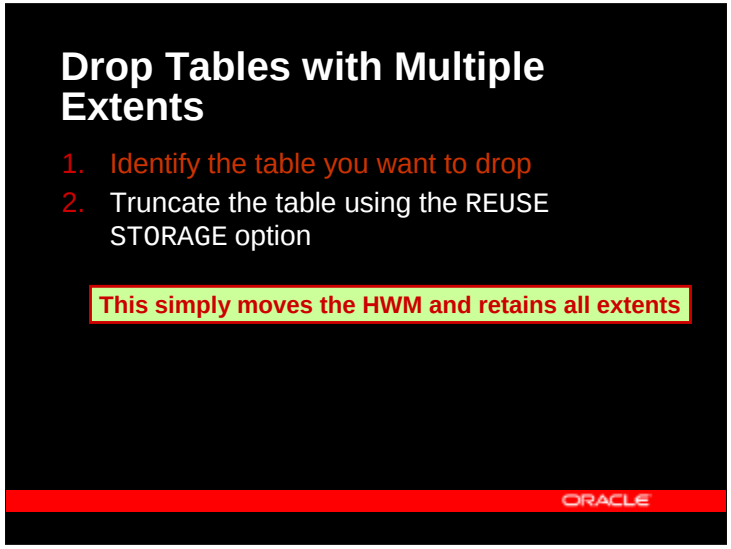
In the last example, the `enormous_tables_ts` tablespace definition contains the `DEFAULT COMPRESS` clause. This causes any heap table you create in the tablespace to be compressed without requiring you to include the `COMPRESS` keyword in your `CREATE TABLE` command.



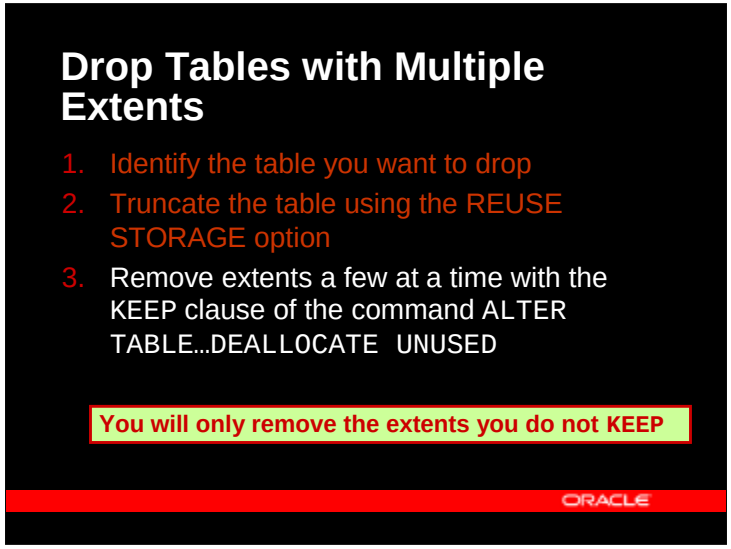
If you drop a table made up of many extents, your process will consume large amounts of CPU while performing the drop activity. This is an inescapable fact. However, with some forethought, you may be able to mitigate the effects of this CPU usage and its impact on other users of system resources. To do this, follow the four steps described on the next few pages.



The assumption is that you have discovered, from past experience with similarly-constructed tables, that dropping this table will degrade performance for an unacceptable length of time due to the large number of extents to be removed.



In this step, you simply move the high water mark of the table to the beginning of the segment, logically deleting all of the rows. By default, this command also drops all the extents except those created with the `MINEXTENTS` value in the original `STORAGE` clause. However, when you include the `REUSE STORAGE` clause, you prevent your process from dropping any of the extents so this statement incurs minimal overhead.



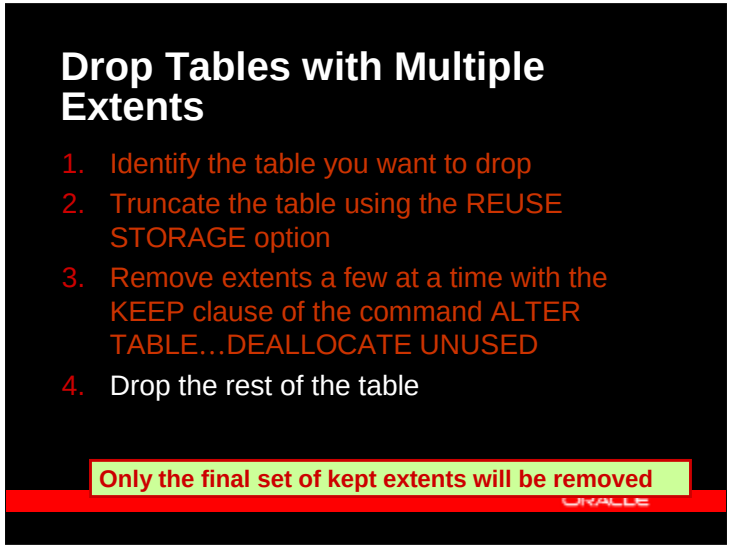
This step is at the heart of this process. The idea is to execute the `ALTER TABLE…DEALLOCATE UNUSED KEEP` command multiple times, using a value in the `KEEP` clause that will allow only a few extents to be dropped each time.

To use this approach, first determine how many extents you can drop before impacting overall system performance, for example, how many you can drop during a typical low-activity times. For the purpose of this illustration, assume this is 20 extents.

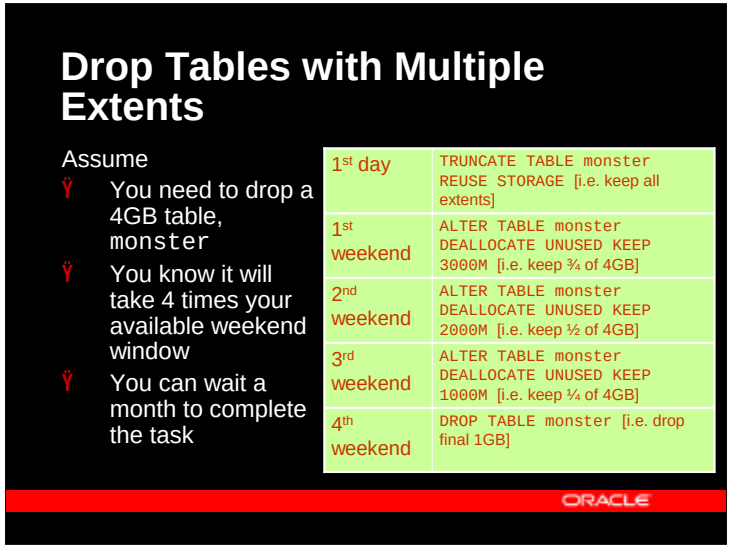
Now, using the `DBA_EXTENTS` view, sum the sizes of the 20 highest-numbered extents. This is the amount of space you want to drop with the next execution of the `ALTER TABLE…DEALLOCATE UNUSED KEEP` command. Let us say this works out to be 10 megabytes.

Subtract this amount, 10 megabytes, from the total size of the table (the sum of all extent sizes listed in `DBA_EXTENTS`. Use the resulting value for the value in the `KEEP` clause and execute the command.

When you are ready to drop more extents, repeat these calculations to obtain the required `KEEP` value. You may find that the total size of the extents you want to drop varies from time to time because they were created with a `PCTINCREASE` multiplier or because they were trimmed during a parallel load operation. Also, if length of time you can use to drop more extents increases or decreases, you obviously need to use a different value for the number of extents whose total size you compute.



Eventually, by repeating Step 3 over and over, you will reduce the table to the number of extents you can safely remove without impacting performance any more. At this point, instead of executing the `ALTER TABLE` command again when you next have an opportunity to drop the extents, you can finish the process with a normal `DROP TABLE` command. Your process will remove the remaining extents and also clean up the data dictionary records associated with the table.

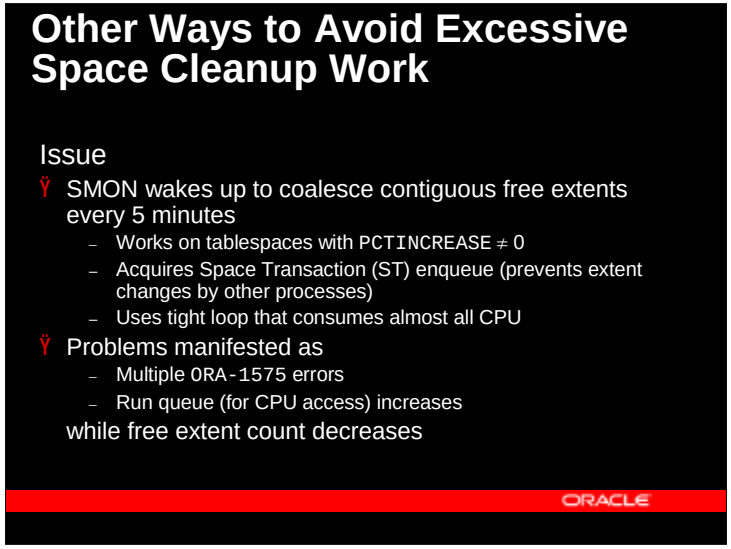


This example makes a number of assumptions, most of which were chosen to keep the arithmetic simple, about a table you need to drop:

- The table name is `monster`
- It contains 4,000 extents
- Each extent is 1MB
- You can drop up to 1,000 extents without impacting other users, during a low-activity period each weekend
- Although you don't want to keep the table, you do not need to free all of the space it is currently occupying for a few weeks

From this information, it should be obvious that you need to drop 1 GB of the table's space (one thousand of the 1MB extents) each weekend. Also, it is easy to determine that 1GB, or 1,000MB, is one quarter of the table's current size. So, following the steps described on the preceding pages, here is what you do.

1. Determine the table you need to drop. This is easy, it is `monster`.
2. Once you have done this, move the high water mark while keeping the extents already allocated. The graphic above provides the SQL command you use for this step in the "1st day" row.
3. Perform Step 3 for the next three weekends. On the 1st weekend, drop the first thousand extents by retaining the other three thousand extents, or 3,000MB. Note that, like other size-related values in SQL commands, provide the number 3000 without commas and use only the letter `M` for the megabyte multiplier. On the 2nd weekend, drop another one thousand extents, this time by retaining 2,000MB. Similarly, on the 3rd weekend, drop all but the final thousand extents by retaining 1,000MB. The specific statement for each weekend's operation is shown in the appropriate row of the graphic.
4. Perform this final step, dropping the table completely, on the 4th weekend. Again, the command is given in the graphic, this



When SMON performs free space coalescing, which it does every five minutes if it finds adjacent free extents, it may consume lots of CPU cycles if there many such extents. If you suspect this is occurring and causing problems, you may want to investigate further.

First, you should check if SMON is consuming large amounts of CPU for a long period. You can run the UNIX utilities `sar` or `vmstat` to see how busy your CPU (or CPUs) are; and use `ps` to find out which process is using the CPU.

Second, if you determine that that SMON is using lots of CPU, you need to determine whether it is performing temporary segment (extent) cleanup, or free space coalescing. Do this by checking if there are a large number of free extents that could be coalesced by running the following query a few times:

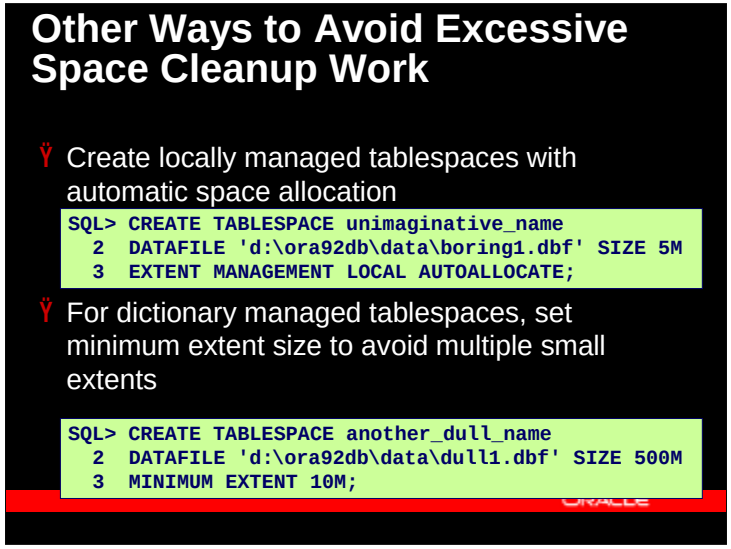
```
SELECT COUNT(*) FROM dba_free_space;
```

If the count returned decreases while SMON is working, it is likely that SMON is coalescing free space.

Third, check for the most common problems that this SMON activity can cause:

- Multiple `ORA-1575` errors. These occur because other processes cannot access the ST enqueue which SMON is using to manage the space activities.
- The run queue length increases. This will occur if the system is CPU-bound because SMON sits in a very tight loop while coalescing, and consumes close to 100% CPU.

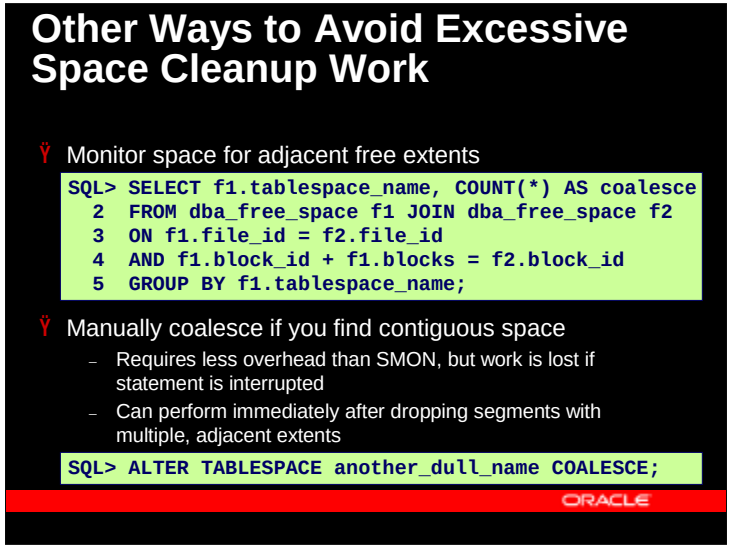
Even if SMON is coalescing multiple free extents, you do not need to do anything unless it is causing problems such as these.



If you are having problems due to SMON coalescing free extents, as described on the previous page, your best solution is to restructure your database to reduce the number of potential free extents. This will avoid the problem during future processing.

The two methods explained above both address the problem by reducing the number of extents needed by large segments. Thus, when the segments are dropped, or modified in some way to reduce the number of extents, they are less likely to leave adjacent free extents behind.

If, for some reason, you are not able to solve the problem using either of these approaches, you can help reduce the performance impact should you catch SMON performing extensive free space cleanup. See the next page for a suggested method on how to do this with commands run from a user process rather than relying on SMON to complete the work.



Note that the first statement may take a long time to execute. You can modify it to search in just one tablespace for adjacent free space, but this may also take a while to run in a large tablespace with many used and free extents. A more efficient query is

```
SELECT tablespace_name, block_id, block_id+blocks FROM dba_extents
ORDER BY file_id, block_id
```

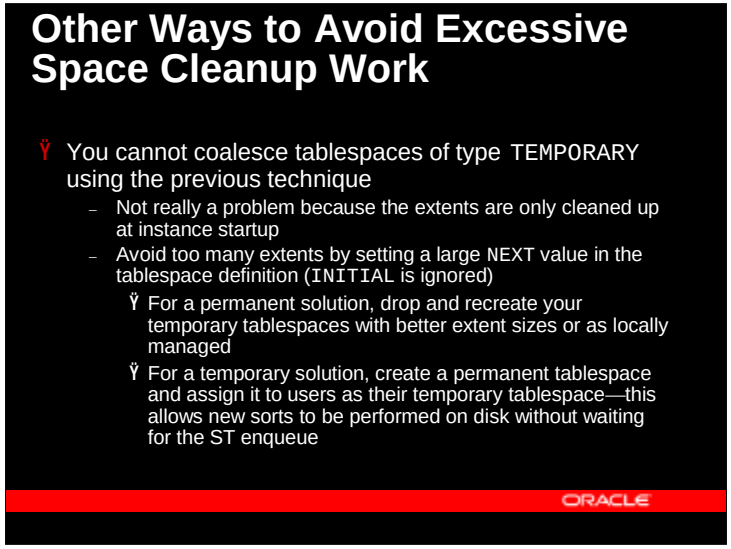
However, you have to examine the results to find line pairs where the value `block_id+blocks` in the first line of the pair equals `block_id` in the second line. This can be time-consuming and it is easy to overlook the rows you need. You can more quickly determine if you have multiple free extents with the following query:

```
SELECT tablespace_name, COUNT(*) FROM dba_free_space GROUP BY tablespace_name
```

Although the results do not indicate which, if any, extents are contiguous, you may assume that if a tablespace contains many free extents, at least a few of them will be adjacent. You can also assume there will be some adjacent free extents immediately after you drop a segment with multiple extents.

Regardless of how you determine if you may have adjacent free extents, you can coalesce any that do exist by executing the `ALTER TABLESPACE` statement given above. Even if you have made a wrong assumption and there is no space to coalesce, the statement will not cause any problems or overhead because it will not have any work to do.

Note: If your database is shutdown while SMON is performing its space management work, the shutdown will not undo the work completed so far. However, if your `ALTER TABLESPACE` command is interrupted, the entire statement will be rolled back and all the processed extents will have to be coalesced again.

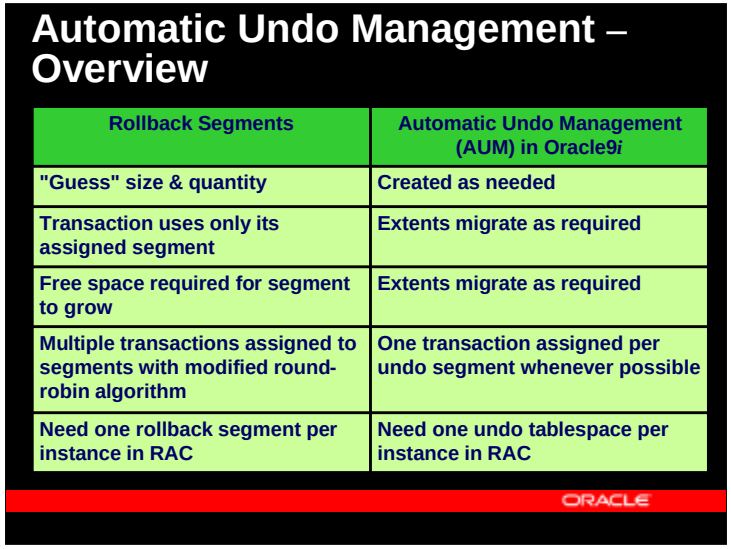


SMON is also responsible for cleaning up (dropping) temporary segments. Temporary segments are created when a command to create a new segment fails, when a segment is dropped, or when a PERMANENT type temporary tablespace is used for segments involved in sorts that are too large to complete in memory. If the user process does not remove the extents, SMON performs this service. Temporary segments used for sort operations may also be dropped by SMON.

You can detect if SMON is performing temporary space cleanup by repeatedly running the following query while SMON is busy. However, because SMON does not work as hard or require as many resources when cleaning up temporary extents as it does when coalescing free extents, it is typically not a big issue.

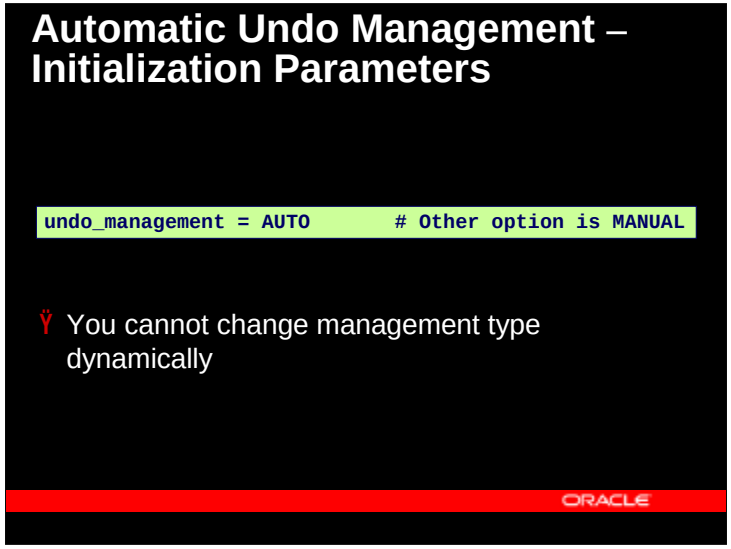
However, if you are using TEMPORARY type temporary tablespaces, then SMON's cleanup of the segment, which occurs after a database restart, may be a problem. This is because SMON does not service sort segment requests while performing cleanup. If the cleanup takes a long time, due to multiple extents in the temporary tablespace, sort operations for users will be delayed. To avoid this problem, do not create TEMPORARY type temporary tablespaces with small storage parameters or with unlimited `MAXEXTENTS`. Be aware that TEMPORARY type tablespaces set `MAXEXTENTS` unlimited automatically and only the `NEXT` storage value is used (`INITIAL` and `PCTINCREASE` values are ignored).

If you should have a large number of extents in a TEMPORARY type temporary tablespace, you can avoid user problems, while the SMON process completes its cleanup, by pointing users at a PERMANENT temporary tablespace. To do this, you may need to create a new tablespace (do not define it as type TEMPORARY) and then alter the users to reassign them to the PERMANENT type temporary tablespace for the time being. Of course, if you are running an Oracle9*i* database with a DEFAULT TEMPORARY tablespace, you can easily redirect the users to the new tablespace by changing the default at the database level.



Automatic Undo Management (AUM) was introduced in Oracle9*i* and greatly simplifies the configuration of the structures needed to manage rollback (undo) information. You can continue to manage rollback segments with the same techniques available under versions Oracle7, Oracle8, and Oracle8i, or you can let the RDBMS do it.

There are now two modes of rollback segments management and usage: * AUTOMATIC or * MANUAL To distinguish between the two types of segments, ROLLBACK segments are called UNDO segments when AUM is enabled. In both cases, rollback/undo segments are still required for transactions to execute and complete so, with either method, rollback/undo segments are present in the database and consume disk space.



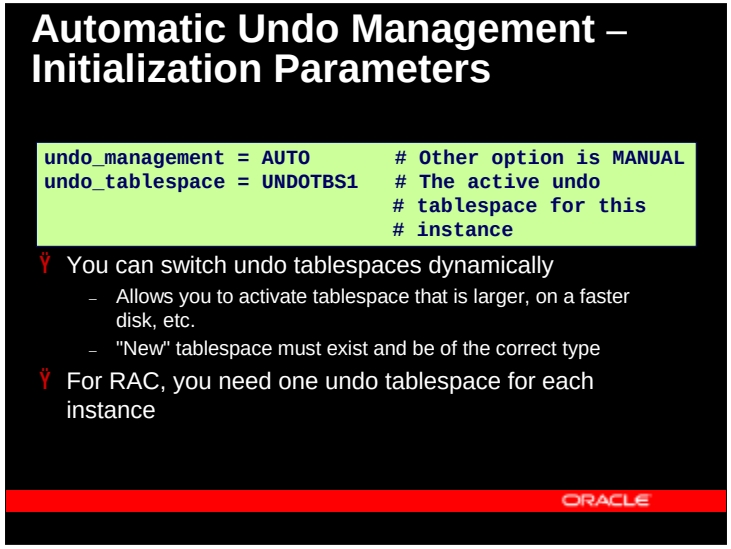
You determine how your undo/rollback will be managed by setting the UNDO_MANAGEMENT initialization parameter. The possible values, and their consequences are

AUTO: The RDBMS manages undo segments automatically:

- Creates an initial set of undo segments when you create a new UNDO tablespace
- Alters them ONLINE/OFFLINE when you choose a specific UNDO tablespace
- Drops them when you drop an UNDO tablespace
- Adds and drops segments based on load
- Reassigns space between segments as required

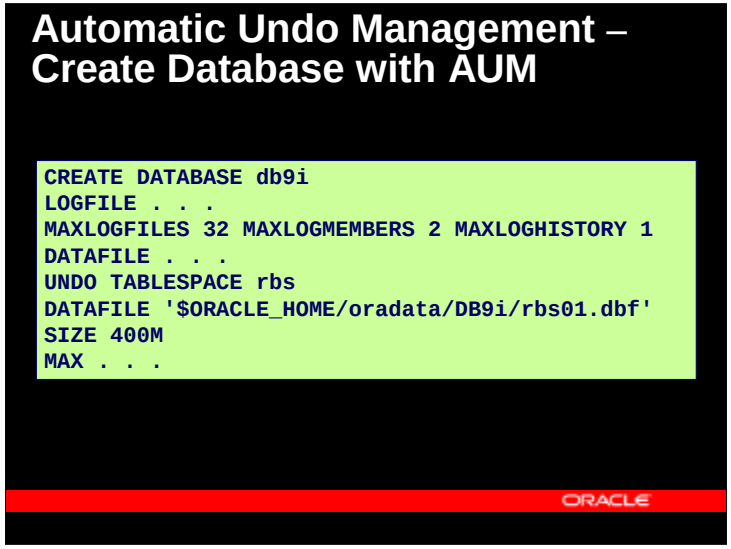
Note: You cannot manage automatic undo segments manually even though they appear as "rollback" segments in your database. You may create rollback segments in UNDO tablespaces, but it is strongly recommended not to do this.

MANUAL: You keep control of your rollback segments.



If you decide to use AUM, you have to create at least one undo tablespace to store the undo segments automatically created. Even though an instance running with AUM uses only one undo tablespace at a time, you can create several undo tablespaces. In this case, specify which undo tablespace is to be used by an instance with the initialization parameter: UNDO_TABLESPACE.

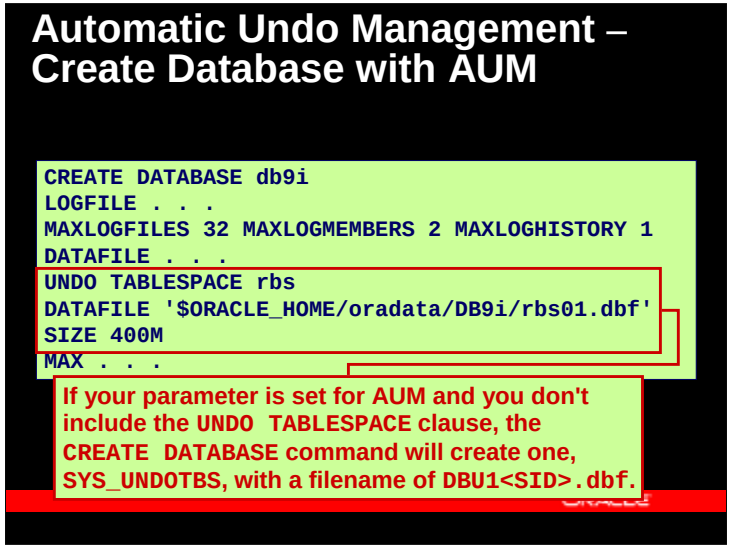
To determine which undo tablespace is currently assigned to an instance, you can query the V$PARAMETER view, or use the SQL*Plus SHOW PARAMETER command for the value of the UNDO_TABLESPACE parameter.



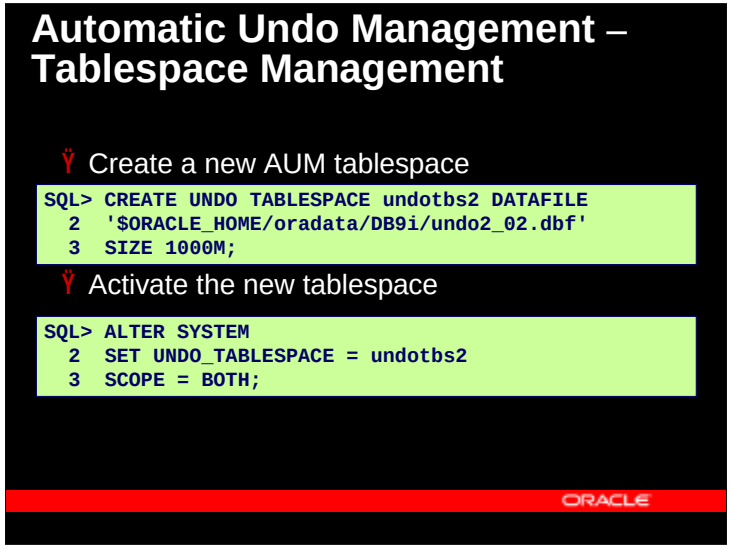
If you are creating a new database and wish to use AUM, you should include the `UNDO TABLESPACE` clause in your `CREATE DATABASE` command.

Here is the complete command from the slide:

```
CREATE DATABASE db9i
LOGFILE
'$ORACLE_HOME/oradata/DB9i/redo01.log' SIZE 1024K,
'$ORACLE_HOME/oradata/DB9i/redo02.log' SIZE 1024K,
'$ORACLE_HOME/oradata/DB9i/redo03.log' SIZE 1024K
MAXLOGFILES 32 MAXLOGMEMBERS 2 MAXLOGHISTORY 1
DATAFILE
'$ORACLE_HOME/oradata/DB9i/system01.dbf' SIZE 264M REUSE
AUTOEXTEND ON NEXT 10240K
UNDO TABLESPACE rbs DATAFILE
'$ORACLE_HOME/oradata/DB9i/rbs01.dbf' SIZE 400M
MAXDATAFILES 254 MAXINSTANCES 1
CHARACTER SET UTF8 NATIONAL CHARACTER SET AL16UTF16;
```



When creating a database with AUM but without the `UNDO TABLESPACE` clause, an undo tablespace called `SYS_UNDOTBS` is automatically created with the filename `DBU1<`*`SID`*`>.dbf`.

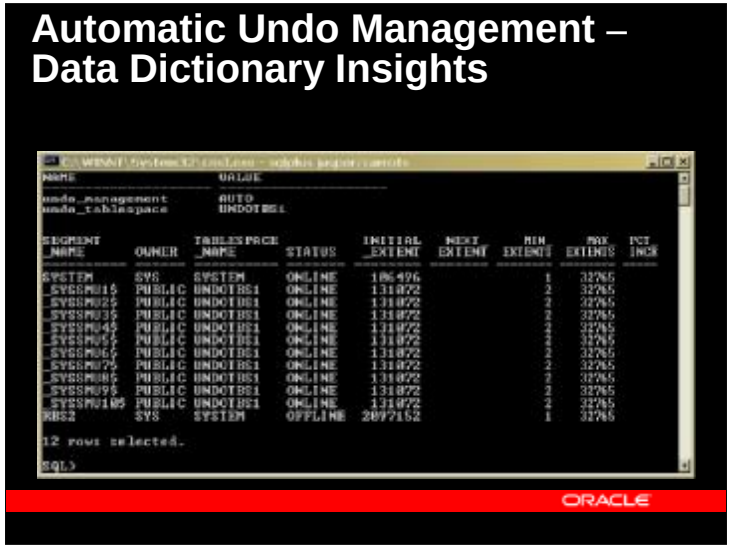


When you switch an instance to a different undo tablespace, only new transactions will be assigned to the newly-identified tablespace. Current transactions will continue to run in the original tablespace and its contents may be needed to satisfy queries or flashback operations that require consistent data. Although you may drop the unused undo tablespace once all current transactions complete, you may wish to leave it available for the length of time your applications typically require rollback information (possibly set with the `UNDO_RETENTION` initialization parameter).

Automatic Undo Management – Best Practice Advice

- It is possible to architect a database containing both rollback and undo segments
- However
  - Some combinations of rollback- and undo-related parameters may cause startup failures
  - Not all operations (e.g. `ALTER ROLLBACK SEGMENT`) will work on the rollback or undo segments

ORACLE



The output shown above was generated from a small test database running in AUM mode. The first query retrieved the values of the AUM-related parameters, UNDO_MANAGEMENT and UNDO_TABLESPACE. The second query was as follows:

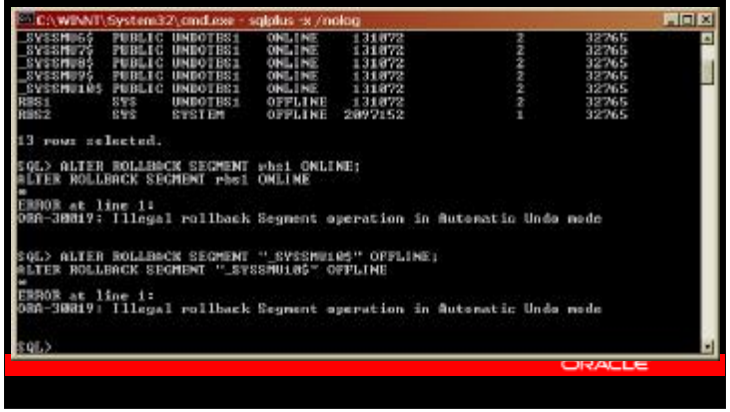
```
SELECT
  segment_name, owner, tablespace_name, status,
initial_extent,
  next_extent, min_extents, max_extents, pct_increase
FROM
  dba_rollback_segs
```

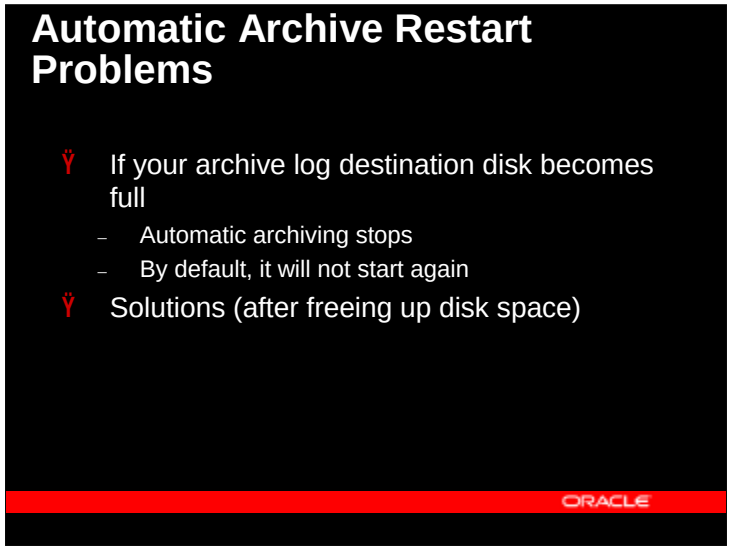
The output was modified with SQL*Plus `COLUMN` commands for display purposes.

Note that there is a traditional rollback segment, `RBS2`, in the database but that it is offline because AUM is active. Also note that the AUM-related segments have Oracle-generated names and are created in the `PUBLIC` schema.

select segment_name, count(*) from dba_extents where segment_name

Automatic Undo Management – Some Common "Gotcha's"





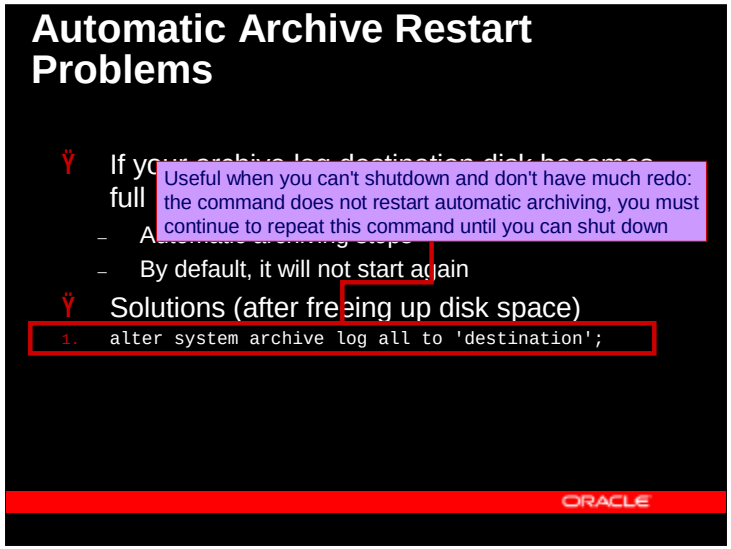
Suppose your database is running in archive log mode with automatic archiving turned on (Oracle's recommended configuration for a production database). If the archive log destination disk space becomes full, the automatic archiving will stop. This is expected behavior and a message should appear on the affected client's screen:

```
ORACLE Instance v816 - Can not allocate log, archival required.
```

Subsequently, after all of the online redo logs fill up, your database will be in a hang state.

After you free up disk space, you will probably discover that the online redo logs are still not archiving. In most cases, this is also the expected behavior. If you attempt to manually archive the files you may receive the errors similar to the following:

```
SQL> archive log next
ORA-16014: log 1 sequence# 199 not archived, no available destinations
ORA-00312: online log 1 thread 1: 'C:\ORACLE\ORADATA\V816\REDO01.LOG'
SQL> archive log all
ORA-16020: less destinations available than specified by
LOG_ARCHIVE_MIN_SUCCEED_DEST
```

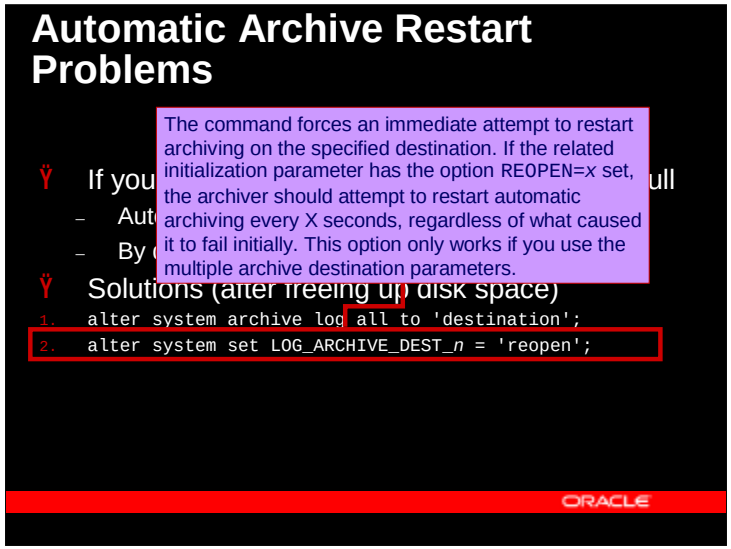


One way to fix this problem is to issue a command such as the following:

```
ALTER SYSTEM ARCHIVE LOG ALL TO 'c:\oradata\archive'
```

NOTE: You must specify a location, even if it is the original archive log directory in which you have just freed up space.

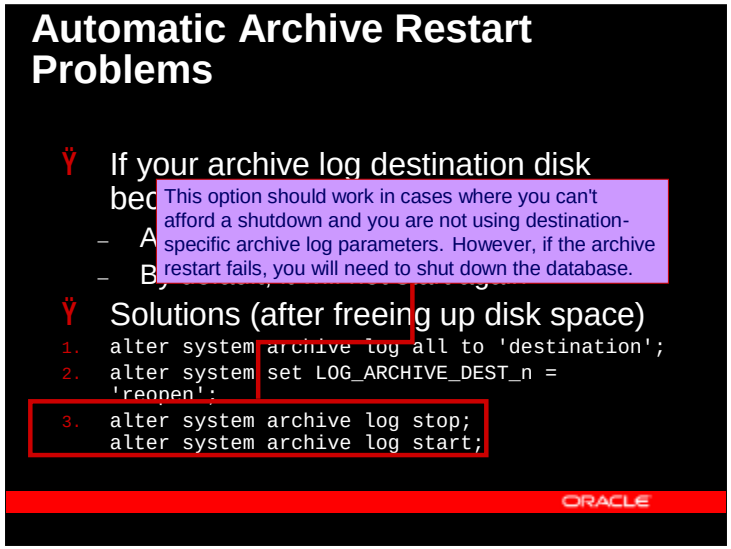
In some cases, although this command succeeds in creating archived log files for the current redo logs, it does not restart automatic archiving. You need to monitor the archive operations to ensure that the next few log switches occur normally. If they do not, you must either continue to create the archived copies manually, using the same `ALTER SYSTEM` command, until you can successfully execute one of the other options, described on the next pages.



Reset the value of the LOG_ARCHIVE_DEST_n parameter to ensure that the REOPEN option is specified with a non-zero value. For example, to force the ARCn process to re-attempt writing archive logs to LOG_ARCHIVE_DEST_1 every 30 seconds, execute the command:

```
ALTER SYSTEM SET LOG_ARCHIVE_DEST_1 = 'REOPEN=30'
```

By default, the retry period is 300 seconds and a value of 0 is the same as turning off the REOPEN option, in other words, ARCn will not attempt to archive after a failure. If you do not specify the REOPEN keyword, ARCn will never reopen a destination following an error.

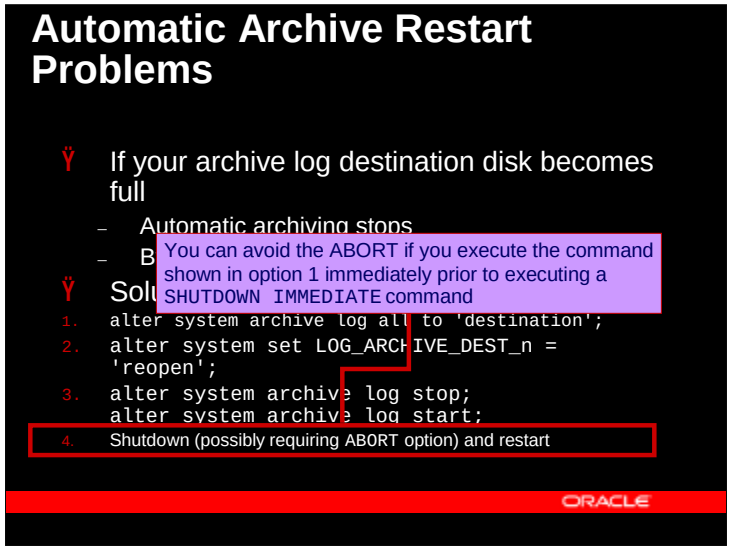


You can also try executing the commands:

`ALTER SYSTEM ARCHIVE LOG STOP`

`ALTER SYSTEM ARCHIVE LOG START`

However, although these commands restart the ARCn process, they may not succeed in sending the archive logs to the previously-full destination unless you have also set the `REOPEN` value for that destination, as described on the previous page. In some cases, even this will not re-enable automatic archiving.



If the `ALTER SYSTEM ARCHIVE LOG ALL` command does not restart the automatic archiving process for you, your next option is to bounce (shutdown and restart) your database. This should also restart the automatic archiving process.

In some cases, the shutdown will not work unless you specify the `ABORT` option:

```
SHUTDOWN ABORT
```

However, if your `ALTER SYSTEM ARCHIVE LOG ALL` succeeded in archiving the current online logs but did not restart the automatic archiving process, the shutdown should not require the `ABORT` option. In this situation, you should use the `IMMEDIATE` option, however, to avoid any further delays in solving the problem.

Build a Clone Database

- You can make a copy of an existing database using a few simple steps
- An OBE (Oracle By Example) course on OTN explains these steps on UNIX
- You can find OBEs at http://otn.oracle.com/obe

ORACLE


Comments
Questions
Answers
ORACLE