**Chapter 3 - Preparing the Database Environment**

All platforms have a hard-coded, platform-specific technique by which the OUI will find an existing inventory. On Linux this is a file:

/etc/oraInst.loc

On Solaris it is also a file: /var/opt/oracle/oraInst.loc

One Windows it is a key in the registry: HKEY\_LOCAL\_MACHINE\SOFTWARE\ORACLE\inst\_loc

The user running the installer for the first time will need permission to write to the appropriate directory. Usually only the root user can write to /etc or /var. OUI will generate a script (the orainstRoot.sh script) to be run by the root user that will create the oraInst.loc file. On Windows, the user running the OUI will need privileges that will let him/her create the registry key.

*Optimal Flexible Architecture a file system directory structure that should make maintaining multiple versions of multiple Oracle products straightforward.* The heart of OFA is two environment variables: ORACLE\_BASE and ORACLE\_HOME. The ORACLE\_BASE directory is one directory on the server, beneath which all the Oracle software (all products, all versions) should be installed.

The OFA standard includes several directories that you should be familiar with:

• Oracle inventory directory

• Oracle base directory (ORACLE\_BASE)

• Oracle home directory (ORACLE\_HOME)

• Oracle network files directory (TNS\_ADMIN)

• Automatic Diagnostic Repository (ADR\_HOME)

Oracle Inventory Directory

The Oracle inventory directory stores the inventory of Oracle software installed on the server. This directory is required and is shared among all installations of Oracle software on a server. When you first install Oracle, the installer checks to see whether there is an existing OFA-compliant directory structure in the format /u[01–09]/app. If such a directory exists, then the installer creates an Oracle inventory directory, such as /u01/app/oraInventory

If the ORACLE\_BASE variable is defined for the oracle operating system (OS) user, then the installer creates a directory for the location of Oracle inventory, as follows:

ORACLE\_BASE/../oraInventory. For example, if ORACLE\_BASE is defined as /ora01/app/oracle, then the installer defines the location of Oracle inventory as /ora01/app/oraInventory

If the installer doesn’t find a recognizable OFA-compliant directory structure or an ORACLE\_BASE variable, then the location for Oracle inventory is created under the HOME directory of the oracle user.

Oracle Base Directory

The Oracle base directory is the topmost directory for Oracle software installation. You can install one or more versions of the Oracle software beneath this directory. The OFA standard for the Oracle base directory is as: /<mount\_point>/app/<software\_owner> (Typical names for the mount point include /u01, /ora01, /oracle, and /oracle01.) /u01/app/oracle

Oracle Home Directory

The Oracle home directory defines the installation location of software for a particular product : ORACLE\_BASE/product/<version>/<install\_name>

/u01/app/oracle/product/12.1.0.1/db\_1

Oracle Network Files Directory

Some Oracle utilities use the value TNS\_ADMIN to locate network configuration files. This directory is defined as ORACLE\_HOME/network/admin. It typically contains the tnsnames.ora and listener.ora Oracle Net files.

Automatic Diagnostic Repository

Starting with Oracle Database 11g, the ADR\_HOME directory specifies the location of the diagnostic files related to Oracle. These files are crucial for troubleshooting problems with the Oracle database. This directory is defined as ORACLE\_BASE/diag/rdbms/lower(db\_unique\_name)/instance\_name.

/u01/app/oracle/diag/rdbms/o12c/O12C

You can verify the location of the ADR\_HOME directory via this query:

SQL> select value from v$diag\_info where name='ADR Home';

**Chapter 4 - Creating an Oracle Database**

These are the steps to follow to create a database:

1. Create a parameter file and (optionally) a password file. Ensure all the directories used in the parameter file are created. (e.g location of control file). Ensure that the oracle user and dba groups are properly set to own the directories, subdirectories

$ orapwd file=orapw<ORACLE\_SID> password=<sys password>

2. Use the parameter file to build an instance in memory. Ensure all the directories that would be used in the create database command are created. (e.g location of data file)

3. Issue the CREATE DATABASE command. This will generate, as a minimum, a controlfile, two online redo log files, two datafiles for the SYSTEM and SYSAUX tablespaces, and a data dictionary.

4. Run SQL scripts to generate the data dictionary views and the supplied PL/SQL packages.

SQL> spool create\_dd.lis

SQL> @?/rdbms/admin/catalog.sql

SQL> @?/rdbms/admin/catproc.sql

After you successfully create the data dictionary, as the SYSTEM schema, create the product user profile tables:

SQL> connect system/<password>

SQL> @?/sqlplus/admin/pupbld

If the pupbld.sql script isn’t run, then all non-sys users see warning message.

5. Run SQL scripts to generate the Enterprise Manager Database Control, and any options (such as Java) that the database will require.

The easiest way to create a database is through the DBCA.

To launch the DBCA on Linux, first set the environment variables that should always be set for any Linux DBA session:

export ORACLE\_BASE=/u02/app/db11g

export ORACLE\_HOME=$ORACLE\_BASE/product/11.1.0/db\_1

export PATH=$ORACLE\_HOME/bin:$PATH

export LD\_LIBRARY\_PATH=$ORACLE\_HOME/lib:$LD\_LIBRARY\_PATH

The oratab file is used in Linux/Unix environments for the following purposes:

• Automating the sourcing of required OS variables

• Automating the start and stop of Oracle databases on the server

The oratab file has three columns with this format:

<database\_sid>:<oracle\_home\_dir>:Y|N

The Y or N indicates whether you want Oracle to restart automatically on reboot of the box; Y indicates yes, and N indicates no.

The oraenv utility automates the setting of required OS variables (such as ORACLE\_HOME, ORACLE\_SID, and PATH) on an Oracle database server.

You can also run the oraenv utility in a noninteractive way by setting OS variables before you run it. This is useful for scripting when you don’t want to be prompted for input:

$ export ORACLE\_SID=o12c

$ export ORAENV\_ASK=NO

$ . oraenv

Oracle first looks in the ORACLE\_HOME/dbs directory for parameter files with specific formats, in this order:

• spfile<SID>.ora

• spfile.ora

• init<SID>.ora

Using the dbca to Create a Database

This utility works in two modes: graphical and silent. To use the dbca in graphical mode, ensure you have the proper X software installed, then issue the xhost + command, and make certain your DISPLAY variable is set; for example,

$ xhost +

$ echo $DISPLAY

:0.0

To run the dbca in graphical mode, type in dbca from the OS command line:

$ dbca

The graphical mode is very intuitive and will walk you through all aspects of creating a database. You may prefer to use this mode if you are new to Oracle and want to be explicitly prompted with choices.

. Locate the dbca.rsp file.

$ find . -name dbca.rsp

./12.1.0.1/database/response/dbca.rsp

2. Make a copy of the dbca.rsp file.

$ cp dbca.rsp mydb.rsp

3. Modify the copy of the dbca.rsp file for your environment.

[CREATEDATABASE]

GDBNAME = "O12DEV"

SID = "O12DEV"

TEMPLATENAME = "General\_Purpose.dbc"

SYSPASSWORD = "foo"

SYSTEMPASSWORD = "foo"

SYSMANPASSWORD = "foo"

DBSNMPPASSWORD = "foo"

DATAFILEDESTINATION ="/u01/dbfile"

STORAGETYPE="FS"

CHARACTERSET = "AL32UTF8"

NATIONALCHARACTERSET= "UTF8"

4. Run the dbca utility in silent mode.

$ dbca -silent -responseFile /home/oracle/orainst/mydb.rsp

. . .

100% complete

Look at the log file . . . for further details.

You must consider several architectural aspects when determining whether to use one database to host multiple applications and users:

• Do the applications generate vastly different amounts of redo, which may necessitate differently sized online redo logs?

• Are the queries used by applications dissimilar enough to require different amounts of undo, sorting space, and memory?

• Does the type of application require a different database block size, such as 8KB, for an OLTP database, or 32KB, for a data warehouse?

• Are there any security, availability, replication, or performance requirements that require an application to be isolated?

• Does an application require any features available only in the Enterprise Edition of Oracle?

• Does an application require the use of any special Oracle features, such as Data Guard, partitioning, Streams, or RAC?

• What are the backup and recovery requirements for each application? Does one application require online backups and the other application not? Does one application require tape backups?

• Is any application dependent on an Oracle database version? Will there be different database upgrade schedules and requirements?

**Chapter 6 - Configuring the Oracle Network Environment**

If the “@” and the connect string are omitted, then the user process will assume that the instance you wish to connect to is running on the local machine, and that the always-available IPC protocol can be used.

To create the listener, you need do nothing more than create an entry in the listener.ora file, and start it.

LIST2 =

(DESCRIPTION =

(ADDRESS\_LIST =

(ADDRESS = (PROTOCOL = TCP)(HOST = 127.0.0.1)(PORT = 1522))

(ADDRESS = (PROTOCOL = TCP)(HOST = jwlnx1.bplc.co.za)(PORT = 1522))

)

)

Database Registration

A listener is necessary to spawn server processes against an instance. In order to do this, it needs to know what instances are available on the computer on which it is running. A listener finds out about instances by the process of “registration.” There are two methods for registering an instance with a database: static and dynamic registration. For static registration, you hard-code a list of instances in the listener.ora file. Dynamic registration means that the instance itself, at startup time, locates a listener and registers with it.

Static Registration

SID\_LIST\_LIST2 =

(SID\_LIST =

(SID\_DESC =

(ORACLE\_HOME = /u01/oracle/app/product/11.1.0/db\_1)

(SID\_NAME = ocp11g)

)

)

This entry will configure the listener called LIST2 to accept connection requests for an instance called ocp11g. It says nothing about whether the instance is running or even exists at all. The directive ORACLE\_HOME is only required if the database listener is not running from the same Oracle Home as the instance. If this is the case, then this directive will let the listener find the executable file that it must run to launch a server process.

Dynamic Instance Registration

SQL> alter system set local\_listener=list2 scope=memory;

SQL> alter system register;

where LIST2 is a entry in the tnsnames.ora file of server containing the listner details.

LIST2=(DESCRIPTION=(ADDRESS\_LIST=(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1522))))

Alternatively following statement can be used.

alter system set LOCAL\_LISTENER='(ADDRESS=(PROTOCOL=TCP)(HOST=localhost)(PORT=1522))' scope=memory;

easy connect

SQL> connect [scott/tiger@jwlnx1.bplc.co.za:1522/ocp11g](mailto:scott/tiger@jwlnx1.bplc.co.za:1522/ocp11g)

Local Naming : nsnames.ora

Directory Naming and External Naming : Directory naming points the user toward an LDAP directory server to resolve aliases.

ocp11g =

(DESCRIPTION =

(ADDRESS\_LIST =

(ADDRESS = (PROTOCOL = TCP)(HOST = jwlnx1.bplc.co.za)(PORT= 1522)

)

)

(CONNECT\_DATA =

(service\_name = ocp11g)

)

)

lsnrctl status list2

lsnrctl start list2

lsnrctl > services

To test a connect string, use the TNSPING utility. This will accept a connect string, locate the Oracle Net files, resolve the string, and send a message to the listener. If the listener is running and does know about the service requested, the test will return successfully.

TNSADMIN Environment Variable

The listener.ora file is a server-side file that defines database listeners.

The tnsnames.ora file is a client-side file used for name resolution.

The sqlnet.ora file is optional and may exist (possibly with different settings) on the server side, the client side, or both. It contains settings that apply to all connections and listeners, such as security rules and encryption.

The Shared Server Architecture

There are two new process types: dispatchers and shared servers. There are also some extra queue memory structures within the SGA.

When a user process contacts a listener, rather than launching a server process and connecting it to the user process, the listener passes back the address of a dispatcher. If there is only one dispatcher, the listener will connect it to all the user processes. If there are multiple dispatchers, the listener will load-balance incoming connection requests across them.

When a user process issues a SQL statement, it is sent to the dispatcher. The dispatcher puts all the statements it receives onto a queue. This queue is called the common queue, because all dispatchers share it. All the shared server processes monitor the common queue. When a statement arrives on the common queue, the first available shared server picks it up. From then execution proceeds through the usual parse-bind-execute cycle, but when it comes to the fetch phase, the shared server puts the result set onto a response queue that is specific to the dispatcher that received the job in the first place. Each dispatcher monitors its own response queue, and whenever any results are put on it, the dispatcher will pick them up and fetch them back to the user process that originally issued the statement.

The PGA for a dedicated server session will store the session’s session data, its cursor state, its sort space, and its stack space. But in the shared server environment, a shared server session stores most of the session data in the SGA, rather than in a PGA. Then whenever a shared server picks a job off the common queue, it will go to the SGA and connect to the appropriate block of memory to find out the state of the session. The memory used in the SGA for each shared server session is known as the user global area (the UGA) and includes all of what would have been in a PGA with the exception of the session’s stack space.

When to Use the Shared Server

You will not find a great deal of hard advice in the Oracle documentation on when to use shared server, or how many dispatchers and shared servers you’ll need. The main point to hang on to is that shared server is a facility you use because you are forced to, not something you use automatically. It increases scalability, but perhaps at the cost of reducing performance. It is quite possible that any one statement will take longer to execute in a shared server environment than if it were executing on a dedicated server, because it has to go via queues. It may also take more CPU resources because of this enqueuing and dequeuing activity. But overall, the scalability of your system will increase dramatically.

It is often said that you should think about using shared server when your number of concurrent connections is in the low hundreds. If you have less than one hundred concurrent connections, you almost certainly don’t need it. But if you have more than a thousand, you probably do. The critical factor is whether your operating system performance is beginning to degrade. It should be apparent that shared server is ideal for managing many sessions doing short transactions, where the bulk of the work is on the client side of the clientserver divide. In these circumstances, one shared server will be able to service dozens of sessions. But for batch processing work, dedicated servers are much better. If you submit a large batch job through a shared server session, it will work—but it will tie up one of your small pool of shared server processes for the duration of the job, leaving all your other users to compete for the remaining shared servers.

Set Up a Shared Server Environment

Set the dispatchers and shared\_servers parameters and register with the listener as follows:

alter system set dispatchers='(pro=tcp)(dis=2)' scope=memory;

alter system set shared\_servers=4 scope=memory;

alter system register;

Confirm that the dispatchers and shared servers have started by querying the view V$PROCESS. Look for processes named S000, S001, S002, S003, D000, and D001:

select program from v$process order by program;

Connect through the listener, and confirm that the connection is through the shared server mechanism:

connect system/oracle@new;

select d.name,s.name from v$dispatcher d,v$shared\_server s, v$circuit c

where d.paddr=c.dispatcher and s.paddr=c.server;

Tidy up the environment, by returning to the original configuration:

alter system set dispatchers='' scope=memory;

alter system set shared\_servers=0 scope=memory;

alter system register;

**Chapter 7: Managing Database Storage Structures**

1 CREATE SMALLFILE TABLESPACE "NEWTS"

2 DATAFILE 'D:\APP\ORACLE\ORADATA\ORCL11G\newts01.dbf'

3 SIZE 100M AUTOEXTEND ON NEXT 10M MAXSIZE 200M

4 LOGGING

5 EXTENT MANAGEMENT LOCAL

6 SEGMENT SPACE MANAGEMENT AUTO

7 DEFAULT NOCOMPRESS;

A tablespace can be renamed while it is in use, but to rename a datafile, the datafiles must be offline. This is because the file must be renamed at the operating system level, as well as within the Oracle environment, and this can’t be done if the file is open: all the file handles would become invalid.

alter tablespace a1 rename to b1;

alter tablespace b1 offline;

host rename d:\oradata\a1.dbf to b1.dbf

alter database rename datafile 'd:\oradata\a1.dbf' to 'd:\oradata\b1.dbf'

alter tablespace b1 online;

alter tablespace gl\_large\_tabs

add datafile 'D:\ORADATA\GL\_LARGE\_TABS\_03.DBF' size 2g;

alter database datafile 'D:\ORADATA\GL\_LARGE\_TABS\_03.DBF'

autoextend on next 100m maxsize 4g;

DROP TABLESPACE tablespacename [INCLUDING CONTENTS [AND DATAFILES] ] ;

ALTER TABLESPACE tablespacename [READ ONLY | READ WRITE];

Following making a tablespace read only, none of the objects within it can be changed with DML statements, But they can be dropped. This is a little disconcerting but makes perfect sense when you think it through. Dropping a table doesn’t actually affect the table.

Resizing a Tablespace

alter tablespace gl\_large\_tabs

add datafile 'D:\ORADATA\GL\_LARGE\_TABS\_03.DBF' size 2g;

alter database datafile 'D:\ORADATA\GL\_LARGE\_TABS\_03.DBF'

autoextend on next 100m maxsize 4g;

There are two techniques for managing extent usage: dictionary management or local management

Dictionary extent management uses two tables in the data dictionary. SYS.UET$ has rows describing used extents, and SYS.FET$ has rows describing free extents. Every time the database needs to allocate an extent to a segment, it must search FET$ to find an appropriate bit of free space, and then carry out DML operations against FET$ and UET$ to allocate it to the segment.

In locally managed tablespace extent management is handled by using bitmaps in each data file. 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. When creating a locally managed tablespace, an important option is uniform size. If uniform is specified, then every extent ever allocated in the tablespace will be that size. This can make the space management highly efficient, because the block ranges covered by each bit can be larger: only one bit per extent.

create tablespace small\_tabs datafile 'small\_tabs\_01.dbf' size 1g extent management local uniform size 160k;

The alternative (and default) syntax would be

create tablespace any\_tabs datafile 'any\_tabs\_01.dbf' size 10g extent management local autoallocate;

When segments are created in this tablespace, Oracle will allocate a 64 KB extent. As a segment grows and requires more extents, Oracle will allocate extents of 64 KB up to 16 extents, from which it will allocate progressively larger extents. Thus fast growing segments will tend to be given space in ever-increasing chunks.

Segment Space Management

The segment space management method is set per tablespace and applies to all segments in the tablespace. There are two techniques for managing segment space usage: manual or automatic.

Manual segment space management exists for backward compatibility and uses free-block lists to identify the data blocks available for inserts together with the parameters PCTFREE and PCTUSED, which control when a block is made available for inserts.

The SEGMENT SPACE MANAGEMENT AUTO clause instructs Oracle to manage the space within the block. When you use this clause, there is no need to specify parameters, such as PCTUSED, FREELISTS, and FREELIST GROUPS.

Every segment created in an automatic management tablespace has a set of bitmaps, The free space map in a segment is spread out into a bitmap block within each extent of the segment. Each process performing insert, update, or delete operations will likely be accessing different blocks instead of one freelist or one of a few freelist groups. In addition, each extent’s bitmap block lists each block within the extent along with a four-bit “fullness” indicator. 0000 Unformatted block, 0001 Block full, 0010 Less than 25 percent free space available, 0011 25 percent to 50 percent free space, 0100 50 percent to 75 percent free space, 0101 Greater than 75 percent free space. . Each block will appear on exactly one bitmap.

When searching for a block into which to insert a row, the session server process will look at the size of the row to determine which bitmap to search.

It’s important to note that blocks are marked as full as soon as the last insert/update crosses the PCTFREE threshold, the point at which ASSM un-links the data block from the freelist chain.

The old manual space management method used a simple list, known as the free list, which stated which blocks were available for insert but without any information on how full they were. This method could cause excessive activity, as blocks had to be tested for space at insert time.

select tablespace\_name,segment\_space\_management from dba\_tablespaces;

It is not possible to convert tablespace from manual to automatic segment space management. The only solution is to create a new tablespace using automatic segment space management, move the segments into it (at which point the bitmap will be generated), and drop the old tablespaces.

You can use the oerr utility to quickly display the cause of an error and simple instructions on what actions to take; for example,

$ oerr ora 01653

alter database tempfile '/u01/dbfile/o12c/temp01.dbf' resize 500m;

alter tablespace temp add tempfile '/u01/dbfile/o12c/temp02.dbf' size 5000m;

As of Oracle Database 12c, you can move and rename data files while they are online and open for use.

Database has to be open to take one tablespace offline, you can't do in mount mode.

When in mount mode, you must use the ALTER DATABASE DATAFILE statement to take a data file offline.

You can specify ALTER TABLESPACE . . . OFFLINE IMMEDIATE when taking a tablespace offline. Your database must be in archivelog mode in this situation, or the following error is thrown:

ORA-01145: offline immediate disallowed unless media recovery enabled

You can also use the ALTER DATABASE DATAFILE statement to take a data file offline. If your database is open for use, then it must be in archivelog mode to take the data file offline. While the database is in mount mode (and not open), you can use the ALTER DATABASE DATAFILE.

New in Oracle Database 12c is the ALTER DATABASE MOVE DATAFILE command. This command allows you to rename or move data files without any downtime. This vastly simplifies the task of moving or renaming a data file, as there is no need to manually place data files offline/online and use OS commands to physically move the files. This once manually intensive (and error-prone) operation has now been simplified to a single SQL command.

A data file must be online for the online move or rename to work. Here is an example of renaming an online data file:

SQL> alter database move datafile '/u01/dbfile/o12c/users01.dbf' to '/u01/dbfile/o12c/users\_dev01.dbf';

You can also specify the data file number when renaming or moving a data file; for example,

SQL> alter database move datafile 2 to '/u02/dbfile/o12c/sysuax01.dbf';

Best Practices for Creating and Managing Tablespaces

Best Practice : Reasoning

Create separate tablespaces for different applications using the same database. : If a tablespace needs to be taken offline, it affects only one application.

For an application, separate table data from index data in different tablespaces. : Table and index data may have different storage requirements.

Don’t use the AUTOEXTEND feature for data files. If you do use AUTOEXTEND, specify a maximum size. : Specifying a maximum size prevents a runaway SQL statement from filling up a storage device.

Create tablespaces as locally managed. You shouldn’t create a tablespace as dictionary managed. : This provides better performance and manageability.

For a tablespace’s data file naming convention, use a name that contains the tablespace name followed by a two-digit number that’s unique within data files for that tablespace. : Doing this makes it easy to identify which data files are associated with which tablespaces.

Try to minimize the number of data files associated with a tablespace. : You have fewer data files to manage.

In tablespace CREATE scripts, use ampersand variables to define aspects such as storage characteristics. : This makes scripts more reusable among various environments.

Log writer flushes the contents of the redo log buffer when any of the following are true:

• A COMMIT is issued.

• A log switch occurs.

• Three seconds go by.

• The redo log buffer is one-third full.

• The redo log buffer fills to one megabyte.

V$LOG Displays the online redo log group information stored in the control file

V$LOGFILE Displays online redo log file member information

Try to size the online redo logs so that they switch anywhere from two to six times per hour. The V$LOG\_HISTORY contains a history of how frequently the online redo logs have switched.

select count(\*)

,to\_char(first\_time,'YYYY:MM:DD:HH24')

from v$log\_history

group by to\_char(first\_time,'YYYY:MM:DD:HH24')

order by 2;

COUNT(\*) TO\_CHAR(FIRST

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

1 2012:10:23:23

3 2012:10:24:03

28 2012:10:24:04

23 2012:10:24:05

68 2012:10:24:06

84 2012:10:24:07

15 2012:10:24:08

From the previous output, you can see that a great deal of log switch activity occurred from approximately 4:00 am to 7:00 am. This could be due to a nightly batch job or users’ in different time zones updating data. For this database the size of the online redo logs should be increased. You should try to size the online redo logs to accommodate peak transaction loads on the database.

You don’t want them switching too often because there is overhead with the log switch. Oracle initiates a checkpoint as part of a log switch. During a checkpoint the database writer background process writes modified (also called dirty) blocks to disk, which is resource intensive. Then again, you don’t want online redo log files never to switch, because the current online redo log contains transactions that you may need in the event of a recovery. If a disaster causes a media failure in your current online redo log, you can lose those transactions that haven’t been archived.

You can also query the OPTIMAL\_LOGFILE\_SIZE column from the V$INSTANCE\_RECOVERY view to determine if your online redo log files have been sized correctly:

SQL> select optimal\_logfile\_size from v$instance\_recovery;

This column reports the redo log file size (in megabytes) that is considered optimal, based on the initialization parameter setting of FAST\_START\_MTTR\_TARGET. Oracle recommends that you configure all online redo logs to be at least the value of OPTIMAL\_LOGFILE\_SIZE. However, when sizing your online redo logs, you must take into consideration information about your environment (such as the frequency of the switches).

Determining the Optimal Number of Redo Log Groups

Every time a log switch occurs, it initiates a checkpoint. As part of a checkpoint the database writer writes all modified (dirty) blocks from the SGA to the data files on disk. Also recall that the online redo logs are written to in a round-robin fashion and that eventually the information in a given log is overwritten. Before the log writer can begin to overwrite information in an online redo log, all modified blocks in the SGA associated with the redo log must first be written to a data file. If not all modified blocks have been written to the data files, you see this message in the alert.log file:

Thread 1 cannot allocate new log, sequence <sequence number>

Checkpoint not complete

There are a few ways to resolve this issue:

• Add more redo log groups.

• Lower the value of FAST\_START\_MTTR\_TARGET. Doing so causes the database writer process to write older modified blocks to disk in a shorter time frame.

• Tune the database-writer process (modify DB\_WRITER\_PROCESSES).

Adding an extra redo log gives the database writer more time to write modified blocks in the database buffer cache to the data files before the associated redo with a block is overwritten. There is little downside to adding more redo log groups. The main concern is that you could bump up against the MAXLOGFILES value that was used when you created the database. If you need to add more groups and have exceeded the value of MAXLOGFILES, then you must re-create your control file and specify a high value for this parameter.

If adding more redo log groups doesn’t resolve the issue, you should carefully consider lowering the value of FAST\_START\_MTTR\_TARGET. When you lower this value, you can potentially see more I/O because the database writer process is more actively writing modified blocks to data files. You can modify this parameter while your instance is up; this means you can quickly modify it back to its original setting if there are unforeseen side effects.

Finally, consider increasing the value of the DB\_WRITER\_PROCESSES parameter. Carefully analyze the impact of modifying this parameter in a test environment before you apply it to production. This value requires that you stop and start your database; therefore, if there are adverse effects, downtime is required to change this value back to the original setting.

alter database add logfile group 3

('/u01/oraredo/o12c/redo03a.rdo',

'/u02/oraredo/o12c/redo03b.rdo') SIZE 50M;

In this scenario I highly recommend that the log group you add be the same size and have the same number of members as the existing online redo logs. If the newly added group doesn’t have the same physical characteristics as the existing groups, it’s harder to accurately determine performance issues.

If the redologfile size is too high then this is very likely to produce the Checkpoint not complete issue described in the previous section. This is because flushing all

modified blocks from the SGA that are protected by the redo in a large log file can potentially take much longer time.

To resize an online redo log, you have to first add online redo log groups that are the size you want, and then drop the online redo logs that are the old size.

A log group must have an INACTIVE status before you can drop it. You can check the status of the log group, as shown here:

SQL> select group#, status, archived, thread#, sequence# from v$log;

Adding Online Redo Log Files to a Group

SQL> alter database add logfile member '/u02/oraredo/o12c/redo01b.rdo' to group 1;

Make certain you follow standards with regard to the location and names of any newly added redo log files.

Removing Online Redo Log Files from a Groups

Make sure the log file you want to drop isn’t in the current group:

SELECT a.group#, a.member, b.status, b.archived, SUM(b.bytes)/1024/1024 mbytes

FROM v$logfile a, v$log b

WHERE a.group# = b.group#

GROUP BY a.group#, a.member, b.status, b.archived

ORDER BY 1, 2;

SQL> alter database drop logfile member '/u01/oraredo/o12c/redo04a.rdo';

Moving or Renaming Redo Log Files

Sometimes, you need to move or rename online redo log files. For example, you may have added some new mount points to the system, and you want to move the online redo logs to the new storage. You can use two methods to accomplish this task:

• Add the new log files in the new location, and drop the old log files.

• Physically rename the files from the OS.

You can do OS rename with the database open or closed. But the risk is you have to ensure that the files you move aren’t part of the current online redo log group.

SQL> shutdown immediate;

$ mv /u02/oraredo/o12c/redo02b.rdo /u01/oraredo/o12c/redo02b.rdo

SQL> startup mount;

alter database rename file '/u02/oraredo/o12c/redo02b.rdo' to '/u01/oraredo/o12c/redo02b.rdo';

alter database open;

There are several architectural decisions you must carefully consider before you enable archiving:

• Where to place the archive redo logs and whether to use the fast recovery area to store them

• How to name the archive redo logs

• How much space to allocate to the archive redo log location

• How often to back up the archive redo logs

• When it’s okay to permanently remove archive redo logs from disk

• How to remove archive redo logs (e.g., have RMAN remove the logs, based on a retention policy)

• Whether multiple archive redo log locations should be enabled

• When to schedule the small amount of downtime that’s required (if a production database)

SQL> alter system set log\_archive\_dest\_1='location=/u01/oraarch/o12c' scope=both;

SQL> alter system set log\_archive\_format='o12c\_%t\_%s\_%r.arc' scope=spfile;

Take care not to set the LOG\_ARCHIVE\_FORMAT to an invalid value; If you do so, when you attempt to stop and start your database, you won’t even get to the nomount phase (because the spfile contains an invalid parameter): In this situation, if you’re using an spfile, you can’t start your instance. You have a couple of options here.

If you’re using RMAN and are backing up the spfile, then restore the spfile from a backup.

The alternative is to create an init.ora file manually from the contents of the spfile. First, rename the spfile that contains a bad value:

The FRA is an area on disk—specified via database initialization parameters—that can be used to store files, such as archive redo logs, RMAN backup files, flashback logs, and multiplexed control files and online redo logs. To enable the use of a FRA, you must set two initialization parameters (in this order):

• DB\_RECOVERY\_FILE\_DEST\_SIZE specifies the maximum space to be used for all files that are stored in the FRA for a database.

• DB\_RECOVERY\_FILE\_DEST specifies the base directory for the FRA.

When you enable a FRA, if you don’t set the initialization parameter LOG\_ARCHIVE\_DEST\_N, then, by default, the archive redo logs are written to the FRA.

Benefit of storing the archivelog files in the FRA: some aspects of database administration are automated. For example, once the FRA is enabled, RMAN backups and archive redo logs are automatically placed in a FRA within directory structures identifiable by database and date.

Another feature he likes is that archivelog files that are already beyond the retention policy (set via RMAN) are automatically deleted when space is needed in the FRA.

Reacting to a Lack of Disk Space in Your Archive Log Destination

The archiver background process writes archive redo logs to a location that you specify. If, for any reason, the archiver process can’t write to the archive location, your database hangs. Any users attempting to connect receive this error: ORA-00257: archiver error. Connect internal only, until freed.

In this situation your database is as good as down and completely unavailable. To fix the issue, you have to act quickly:

• Move files to a different location. (Be careful not to move an archive redo log that is currently being written to.)

• Compress old files in the archive redo log location. (Be careful not to compress an archive redo log that is currently being written to.)

• Permanently remove old files.

• Switch the archive redo log destination to a different location (this can be changed dynamically, while the database is up and running).

Moving files is usually the quickest and safest way to resolve the archiver error. You can use an OS utility such as mv to move old archive redo logs to a different location. If they’re needed for a subsequent restore and recovery, you can let the recovery process know about the new location. Be careful not to move an archive redo log that is currently being written to. Another option is to use an OS utility such as rm to remove archive redo logs from disk permanently. This approach is dangerous because you may need those archive redo logs for a subsequent recovery. If another location on your server has plenty of space, you can consider changing the location to which the archive redo logs are being written. You can perform this operation while the database is up and running; for example,

SQL> alter system set log\_archive\_dest\_1='location=/u02/oraarch/o12c';

Redundant array of inexpensive disks. RAID is essentially about different ways of storing data twice, or more than twice, on different disks. I’m oversimplifying just a bit, but a core goal of RAID is to let you survive losing a disk without also losing any of the data that was on that disk.

You can use RAID to increase your I/O throughput by taking advantage of striping your data across several disks at once.

**Chapter 8: Administering User Security**

The query against DBA\_TS\_QUOTAS confirms the quota of a user; the figure “–1” is how “unlimited” is represented.

ALTER USER username ACCOUNT UNLOCK;

To force a user to change his password, use this command:

ALTER USER username PASSWORD EXPIRE;

System privileges that (generally speaking) let users perform actions that affect the data dictionary and object privileges that let users perform actions that affect data.

System Privileges - CREATE SESSION, RESTRICTED SESSION (If the database is started with STARTUP RESTRICT, or adjusted with ALTER SYSTEM ENABLE RESTRICTED SESSION, ), ALTER DATABASE (for modifying physical structures), ALTER SYSTEM (for changing instance parameters and memory structures), GRANT ANY OBJECT PRIVILEGE, CREATE ANY TABLE , DROP ANY TABLE , INSERT ANY TABLE, UPDATE ANY TABLE, DELETE ANY TABLE , SELECT ANY TABLE.

There is no record kept of the granter of a system privilege, so it is not possible for a REVOKE to cascade. Hence if a user had been granted and had used the ADMIN OPTION, any users to whom he/she passed on the privilege will retain it. But for object privileges (Using WITH GRANT OPTION) a record is maintained and is revoked in the chain.

Managing security with directly granted privileges works but has two problems.

it can be a huge workload: an application with thousands of tables and users could need millions of grants.

Second, if a privilege has been granted to a user, that user has it in all circumstances: it is not possible to make a privilege active only in certain circumstances.

Roles are not schema objects: they aren’t owned by anyone and so cannot be prefixed with a username. However, they do share the same namespace as users:

SELECT\_CATALOG\_ROLE Has over 2000 object privileges against data dictionary objects, but no system privileges or privileges against user data.

select \* from dba\_role\_privs where granted\_role in ('USR\_ROLE','MGR\_ROLE');

select grantee,owner,table\_name,privilege,grantable from dba\_tab\_privs where grantee in ('USR\_ROLE','MGR\_ROLE')

union all

select grantee,to\_char(null),to\_char(null),privilege,admin\_option from dba\_sys\_privs where grantee in ('USR\_ROLE','MGR\_ROLE')

order by grantee;

A profile has a dual function: to enforce a password policy and to restrict the resources a session can take up. Password controls are always enforced; resource limits are only enforced if the instance parameter RESOURCE\_LIMIT is on TRUE—by default it is FALSE.

if you have a user with an expired password, and you don’t know the current password and are hesitant to change it (because it’s in use in a production environment.

SQL> select username, profile from dba\_users where username='MV\_MAINT';

create a temporary profile:

CREATE PROFILE temp\_prof LIMIT

PASSWORD\_REUSE\_MAX unlimited

PASSWORD\_REUSE\_TIME unlimited;

SQL> alter user mv\_maint profile temp\_prof;

SQL> select password from user$ where name='MV\_MAINT';

SQL> alter user mv\_maint identified by values 'E88FDA313EC0F3F4';

SQL> alter user mv\_maint profile default; -- switch back to the original profile

You can alter your current user’s session to point at a different schema via the ALTER SESSION statement:

SQL> alter session set current\_schema = hr;

**Chapter 9: Managing Schema Objects**

These object types each have their own namespace:

¦ Indexes ¦ Constraints ¦ Clusters ¦ Database triggers ¦ Private database links ¦ Dimensions

It is not possible to drop or truncate the parent table in a foreign key relationship, even if there are no rows in the child table. This still applies if the ON DELETE SET NULL or ON DELETE CASCADE clauses were used.

At any time, every constraint is either enabled or disabled, and validated or not validated. Any combination of these is syntactically possible.

Changing the status of a constraint between ENABLED/DISABLED and VALIDATE/ NOVALIDATE is an operation that will affect all sessions. The status is a data dictionary update. Switching a deferrable constraint between IMMEDIATE and DEFERRED is session specific, though the initial state will apply to all sessions.

Index Type Options

There are six commonly used options that can be applied when creating indexes:

¦ Unique or non-unique ¦ Reverse key ¦ Compressed ¦ Composite ¦ Function based ¦ Ascending or descending

All these six variations apply to B\*Tree indexes, but only the last three can be applied to bitmap indexes:

If an index was created implicitly by creating a constraint, then dropping the constraint will also drop the index. If the index had been created explicitly and the constraint created later, then if the constraint were dropped the index would survive.

Indexes

An index is an optionally created database object used primarily to increase query performance. In addition to improving performance, Oracle uses indexes to help enforce enabled primary key and unique key constraints. Additionally, Oracle can better manage certain table-locking scenarios when indexes are placed on foreign key columns.

B-tree : Default index; good for columns with high cardinality (i.e., high degree of distinct values). Use a normal B-tree index unless you have a concrete reason to use a different index type or feature.

IOT : This index is efficient when most of the column values are included in the primary key. You access the index as if it were a table. The data are stored in a B-tree-like structure.

Unique : A form of B-tree index; used to enforce uniqueness in column values; often used with primary key and unique key constraints but can be created independently of constraints.

Reverse key : A form of B-tree index; useful for balancing I/O in an index that has many sequential inserts.

Reverse-key indexes can perform better in scenarios in which you need a way to evenly distribute index data that would otherwise have similar values clustered together. Thus, when using a reverse-key index, you avoid having I/O concentrated in one physical disk location within the index during large inserts of sequential values. Use the REVERSE clause to create a reverse-key index:

SQL> create index cust\_idx1 on cust(cust\_id) reverse;

In a reverse key index, the values are stored backward—for example, a value of 2201 is stored as 1022. If you use a standard index, consecutive values are stored near each other. In a reverse key index, consecutive values are not stored near each other. If your queries do not commonly perform range scans and you are concerned about I/O contention (in a RAC database environment) or concurrency contention (buffer busy waits statistic in ADDM) in your indexes, reverse key indexes may be a tuning solution to consider.

There is a downside to reverse key indexes, however: they need a high value for pctfree to allow for frequent inserts, and must be rebuilt often, more often than a standard B-tree index.

Bitmap : Excellent in data warehouse environments with low cardinality (i.e., low degree of distinct values) columns and SQL statements using many AND or OR operators in the WHERE clause. Bitmap indexes aren’t appropriate for OLTP databases in which rows are frequently updated. You can’t create a unique bitmap index. Bitmap join Useful in data warehouse environments for queries that use star schema structures that join fact and dimension tables.

Bitmap indexes are commonly used in data warehouse environments. A typical star schema structure consists of a large fact table and many small dimension (lookup) tables. In these scenarios it’s common to create bitmap indexes on fact table foreign key columns. The fact tables are typically inserted into on a daily basis and usually aren’t updated or deleted from.

Structure of the LOCATIONS\_BMX1 Bitmap Index

Value/Row Row 1 Row 2 Row 3 Row 4 Row 5 Row 6 Row 7

EAST 0 1 0 0 1 0 0

NORTH 1 0 1 0 0 1 1

WEST 0 0 0 1 0 0 0

Creating Bitmap Join Indexes

Bitmap join indexes store the results of a join between two tables in an index. Bitmap join indexes are beneficial because they avoid joining tables to retrieve results. The syntax for a bitmap join index differs from that of a regular bitmap index in that it contains FROM and WHERE clauses.

create bitmap index f\_shipments\_bmx1

on f\_shipments(cust.first\_name, cust.last\_name)

from f\_shipments, cust

where f\_shipments.cust\_id = cust.cust\_id;

Now, consider a query such as this:

select c.first\_name, c.last\_name

from f\_shipments s, cust c

where s.cust\_id = c.cust\_id

and c.first\_name = 'JIM'

and c.last\_name = 'STARK';

Bitmap join indexes are appropriate in situations in which you’re joining two tables, using the foreign key column (or columns) in one table relating to the primary key column (or columns) in the other table.

Function based : Good for columns that have SQL functions applied to them; can be used with either a B-tree or bitmap index.

Indexed virtual column : An index defined on a virtual column (of a table); useful for columns that have SQL functions applied to them; a viable alternative to a function-based index.

Virtual : Allows you to create an index with no physical segment or extents via the NOSEGMENT clause of CREATE INDEX; useful in tuning SQL without consuming resources required to build the physical index. Any index type can be created as virtual.

The B-tree index has a hierarchical tree structure. When Oracle accesses the index, it starts with the top node, called the root (or header) block. Oracle uses this block to determine which second-level block (also called a branch block) to read next. The second-level block points to several third-level blocks (leaf nodes), which contain a ROWID and the name value.

Prior to Oracle Database 12c, you could not have multiple indexes defined on the exact same combination of columns in one table. This has changed in 12c.

Implementing Function-Based Indexes

select first\_name from cust where UPPER(first\_name) = 'JIM';

create index cust\_fnx1 on cust(upper(first\_name));

You must use the keyword DETERMINISTIC when creating a user-defined function that you want to use in a function-based index.

If you want to see the definition of a function-based index, select from the DBA/ALL/USER\_IND\_EXPRESSIONS

Implementing Invisible Indexes

In Oracle Database 11g and higher, you have the option of making an index invisible to the optimizer. Oracle still maintains an invisible index (as DML occurs on the table) but doesn’t make it available for use by the optimizer.

Invisible indexes have a couple of interesting uses:

• Altering an index to be invisible before dropping it allows you to quickly recover if you later determine that the index is required.

• You can add an invisible index to a third-party application without affecting existing code or support agreements.

SQL> create index cust\_idx2 on cust(first\_name) invisible;

SQL> alter index cust\_idx1 invisible;

There are a couple of good reasons to rebuild an index:

• Modifying storage characteristics, such as changing the tablespace.

• Rebuilding an index that was previously marked unusable to make it usable again.

SQL> alter index cust\_idx1 rebuild;

Oracle attempts to acquire a lock on the table and rebuild the index online. If there are any active transactions that haven’t committed, Oracle won’t be able to obtain a lock,

ORA-00054: resource busy and acquire with NOWAIT specified or timeout expired

The DDL\_LOCK\_TIMEOUT initialization parameter is available in Oracle Database 11g and higher. It instructs Oracle to repeatedly attempt to obtain a lock (for 15 seconds, in this case).

SQL> alter session set ddl\_lock\_timeout=15;

SQL> alter index cust\_idx1 rebuild tablespace reporting\_index;

SQL> alter index cust\_idx1 rebuild parallel nologging;

If no B-tree index exists on the foreign key columns, when you insert or delete a record from a child table, all rows in the parent table are locked. For applications that actively modify both the parent and child tables, this will cause locking and deadlock issues

Resetting a Sequence

SQL> alter sequence myseq increment by -1000;

SQL> select myseq.nextval from dual;

SQL> alter sequence myseq increment by 1;

A temporary table is similar to a permanent table. The difference is that the data is transient and private to the session, and that all SQL commands against it will be far faster than commands against permanent tables.

The first reason for the speed is that temporary tables are exist only in the PGAs of the sessions that are using them, so there is no disk activity (unless PGA can’t grow sufficiently to accommodate the temporary table, the table gets written out to a temporary segment in the user’s temporary tablespace ) or even database buffer cache activity involved.

A second reason for speed is that DML against temporary tables does not generate redo. Since the data only persists for the duration of a session (perhaps only for the duration of a transaction), there is no purpose in generating redo.

Here is an example of creating a table with a virtual column:

create table inv(

inv\_id number

,inv\_count number

,inv\_status generated always as (

case when inv\_count <= 100 then 'GETTING LOW'

when inv\_count > 100 then 'OKAY'

end)

);

Specifying GENERATED ALWAYS is optional. If you insert data into the table, nothing is stored in a column set to GENERATED ALWAYS AS. The virtual value is generated when you select from the table.

The main use for an invisible column is to ensure that adding a column to a table won’t disrupt any of the existing application code. If the application code doesn’t explicitly access the invisible column, then it appears to the application as if the column doesn’t exist.

create table inv

(inv\_id number

,inv\_desc varchar2(30 char)

,inv\_profit number invisible);

The main use for an invisible column is to ensure that adding a column to a table won’t disrupt any of the existing application code. If the application code doesn’t explicitly access the invisible column, then it appears to the application as if the column doesn’t exist.

create table inv

(inv\_id number

,inv\_desc varchar2(30 char)

,inv\_profit number invisible);

Starting with Oracle Database 11g, you can place individual tables in read-only mode. Doing so prevents any INSERT, UPDATE, or DELETE statements from running against a table.

SQL> alter table inv read only;

Starting with Oracle Database 12c, you can define a column that is automatically populated and incremented when inserting data. This feature is ideal for automatically populating primary key columns. Prior to Oracle Database 12c, you would have to create a sequence manually and then access the sequence when inserting into the table.

create table inv(

inv\_id number generated as identity

,inv\_desc varchar2(30 char));

Avoiding Redo Creation

The NOLOGGING feature can greatly reduce the amount of redo generation for certain types of operations. The downside to eliminating redo generation is that you can’t recover the data created via NOLOGGING in the event a failure occurs after the data are loaded (and before you can back up the table).

The NOLOGGING feature can significantly reduce redo generation for the following types of operations:

• SQL\*Loader direct-path load

• Direct-path INSERT /\*+ append \*/

• CREATE TABLE AS SELECT

• ALTER TABLE MOVE

• Creating or rebuilding an index

If your database is in FORCE LOGGING mode, then redo is generated for all operations, regardless of whether you specify NOLOGGING.

Undropping a Table

Suppose you accidentally drop a table, and you want to restore it. First, verify that the table you want to restore is in the recycle bin:

SQL> show recyclebin;

SQL> flashback table inv to before drop;

Prior to Oracle Database 12c, there were only three levels of static views:

• USER • ALL • DBA

Starting with Oracle Database 12c, there is a fourth level that is applicable when using the container/pluggable database feature: • CDB . The CDB-level views are only applicable if you’re using the pluggable database feature.

The static views are based on internal Oracle tables, such as USER$, TAB$, and IND$.

Dynamic Performance Views

These views are constantly updated by Oracle and reflect the current condition of the instance and database. Dynamic views are critical for diagnosing real-time performance issues.

At the top layer, the V$ views are actually synonyms that point to underlying SYS.V\_$ views.

At the next layer down, the SYS.V\_$ objects are views created on top of another layer of SYS.V$ views.

The SYS.V$ views in turn are based on the SYS.GV$ views.

At the bottom layer, the SYS.GV$ views are based on the X$ memory structures.

The top-level V$ synonyms and SYS.V\_$ views are created when you run the catalog.sql script, which you usually do after the database is initially created.

**Chapter 10: Managing Data and Concurrency**

A Is for Atomicity : The principle of atomicity states that either all parts of transaction must complete, or none of them complete. Typical example is balance transfer between two bank A/c.

C Is for Consistency : The principle of consistency states that the results of a query must be consistent with the state of the database at the time the query started.

I Is for Isolation : The principle of isolation states that an incomplete (that is, uncommitted) transaction must be invisible to the rest of the world.

D Is for Durable : The principle of durability states that once a transaction completes, it must be impossible for the database to lose it.

Redo protects all block changes, no matter whether it is a change to a block of a table segment, an index segment, or an undo segment. As far as redo is concerned, an undo segment is just another segment, and any changes to it must be made durable.

Only one session can take an exclusive lock on a row, or a whole table, at a time—but shared locks can be taken on the same object by many sessions. The purpose of taking a shared lock on a table is to prevent another session acquiring an exclusive lock on the table: you cannot get an exclusive lock if anyone else already has a shared lock. Exclusive locks on tables are required to execute DDL statements.

To execute DML on rows, a session must acquire exclusive locks on the rows to be changed, and shared locks on the tables containing the rows. If another session already has exclusive locks on the rows, the session will hang until the locks are released by a COMMIT or a ROLLBACK. If another session already has a shared lock on the table and exclusive locks on other rows, that is not a problem.

If you do not want a session to queue up if it cannot get a lock, the only way to avoid this is to use the WAIT or NOWAIT clauses of the SELECT...FOR UPDATE command. It is possible to append the keywords SKIP LOCKED to a SELECT FOR UPDATE statement, which will return and lock only rows that are not already locked by another session. This command existed with earlier releases but is only supported from release 11g.

**Chapter 11 Managing Undo Data**

select value from v$parameter where name='undo\_management'; -- auto means undo else rollback

alter system set undo\_management=auto scope =spfile;

what are the undo tablespaces available

select tablespace\_name from dba\_tablespaces where contents='UNDO';

Which undo tablespaces are in use

select value from v$parameter where name='undo\_tablespace';

Determine what undo segments are in use in the database, and how big they are:

select tablespace\_name,segment\_name,segment\_id,status from dba\_rollback\_segs;

select usn,rssize from v$rollstat;

When a transaction starts, Oracle will assign it to one (and only one) undo segment.

Any one transaction can only be protected by one undo segment—it is not possible for the undo data generated by one transaction to cut across multiple undo segments.

If a transaction does manage to fill its undo segment, Oracle will automatically add another extent to the segment.

It is possible for multiple transactions to share one undo segment, but in normal running this should not occur.

One feature of undo management is that Oracle will automatically spawn new undo segments on demand, in an attempt to ensure that it is never necessary for transactions to share undo segments. when the workload drops Oracle will shrink and drop the segments, again automatically.

Active undo is undo data that might be needed to roll back transactions in progress.

Expired undo is undo data from committed transactions, which Oracle is no longer obliged to store

Unexpired undo is an intermediate category; it is neither active nor expired: the transaction has committed, but the undo data might be needed for consistent reads, if there are any long-running queries in progress.

If a transaction runs out of undo space, it will fail with the error ORA-30036, “unable to extend segment in undo tablespace.” The statement that hit the problem is rolled back, but the rest of the transaction remains intact and uncommitted. The algorithm that assigns space within the undo tablespace to undo segments means that this error condition will only arise if the undo tablespace is absolutely full of active undo data.

If a query encounters a block that has been changed since the query started, it will go to the undo segment to find the preupdate version of the data. If, when it goes to the undo segment, that bit of undo data has been overwritten, the query fails on consistent read with a famous Oracle error ORA-1555, “snapshot too old.”

If the undo tablespace is undersized for the transaction volume and the length of queries, Oracle has a choice: either let transactions succeed and risk queries failing with ORA-1555 or let queries succeed and risk transactions failing with ORA-30036. The default behaviour is to let the transactions succeed, to allow them to overwrite unexpired undo.

There are three parameters controlling undo: UNDO\_MANAGEMENT, UNDO\_TABLESPACE, and UNDO\_RETENTION.

UNDO\_RETENTION, set in seconds, is usually optional. It specifies a target for keeping inactive undo data and determines when it becomes classified as expired rather than unexpired. If, for example, your longest running query is thirty minutes, you would set this parameter to 1800. Oracle will then attempt to keep all undo data for at least 1800 seconds, and your query should therefore never fail with ORA-1555. If, however, you do not set this parameter, or set it to zero, Oracle will still keep data for as long as it can anyway.

No matter how many undo tablespaces there may be in a database, generally speaking only one will be in use at a time. The undo segments in this tablespace will have a status of online (meaning that they are available for use); the segments in any other undo tablespaces will have status offline.

The Oracle database guarantees transactional integrity absolutely, but not necessarily read consistency. If the undo system is not appropriately configured, queries may fail because a lack on undo data—but if a query succeeds, it will be consistent. This behavior can be modified by enabling the RETENTION GUARANTEE, though this may mean that transaction fail.

**Chapter 12 - Implementing Oracle Database Security**

create user jon identified externally;

REMOTE\_OS\_AUTHENT – users don’t have access to db server can be authenticated using identified externally

OS\_AUTHENT\_PREFIX – prefix to be appended before os users name for corresponding db user.

O7\_DICTIONARY\_ACCESSIBILITY – false default, select any privilege user can’t access the sys owned objects, oracle data dictionary

REMOTE\_LOGIN\_PASSWORDFILE - instance parameter controls whether it is possible to connect to the instance as a user with the SYSDBA or SYSOPER privilege over the network.

V$PWFILE\_USERS view shows you which users have their passwords entered in the password file, and whether they have the SYSOPER privilege, the SYSDBA privilege, or both.

Orapwd

**Chapter 13 -Database Maintenance**

Statistics are visible in the DBA\_TABLES include: number of rows, number of blocks, avg row length, number of chained rows, free space available in the blocks.

“chained” rows—rows that cut across two or more blocks, either because they are very long or because of poor storage settings

Column statistics in the DBA\_TAB\_COLUMNS view include : number of distinct values, highest and lowest value, number of nulls, depth of index, clustering factor.

• If the value is near the number of blocks, then the table is very well ordered. In this case, the index entries in a single leaf block tend to point to rows in the same data blocks.

• If the value is near the number of rows, then the table is very randomly ordered. In this case, it is unlikely that index entries in the same leaf block point to rows in the same data blocks.

For unordered table optimizer may choose full table scan as opposed to index range scan depending on the number of estimated rows of output.

The gathering and visibility of statistics can be controlled by the STATISTICS\_LEVEL : ■ BASIC, ■ TYPICAL, ■ ALL

An AWR snapshot can be thought of as a copy of the contents of many V$ views at the time the snapshot was taken. Snapshots of statistics data are kept in the AWR, by default, for eight days. This period is configurable. As a rough guide for sizing, if the snapshot collection is left on every hour and the retention time is left on eight days then the AWR may well require between 200 MB and 300 MB of space in the SYSAUX tablespace. But this figure is highly variable and will to a large extent depend on the number of sessions.

Gather an AWR snapshot: execute dbms\_workload\_repository.create\_snapshot;

Find out how many snapshot there are, and what date range they cover:

select min(begin\_interval\_time), max(begin\_interval\_time),count(snap\_id) from dba\_hist\_snapshot;

You can enable and disable the automatic optimizer statistics gathering by using the DBMS\_AUTO\_TASK package.

BEGIN

DBMS\_AUTO\_TASK\_ADMIN.DISABLE (

client\_name=>’auto optimizer stats collection’,

operation=>NULL, window\_name=>NULL);

END;

BEGIN

DBMS\_AUTO\_TASK\_ADMIN.ENABLE (

client\_name=>’auto optimizer stats collection’,

operation=>NULL, window\_name=>NULL);

END;

SELECT client\_name, status FROM dba\_autotask\_client;

Extended Statistics

In Oracle 11g, you can tell the optimizer the relationship between columns by using the extended statistics feature (multicolumn statistics). The extended statistics feature also includes statistics on columns where a function is applied (function-based statistics).

exec dbms\_stats.gather\_table\_stats(null, ‘customers’, method\_opt=>’for all columns size skewonly for columns (cust\_country, cust\_state)’);

The 30-day retention value shown here is expressed in minutes: 60 minutes per hour × 24 hours per day × 30 days = 43,200 minutes.

execute dbms\_workload\_repository.modify\_snapshot\_settings (interval=>60,retention=>43200);

The values for \_DUMP\_DEST parameters are ignored by Oracle 11g. The new view V$DIAG\_INFO gives file locations:

SELECT name, value FROM v$diag\_info;

BEGIN

DBMS\_MONITOR.SESSION\_TRACE\_ENABLE(session\_id=>324,

serial\_num=>54385,

waits=>TRUE,

binds=>TRUE);

END;

BEGIN

DBMS\_MONITOR.SESSION\_TRACE\_DISABLE(session\_id=>324,

serial\_num=>54385);

END;

$oracle\_home/rdbms/admin @utlxplan

create public public synonym plan\_table for plan\_table

C:\oracle\app\product\11.2.0\dbhome\_1\sqlplus\admin @plustrce

set autotrace traceonly;

or

ALTER SESSION SET SQL\_TRACE=TRUE|

Finding the trace file in oracle 11g

SELECT value FROM v$diag\_info WHERE name = 'Default Trace File';

You may require to append a separate identifier in the trace file name to identify the file easily.

ALTER SESSION SET TRACEFILE\_IDENTIFIER = "MY\_TEST\_SESSION";

**14 Performance Management**

Automatic PGA memory management is enabled with two instance parameters:

¦ WORKAREA\_SIZE\_POLICY = AUTO

¦ PGA\_AGGREGATE\_TARGET = VALUE

To enable automatic SGA management, leave all of these on default (or set to zero) and set one parameter to enable automatic shared memory management (ASSM):

¦ SGA\_TARGET

Automatic Memory Management : The Automatic Memory Management mechanism lets the Oracle instance manage server memory usage as a whole, by setting one parameter: MEMORY\_TARGET.

SELECT

PGA\_TARGET\_FOR\_ESTIMATE,

PGA\_TARGET\_FACTOR,

ESTD\_EXTRA\_BYTES\_RW

FROM

V$PGA\_TARGET\_ADVICE;

SELECT

SGA\_SIZE,

SGA\_SIZE\_FACTOR,

ESTD\_DB\_TIME,

ESTD\_DB\_TIME\_FACTOR

FROM V$SGA\_TARGET\_ADVICE

SELECT

MEMORY\_SIZE,

MEMORY\_SIZE\_FACTOR,

ESTD\_DB\_TIME

FROM V$MEMORY\_TARGET\_ADVICE

select \* from v$parameter where name in

('workarea\_size\_policy',

'pga\_aggregate\_target',

'sga\_target',

'memory\_target'

)

Oracle Invisible Index

An invisible index is an index that is maintained by the database but ignored by the optimizer unless explicitly specified. Invisible indexes are an attractive feature for the process of dropping an index. They are also useful when a specific application needs the benefit of a temporary index without impacting the database on a wider scale.

**Chapter 15 Backup and Recovery Concepts**

Setting FAST\_START\_MTTR\_TARGET to a non-zero value has two effects.

First, it sets a target for recovery,

Also a secondary effect: enabling checkpoint auto-tuning. The checkpoint auto-tuning mechanism inspects statistics on machine utilization, such as the rate of disk I/O and CPU usage, and if it appears that there is spare capacity, it will use this capacity to write out additional dirty buffers from the database buffer cache, thus pushing the checkpoint position forward.

Instance recovery timing related information can be gained from querying the V$INSTANCE\_RECOVERYview

**Column : Meaning**

recovery\_estimated\_ios : The number of read/write operations that would be needed on datafiles for recovery if the instance crashed now

actual\_redo\_blocks : The number of OS blocks of redo that would need to be applied to datafiles for recovery if the instance crashed now

estimated\_mttr : The number of seconds it would take to open the database if it crashed now

target\_mttr : The setting of fast\_start\_mttr\_target

writes\_mttr : The number of times DBWn had to write, in addition to the writes it would normally have done, to meet the target mttr

**Checkpointing** is the process of forcing the DBWn to write dirty buffers from the database buffer cache to the datafiles. The CKPT process controls the amount of time required for instance recovery. During a checkpoint, CKPT updates the control file and the header of the data files to reflect the last successful transaction by recording the last system change number (SCN).

The checkpoint position is advanced automatically by the DBWn. This process is known as incremental checkpointing. In addition, there may be full checkpoints and partial checkpoints.

A full checkpoint occurs when all dirty buffers are written to disk. This will entail a great deal of work: very high CPU and disk usage while the checkpoint is in progress, and reduced performance for user sessions. Full checkpoints are bad for business. Because of this, there will never be a full checkpoint except in two circumstances: an orderly shutdown, or at the DBA’s request.

A partial checkpoint is necessary and occurs automatically as part of certain operations. Depending on the operation, the partial checkpoint will affect different buffers. These are

Taking a tablespace offline : All blocks that are part of the tablespace will Be Flushed from the Cache

Taking a datafile offline : All blocks that are part of the datafile will Be Flushed from the Cache

Dropping a segment : All blocks that are part of the segment will Be Flushed from the Cache

Truncating a table : All blocks that are part of the table will Be Flushed from the Cache

Putting a tablespace into backup mode : All blocks that are part of the tablespace will Be Flushed from the Cache

The LGWR process writes redo information from the redo log buffer to the online redo log files under a variety of circumstances:

When a user commits a transaction, even if this is the only transaction in the log buffer

When the redo log buffer becomes one-third full

When the buffer contains approximately 1MB of changed records; this total does not include deleted or inserted records

When a database checkpoint is performed

LGWR always writes its records to the online redo log file before DBWn writes new or modified database buffer cache records to the data files.

You can create and add more redo log groups to the database by using the ALTER DATABASE command. The following statement creates a new log file group with two members:

ALTER DATABASE ADD LOGFILE

GROUP 3 (‘/ora02/oradata/MYDB01/redo0301.log’,

‘/ora03/oradata/MYDB01/redo0302.log’) SIZE 10M;

If you omit the GROUP clause, Oracle assigns the next available number. For example, the following statement also creates a multiplexed group:

ALTER DATABASE ADD LOGFILE

(‘/ora02/oradata/MYDB01/redo0301.log’,

‘/ora03/oradata/MYDB01/redo0302.log’) SIZE 10M;

You can add more than one redo log group by using the ALTER DATABASE command—just use a comma to separate the groups.

**Chapter 16 Performing Database Backups**

User-Managed Consistent Backups (Also known as: closed/cold/consistent/offline)

Server-Managed Consistent Backups

An RMAN offline consistent backup has one crucial difference from an operating system offline consistent backup: it can be accomplished only when the database is in mount mode. This is because RMAN needs to read the controlfile in order to find the datafiles.

User-Managed Open Backups

An open backup with operating system commands is three steps:

■ Back up the controlfile with ALTER DATABASE BACKUP CONTROLFILE.

■ Copy the datafiles, while they are in backup mode.

■ Archive the online redo log files.

alter database backup controlfile to 'file\_name' ; -- It will be byte-for-byte the same as the live controlfile, and the snapshot mechanism ensures that it is consistent.

alter database backup controlfile to trace as 'file\_name' ; --The second command constructs a CREATE CONTROLFILE command and saves it in a script called file\_name. This script can be executed while the instance is in NOMOUNT mode to create a new controlfile, with the same contents as the original controlfile from which it was generated. This script is an ASCII file that can be viewed (and edited) with any test editor.

The datafiles can be copied with any operating system utility, but before making the copy they must be put into backup mode with an ALTER TABLESPACE command:

alter tablespace tablespace\_name begin backup;

This command has two effects. First, it forces a partial checkpoint: all the dirty buffers in the database buffer cache that contain blocks from the tablespace’s datafiles are written to disk.

Second, it adjusts the redo generation mechanism for changes made to blocks from those datafiles. This is necessary to manage the situation where a block is updated while it is being copied, and it requires some explanation.

Consider what actually happens if you copy a file with an operating system utility while the file is in use. Take an example where the operating system block size is 512 bytes (the default on many Unix and Windows systems) but the Oracle block size, as determined by the DB\_BLOCK\_SIZE parameter, is 8 KB. Each Oracle block will be sixteen operating system blocks.

If the file is many megabytes big, this copy will take several seconds or minutes to complete, and during that time it is more than likely that DBWn will flush some changed blocks from the database buffer cache to the datafile; the file will be changed while it is being copied. The granularity of the operating system copy is the operating system block, but the granularity of the DBWn write is the Oracle block. It is thus possible that the copy command will take the first part of an Oracle block, then DBWn will overwrite the whole Oracle block, and the copy command will then take the remaining part of the Oracle block. You have no control over this;

So in the output file produced by the copy, there will be what is called a fractured block: parts of it will be as of different versions. Such a block is completely useless; you have a data corruption in your backup.

If you are using operating system utilities to perform online backups, you avoid the fractured block problem by putting the tablespace containing the datafile into backup mode, with an ALTER TABLESPACE...BEGIN BACKUP command, for the duration of the copy. From that point on, whenever a server process updates a block in the database buffer cache, rather than writing out the minimal change vector to the log buffer it will write out the complete block image to the log buffer. This block image will be read-consistent.

Thus you accept that an online backup will have fractured blocks, but Oracle can repair them if necessary. The downside of this is that the rate of redo generation may accelerate astronomically while the tablespace is in backup mode: 8KB for each change, instead of just a few bytes. You may find that you are log-switching thirty times an hour rather than thirty times a day when you put tablespaces into backup mode, with a consequent drop in performance.

**Chapter 18 – Moving Data**

Architecturally, SQL\*Loader is a user process like any other: it connects to the database via a server process. To insert rows, it can use two techniques: conventional or direct path. A conventional insert, the SQL\*Loader user process constructs an INSERT statement with bind variables in the VALUES clause and then reads its source data file to execute the INSERT once for each row to be inserted. This method uses the database buffer cache and generates undo and redo data: these are INSERT statements like any others, and normal commit processing makes them permanent.

The direct path bypasses the database buffer cache. SQL\*Loader reads the source data file and sends its contents to the server process. The server process then assembles blocks of table data in its PGA and writes them directly to the datafiles. The write is above the high water mark of the table and is known as a data save. SQL\*Loader shifts the high water mark up to include the newly written blocks, and the rows within them are then immediately visible to other users. This is the equivalent of a commit; No undo is generated, and if you wish, you can switch off the generation of redo as well. For these reasons, direct path loading is extremely fast, and furthermore it should not impact on your end users, because interaction with the SGA is kept to a minimum.

Direct path loads are very fast, but they do have drawbacks:

■ Referential integrity constraints must be dropped or disabled for the duration of the operation.

■ Insert triggers do not fire.

■ The table will be locked against DML from other sessions.

■ It is not possible to use direct path for clustered tables. These limitations are a result of the lack of interaction with the SGA while the load is in progress.

SQL\*Loader’s Capabilities

• SQL\*Loader can read from multiple input files in a single load session.

• SQL\*Loader can handle files with fixed-length records, variable-length records, and stream-oriented data.

• Not only can SQL\*Loader read from multiple input files, but it can load that data into several different database tables, all in the same load session.

• SQL\*Loader allows you to use Oracle’s built-in SQL functions to manipulate the data being read from the input file.

• SQL\*Loader includes functionality for dealing with whitespace, delimiters, and null data.

• In addition to standard relational tables, SQL\*Loader can load data into object tables, varying arrays (VARRAYs), and nested tables.

• SQL\*Loader can load data into large object (LOB) columns.

• SQL\*Loader can handle character set translation between the input data file and the database.

sqlldr userid=scott/tiger@ocp11g control=emp.ctl data='C:\Saroj\StudyMaterial\Oracle\SQLLDRPractice\emp.csv' discard=emp.dis BAD=path\_file\_name

The bad file is where SQL\*Loader writes input records that cause errors. The discard file is where SQL\*Loader writes records that do not match conditions that you specify in your LOAD statement.

You can even mix the positional and keyword methods of passing command-line parameters. The one rule when doing this is that all positional parameters must come first.

Parameters read from a parameter file as a result of using the PARFILE parameter may override those specified on the command line. SQL\*Loader processes parameters from left to right, and the last setting for a given parameter is the one that SQL\*Loader uses.

Control file

LOAD DATA

INFILE 'mi\_deci.'

REPLACE

INTO TABLE gc\_gnis\_county

WHEN (feature\_type='school')

(

gc\_state\_abbr CHAR TERMINATED BY "," ENCLOSED BY '"',

gc\_feature\_name FILLER CHAR TERMINATED BY "," ENCLOSED BY '"',

gc\_feature\_type FILLER CHAR TERMINATED BY "," ENCLOSED BY '"',

gc\_county\_name CHAR TERMINATED BY "," ENCLOSED BY '"'

)

INTO TABLE gfn\_gnis\_feature\_names

WHEN (feature\_type='airport')

(

gfn\_state\_abbr POSITION(1) CHAR TERMINATED BY "," ENCLOSED BY '"',

gfn\_feature\_name CHAR TERMINATED BY "," ENCLOSED BY '"',

gfn\_feature\_type CHAR TERMINATED BY "," ENCLOSED BY '"',

gfn\_county\_name CHAR TERMINATED BY "," ENCLOSED BY '"',

visit\_date DATE 'dd-mon-yyyy' NULLIF = blanks TERMINATED BY "," ENCLOSED BY QUOTES,

id “seq.nextval”

)

You can use **Data Pump** in a variety of ways:

• Perform point-in-time logical backups of the entire database or subsets of data

• Replicate entire databases or subsets of data for testing or development

• Quickly generate DDL required to recreate objects

• Upgrade a database by exporting from the old version and importing into the new version

The worker process names have the format ora\_dwNN\_<SID>.

The name of the status table is dependent on what type of job you’re running. The table is named with the format SYS\_<OPERATION>\_<JOB\_MODE>\_NN, where OPERATION is either EXPORT or IMPORT. JOB\_MODE can be one of the following types:

• FULL • SCHEMA • TABLE • TABLESPACE • TRANSPORTABLE

e.g SYS\_EXPORT\_SCHEMA\_NN

$ impdp mv\_maint/foo directory=dp\_dir dumpfile=exp.dmp logfile=imp.log

userid=mv\_maint/foo

directory=dp\_dir

dumpfile=exp.dmp

logfile=exp.log

tables=inv

reuse\_dumpfiles=y

$ expdp parfile=exp.par

Same like sql loader

Exporting and Importing an Entire Database

When you export an entire database, this is sometimes referred to as a full export. In this mode the resultant export file contains everything required to make a copy of your database. A full export consists of

• all DDL required to recreate tablespaces, users, user tables, indexes, constraints, triggers, sequences, stored PL/SQL, and so on.

• all table data (except the SYS user’s tables)

A full export is initiated with the FULL parameter set to Y and must be done with a user that has DBA privileges or that has the DATAPUMP\_EXP\_FULL\_DATABASE role granted to it.

To initiate a full database import, you must have DBA privileges or be assigned the DATAPUMP\_IMP\_FULL\_DATABASE role.

When you initiate an export, unless otherwise specified, Data Pump starts a schema-level export for the user running the export job.

The following command starts a schema-level export for the MV\_MAINT user:

$ expdp mv\_maint/foo directory=dp\_dir dumpfile=mv\_maint.dmp logfile=mv\_maint.log

$ expdp mv\_maint/foo directory=dp\_dir dumpfile=user.dmp schemas=heera,chaya

When you initiate a schema-level import, there are some details to be aware of:

• No tablespaces are included in a schema-level export.

• The import job attempts to recreate any users in the dump file. If a user already exists, an error is thrown, and the import job continues.

• The import job will reset the users’ passwords, based on the password that was exported.

• Tables owned by the users will be imported and populated. If a table already exists, you must instruct Data Pump on how to handle this with the TABLE\_EXISTS\_ACTION parameter.

You can also initiate a schema-level import when using a full-export dump file.

You can instruct Data Pump to operate on specific tables via the TABLES parameter. For example, say you want to export

$ expdp mv\_maint/foo directory=dp\_dir dumpfile=tab.dmp tables=heera.inv,heera.inv\_items

A tablespace-level export/import operates on objects contained within specific tablespaces. This example exports all objects contained in the USERS tablespace:

$ expdp mv\_maint/foo directory=dp\_dir dumpfile=tbsp.dmp tablespaces=users

Transferring Data

One of the main uses of Data Pump is the copying of data from one database to another. Often, source and destination databases are located in data centers thousands of miles apart. Data Pump offers several powerful features for efficiently copying data:

• Network link - Allows you to take an export and import it into the destination database without having to create a dump file.

• Copying data files (transportable tablespaces) - Lets you copy the data files from a source database to the destination and then use Data Pump to transfer the associated metadata.

• External tables

$ impdp darl/engdev directory=engdev network\_link=dk \

schemas='STAR2,CIA\_APP,CIA\_SEL' \

remap\_schema=STAR2:STAR\_JUL,CIA\_APP:CIA\_APP\_JUL,CIA\_SEL:CIA\_SEL\_JUL

Copying Data Files

Oracle provides a mechanism for copying data files from one database to another, in conjunction with using Data Pump to transport the associated metadata.

This is known as the transportable tablespace feature. The amount of time this task requires depends on how long it takes you to copy the data files to the

destination server. This technique is appropriate for moving data in DSS and data warehouse environments.

SQL> exec dbms\_tts.transport\_set\_check('INV\_DATA,INV\_INDEX', TRUE);

SQL> select \* from transport\_set\_violations;

SQL> alter tablespace inv\_data read only;

SQL> alter tablespace inv\_index read only;

$ expdp mv\_maint/foo directory=dp\_dir dumpfile=trans.dmp transport\_tablespaces=INV\_DATA,INV\_INDEX

Copy the Data Pump export dump file to the destination server and Copy the data file(s) to the destination database. Place the files in the directory where you want them in the destination database server. The file name and directory path must match the import command used in the next step.

Import the metadata into the destination database. Use the following parameter file to import the metadata for the data files being transported:

userid=mv\_maint/foo

directory=dp\_dir

dumpfile=trans.dmp

transport\_datafiles=/ora01/dbfile/rcat/inv\_data01.dbf, /ora01/dbfile/rcat/inv\_index01.dbf

you can use the SQLFILE parameter to view the DDL associated with the tablespaces that were exported:

$ impdp mv\_maint/foo directory=dp\_dir dumpfile=inv.dmp sqlfile=tbsp.sql

When you use the SQLFILE parameter, nothing is imported. In this example the prior command only creates a file named tbsp.sql, containing SQL statements pertaining to tablespaces.

Specifying Different Data File Paths and Names

userid=mv\_maint/foo

directory=dp\_dir

dumpfile=inv.dmp

full=y

include=tablespace:"like 'INV%'"

remap\_datafile="'/ora03/dbfile/O12C/inv\_data01.dbf':'/ora01/dbfile/O12C/tb1.dbf'"

remap\_datafile="'/ora03/dbfile/O12C/inv\_index01.dbf':'/ora01/dbfile/O12C/tb2.dbf'"

This example remaps the user as well as the tablespace. The original user and tablespaces are HEERA and INV\_DATA. This command imports the INV table into the CHAYA user and the DIM\_DATA tablespace:

$ impdp mv\_maint/foo directory=dp\_dir dumpfile=inv.dmp remap\_schema=HEERA:CHAYA \

remap\_tablespace=INV\_DATA:DIM\_DATA tables=heera.inv

The following example exports only the procedures and functions that a user owns:

$ expdp mv\_maint/foo dumpfile=proc.dmp directory=dp\_dir include=procedure,function

exporting only specific PL/SQL objects,

directory=dp\_dir

dumpfile=ss.dmp

include=function:"='ISDATE'",procedure:"='DEPTREE\_FILL'"

Cloning a User

$ expdp mv\_maint/foo directory=dp\_dir schemas=inv dumpfile=inv.dmp

$ impdp mv\_maint/foo directory=dp\_dir remap\_schema=inv:inv\_dw dumpfile=inv.dmp

Exporting a Percentage of the Data

$ expdp mv\_maint/foo directory=dp\_dir tables=inv,reg sample=reg:30 dumpfile=inv.dmp

Estimating the Size of Export Jobs

To estimate the size, use the ESTIMATE\_ONLY parameter. This example estimates the size of the export file for an entire database:

$ expdp mv\_maint/foo estimate\_only=y full=y logfile=n

The following command takes a consistent full export of the database, using the FLASHBACK\_SCN parameter:

$ expdp mv\_maint/foo directory=dp\_dir full=y flashback\_scn=5715397 dumpfile=full.dmp

SQL> select current\_scn from v$database;

The TABLE\_EXISTS\_ACTION parameter takes the following options:

• SKIP (default if not combined with CONTENT=DATA\_ONLY)

• APPEND (default if combined with CONTENT=DATA\_ONLY)

• REPLACE

• TRUNCATE

The SKIP option tells Data Pump not to process the object if it exists. The APPEND option instructs Data Pump not to delete existing data, but rather, to add data to the table without modifying any existing data. The REPLACE option instructs Data Pump to drop and recreate objects; this parameter isn’t valid when the CONTENT parameter is used with the DATA\_ONLY option. The TRUNCATE parameter tells Data Pump to delete rows from tables via a TRUNCATE statement.

The CONTENT parameter takes the following options:

• ALL (default)

• DATA\_ONLY

• METADATA\_ONLY

The ALL option instructs Data Pump to load both data and metadata contained in the dump file; this is the default behavior. The DATA\_ONLY option tells Data Pump to load only table data into existing tables; no database objects are created. The METADATA\_ONLY option only creates objects; no data are loaded.

The next example instructs Data Pump to append data in any tables that already exist via the TABLE\_EXISTS\_ACTION=APPEND option. Also used is the CONTENT=DATA\_ONLY option, which instructs Data Pump not to run any DDL to create objects (only to load data):

$ impdp mv\_maint/foo directory=dp\_dir dumpfile=inv.dmp table\_exists\_action=append content=data\_only

Renaming a Table

Starting with Oracle Database 11g, you have the option of renaming a table during import operations. Note that this syntax doesn’t allow you to rename a table into a different schema.

$ impdp mv\_maint/foo directory=dp\_dir dumpfile=inv.dmp tables=heera.inv remap\_table=heera.inv:invent

Use the PARALLEL parameter to parallelize a Data Pump job. For instance, if you know you have four CPUs on a box, and you want to set the degree of parallelism to 4, use PARALLEL as follows:

$ expdp mv\_maint/foo parallel=4 dumpfile=exp.dmp directory=dp\_dir full=y

$ expdp mv\_maint/foo encryption=all directory=dp\_dir dumpfile=inv.dmp

Starting with Oracle Database 12c, you can specify that objects be loaded with nologging of redo. This is achieved via the DISABLE\_ARCHIVE\_LOGGING parameter:

$ impdp mv\_maint/foo directory=dp\_dir dumpfile=inv.dmp transform=disable\_archive\_logging:Y

One powerful feature of Data Pump is that you can attach to a currently running job and view its progress and status.

SQL> select owner\_name, operation, job\_name, state from dba\_datapump\_jobs;

$ expdp system/foobar attach=mv\_maint.sys\_export\_schema\_01

Interactive Commands

CONTINUE\_CLIENT : Continues with interactive logging mode

EXIT\_CLIENT : Exits the client session and returns to the OS prompt. Leaves the current job running

HELP : Displays the available interactive commands

KILL\_JOB : Terminates the job currently connected to in the client

PARALLEL : Increases or decreases the degree of parallelism

START\_JOB : Restarts a previously stopped job. START\_JOB=SKIP\_CURRENT restarts the job and skips any operations that were active when the job was stopped

STATUS : Specifies the frequency at which the job status is monitored. Default mode is 0; the client reports job status changes whenever available in this mode.

STOP\_JOB [=IMMEDIATE] : Stops a job from processing (you can later restart it). Using the IMMEDIATE parameter quickly stops the job, but there may be some incomplete tasks.

select name, object\_name, total\_bytes/1024/1024 t\_m\_bytes

,job\_mode

,state ,to\_char(last\_update, 'dd-mon-yy hh24:mi')

from SYS\_EXPORT\_TABLE\_01

where state='EXECUTING';