DBA101: A Refresher Course

Marlene Theriault and Rachel Carmichael

Abstract

There are many tasks that a database administrator (DBA) should perform on a routine basis. Often, ORACLE documentation is unclear or does not address these tasks well. New DBAs are either unaware that the tasks should be performed or do not know how to do them. Even senior DBAs sometimes are unaware of actions that they can take to make their job easier or more effective.

This paper is a collection of tasks and overview information that DBAs should know. The information was gathered by two DBAs over a period of several years and includes:

- Those things you do (actions that a DBA takes before and immediately after creating a database)
- Scripts to gather information to pro-actively monitor and document your database(s)
- Tips and traps that can cause a DBA problems and many wasted hours

Scope

This paper is intended to be an overview of some of the activities that a DBA might perform. We assume that the reader has at least an introductory level of knowledge about both ORACLE and database administration in general. Because of the limited amount of space allotted for a conference paper, only a subset of DBA tasks will be addressed here.

The majority of the ORACLE version 8.0.5 information within this paper is based on the delivered Windows NT and Digital Unix v8.0.5 software and may vary with your operating system version.

About Software Installation

Although specific information about installing the ORACLE-supplied software is out of the scope of this paper, we would like to briefly cover the steps that a DBA normally follows when he or she receives the ORACLE software CDROM disk and any hardcopy documentation (which is becoming more and more scarce).

When you get your software, the first thing you should do is mount the CDROM and look for any files with “readme” or “installation” in their names. This may be all of the installation documentation that you are going to receive and there can be very important information that you need to know BEFORE you begin to install the ORACLE software.

Verify that the operating system that you are using is the version – with all associated operating system patches installed – that is required for the specific version of the Oracle server software that you have received. Do not assume anything!! It’s very easy to believe that you are on version 4.0.d of an operating system and find out later that you are REALLY on version 4.0.b or that you need operating system patch number XXXXXXX in order to run Oracle’s version 8.X on your Unix, OpenVMS, or Windows NT system – and, yes, this just recently happened to one of us. Marlene was asked to help a group who was installing Oracle version 8.0.5 on a Digital Unix box. The link kept failing with two undefined symbols. After several hours, it was finally determined that the operating system level that was

installed was not the correct version required for this Oracle installation. Everyone assumed that the system administrator had upgraded the operating system.

Once you have verified that the operating system versions and patches are correct, read through the installation instructions to verify that your system memory and disk resources are adequate for the Oracle version you are going to use. Be sure to read through ALL of the instructions to ensure that you have the appropriate answers for each question for which you will be prompted. Decide before you begin the installation process whether you will want the Oracle installation to create a default database or not and, in later versions of the software, whether you want Oracle to link the appropriate software, install, configure and startup your listener and intelligent agent and install and configure a Names Server.

We believe that you should allow Oracle to create a small, default database during your software installation for the following reasons:

- If the database creation succeeds, you will have probably verified your installation as well as proven that a database CAN be built
- You will be able to "poke around in" the default database to see what roles, role composition, and default users (along with their privileges) are being created by Oracle
- You will have a model of the naming conventions and files that Oracle feels are important – although the sizing of these files may be unrealistic for your development or production needs
- You will have a small, test database that you can use as a "sandbox" to perform practice scenarios for disaster recovery and to perform sanity checks if things fail to work correctly in another database

If the options are presented to you, we recommend that you allow Oracle to link the "application software" and configure the listener and intelligent agent so that you will be able to see how Oracle performed these tasks. Whether or not you will want to use the Oracle Names Server is another issue (and probably another paper) but is outside the scope of this paper.

We suggest that you script everything you do – as you do it – so that you will be able to recreate your actions later. After the software has been installed and proven to work correctly (by creating a default database, etc.), you will want to create your own REAL database. Before you create your first database, decide how many copies of the control file and how many redo log files you are going to want. You will also need to determine the database block size for your database – the default from Oracle is 2k and that's generally just too small to be practical in anything other than a very small test database!

When you have completed creating your database, you will need to run the "CAT" scripts (CATALOG.SQL, CATPROC.SQL, CATEXP.SQL, etc.). When you get a chance, take the time to look at what the "CAT" scripts are used for. Be sure to always run CATALOG.SQL first and CATPROC.SQL second and then any of the other "CAT" scripts in whatever order you want. If you have already run CATALOG.SQL and CATPROC.SQL, you do not have to run them again to run any of the other "CAT" scripts.

Your next step is to create one small rollback segment in the SYSTEM tablespace. You will not be able to create any other objects or segments until you have created this rollback segment and brought it on line. Once you have created the SYSTEM rollback segment, you will be able to create the other tablespaces, rollback segments, etc. that you will want in your database. In the next section, we will look at how to determine what your base tablespace configuration should look like.

On the Number of Tablespaces and Their Layout

In 1991 at the International Oracle User Week conference, Cary Milsap presented a paper on a standardized approach to naming Oracle directories, placement of files on a system, and the amount and type of tablespaces that a basic database should have. The approach was developed to improve Oracle performance in earlier versions of the RDBMS and to enable consistency in database file location. Cary called his approach the "Optimal Flexible Architecture" (OFA).

Over time, Cary's concepts have proven to be an invaluable approach to configuring a database for optimal performance. Although his approach to directory structure and file-naming conventions is excellent, we'll concentrate more on looking at the part of his recommendations for database architecture. While performing research as he developed his concepts, Cary found that a database that has at least seven initial, basic tablespaces located on different disks – preferably on different controllers – will yield better performance. Here are the recommended minimum tablespaces that should be created for a database and the currently proposed naming conventions for them:

Table 1. Seven Basic Tablespaces

Tablespace Name	File Name
SYSTEM	SYSTEM01.DBF
RBS	RBS01.DBF
TEMP or TEMPORARY	TEMP01.DBF
TOOLS	TOOLS01.DBF
USERS	USER01.DBF
DATA	DATA01.DBF
INDEX	INDEX01.DBF

Now, let's take a look at each of these tablespaces and see what type of segments should be stored in each. At first glance, it would seem like the SYSTEM tablespace is self-explanatory – a tablespace to be used to store the objects that belong to the user SYSTEM. However, the SYSTEM tablespace should be used to store only those objects that belong to the user SYS. Once you have created the first five tablespaces listed in Table 1, we suggest that you alter the SYSTEM user to use the TOOLS tablespace as the default tablespace and the TEMP tablespace as the temporary tablespace. We also suggest that you alter the SYS user to point to the TEMP tablespace for sort operations. The commands you use are as follows:

```
alter user SYSTEM default tablespace TOOLS temporary tablespace TEMP;

alter user SYS temporary tablespace TEMP;
```

Why would you want to do this? To ensure that the only user who will be using the SYSTEM tablespace is SYS and that the SYSTEM tablespace will not be used to store temporary segments during sorting operations! Since, by default, Oracle creates users with their default tablespace as SYSTEM and their temporary tablespace as SYSTEM, as you create each new user in your database, you will want to ensure that you assign them a specific default tablespace and temporary tablespace. If you do not ensure that SYS is the only user assigned to the SYSTEM tablespace, that tablespace will:

- Tend to get really fragmented
- Will have potentially severe contention against it

Okay, so the only user in the SYSTEM tablespace is SYS. The RBS tablespace (and you can have more than one of these tablespaces) is used to specifically store rollback segments. If you have enough disks and controllers and high rollback segment contention, you might even want to have two or more sets of rollback segment tablespaces with the rollback segments spread evenly over them.

The next tablespace on the list is TEMP or TEMPORARY. As the name implies, this tablespace is assigned to each user as the "temporary tablespace" that will be used each time a sort operation is performed. What causes a sort operation? Any time a SQL statement includes "order by", "group by", "union", etc. operations, a sort will be performed. We want these sorts to be performed in a separate tablespace that will be used specifically to store temporary segments so that other tablespaces will not get fragmented. We have found that it is a really good policy to ONLY store temporary segments in a TEMP tablespace. Do not put any other objects in there – even if they are viewed as "temporary" objects. You can also have more than one temporary tablespace but each one should have a unique name that will help indicate with which application it is associated. If you have a payroll application in your database in which there is a great deal of sort activity, you might create an additional temporary tablespace named TEMP_PAYROLL and assign it as the temporary tablespace for the PAYROLL_USER schema. We'll talk more about temporary segments and tablespaces defined as "temporary" later in this paper.

The fourth tablespace on our list is the TOOLS tablespace. As we mentioned earlier, you will want to alter the SYSTEM default tablespace to point to the TOOLS tablespace. If you are licensed to use the Oracle Developer or Oracle Discoverer or Oracle Web Applications software or another Oracle product, you will install the base tables for these products either in their own schema area tablespaces or in the TOOLS tablespace. You can also use this tablespace for third-party vendor products that require a "SYSTEM" tablespace in which to house their objects.

Our fifth tablespace is USERS and can be used to provide an area where a user can create and store small objects such as test tables or the prototype of a new application under development. If the application begins to appear to be turning into a "real" application, you will want to move it from the USERS tablespace to an area of its own – a set of DATA and INDEX tablespaces.

If your database is going to be used for one very small application, you would assign your schema developer to the DATA tablespace and request that the developer use the INDEX tablespace to store indexes for the application. If you are going to house many applications or one large application in the database, you might want to create several tablespaces for data and indexes. The developer will have to name the specific index tablespace to be used in the index creation script since, by default, Oracle will place all objects created by a user in his or her default tablespace. In general, it is a good idea for the developer to specify the tablespace for each table or index he or she creates, even if they want it in their default tablespace. The DBA may decide to change the default tablespace of a user at any time and the object may not end up where it was intended to go.

Let's say that the PAYROLL application developer is working in the PAYROLL schema that was created with the statement:

```
create user PAYROLL identified by payme
default tablespace       PAYROLL_DATA
temporary tablespace TEMP_PAYROLL
quota unlimited on      PAYROLL_DATA
quota unlimited on      PAYROLL_INDEX
```

```
/
```

and was granted the Oracle-defined CONNECT role using the statement:

```
grant connect to PAYROLL
/
```

The creation statement for a basic table for the PAYROLL application might look like this:

```
create table PAY_SALARY (
 Empname      VARCHAR2(20),
 Empnum              NUMBER,
 Job_Title    VARCHAR2(10),
 Salary              NUMBER(9,2))
/
```

The creation statement for a basic unique index for the PAYROLL application might look like the following:

```
create unique index PAY_SALARY_IDX
on SALARY (Empname, Empnum)
tablespace PAYROLL_INDEX
/
```

Notice that the table creation statement does not have a tablespace explicitly named while the index does. By default, since the user was created pointing to the default tablespace PAYROLL_DATA any objects that the PAYROLL user creates with no "tablespace" statement will be created in the default tablespace. Notice also the granting of an amount of quota on each of the tablespaces to which the user will have access. Even though the user has been granted the ability to create tables through the CONNECT role, without quota being granted on a tablespace, the user could not create objects because there would be no place for them to be stored.

Instance Versus Database

What comprises an instance and what makes up a database? Although the terms are often used interchangeably, the words "instance" and "database" in the Oracle world have very specific meanings. Since nomenclature is often important, let's take a quick look at the meanings of each of these words.

An Oracle **instance** is comprised of the background processes that always consist of:

- SMON – the system monitor
- PMON – the process monitor
- DBWR – the database writer
- LGWR -- the log writer

and may include other processes like:

- RECO – the recovery manager for distributed databases
- ARCH – the archive log writer

- CHKR – the checkpoint writer

and any other detached processes used to support parallel servers or multithreaded servers, etc. and the System Global Area (SGA) which is a reserved shared memory area.

The **database** is the set of physical files in which all of the objects (tables, indexes, views, synonyms, procedures, programs, etc. and all of the metadata (data dictionary information) are stored. There is often a mix of logical and physical references when we talk about a database. The logical entities are the objects like tablespaces, tables, indexes, views, etc. while the physical entities are the actual datafiles that reside at the operating system level and the blocks allocated within those datafiles.

Those Things You Do

CATDBSYN.SQL

An Oracle database uses memory-resident structures known as "x$" tables to store information dynamically while the database is opened. Oracle provides "v$" performance tables for the DBA to view to help in performance tuning. The metadata is stored in the Oracle data dictionary and has three levels of information available. The three levels of information available are:

- USER_ – Any objects that the user personally owns
- ALL_ – Any objects to which the user has access
- DBA_ – All objects in the database (reserved for users who have been granted DBA privileges)

In an Oracle version 7.3 or earlier database, after you create the database, run the "CAT" scripts, create the tablespaces and rollback segments, you will need to log on to the SYSTEM account and run the script CATDBSYN.SQL that is generally located in the $ORACLE_HOME/rdbms/admin directory. This script creates the data dictionary views so that the DBA can easily see the database metadata. Each user who will be performing DBA tasks and has DBA privileges granted to him or her will also need to run this script from their account.

In Oracle version 8.0.2 and higher, the approach used by Oracle is slightly different. Anyone who has the DBA role assigned will automatically be able to "see" the data dictionary views.

PUPBLD.SQL

If you've ever logged onto a newly created database version earlier than v8.0.5, you may have gotten a message stating that the "product user profile does not exist" with a notation to run PUPBLD.SQL. What's going on here? Why did you get this message?

From the early days of Oracle databases, you, as the DBA, have had the ability to block a user's access to the database by any method other than through an application. You could prevent a user from logging directly into SQL*Plus and could block their ability to "Host" out to the operating system level. You could even block their use of products like SQL*Forms and SQL*ReportWriter, etc. When a user attempted to connect to the database using SQL*Plus or one of the early products, Oracle would check a table called the PRODUCT_USER_PROFILE table to determine if the user was allowed to perform the action requested. If Oracle went to check the table and it did not exist, the warning would be issued that the table did not exist and Oracle would assume that, because there was no PRODUCT_USER_PROFILE

table, there was no requirement to block SQL access and the user would be permitted to log on to the database from the SQL*Plus level. Of course, each user received the error message each time he or she logged on to the database and you would receive irate calls from the users. If you were lucky, you either remembered to run PUPBLD.SQL or you were the first person to log on to the database and the error message was enough to "spur you to take action" and you'd run the script. Once the table was created in the database and the appropriate views were created, you no longer got the error message because Oracle now had a table that it could check to verify if the user should be permitted to access the database through SQL*Plus or any other designated Oracle product.

Over the years, the complexion of what you can block and what you can't block has changed. However, the need to run PUPBLD.SQL in your newly created database has remained until version 8.0.5 when Oracle started to automatically run this script when it creates a default database. If you manually create your database, you will still need to run PUPBLD.SQL. You can use this utility to block access to SQL, PL/SQL, and SQL*Plus commands including blocking one specific user or an entire group of users from being able to reach the operating system level from within an application or from the SQL*Plus command line. This utility is a way to enhance your database security but should be used with caution since you might inadvertently block someone's access when that access is really required for the person to perform his or her job.

HELPINS.SQL

Up until Oracle version 8.0.5, if you wanted to provide online information about SQL and SQL*Plus commands, you could run the script HELPINS.SQL to build the information area. You could then just type "help <subject>" at the SQL*Plus command line and receive information immediately. You can still do this with versions of Oracle earlier than 8.0.5. We have both found this feature very helpful and appropriate to use in a development environment. We do no recommend that you install the help utility in a production database since no one should be doing development there and there is a certain amount of storage resource used with the utility.

If you type "help" in an Oracle version 8.0.5 database on a Windows NT platform, you will receive the following message:

```
SQL> help

SQL*Plus Help Files are included in Oracle Documentation.

You have to read SQL*Plus Help Files from Oracle Documentation.
```

About the Demo Tables

When you create a database, there are several scripts that are run – either by Oracle or by you. If you accept Oracle's offer to build a default database (and even if you don't), we recommend that you take some time immediately after creating the database to examine the DBA_USERS area to determine what default users are created there. You can do this by logging on as SYSTEM/MANAGER and typing:

```
select username

from DBA_USERS

/
```

The results from a default database created in version 8.0.5 on a Windows NT system were:

```
USERNAME
```

```
------------------------------
SYS
SYSTEM
ORDSYS
DEMO
DBSNMP
SCOTT
MDSYS

7 rows selected.
```

After you have determined the usernames, you will want to find out what privileges each user has been granted in your database. Most of the users listed above have been granted CONNECT and usually RESOURCE privileges. The MDSYS user was not granted any Oracle-supplied roles but, instead, has been granted 90 system privileges and owns 21 procedures. The problem with the default users created here are two-fold:

1. The RESOURCE role that is supplied by Oracle explicitly grants UNLIMITED TABLESPACE privilege to the user who has been granted this role. This privilege means that a user can override any assigned quotas and place objects anywhere in the database that he or she desires – even the SYSTEM tablespace.
2. The default password – that may not be permitted to be changed – is very easily guessed. For example, the DEMO password is "demo" and the MDSYS password (with 90 system privileges available) is "mdsys". The only passwords that differ from the usernames are SYS (change_on_install), SYSTEM (manager), and SCOTT (tiger) that are all well documented and well known.

You will need to determine if the default users that have been supplied are necessary in your production database and if their passwords can be changes. SYS, SYSTEM, and SCOTT can all be safely modified and we strongly urge you to do so as soon as possible after your database has been created. The DBSNMP user is used by the intelligent agent to perform tasks within your database from a remote console. In earlier versions of Oracle, the DBSNMP user has been granted the DBA role and may or may not be able to work correctly if you change its password. This can pose a severe security problem.

What We Do Daily:

General Monitoring

The Alert Logs

When your database is created, Oracle also creates a log, known as the "alert log" in which it writes information about:

- Each time the database is started
- Any recovery that the was performed on database startup

- Each time the database is shut down
- Each time a log switch occurs (referred to as a "thread switch")
- What parameters (from the init.ora file) that were used within the database at startup – that are not default parameters
- Any DDL commands that change the structure of the database – like "alter tablespace PAYROLL_TS add datafile …"
- Many errors that occur – usually with an error number of "600" or "602" – and the trace file name and location where the errors have been more fully documented

If the alert log "disappears" from the directory – as in the case of the file being renamed, when Oracle needs to write any entry to the alert log, a new log will be created with the default alert log name. We have a cron job that runs nightly and renames each of our database alert log files to the current date with the database name as the extension. The routine then emails the log to our email account so that we can quickly scan through the file to verify if anything looks like a problem area such as a 600 error occurring or too frequent thread switches, etc. Some DBAs have created routines that check for errors and email them only if an error is found. We prefer to check the files manually to ensure that our redo logs are not switching too frequently and to verify that the database writer is not being overwhelmed – that would indicate that there are not enough and/or not large enough redo logs on our system.

LISTENER.LOG

The SQL*Net listener generates a log when it is started and writes to the log – either with a great deal of information or minimally depending on what level of logging has been enabled. With the minimal message information enabled on a Net8 listener, the information that is provided shows when the listener was started, the ports being listened on, the machine on which the listener is running, etc. A sample entry looks like the following:

```
TNSLSNR80 for 32-bit Windows: Version 8.0.5.0.0 - Production on 25-SEP-98 14:24:30
(c) Copyright 1997 Oracle Corporation.  All rights reserved.
System parameter file is C:\orant\NET80\admin\listener.ora
Log messages written to C:\orant\NET80\log\listener.log
Listening on: (ADDRESS=(PROTOCOL=ipc)(PIPENAME=\\.\pipe\oracle.worldipc))
Listening on: (ADDRESS=(PROTOCOL=ipc)(PIPENAME=\\.\pipe\ORCLipc))
Listening on: (ADDRESS=(PROTOCOL=ipc)(PIPENAME=\\.\pipe\EXTPROC0ipc))
Listening on: (ADDRESS=(PROTOCOL=nmp)(PIPENAME=\\MYOWN-PC\pipe\ORAPIPE))
Listening on: (ADDRESS=(PROTOCOL=tcp)(DEV=136)(HOST=128.299.5.45)(PORT=1521))
Listening on: (ADDRESS=(PROTOCOL=tcp)(DEV=124)(HOST=128.299.5.45)(PORT=1526))
Listening on: (ADDRESS=(PROTOCOL=tcp)(DEV=144)(HOST=110.0.0.1)(PORT=1521))
TIMESTAMP * CONNECT DATA [* PROTOCOL INFO] * EVENT [* SID] * RETURN CODE
```

If you enable logging at a more intensive level, more information will be written to the log but more overhead traffic will be incurred. We recommend you periodically check your LISTENER.LOG for errors or when you seem to be encountering SQL*Net problems. Note that earlier versions of SQL*Net will give different log information. The only way to remove a too-large listener.log is to stop the listener

process, delete the log file and restart the listener process. If you rename the log file, Oracle will continue to write to the renamed file.

Monitoring Extents

For versions of Oracle earlier than 7.3.3, we, as DBAs, had to keep track of how many extents had been allocated in a table or index. We feared that we would "blow extents" and receive one of the dreaded error messages with ORA-01650 through ORA-01655 with the message "unable to extend …". Let's talk a bit about what values we see for the parameter MAXEXTENTS and why there's been a problem with these values. Let's also talk about why we still should be monitoring what our MAXEXTENTS are set to and what our actual extent values are.

When you create a default Oracle database, what is the block size used by Oracle? If you answer "2k", you are correct. Now, how many rows of header information can a 2k block hold? If you answer "121", you are right again. So, for a 2k block size database, what is the maximum number of extents that you can have prior to version 7.3 of the RDBMS? If you answer "121", you are really on a roll! Hmm. Do we see a correlation here? Of course! Now, if a 2k block size database can support 121 rows of header information and 121 MAXEXTENTS per object, how many extents can a 4k database support? The answer is 249. For an 8k block size, the maximum number of extents is 505. As you increase the database block size, the number of maximum extents that you can have increases.

In version 7.3, the software has been modified to enable the header block to also extend so you can actually have as many extents as your operating system will support and you can create your table with MAXEXTENTS UNLIMITED instead of stating a number if you so choose. However, if you do not specify a MAXEXTENTS value when you create a table and you have not specified a MAXEXTENTS value when you created the tablespace where the table will be stored, the MAXEXTENTS value that will be assigned for the table will be the default maximum extent limit based on the block size for that database. For example, let's create a tablespace in an 8K block size database as follows:

```
SQL> create tablespace mY_TS
  2  datafile 'mydisk1:[Oracle.mydb]my_ts01.dbf' size 10M
  3  default storage(initial 500k next 500k
  4              minextents 1)
  5  online;

Tablespace created.
```

We have not specified a MAXEXTENTS value. Now, let's create a table and see what the default value for MAXEXTENTS will be.

```
SQL> create table MY_TAB (Ename varchar2(20), Empno number);


Table created.
```

Next, we'll select the values from user_tables and see what our MAXEXTENTS value is by default.

```
SQL> select Table_Name, Initial_Extent, Max_Extents
  2  from USER_TABLES
  3  where Table_Name = 'MY_TAB';

TABLE_NAME                     INITIAL_EXTENT MAX_EXTENTS
```

```
---------------------------- ------------- -----------
MY_TAB                                524288         505
```

We see that the value for MAXEXTENTS is equal to the default value for an 8k block size database.

To monitor the growth of tables in a database, you can use the following script:

```
prompt CHECKING FOR FRAGMENTED DATABASE OBJECTS:
prompt
column Owner noprint new_value Owner_Var
column Segment_Name format a30 heading 'Object Name'
column Segment_Type format a9 heading 'Table/Indx'
column SUM(Bytes) format 999,999,999 heading 'Bytes Used'
column COUNT(*) format 999 heading 'No.'
break on Owner skip page 2
ttitle center 'Table Fragmentation Report' skip 2 -
  left 'creator: ' Owner_Var skip 2
select a.Owner, Segment_Name, Segment_Type, SUM(Bytes), Max_Extents, COUNT(*)
from DBA_EXTENTS a, DBA_TABLES b
where Segment_Name = b.Table_Name
having COUNT(*) > 12
group by a.Owner, Segment_Name, Segment_Type, Max_Extents
order by a.Owner, Segment_Name, Segment_Type, Max_Extents
/
ttitle center 'Index Fragmentation Report' skip 2 -
  left 'creator: ' Owner_Var skip 2
select a.Owner, Segment_Name, Segment_Type, SUM(Bytes), Max_Extents, COUNT(*)
from DBA_EXTENTS a, DBA_INDEXES b
where Segment_Name = Index_Name
having COUNT(*) > 12
group by a.Owner, Segment_Name, Segment_Type, Max_Extents
order by a.Owner, Segment_Name, Segment_Type, Max_Extents
/
```

In this set of scripts, we check for any table or index with a number of extents exceeding 12. You can, of course, set your threshold to whatever value makes sense for your environment. The idea is to keep track of those tables that are growing and may reach a point where they will exceed the amount of extents allocated for them.

Monitoring Object Modifications

You can check on changes to database objects using the data dictionary views USER_OBJECTS or DBA_OBJECTS. Why do you care when objects were last changed? If you are running a production system, you

want to control when changes are made and who makes them. You can use auditing to monitor who has made use of a privilege that changes an object, but auditing won't tell you which object was changed. The DBA_OBJECTS and USER_OBJECTS views contain a column called Last_DDL_Time. This column contains the time that the object was last modified. However, ORACLE considers adding grants or an index a modification to the object, so this field doesn't really reflect the last time the object itself was changed. To find this out, look in the Timestamp column of the DBA_OBJECTS or USER_OBJECTS view. This field is varchar2(75) so you will need to convert it to a date. Set up a batch job to run once a day that runs the following SQL script:

```
col Owner            format a20
col Object_Name      format a30
col Timestamp        format a20
select Owner, Object_Name, Object_Type, Status, Timestamp
  from DBA_OBJECTS
 where SUBSTR(Timestamp,1,10) = TO_CHAR(sysdate-1,'YYYY-MM-DD')
 order by Owner, Object_Name
/
```

This script will list all objects that have changed since the prior day, in owner order. The status column is included because procedures, functions and packages are objects and can become invalid

when changed.

Documenting Your Database

If a disaster were to occur today, would you know where every one of your files in each of your databases was? Could you be sure that the list of files you have today will be current and valid next week? Here is a script that you can run from a nightly batch or cron job that will produce a report of the location of all of your database files. You can set it up to run for each of your databases. We recommend that you set the job up and have it print the results to a printer so that you will have a hard-copy document of your file locations on a daily basis to use for disaster recovery and database documentation.

```
set pages 999
col file_name format a45
col tablespace_name format a20
col bytes format 9999999999
col blocks format 9999999999
col member format a38
col group# format 99999
set head off feedback off termout off
col name format a10
column dbname new_value xdb noprint
column today new_value tdy noprint
select
```

```
 substr(sysdate,1,9) today
from dual
/
select
  value dbname
from
  v$parameter
where
  name = 'db_name'
;
spool &tdy..&xdb
select 'Datafile Information for '||name||' - '||sysdate
from v$database, dual;
prompt
prompt
set head on feedback on termout on
select * from v$database;
prompt
prompt
select a.group#, a.member, b.bytes
  from v$logfile a, v$log b
 where a.group# = b.group#;
prompt
prompt
col value heading 'CONTROL FILE INFO'
select value from v$parameter
where name like '%control%';
prompt
prompt
select tablespace_name, file_name, bytes
from dba_data_files
order by 2,1;
spool off
exit
```

This script creates a file with the name in the form of “DD-MON-YY.<database_name>” like 10-OCT-98.MYDB and produces a report like the following:

```
Datafile Information for MYDB - 09-OCT-98
NAME       CREATED            LOG_MODE     CHECKPOINT_CHANGE# ARCHIVE_CHANGE#
---------- -----------------  ------------ ------------------ --------------
```

```
MYDB       11/21/97 08:22:17  NOARCHIVELOG              14274          14144
1 row selected.
GROUP# MEMBER                                      BYTES
------ ------------------------------------ -----------
     1 MYDISK01:[ORACLE.MYDB]ORA_LOG1.RDO       3145728
     2 MYDISK01:[ORACLE.MYDB]ORA_LOG2.RDO       3145728
     3 MYDISK01:[ORACLE.MYDB]ORA_LOG3.RDO       3145728
3 rows selected.
CONTROL FILE INFO
----------------------------------------
MYDISK38:[ORACLE.MYDB]ORA_CONTROL1.CON, MYDISK39:[ORACLE.MYDB]ORA_CONTROL2.CON,
MYDISK40:[ORACLE.MYDB]ORA_CONTROL3.CON
1 row selected.
TABLESPACE_NAME      FILE_NAME                                        BYTES
-------------------- ------------------------------------------- -----------
MY_TS                MYDISK41:[ORACLE.MYDB]MY_TS01.DBF             10485760
SYSTEM               MYDISK01:[ORACLE.MYDB]ORA_SYSTEM.DBF          62914560
RBS                  MYDISK702:[ORACLE.MYDB]RBS01.DBF             314572800
TEMP                 MYDISK703:[ORACLE.MYDB]TEMP01.DBF            314572800
TOOLS                MYDISK704:[ORACLE.MYDB]TOOLS01.DBF           157286400
USERS                MYDISK704:[ORACLE.MYDB]USERS01.DBF           157286400
6 rows selected.
```

About Roles (To role or Not to role)

Roles are collections of privileges that can be granted to users. Why use roles rather than grant privileges directly? Let's look at an example. If you have an application with 100 objects, and you have 100 users, you need to make 10,000 grants (100 grants each for 100 users). If one user leaves, you have to revoke 100 grants to maintain security. And if you add a new object, you need to grant access to it to 100 users. Magnify this by multiple tables and multiple users and you end up spending all your time just granting access to objects. Now complicate this even further by assuming that your users fall into different categories, and each category of user should have access to only some of the objects. How do you ensure that users get access to only those objects that they should?

To help control this problem, ORACLE introduced roles. Rather than grant privileges to users directly, you grant privileges to a role and then grant the role to users. Now, when you add a new object, you only have to grant access to the role and all users who are granted that role automatically inherit that access. And when a user leaves, you don't have to revoke access at all.

Sounds perfect, doesn't it? Before you rush out and create roles for everything and grant them to all your users, there are a few drawbacks to using roles of which you should be aware. First, roles cannot own objects. Why is this important? Well, if you do a "select distinct object_type from dba_objects" one of the object types that is returned is SYNONYM. So roles can't own synonyms. That means that you either have to hard-code the owner of the object into every object access (scott.emp instead of emp) or you

have to create public synonyms. And the problem with public synonyms is that if you have two applications in your database, and each one has a table named HOLIDAYS that contains different data for each application, you can't create the same public synonym name for the two tables.

Second, you can't create stored procedures, functions or packages using the access privileges you get from a role. So all your developers will have to have access granted directly. Because of this restriction, it's generally a good idea in production to have a single userid that owns all the objects and the procedures that access the objects.

About the Authors

Marlene Theriault has been an Oracle DBA for over 15 years and is Senior Oracle DBA at The Johns Hopkins University Applied Physics Laboratory. She has been published in several magazines and conference proceedings throughout the world. Ms Theriault tied for first place with Steve Feuerstein and received the “Outstanding Speaker” award at ECO’98 and has received the “Distinguished Speaker” award for presentations at both ECO’95 and ECO’96. She is responsible for re-activating the DBA Special Interest Group for the Mid-Atlantic Association of Oracle Professionals (MAOP), is current Chair of that group, and publishes an “Ask the DBA” column for the MAOP newsletter. Her book, co-authored with William Heney, is **Oracle Security**, published by O’Reilly and Associates, October, 1998, ISBN# 1-56592-450-9.

Rachel Carmichael has presented at various Oracle conferences including ECO and IOUW. She chairs the DBA SIG for the New York Oracle Users Group. Her book, co-authored with Kevin Loney, is **Oracle SQL & PL/SQL Annotated Archives**, published by Oracle Press, September, 1998, ISBN# 0-07-882536-9.