DBA Best Practices from the Field

Arup Nanda

Introduction

The very mention of best practices evokes mixed reactions – a huge glimmer of hope shadowed by a hint of doubt. A map to chart the course set by the best practices gives the hope to anyone at any level of expertise; but it's the doubt that dampens the enthusiasm. The primary reason of the doubt is the way the notion of best practices is promulgated. Many so called "experts" either manufacture these best practices or reiterate those already there. Of course, there are genuine experts who make the rules and guidelines based on their experience and thorough analysis; but unfortunately these experts are in minority. Most of the best practices have been products of empirical opinions or, worse, rules of thumb, which has just been passed around. Their legitimacy stems from only one fact – someone has told them somewhere, it has been repeated by several other people and *hence*, it must be true. These could be inadequate or irrelevant advice or just downright wrong.

I often question these so called "best practices" for their relevance in real life situations. Almost always the questions bring out a deeper understanding of the issue the practice is supposed to solve; and in many cases, the underlying issue, well, is not an issue at all. Sometimes, the issue may be real; but the supposed solution does nothing to address it. In many other cases, the solution is inadequate or plain wrong. As a lead DBA in many corporations I am expected set some guidelines and provide guidance for others. Needless to say, I'm often challenged with why a best practice should or should not be followed and also come up with some practices of my own. These best practices stem from my readings; but more as a product of working as a production support and architect DBA for a long period of time. This paper is the summary of some of those practices that cater to "real life".

What Makes a Practice Best?

I have two simple rules of thumb, when considering something as a best practice.

It Must be Justified

Anyone promoting anything as a best practice must be able to justify it. The fact that it has been around, or that it has been told by someone important in some very important conference are not the criteria for being considered a best practice. If you come across someone stating a best practice, always question, politely but firmly, why that is so. If you get the answer that essentially tells you not to question the authority without an explanation, most likely it's an urban myth. Even if the best practice is said to be a quote from an acknowledged expert, such as Jonathan Lewis and Tom Kyte (whose words, by the way, should be treated as gospel), may still be misinterpreted, leading to false practices. Again, the speaker should have understood that quote before making a statement and should be able to justify. Sometimes, the original quotes were correctly portrayed; but they may have lost the relevance in course of time and in the new version. For instance, locally managed tablespace is a great best practice; but not in a RAC environment, where ASSM may be great.

It Has to Be Situation Aware

A best practice must be able to adapt to the situation. Very few best practices universal. For instance, I saw this on a book on best practices:

Run the database in archivelog mode

Why? Because it offers the possibility media recovery up to the very moment of failure. That part is true; but what about the performance penalties? Running a small OLTP database in archivelog mode is one thing; but running a 64TB datawarehouse is another. In the latter case, you may be better off recreating the data from the source system instead of from the backup. So, a different backup strategy such as taking disk level copies might be a better alternative to RMAN hot backup. So, a blanket statement such as above is not generally a best practice. It has to be specific to a situation; and it has to offer pros and cons. It's somewhat tied to the previous requirement.

So, that being said, let's see what these best practices are.

(1) Set Audit Trail to DB

Here is a question. Is it true or false: setting audit_trail to DB in the initialization parameter file will start auditing in the database and increase I/O?

Most people tend to think it is true; actually it's false. You need to issue AUDIT statement to start auditing. For instance to start auditing SELECT statements on table EMP, you need to give the command:

```
SQL> audit select on emp by access;
```

The audit_trail initialization parameter merely specifies where the trail should be written, how much information and in what format the information is written. But if the parameter is not set to anything explicitly, AUDIT statement on the table EMP does not do any thing. When you want to start auditing, you need to set this parameter to DB, DB_EXTENDED (10gR1), OS or XML (in 10gR2).So, what's the problem?

This parameter, unfortunately, is not a dynamic one. You can't use ALTER SYSTEM command to set this; you need to recycle the database for this to take effect. So, even if you are not planning on auditing now, you need to set it when you do plan to; but unfortunately, you have to bounce the database at that time. There is no problem in leaving it set to DB; and it will save you a recycle later, when you plan to use auditing. Oracle 11g already has this set to DB by default.

(2) Think outside the "OFA"

Oracle Flexible Architecture (OFA), introduced some 14 years ago has been somewhat of a gold standard in database layout. When it was introduced, there was a good reason for following it. At least it provided a common layout for the files which allowed with new DBAs to get acclimatized with the layout faster. In summary, it requires you define an Oracle Base directory such as /u01/app/oracle. Under that, it specifies that you define the following directories.

➥admin/SID/bdump – for background dumps
➥admin/SID/udump – user dumps
➥oradata/SID/datafiles – datafiles
and so on.

When you use Oracle Installer to create a starter database or use DBCA to create a database, it follows the OFA layout for the different directories. So, it has been best practice for some time.

But things changed, as all things do. Although it's still a good idea; OFA may not be the right choice in all cases. So, this is not a universal best practice. Some of the issues with the OFA plan are due to the lack of separation of the filesystems:

- All directories branch out from the same source - the base, which makes it difficult to lay them out on different types of storage. For instance, the datafiles should be on a high performance disk while the others such as the Oracle binaries can be on cheaper and slower disks.
- Security is another consideration. Non-DBAs may want access to the user dump directory where the trace files are created; and you can't give then permission to go there unless all the parent and grand-parent directories are allowed execute permission. This is not very desirable.
- You may want to keep the Oracle Home on cheaper and available free space on the internal disks in the server while the datafiles may be on more expensive SAN. Similarly, archived logs can be on cheapest storage, perhaps NAS.

Sometimes, databases are set up with active-passive clusters. When the main host fails, the filesystem gets unmounted and then gets remounted on the passive server. But the passive host may already have another Oracle Home (such as that of a QA system). So, you don't want to failover the entire Oracle Base; just the datafiles.

So, OFA may not be the right solution for the database systems. Here is an alternative strategy.

Create a base directory: /oracle. Under that create a directory admin/SID, under which create bdump (background), adump (audit), cdump (coredumps), pfile (parameter file) and so on. Do not create user dump destination directories here. Make the permissions of the directory /oracle and below no privs for "others".

Another directory /u01/udump/SID/* is defined as user dump destination and is allowed free for the users.

A third types of directory /prddata1/SID/datafiles is used for datafiles. This probably requires high performance

disks as well as high security – this directory should have no permissions for anyone other than oracle. Create more than one filesystem to spread it over more than one disk spindles, e.g. /prddata2, /prddata2 and so on. You may want to place undo, redo and control files on separate mount points to isolate disk and I/O paths. Some of the datafiles containing historical may not need to be on high performance disks. These have to be on a different set of disks, and need a different filesystem.

Finally, the archive logs need to be on a different filesystem, due to the crucial nature of the contents – the archived logs. If you ever perform media recovery, you need to have the archived log. If an archived log is missing, the recovery will stop right there even if all other archived logs after that are available. So, you may want to put them on disks that are more reliable.

Note the mount point naming convention –/prddata, instead of /u01. This allows the filesystem to be mounted to a new host where QA database could be running, which may be on filesystems named /qadata. If you had followed the convention of /u01, this would not have been possible.

As you can see, it pays to rethink the option of following OFA for the filesystem layout. Of course, you can create the directories as per the OFA conventions and create symbolic links to all the actual directories. This approach works; but makes it complicated to follow which content goes where and might prove to be worse than just referring the directories directly.

(3) Different Oracle Homes

How do you apply a patch to the Oracle Home? You probably follow the general steps as shown:

1. Shutdown the database
2. Patch oracle software
3. Bring up the database
4. If the patch has some database component, run any scripts that need to be run, e.g. catalog.sql and catproc.sql.

Now, consider this: in step 2, something goes wrong. The patch application did not go well. Of course there is a patch backout option in OPatch; but it may or may not do a clean uninstallation. You run the risk of corrupting the database software.

A second problem is timing. Between steps 1 and 3 the database remains unavailable. Depending on the patch or patchkit, this outage could be substantial and unacceptable.

So, here is a best practice –modify the process a little bit. Name your Oracle Homes in the following manner - /u01/app/oracle/product/10.2/db1. Note how the directory has been named “db1”, instead of just “db”. When you are ready to apply a patch, do not apply it to this Oracle Home. Instead, create another home, called /u01/app/oracle/product/10.2/db2 (note “db2”). Install the same software in db2 as it is in db1. Then patch “db2” home, not “db1”. Since the running database is on home “db1”, this application could be done even when the database is running. When the outage time comes, just shutdown the database, change ORACLE_HOME to db2 and startup the database.

If something goes wrong in the process, you can simply shutdown the database and reset the ORACLE_HOME to db1 and startup as usual.

Follow this for all the patch applications. When the next patch is applied, create another home, called “db3” and install everything you had done in db2 and then the new patch. And follow the process for all the patches. You may also reuse the home “db1” for this patch instead of creating a brand new home. Let’s see an example illustrating this approach:

1. Brand new install with Oracle 10.2.0.2 software. Install it on /u01/app/oracle/product/10.2/db1.
2. Then a patch 1868560 comes out that needs to be applied.
3. Once again, install the base 10.2.0.2 software on the directory /u01/app/oracle/product/10.2/db2.
4. Apply the patch to this new Oracle Home
5. When the outage time comes, shutdown the database
6. Execute ORACLE_HOME=/u01/app/oracle/product/10.2/db2. Set the appropriate PATH and LD_LIBRARY_PATH as well.
7. Startup the database
8. If something goes wrong in the patch application, your original Oracle Home is still intact. You can set the OH variable to that home and restart the database.

What Are The Benefits:

This keeps the existing software intact, reducing the risk of failed patch

1. The database need not be down during the patch application time, which could be considerable.
2. Since the database is not down during patch application, you can do it at a time that is less stressful rather than at a crunch time.
3. Patch undo is as trivial as changing the Oracle Home, for non-database patches. This is perfect for CRS and ASM patches.
4. Since you have copies of both database software versions, you can perform a comparison to identify differences, if any. This is a lifesaver if the new patchkit corrupts some binary. You may be able to get the binary from the old home.

Therefore, best practice is to install the patch on a new Oracle Home each time. If you have the space, keep on adding new Oracle Homes instead of reusing them.

(4) Keep ASM Home Separate from Database

If you are using Oracle RAC, you already know that you need to install the CRS software on a new home. But there is no such requirement for ASM. ASM software is part of the RDBMS codebase; so if you have installed Oracle database software, you already have ASM code in it.

However, it's a best practice to install a separate Oracle Home for ASM software alone and run ASM from that home. So, you may have two homes:

- /u01/app/oracle/product/10.2/db1 for the Database software
- /u01/app/oracle/product/10.2/asm1 for the ASM instance alone.

So, you will perform the following

```
$ export
ORACLE_HOME=/u01/app/oracle/product/10.2/
asm1
$ . oraenv
$ sqlplus / as sysdba
SQL> startup
```

Once ASM comes up, use the following to startup the database:

```
$ export
ORACLE_HOME=/u01/app/oracle/product/10.2/
db1
$ . oraenv
$ sqlplus / as sysdba
SQL> startup
```

The content of ASM home and DB home might be identical; but by putting them in separate homes, you eliminate a lot of risk. Sometimes patches come out for RDBMS but not applicable for ASM. If you have a common home, the patches are applicable to both of them, which may cause instability in ASM instance. Oracle software is known to be 100% error free (not even close to that); so a patch not designed for some product is not something you want to apply. Similarly, there may be a ASM specific patch, which is applied to the ASM home only; not to the RDBMS home. Most likely, both homes will identical as most patches are applicable to both; but that is not always the case.

The downside of this approach is you have to have double the space and your patch application time is now increased as you have to patch both the homes. But you can apply them in parallel, which makes the elapsed time the same as a common home. If you follow the new home approach suggested elsewhere in this paper, you can reduce the downtime even further.

(5) Set Parameters at Start

Some initialization parameters found in the Oracle database are static and not modifiable by an ALTER SYSTEM. If you want to modify them, you will have to bounce the database. Many of these parameters are not needed to be changed after the database creation; but some stand out a possibility. There are two in particular that may be required and I recommend setting them to non-default value. These are:

1. _trace_files_public=true

   By default, it's set to FALSE, which makes the trace files generated in user dump destination directories set to no permission for others. Trace files are usually requested by developers to diagnose performance issues and sooner or later you will a request to have access to these files. Rather than copying these to some other common developer-accessible location, why not just let them grab it directly from the location where it gets generated? By setting this parameter to TRUE, the files will be

generated as publicly readable, and developers can directly access them.

2. utl_file_dir='/tmp'

   The OS file manipulation utility in Oracle is implemented through the supplied package utl_file. The package opens the file by specifying a directory. However, this directory must be mentioned in the parameter utl_file_dir. If this parameter is not specified in the initialization parameter file, then you can't call utl_file to open a file in that directory. Since DBAs didn't know in advance what directories will be opened by developers, they usually put a "*" in there, a wild card that denotes all directories readable by oracle user. Nothing can be more dangerous than that. A malicious developer can overwrite database files since they are owned by oracle user.

From Oracle 9i onwards, the directory does not have to be a physical directory; but can be a directory object. This eliminated the need to have utl_file_dir parameter and most DBAs do not place it. However, unfortunately, one program still needs it – the Logminer. If you want to create a dictionary flat file, Logminer uses the OS directory name rather than the directory object name. If you haven't defined the parameter, you wil not be able to use that feature, unless you recycle the database. To avoid this possibility, set this parameter to some harmless directory such as /tmp. Once set, you can use Logminer without bouncing the database.

(6) Kill Inactive Sessions

Here is a typical problem: some applications, especially web apps using connection pool, remain inactive for considerable amounts of time. Some considerate middleware teams set the parameters in such a way that the inactive sessions in the pool are disconnected automatically. What if you don't have such a partnership? The inactive sessions keep draining resources.

Suggestion

A very simple solution is to use Resource Manager and set the inactive session disconnect timeout. This automatically disconnects the sessions that have been inactive for a period of time specified by you. Even if you don't use Resource Manager for anything else, this is a good use of the tool.

You may be wondering, why RM, why not use user profile, which also has a inactive session disconnect feature?

The answer is simple. Profiles are rather static. You attach a profile to a user, not to a session. Once attached the profile remains attached to the user unless you disassociate it; or turn off profiles using resource_limit = false in the initialization parameter file. On the other hand, RM allows you to turn on and off via scheduling and based on some events. You can even temporarily disable resource manager based limits.

The second benefit is that RM allows service name based control. So, a user SCOTT maybe subject to a resource plan while using a service SN1; but another resource plan while connecting through service name SN2. Profiles are attached to user SCOTT and therefore the same limit applies in any case.

(7) Check Listener Log

Listener log files contain valuable information about the database connections such as which clients are trying to connect but failing, and what service names they are using. A lot of information is captured anyway there, even if auditing is not turned on. However, the information source is a text file; how can you effectively use it.

The answer: create External Tables on listener log. Once created this way, you can extract the information using the best language you know – SQL. Rather than explaining the mechanics here, I will direct you to read a three-part series I wrote for DBAZINE.com at: http://www.dbazine.com/oracle/or-articles/nanda14

(8) Service Names

An Oracle database can be accessed via either SID or Service Name. Here is how a conventional TNS entry using Service Name looks like:

```
prodb1 =
  (DESCRIPTION =
   (ADDRESS_LIST =
    (ADDRESS = (PROTOCOL = TCP)(HOST =
prolin1)
       (PORT = 1521)))
       (CONNECT_DATA = (SID = PRODB1))
  )
```

When you use service name, the connect data section simply changes to:

```
(CONNECT_DATA = (SERVICE_NAME = PRODB1)))
```

Here we are assuming you are using the default service name – the database name itself. That is not something you want if you want to switch over to service names.

Enable Service Names

In the instance, check service names present already:

```
SQL> show parameter service_names
```

This will show service names already defined. To create additional service names:

```
SQL> alter system set service_names =
'SVC1', 'SVC3', 'SVC3';
```

Check is listener is listening for these:

```
$ lsnrctl services
```

In RAC, you should use SRVCTL to add services:

```
$ srvctl add service –d MYDB –s SVC1 ...
```

Why Service Names?

- First of all, service names bring no change in functionality; but they enable separating use from user, e.g. user SCOTT logging from laptop uses service SVC1; but from app server SVC2. you may want to set up different resource plans for each type of service. For instance, SVC2 (from app server) may be allowed 50% of the CPU; but SVC1 is allowed only 20%.
- It allows the load balancing and failover in RAC or Data Guard databases
- Allows fine grained failover capabilities. For instance you can say, service SVC1 fails from node1 to node2; but SVC2 fails to node3.

(9) Analyze CPU and IO Usage

Auditing in database is expensive because it involves additional I/O. So, we need to have the biggest bang for the buck, i.e.. accomplish the most with little auditing and that is Session Auditing. To enable session auditing, simply issue:

```
SQL> audit session;
```

It records when users connect to and disconnect from the database. So, one record gets created in audit trail when the user connects. When the user disconnects, the same record is updated with a whole lot of information, such as logical and physical I/O used, CPU used and so on, in addition to the disconnect time. Since there are only two activities associated with a session – one insert and the other update, the I/O impact is low. On the other hand, the information captured is a gold mine. Let's see how.

Purpose

You can calculate CPU consumption for the users per day as shown below:

```
select username,
to_char(logoff_time,'mm/dd') ts,
   count(1) cnt,
   sum(session_cpu) sum_cpu,
   avg(session_cpu) avg_cpu,
   min(session_cpu) min_cpu,
   max(session_cpu) max_cpu
from dba_audit_trail
where logoff_time between '&start_date'
and '&end_date'
group by username,
to_char(logoff_time,'mm/dd')
order by username,
to_char(logoff_time,'mm/dd')
```

Here is the output.

USERNAME	TS	CNT	SUM_CPU	AVG_CPU	MIN_CPU	MAX_CPU
USER1	04/04	3	918	306	17	859
USER2	04/04	36	15,286	425	0	4,094
USER3	04/04	3	794	265	174	379
USER4	04/04	187	396,299	2,119	1	124,274

This output tells you which users are consuming how much CPU. In addition to just getting a list of users that hog CPUs, you get valuable insight into planning for the capacity of the server. For instance, if your server is 65% utilized now and you see that user USER1 is consuming CPU at an increasing rate, then you know who to ask for the increased usage. This information also helps to decide how to configure the resource manager.

You can also use the report to calculate the IO used by the users. Here is an example

```
select username,
to_char(logoff_time,'mm/dd') ts,
  sum(logoff_lread) lread,
  sum(logoff_pread) pread,
  sum(logoff_lwrite) lwrite,
  sum(session_cpu) scpu
from dba_audit_trail
where logoff_time between '&start_date'
and '&end_date'
group by username,
to_char(logoff_time,'mm/dd')
order by username,
to_char(logoff_time,'mm/dd')
```

Here is the output:

```
USERNAME        TS            LREAD        PREAD       LWRITE         SCPU
--------------- ----- ------------ ------------ ------------ ------------
USER1           04/04       283,271       10,858           33          918
USER2           04/04     4,570,965        6,225        2,854       15,286
USER3           04/04       601,838        1,988           26          794
USER4           04/04    33,639,028    4,545,505    1,083,473      396,299
```

From the output you can see the logical reads, writes, physical reads and CPU used in the session for the users. This type of information allows you to understand how a system is used as far as I/O is concerned and how you can do capacity planning for future I/O.

This information can also be used to identify if someone's account was locked after repeated wrong passwords. You can issue:

```
select to_char(timestamp,'mm/dd/yy
hh24:mi') ts,
  os_username, userhost, returncode
from dba_audit_trail
where username = 'ARUP'
order by timestamp;
```

Here is the output:

```
01/10/07 14:12  arupnan         CORP\UPNANT             0
01/10/07 15:12  arupnan         CORP\UPNANT             0
01/11/07 04:01  orandsp         hndspdb1             1017
01/12/07 04:02  orandsp         hndspdb1             1017
01/13/07 04:03  orandsp         hndspdb1             1017
01/14/07 04:04  orandsp         hndspdb1             1017
01/15/07 04:00  orandsp         hndspdb1            28000
```

The output clearly shows that the user ARUP tried to login at 4:01 AM on 1/11/07 but got an error: "ORA-01017 Invalid Username/Password", which shows up in the above output with 1017 in the last column. After four attempts, the account was locked and the user got the error: "ORA-28000 Account is locked", which shows up as in the audit trail as the last record. Using this you can track how the account was locked.

(10) Audit DDL

How many times you were paid a visit by someone who asks what happened to his/her table? If you cast an incredulous look that shouts out "how the heck do I know", you were probably rewarded with an equally bewildered look "You are the DBA and you are saying you don't know what happened to it?"

To prevent such unpleasant moments, just turn on auditing to track all DDL statements. It's conceivable that many statements in the database will be DML, not DDL; so this action will not overwhelm the audit trail but provide important information later. Suppose you want to audit table creation, truncations and drop, you will issue:

```
SQL> audit table by session;
```

This tracks all those statements. To find out all the statements you can issue, check the table stmt_audit_option_map in the SYS schema.

Another shortcut is the statement: AUDIT ALL BY SESSION, which audits most DDLs.

There is a little caveat: in DW environments, users create and drop a large number of tables; so this may not be advisable.

(11) OS Specific Tweaks

Many DBAs do not pay attention to the scope of tuning possibilities in the specific platforms. For instance, in HP/UX, you can set the initialization parameter sched_noage to a low value to set the priority for the user. You can also make "dba" group part of MLOCK to take advantage of the asynch I/O. Similarly on Solaris you can use Intimate Shared Memory. The effect of these OS tuning is sometimes dramatic and you should explore the possibility of such tuning.

(12) Redo is Not a LOG

A very common practice many DBAs follows is naming the redo log files with an extension ".log"; even DBCA does it. There are two problems in that case:

Sometimes you may want to run a script or a program to delete log files via *cron*, e.g. sqlnet.log. Naming the redo log file with .log extension makes it likely for it to be deleted by mistake.

A listener attack that can change the listener log to redo1.log using the command set log_directory and set log_file commands. If the redo log file name is different, then listener attack will fail.

The best practice is to choose the extension .redo or .rdo for redo log files.

(13) To ASSM, or not?

Automatic Segment Space Management (MSSM), a new way to manage free space in the block, has been around since 9i days. It uses bitmap of free space on the block; so a session trying to fit more data does not need to check the UET$ table, as it does in case of Manual Segment Space Management (MSSM) extent management. It's great for performance and eliminates or reduces some waits like buffer busy waits.

But, bitmap states are only for full, 25, 50 and 75% free and then empty block. So it can potentially lose up to 25% space on each block.

So, it's a good idea to use:

- ASSM for non-DW databases
- MSSM for DW databases

Buffer busy waits not common on DW anyway; so using MSSM may not cause a problem.

(14) Aliases

Aliases make some repetitive job faster and quicker. Here is a set of common aliases I use:

```
alias bdump='cd
$ORACLE_BASE/admin/$ORACLE_SID/bdump'
alias pfile='cd
$ORACLE_BASE/admin/$ORACLE_SID/pfile'
alias obase='cd $ORACLE_BASE'
alias tns='cd $ORACLE_HOME/network/admin'
alias oh='cd $ORACLE_HOME'
alias os='echo $ORACLE_SID'
```

(15) Protect the Listener

Restrict Parameter Changes

Did you know that you can change the location of the listener log file by issuing the SET LOG_DIRECTORY command at the LSNRCTL prompt?

This is a favorite tactic by hackers to change the listener log to the redo log directory and corrupt the redo log files. To prevent such an event, simply place this parameter in the listener.ora file.

```
admin_restrictions_listener=on
```

This prevents online modification of the listener parameters. To change a parameter, you have to modify the listener.ora file and then reload the listener.

```
$ lsnrctl reload
```

This does not stop the listener; just loads the new parameters into memory. So your applications will not be affected.

Different Listener for External Procedures

External procedures are particularly vulnerable to attacks. Do you use external procedures? If so, they are placed in the listener.ora file as well. If you do not use the external listener, you should remove the entry from the listener.ora file. The entry is placed by default.

If you use external procedures, always create a new listener for them alone and let them run on a different port. This allows you to be more restrictive on that port and add more security in the external procedures.

(16) Build a Metadata Repository

How do you know when some structural modification occurred in the database? For instance a tablespace was added, a table was added, someone changed the tablespace parameter, a user was added and so on? Once way to go through the change control tickets to figure out the changes – a rather painful and error prone way. The other way is to carefully document every time the change occurs. Both options are not particularly attractive.

A much easier option, in Oracle 10g is to use Data Pump to build a repository automatically:

```
$ expdp u/p content=metadata_only full=y
diectory=tmp_dir dumpfile=md.dmp
```

Import this to create an SQL File

```
$ impdp u/p diectory=tmp_dir
dumpfile=md.dmp sqlfile=md.sql
```

This creates the DDLs of all objects in the database in the file md.sql. You can create a separate file everyday and perform a comparison to see if something changed. For more information and several other tips for Data Pump, see my paper: Datapump: Not Just for Data Movement.

(17) Validate Database

Many people are not aware of a powerful command in RMAN – the VALIDATE command, which allows you to check the integrity of the database components such as backupsets, archived logs and even datafiles. Here is how you use them for archived logs:

```
RMAN> backup validate database archivelog all;
```

Then check for corrupt blocks in view v$database_block_corruption. This allows you to proactively check for archived log errors that might prevent a failure in recovery later.

To detect logical corruption, you can use a slightly different command:

```
RMAN> backup validate check logical
database archivelog all;
```

This command checks for logical errors in the archived logs. As a best practice, always check for consistency and integrity of the archived logs after they are created. Remember, if an archived log is bad, you can't recover beyond that point. So, if you detect a corruption in the archived log, immediately take an incremental backup (or a full one, if you take only full backups).

(18) Preview RMAN Restore

You have been taking database backups diligently; but do you know if all the backups are available? That's a very critical question: what good are the backups if they are missing? How can you make sure all the backup files are available without actually restoring it?

Another under-utilized and relatively unknown option in RMAN is the PREVIEW clause. The clause can be added to any restore command. Here is an example:

```
RMAN> restore tablespace users preview;
```

It does not actually restore but checks the availability of files that will be used in the recovery. If a file is not present, it errors out, alerting you.

It's not the same as the VALIDATE command. Preview checks for the availability of the files that are required; validate assumes you know that and validates the physical and logical integrity of those files. So, they are complementary in function.

It's not the same as TEST clause, either. Test clause expects the tablespace (or the datafile) to be offline:

```
RMAN> restore tablespace users test;
```

Preview does not actually start the recovery process; so the tablespace need not be offline. In a production database that's the best option.

Here is an example of the preview clause:

```
RMAN> restore tablespace users preview;

List of Datafile Copies
Key     File S Completion Time Ckp SCN    Ckp Time        Name
------- ---- - --------------- ---------- --------------- ----
173716  238  A 30-MAR-07       62872433554 30-MAR-07       /f.rman
... And so on ...
173775  2074 A 31-MAR-07       62918498516 31-MAR-07       /j.rman

no backup of log thread 1 seq 92170 lowscn 62872343042 found to restore
... And so on ...
no backup of log thread 1 seq 92173 lowscn 62902345362 found to restore

List of Archived Log Copies
Key     Thrd Seq     S Low Time  Name
------- ---- ------- - --------- ----
92212   1    92174   A 30-MAR-07 /PROPRD1_1_92174_525355299.arc
... And so on ...
92239   1    92201   A 01-APR-07 /PROPRD1_1_92201_525355299.arc
Media recovery start SCN is 62872433554
Recovery must be done beyond SCN 62948207913 to clear data files fuzziness
Finished restore at 06-APR-07
```

The output clearly states the files that will be required for restore and what archived logs are needed and which ones are not found. All this is done without actually restoring anything. Performing this in regular intervals allows you to check if all the backups are available. The best practice is to run this after every backup.

(19) Save the RMAN Log

If you take RMAN backups, I'm sure you backup those backuppieces to tape or some other offline media. But do you copy the RMAN log files? I bet you don't.

RMAN Logs contain information about the backup pieces, names, location, etc.,
which proves invaluable during recovery. Here is one snippet from the RMAN log file:

```
input datafile fno=00084 name=/f1.dbf
output filename=/backup/loc3/data_D-
CRMPRD_I-79785763_TS-DWT_ODS8_RES_FN
O-96_43ie2scm.rman tag=FULLBKPFS
recid=174298 stamp=618757792
```

The logs show filename and the backuppice associations. The same information can also be retrieved from catalog or the controlfile; but what if the controlfile is lost? If all is lost, you can fall back on the logfiles that allows you to look for specific files from backup sets. It also tells you the controlfile backups should you decide to get them later.

It's not difficult to backup the RMAN logfiles; but it may save you a ton of headache later. So, a best practice is save the RMAN log file to tape as a part of the backupsset and separately on a tape.

(20) Auto-Document DBID

Do you know the DBID of the database? You probably don't. Who would? The number is pretty meaningless except under some circumstances during recovery. The database ID (DBID) is very important for recovery. Of course, if you have the current controlfile, you need not know the DBID; but in some cases of recovery, you need to know the DBID. Where do you find it?

As a best practice, note the DBID and keep it in a separate place somewhere; but don't stop there. Document the DBID every time you get a chance. What is the best place to document it? How about the alert log?

Write DBID to alert log every time backup is taken. This simple PL/SQL code does the trick:

```
declare
  l_dbid number;
begin
  select dbid into l_dbid from
v$database;
  dbms_system.ksdwrt(2,'DBID='||l_dbid);
end;
```

You can call the code segment after the backup is completed; or put in a scheduler job to be called at regular intervals.

(21) Do Not Use SPFILE

You have heard about SPFILE, which is a pseudo-binary version of the initialization parameter file – init.ora (or PFILE). It has been sort of a rule of thumb to use spfile in lieu of pfile, which then transformed into a best practice. Let's see the pros and cons of using spfile:

SPFILE Advantages:

- Can be on shared filesystem, very useful for RAC databases, where only one SPFILE is necessary.
- It can be created on ASM
- Can be backed up automatically by RMAN
- Can be updated by command line by ALTER SYSTEM SET … SCOPE = SPFILE;

But Here Are The SPFILE Disadvantages:

- The biggest issue is that spfile has only version of a parameter. If you change the parameter, there is no way of knowing what the value was earlier, who changed it and, most important, why.
- Comments are possible; but only for the current entry. Once the entry is overwritten, the older comments are gone.

Instead of using spfile, if you use pfile, you have a chance to record the comments that led to the changes. I recommend placing extensive comments on what was changed, why and who changed it, along with anything that may be considered relevant. Here is a sample from the pfile:

```
# AKN 3/20/06 added because ...
# RJN 4/10/06 changed from 1M to 2M
# JER 10/3/06 changed from 2M to 4M
# DFW 12/7/06 changed from 4M to 6M SR# …
log_buffers = 6M
```

As you can see, it has a history of changes, with the names and dates of changes and reason for the change. In one case the DBA, whose acronym is DFW also put a SR# (TAR#) on the pfile as well.

In case of descriptive parameters, such as log_arch_dest_1, do not update the line; rather comment that line and put a new line with the new value. Here is an example of what not to do. Suppose the original line was `log_archive_dest_1 = '/u01/arch'`, which you want updated. You may have been following the practice of updating the loine in the file as:

```
# updated DFW 9/12/07
log_archive_dest_1 = '/arch/u01'
```

Instead, use this line:

```
log_archive_dest_1 = '/u01/arch'
# updated DFW 9/12/07
log_archive_dest_1 = '/arch/u01'
```

This maintains a history of the changes. It is very useful for troubleshooting purposes. When a problem occurs you may be able to draw a connection between the changes and the problems. If you spfile, this is not possible. So, I recommend not using spfile, even though it offers all those advantages mentioned earlier.

If you must use spfile, then make sure you have a good version control system in place to track parameter changes. Since the spfile is binary (sort of), it's difficult to track changes as you would be in case of a text file. So one option is to use strings function to extract the text portion of the spfile and compare them. Other option, more preferred one, is to convert the spfile to pfile, e.g.

```
SQL> create pfile='/tmp/a' from spfile;
```

This creates a file called /tmp/a as a pfile which you can use. Now you can use this file to check differences between this and the previous spfile. Write the differences to a log file; update the logfile with the name and other details of the changes and save it. This becomes your version control system. Needless to say, it becomes convoluted and highly error prone. Instead the pfile offers none of those risks.

In Oracle 11g, you can create PFILE from memory, which allows you to create a pfile from the parameters that has been already changed:

```
SQL> create pfile='...' from memory;
```

You can create an spfile from memory as well.

(22) Use ORAENV

The ORAENV is an Oracle supplied tool, found in $ORACLE_HOME/bin directory. The purpose of this tool is to set the appropriate parameters such as PATH and ORACLE_HOME. The ORACLE_SID and ORACLE_HOME values are defined in the file `/etc/oratab` (`/var/opt/oracle/oratab` in Solaris). If you have multiple Oracle Homes on the server, it makes it easier to switch from one to the other.

Why use this tool instead of using your own script that has lines something like this:

```
export ORACLE_HOME=…
export ORACLE_SID=…
```

The purpose is simple. Oraenv provides a consitent interface and functionality. Regardless how many environments you deal with or how many other DBAs you have to interact with, this is a common framework for setting variables. And the best of all, it's already there; you don't have to write the script. It makes your job easier while changing Oracle Home, especially in jobs and commands

(23) New Oracle User for Clients

In many cases the server is exclusively used as one for Oracle database and no client is allowed to run on it. Sometimes you may have other client running there who just need access to client tools like SQL*Plus and SQL*Loader. Since the Oracle Home already has these tools, you may be inclined to set the Oracle Home and PATH of the client to the existing Oracle Home and client.

But that also causes a huge security issue. App running on the DB server needs SQL*Plus; but the $ORACLE_HOME/bin/sqlplus is not executable to others. So there are two solutions:

1. Change $ORACLE_HOME permissions to execute by all others. This allows any app to call the sqlplus or sqllldr executables. But this is hardly desirable since any user on the server can access any executable in Oracle Home.
2. Make app part of the "dba" group, which of course does not need the execute by others permission set; but now the app, being in the dba group can startup or shutdown the database. Again not a very good option.

Instead, I recommend creating a separate user: "appora". Install the Oracle client under that user under a different Oracle Home. This makes the Oracle software running the database different from the client; so there is no risk to one from the other. This also conforms to many security and privacy regulations and mandates.

(24) Separate Instance and DB Names

When you create a single instance database, you probably keep the name of the instance the same as the database. For instance, if the name of the database is MYDB, you call the instance MYDB as well. In a RAC database, you have to keep the same database name; but different instance names. In a single instance, you don't have to:

This is not wrong; but I recommend separating the names of database and instance. You can follow any convention as you find fit. Here is an example: append "1" after DB name for the instance name. If the DB name is PRODB, the instance can be PRODB1, even though this is a single instance database.

Why? If you ever need to convert the DB to a RAC one, you will not need to change the instance name. You simply create the other instances as PRODB2, PRODB3 and so on. Since you are not changing the instance name, you don't have to create instance name specific files such as pfile (or spfile), password file, background dump destiinations and so on. Of course you can make the change when instance name changes; but it just makes it easier.

(25) Uniform-sized Raw Devices

Do you use raw devices (or raw logical volumes) for datafiles? In some cases such as some RAC databases, or databases requiring high I/O performance you may be using raw devices or volumes. Typically, the raw devices are created as per the space requirement in the datafiles. For instance, if you need a 100GB tablespace, you create a raw device /dev/rdsk/users01.dbf of 100GB and then create a tablespace USERS with the raw device as the datafile. When the tablespace USERS needs more

room, you typically expand the raw device and then resize the device.

The practice works; but it needs coordination among DBAs and Sys Admins. This may not be real challenge; but there some additional issues that makes this unattractive under any circumstance:

1. You have to know precisely how much each tablespace would be and create raw devices accordingly.
2. When the raw device is resized, it has to be offlined; so the tablespace is not available during that time. In a true production database that may not be possible; at least it's not desirable.
3. When you need to shrink the file, you can; but the raw volume is already allocated. You can't put two files on the same raw volume; so you can't reuse the space freed up by the first datafile.
4. When you want to drop the tablespace, you can't drop the raw volume. You can reuse the raw volume for another tablespace but the name will be mismatched leading to confusion. For instance you don't want the datafile of tablespace USERS be named system01.

So, as a best practice, you should create uniform size raw volumes (or devices), say, of 30GB each. Name them as generically as possible without any tablespace name references, e.g. /dev/rdsk/d1, /dev/rdsk/d2, etc. When you need to create a tablespace, use datafiles as the devices d1, d2 and d3. When USERS tablespace needs more room, add a new device to the tablespace instead of expanding it. This is done without any outage. When you drop the tablespace, all these raw volumes get freed up and can be used in any other tablespace. The names are generic – d1, d2, etc.; so they will not mismatch with the tablespace name.

So, in summary, the advantages are:

- No outage while adding room
- Reuse of devices possible
- No need to have sys admin involved in the process.

(26) Archivelog Location

Here is a simple question: rate the most critical of the three below:

- Datafiles
- Archivelogs
- Backup of datafiles

All three are critical. But if you had a choice to pick the only one you need to preserve, which one will you choose? The most critical one is archivelog. If datafiles are lost, they can be recreated, provided you have backup. If you lose the backup, you can still recreate the datafile using the archivelog. If archived logs are lost, you have no progress option; you can't recreate the archived logs. You can recover up to that point only.

This is a fact you should careful consider in your storage and fileystem design. If you are short on space and you have to cut corners somewhere such as making RAID0 diskgroup instead of RAID5, you know better not to put the archived logs on those disks. In many cases I see people putting archived logs on those disks.

A related best practice is directly contrary to what Oracle Corp usually pushes for – the use of Flash Recovery Area (FRA). If you use FRA, do not place your archived logs there. If you lose the datafile backup and archived log backup on FRA, make sure the archived logs are on a separate location on a different set of disks. This allows you to recover the database from either the archived log or the backup of the archived log.

(27) Create a Controlfile on Trace

You can run the following SQL statement on a database anytime:

```
SQL> alter database backup controlfile to
trace as '/path/cr_db.sql' reuse;
```

It creates a CREATE CONTROLFILE script as /path/cr_db.sql. This file is used to recreate the controlfile by a SQL script. But you can use this to recreate the database. Of course, you do not create the controlfile or the database everyday; but this file serves a very important purpose. It documents the components of the database such as datafiles, redo log files, database parameters, controlfiles etc. This documentation comes extremely handy when you need to recreate the datafiles (or tempfiles) after the recovery process ends. In any case you have created a documentation of the system. Since you used the "reuse" option, the same gets overwritten every time; so you don't create a large number of files.

Case for Change Control

As a special case, you can write a separate file for each day, instead of using the "reuse" option. Then, you can

perform a "diff" operation to find out what changed. It's crude but effective and simple.

(28) Using ORADEBUG

The database is not responding and you want to take some dumps to analyze the results or kill some processes. You can use the ORADEBUG tool. But, alas! The database is hung. How can you connect to the database to run the tool?

When SQL*Plus does not work, use the `prelim` flag.

$ sqlplus –prelim

It does not establish a connection. So if the database is hung, it will still work. You can run ORADEBUG now. Here are some tips for dumping different components of the database.

1. To dump a data block b of datafile d:

```
alter system dump datafile d block b;
```

2. To dump the other components, here is the command:

```
alter session set events 'immediate trace
name <Key> level 10';
```

The value of *<Key>* depends on what you want to dump, which is as follows:

To dump this:	**<Key> should be:**
Controlfile	`CONTROLF`
File Headers	`FILE_HDRS`
Redo Headers	`REDOHDR`
System State	`SYSTEMSTATE`
Process State	`PROCESSSTATE`
Library Cache	`LIBRARY_CACHE`

For instance, to dump the library cache, use the following statement:

```
SQL> alter session set events 'immediate
trace name library_cache level 10';
```

Conclusion

In conclusion, I want to reiterate the characteristics of the best practices:

1. It's not a best practice, if it is not justified
2. You have to understand why; not just what
3. Best practice needs to be situation-aware, which goes back to "you have to understand"

Always question whenever someone tells you it's a best practice and understand why it is so. If this is not justified, it may not be a best practice.

About the Author

Arup Nanda has been an Oracle DBA for more than 13 years working in all aspects of database management from modeling to performance management and disaster recovery. He is the author of books Oracle PL/SQL for DBAs (O'Reilly 2005) and Oracle Privacy Security Auditing (Rampant 2003). He was awarded the DBA of the Year by Oracle in 2003.