10 PROBLEMS WITH YOUR RMAN BACKUP SCRIPT

Yury Velikanov, The Pythian Group
Michael S Abbey, The Pythian Group

INTRODUCTION

The authors have been called to audit RMAN backup scripts on a regular basis for several years now as part of their day to day duties. They see the same errors in scripts that Oracle DBAs use to backup critical databases over and over again. These errors may play a significant role in the recovery process when you are working under stress. This paper gives you very practical advice on how to improve your RMAN backup scripts and recovery procedures, and how to get ready to perform a possible recovery and minimize backup and recovery related risks.

We should mention that this isn't just the authors' work. Many Oracle DBAs at Pythian and in the community reviewed and contributed to several points we are providing in this paper. The authors have discussed typical issues with many Oracle DBAs while working on this paper, and therefore it is a common work by many experienced IT professionals. We hope this work will add an immediate value to your backup and recovery practice.

10 PROBLEMS WITH YOUR RMAN BACKUP SCRIPT

While it is practically impossible to list all of the problems that Oracle DBAs face managing Oracle backup and recovery procedures, we mention our top 10. Your environment may be relatively unique; therefore, we do not intend to discuss specific cases but more typical and generic issues. This makes this paper more valuable for many Oracle DBAs. Let's start with a simple topic such as log files.

#1 RMAN LOG FILES

It is noticed that quite often Oracle DBAs don't pay necessary attention to what information is provided in RMAN log files. It doesn't take much time and effort to ensure that the information provided is reasonably detailed. An additional bit of information may provide vital input at troubleshooting or recovery time. Have a look at the following output.

part of a log file ...

```
RMAN>
Starting backup at 18-OCT-11
current log archived
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=63 device type=DISK
channel ORA_DISK_1: starting compressed archive log backup set
channel ORA_DISK_1: specifying archive log(s) in backup set
input archive log thread=1 sequence=4 RECID=2 STAMP=764855059
input archive log thread=1 sequence=5 RECID=3 STAMP=764855937
...
Finished backup at 18-OCT-11
```

Did you notice any issues? How about comparing it with the output below?

part of a log file ...

```
RMAN> backup  as compressed backupset database
2> include current controlfile
```

```
3> plus archivelog delete input;

Starting backup at 2011/10/18 12:30:46
current log archived
allocated channel: ORA_DISK_1
channel ORA_DISK_1: SID=56 device type=DISK
channel ORA_DISK_1: starting compressed archive log backup set
channel ORA_DISK_1: specifying archive log(s) in backup set
input archive log thread=1 sequence=8 RECID=6 STAMP=764856204
input archive log thread=1 sequence=9 RECID=7 STAMP=764857848
...
Finished backup at 2011/10/18 12:33:54
```

We are sure that while most of you recognize value of setting the right date format before you call the RMAN executable, to reflect hours/minutes/seconds rather than leaving the default, many of you may miss the *set echo on* option. Let's go through each point one by one.

SET NLS_DATE_FORMAT

If you leave the default date format you will see just Year/Month/Day in the log file. The simple *export NLS_DATE_FORMAT="YYYY/MM/DD HH24:MI:SS"* command just before the RMAN call gives us vital timing information.

SET ECHO ON

You will be surprised to find (the same way we were some time ago) that your RMAN log files don't contain the commands that have been used during a backup run. This could make you wonder what exact command had been used when looking at a several month old log file. If you use the *SET ECHO ON* command at the very beginning of your RMAN script, then RMAN will reflect all the commands in the log files. You may say, "Why should I bother, if I use the same script over and over again ,and know exactly what commands are used there?" Well, changes are unavoidable. If it isn't you, then your workmate may adjust a script. You can't assume that the commands are always the same. On the other hand, it doesn't cost you much to document the exact command used for an execution, rather than rely on your memories and assumptions that the backup script wasn't changed.

SHOW ALL

Besides the commands that have been used to make a backup, it is a good idea to document the default RMAN configuration used for a backup session. Just add the *SHOW ALL* command somewhere in your backup script, and it will allow you to understand why RMAN behaved in one way or another. Needless to say, the default configuration could be changed at any time, and assumptions you made at the time of developing your backup script may not be true anymore.

DOCUMENT SCRIPT'S EXECUTION LENGTH

It always helps if you can easily find how long it took to execute a script (especially a backup script). Typically if you got a page on failed backup then one of the first questions you may ask yourself is - can I run the backup again? How long will it take to complete? Will it interfere with any business activity or not? The following set of simple commands document a session's execution length:

```
c_begin_time_sec=`date +%s`
...
c_end_time_sec=`date +%s`

v_total_execution_time_sec=`expr ${c_end_time_sec} - ${c_begin_time_sec}`

echo "Script execution time is $v_total_execution_time_sec seconds"
```

This makes your task of judgment relatively easy. You just check the previous backups' log files to find the answer.

DO NOT OVERWRITE LOG FILE

It may be very frustrating if, at the time of troubleshooting a long running backup session, you find that the current backup session has overwritten the previous backup session's log file, and you have no evidence of how long the previous backup took. To avoid this, either make a copy of the previous log file or use a unique log file name in the first place. The following command would do the job.

```
v_log_file=full_backup_${ORACLE_SID}.`date +%Y%m%d_%H%M%S`.log
```

KEEP BACKUPS' LOG FILES EASILY ACCESSIBLE

If you followed most of the suggestions provided above, your backups' log files contain quite useful information for a potential troubleshooting or recovery session. Keep the log files easily accessible. Ensure that the log files have at least the same retention policy as your database backups. The log files could be a good alternative source of information in case the RMAN catalog isn't available.

#2 DO NOT USE CROSSCHECK

You would be surprised how often we see a *crosscheck* command used in RMAN scripts. Consider the following script:

```
crosscheck archivelog all;
delete noprompt expired archivelog all;

backup database
include current controlfile
plus archivelog delete input;

delete noprompt obsolete;
```

Do you see any problems with this? Think twice before answering. The *crosscheck* command shouldn't be used in your day to day backup scripts. In fact, it just lets your script silently ignore potentially dangerous issues, and possibly makes database recovery impossible. The script provided above will run successfully with no problem reported, even if there is a missing archived log. It will make point in time recovery or even full recovery impossible. The *crosscheck* command should be used by an Oracle DBA as a corrective action in response of a problem rather than built-in to a day-to-day RMAN script.

#3 BACKUP CONTROL FILE AS THE LAST STEP

Have another look at the sequence of commands provided in the previous section. Do you see any issues related to the control file backup? The *delete ... obsolete* command makes the control file copy we made inconsistent immediately. Make sure that the control file backup command is executed as the very last step in your backup script. The following output illustrates just that.

```
backup  as compressed backupset database
plus archivelog delete input;

delete noprompt obsolete;

backup spfile;

backup current controlfile;

exit
```

#4 DO NOT RELY ON ONE BACKUP ONLY

Do not rely on one backup only. You should always have a second option for a potential recovery. There is a good chance that a single backup, or the media it is stored on, has been corrupted. In such a case, it shouldn't be a problem for you to use another backup to recover a database. It typically isn't a problem for the data files. Even if this week's backup copy is

corrupted, you can always use the previous week's backup and apply all archived logs generated since then to get it current. This becomes a bigger problem if an archived log's backup is corrupted. If a single archived log backup is unavailable or corrupted then database recoverability is compromised. You wouldn't be able to recover from the previous weeks' backup, as there is a gap in the archived logs' stream. Consider making a several backup copies of the same archived log using the following command.

```
BACKUP ARCHIVELOG ALL NOT BACKED UP $v_del_arch_copies TIMES;
```

You may say that this significantly increases backup volumes (*$v_del_arch_copies* times), and you are absolutely right. However think about the risk to the system's recoverability. Discuss it with the system's owner and let him to decide if he willing to pay the price or take the risk.

#5 DO NOT DELETE ARCHIVED LOGS BASED ON TIME ONLY

Check out the following archived log deletion command:

```
DELETE NOPROMPT BACKUP OF ARCHIVELOG ALL COMPLETED BEFORE 'SYSDATE-2' DEVICE TYPE DISK;
```

We see commands like this used in production environments relatively often. A typical explanation from Oracle DBAs is: we delay deletion of the archived logs to make sure that they are applied on our standby database. The problem here is that nothing prevents RMAN from deleting the archived logs, even archived logs that weren't applied on the standby and not backed up even a single time. In some circumstances you may lose a single or several archived logs.

The following command is a much safer way to achieve the same result. It makes sure that all archived logs have been backed up at least *$v_del_arch_copies* times before they are deleted.

```
DELETE NOPROMPT ARCHIVELOG ALL BACKED UP $v_del_arch_copies TIMES TO DISK COMPLETED BEFORE
'$p_start_of_last_db_backup';
```

In the case of a standby database configuration consider using the following configuration command to ensure that RMAN doesn't delete an archived log that wasn't sent to a remote site (please refer to Oracle Support related articles for more details; it would take too much time to describe them here).

```
CONFIGURE ARCHIVELOG DELETION POLICY TO APPLIED ON STANDBY;
```

#6 USE CONTROLFILE IF CATALOG DB ISN'T AVAILABLE

Have a look at the way RMAN is called in the output below. Are you using the same method to call RMAN in your scripts?

```
[oracle@host01 ~]$ rman target / catalog rdata/xxx
Recovery Manager: Release 11.2.0.2.0 - Production on Tue Oct 18 15:15:25 2011
...
connected to target database: PROD1 (DBID=1973883562)
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-00554: initialization of internal recovery manager package failed
RMAN-04004: error from recovery catalog database: ORA-28000: the account is locked
[oracle@host01 ~]$
```

The problem here is that if the catalog database isn't available, the RMAN session fails immediately. However, the catalog database unavailability shouldn't stop RMAN from backing up your database.

As a solution, you may check whether the catalog database is available first and call RMAN with or without the *catalog* option in your script. Please note that we do not advocate ignoring catalog unavailability. The script should send you a warning if the catalog database isn't available. However, it isn't a significant reason for skipping a database's regular backup.

USE CONNECT TARGET WITHIN RMAN SCRIPT

There are several other methods you can use to skip the catalog connection if the catalog database isn't available. One of them is demonstrated below. You just use the *connect catalog* command within the script rather than passing it as an RMAN executable parameter.

```
rman target /
RMAN> echo set on
RMAN> connect target *
connected to target database: PROD1 (DBID=1973883562)
RMAN> connect catalog *
RMAN-00571: ===========================================================
RMAN-00569: =============== ERROR MESSAGE STACK FOLLOWS ===============
RMAN-00571: ===========================================================
RMAN-04004: error from recovery catalog database: ORA-28000: the account is locked

RMAN> backup as compressed backupset database
2> include current controlfile
3> plus archivelog delete input;

Starting backup at 2011/10/18 15:22:30
current log archived
using target database control file instead of recovery catalog
```

RMAN is proceeding with database backup even though the catalog isn't available.

SYNCHRONIZE RMAN CATALOG AS A LAST STEP

Another alternative you may consider is to delay the synchronization with the RMAN catalog to the very end. The following example demonstrates this approach.

```
-- Backup part
rman target / <<!
backup  as compressed backupset database
...
!

-- Catalog synchronization part
rman target / <<!
connect catalog rdata/xxx
resync catalog;
!
```

#7 DO NOT RELY ON RMAN STORED CONFIGURATION

Someone may say this isn't a relevant point in the environments he/she is managing, as he/she is in a control of all RMAN default configurations parameters. However, think about what happens if a database's configuration file has been recreated, and you forget to reset the configurations option you rely on? Or what if you are on vacation, and a DBA replacing you decides to change the configuration for one reason or another? Or what if Oracle decides to change the default configuration values in a next patch set? We would rather suggest do not rely on the default configuration. The typical default configuration items we can mention in this context are a control file autobackup, backup locations, number of parallel backup processes etc. Specify explicitly the options within your RMAN scripts rather than relying on the default values. In many cases you can overwrite the default values by using a *set* command in an RMAN *run*{} block.

RESTORE CHANGED RMAN DEFAULT CONFIGURATION OPTIONS

Based on our experience, in some specific cases changes to the default parameters are inevitable. However, a default value you may change within your scripts may be used by other RMAN related activities (e.g. other DBAs' scripts). It is a good idea to

restore default configuration values at the very end of your script's execution. The following set of commands is an example how you can do it.

```
-- Save the default configuration
v_init_rman_setup=`$ORACLE_HOME/bin/rman target / <<_EOF_ 2>&1|
grep "CONFIGURE " |
sed s/"# default"/""/g
show all;
_EOF_`

...
< script body >
...
-- Restore the configuration at the end of the script
echo $v_init_rman_setup | $ORACLE_HOME/bin/rman target /
```

#8 BACKUPS' CONSISTENCY CONTROL

FAILURE VERIFICATION AND NOTIFICATION PROCESS

We strongly believe that any backup's failure in a production environment is significant enough to be addressed immediately after it happens. Let us ask you how and when an Oracle DBA (or SA) is notified of a backup failure in your organization? Some typical answers may be as follows:

- We don't report failures
- A DBA checks logs sometimes (when he has time)
- Backup logs are sent to a shared email address
- The DBA on duty checks emails (what if no one available/no email received?)
- Scripts check RMAN command's errors code $? and sending an email in case of an error is returned

We suggest sending a page to an oncall DBA immediately if a failure is detected. The Oracle DBA should be notified and involved in a failure's analysis and take corrective action.

BACKUP'S FAILURE CHECK

Some Oracle DBAs prefer to checks RMAN's exit code *$?* only to verify if the run was successful or not. While sometimes this is a valid check, we believe it isn't enough. You should take as pessimistic an approach as possible to checking the backups. The following may be an additional check you may consider implementing.

```
egrep "ORA-|RMAN-" < log file >
```

Better safe than sorry. Be creative in the way you check if a backup has been successful or not.

LONG RUNNING BACKUPS

If a backup operation takes longer than the time frame allocated, then it can seriously impact business operations. An Oracle DBA should be notified about it. Consider implementing a check that will page an oncall DBA if a backup is at risk to interfere with business activities.

NOTIFICATIONS ARE NOT ENOUGH

A fact that there are no failures reported (no pages) doesn't necessary mean that a database safely backed up. What if for one reason or another a backup script was excluded from running at all? What if there is a logical error in the script and some data files are excluded (e.g. read only)? What if there is an unrecoverable operation executed against one of the data files and you are not aware about it (happens quite often)?

To verify that your database was backed up based on your business requirements, and that you can recover the database in an expected timeframe, implement additional (separate) verification scripts. A script could contain several checks and if the current status of the database's backups doesn't pass the checks a page can be sent to the oncall DBA to take immediate action. The following examples demonstrate some of the checks you may consider including in such a verification script.

```
REPORT NEED BACKUP ...
-- report data files that weren't backed up last 48 hours
REPORT NEED BACKUP RECOVERY WINDOW OF 2 DAYS;
REPORT NEED BACKUP REDUNDANCY 10;
REPORT NEED BACKUP DAYS 2;

REPORT UNRECOVERABLE;
```

#9 If you don't have RMAN and MML integration

If you don't use RMAN and Media Management Layer (MML) integration you should seriously consider using it. A popular alternative is an approach where RMAN is used to store backups on a file system and other backup software is used to back up the file system to a tape. In such configuration an Oracle DBA should address several significant challenges including the following:

- How to ensure that all RMAN pieces are backed up on tape?
- How to ensure that tapes don't contain more than necessary number of copies of the same backup piece?
- How to restore necessary backup files during recovery in a shortest time possible?

Let's discuss some of the challenges.

Ensure 3 times full backups space + ARCH

Without MML integration, you must have space necessary to store at least 3 full RMAN backups on your file system, otherwise you put your production's database recoverability at risk. The minimal recommended retention policy is REDUNDANCY 2.

Let 's assume for a moment that database retention is set to REDUNDANCY 1. Consider the following scenario:

- There is an RMAN script that makes archived logs' backups on hourly basis
- There is a full database RMAN backup executed between 9pm and 10pm on a daily basis
- There is a *delete obsolete* command at the end of the full backup that deletes all backups' files based on the retention policy (REDUNDANCY 1). It effectively deletes the oldest full backup and archived logs' backups between the current backup and the previous one.
- Your tape backup transfers any files located under an RMAN backups' directory at 3am on nightly basis

In such a typical setup, all archived logs' backups generated since last tape (previous day's 3am) backup and the beginning of a full database backup (9pm) are not transferred to a tape, as they got deleted by the *delete obsolete* command at 10pm.

This drives us to the conclusion that the minimal acceptable retention policy in such a setup is REDUNDANCY 2. In order to store at least 2 successful full backups, we need space for a third backup, as we can't delete an older backup until the newest backup is completed.

Please note that if the tape software you are using isn't smart enough then it will transfer some of the RMAN backups' files to tapes at least twice (unnecessary redundancy).

Don't use "delete obsolete"

If you are using the *delete obsolete* command as part of your RMAN backup procedure (with no MML integration) you effectively "wipe out" RMAN's memory. There is no way RMAN could know about backups available on tapes in such case. If you need to recover your database you should go through relatively complex and time consuming steps listed below:

- Recover a control file (possibly from offsite backups) # several hours
- Find from the control files all the backups' pieces necessary for the restore
- Bring onsite all tapes involved and recover all backups' pieces # several hours
- Run though the RMAN's restore operation # several hours
- Find from the control files all the backups' pieces necessary for the recovery
- Bring onsite all tapes involved and recover all backups' pieces # several hours
- Run though the RMAN's recovery operation # several hours

As an alternative to the *delete obsolete* command usage, you could implement the following procedure to optimize your backup and recovery procedure and keep information about backups available on the tapes.

-A- Generate a list of RMAN backups' files to be removed based on the expected disk's retention policy.

```
report obsolete recovery window of ${DISK_RETENTION_OS} days device type disk;
```

-B- Check if each of the reported files has been backed up to a tape (transfer it to the tape if necessary) and remove backups' files using an operating system *rm* command

-C- Remove entries from RMAN repository based on tape retention policy.

```
delete force noprompt obsolete recovery window of ${TAPE_DAY_RETENTION} days device type
disk;
```

This method keeps records about RMAN backups' files as long as those are available on tapes. An optimized recovery procedure looks as following in such case:

- A control file contains information about all the necessary backups for the recovery (no needs to restore it from tapes)
- Find from the control file all the backups' files necessary for the data files restore and recovery (including archived logs) using the following RMAN command (note that the command doesn't check files on the file system and therefore executed successfully in few seconds time)

```
RUN
{SET UNTIL SCN 898570;
RESTORE DATABASE PREVIEW;}
```

- Based on the list generated, bring onsite all tapes involved and recover all backups' files necessary for the complete recovery operation # several hours
- Run though the RMAN's restore and recovery operation # several hours

As you can see, avoiding usage of the *delete obsolete* command allows simplifying and significantly reducing recovery times.

NEVER KEEP DEFAULT RETENTION POLICY

If you don't use an RMAN retention policy for managing your backups (e.g. leaving it up to media management layer) don't keep the default retention policy. Set it to relatively high values using the following commands:

```
CONFIGURE RETENTION POLICY TO REDUNDANCY 1000;
CONFIGURE RETENTION POLICY TO RECOVERY WINDOW OF 1000 DAYS;
```

If you keep the default values then someone (e.g. an Oracle DBA who isn't aware about the setup) occasionally may run *delete obsolete* command (e.g with intention to clean some space on the file system). It doesn't just clean all the records about all backups taken from an RMAN catalog (or a control file) but possibly deletes some of the backups' files that still didn't find their way to tapes. We strongly suggest that you adjust the default settings to relatively high values to avoid unnecessary risks.

DELETE OLDEST BACKUP BEFORE MAKING NEXT BACKUP

A typical RMAN backup script would delete oldest backup right *after* the new full database backup is completed. The following set of commands demonstrates it:

```
backup database
include current controlfile
plus archivelog delete input;

< delete command's implementation >
```

However this leaves free space equal to the size of one full backup unused until the next backup time. Instead, you may use the file system space a bit more efficiently by deleting oldest backup right *before* completing the next full backup, as is demonstrated below.

```
< delete command's implementation >

backup database
include current controlfile
plus archivelog delete input;
```

This will make better use of your file system space for a small price of additional archived logs' backups to be stored on the file system.

#10 HALF WAY BACKED UP FILE SYSTEM FILES

Let us ask you: if you don't use the RMAN media management layer integration, how do you ensure that half-created RMAN backups' files are not copied to a tape? If your answer is "my file system backup procedures are separate from RMAN's backups in time," we think it isn't sufficient. The configuration may change (e.g. someone may change file systems' backup times and not warn you), and it could impact your database recoverability when it is too late to fix anything.
We would suggest you to implement an additional measure to make sure that your tapes store completed RMAN backup files only. The following approach could be used to ensure that.

```
-A- BACKUP AS TMP
BACKUP DATABASE FORMAT '${tmp dir}/${file}.tmp_rman';
-B- MOVE TO PERMANENT
mv ${tmp dir}/${file}.tmp_rman ${backup dir}/${file}.rman
-C- MAKE CATALOG AWARE
CHANGE BACKUPPIECE '${tmp dir}/${file}.tmp_rman' UNCATALOG;
CATALOG BACKUPPIECE '${backup dir}/${file}.rman';
```

Configure tape backups to avoid *${tmp dir}* and backup *${backup dir}* directory.

CONCLUSION

We hope that you do not have the problems described in this paper, and if you do, that these solutions help you to minimize possible recoverability risks and make your RMAN backups even more reliable. Think about how you can make your recovery more comfortable at the time you work on your backup script. Working out missing bits and pieces may be way too late at recovery time.