

# Is RMAN REALLY WORTH THE TROUBLE?

Tim Gorman, SageLogix, Inc.

## Abstract

Oracle's Recovery Manager (RMAN) was first introduced with Oracle8 release 8.0 in 1997, and acceptance of the product has been slowed due to the well-deserved reputation of being difficult to install. Add this to the fact that RMAN is entering a market that is already well-served with products such as BMC SQL BackTrack, and you have even more reluctance to accept Oracle Corporation's late entry to the database backup-and-recovery market. But RMAN provides three crucial features that no other solution, whether purchased from another vendor or custom-built, can provide. And, depending on your requirements, these three features should make the difference between deciding to use RMAN or using another backup-and-recovery solution, whether it is a purchased from a vendor or custom-built.

So, the decision of whether to use RMAN or not should really not hinge on the widely discussed difficulties in integrating it with media-management vendor (MMV) software. Arguably, it also shouldn't hinge on a comparison of look-and-feel and ease-of-use with the more-established products against which it competes. After all, it is a relative latecomer to the arena, and Oracle's track record has been positive in the area of improvements.

Instead, this paper will explain the three crucial features: the background behind them and what problems they resolve. Then, you should be able to consider the advantages of these features and decide whether they are important to your installation, or not. If they are important, then going through the hassles of getting RMAN up and running are worth it.

This paper and its associated presentation is intended to clarify your decision on whether to utilize RMAN or not, if you haven't yet made such a decision; or to validate your decision, if it has already been made.

## What is all the fuss about?

The Recovery Manager has been architected to support backup and recovery operations on all of the 80+ platforms on which Oracle runs, including the Windows platforms, all of the UNIX variants, mainframes, and a variety of other proprietary operating systems such as Compaq Open-VMS.

Because of this portability requirement, Oracle designed RMAN in two halves: the generic Oracle half and the extremely specialized media-management half.

The generic Oracle half of RMAN is, indeed, common across all of the platforms upon which Oracle resides. It includes:

- The end-user presentation layer, consisting of either the graphical user-interface (GUI) presented from the Backup Manager module of Oracle Enterprise Manager (OEM) or the command-line interface (CLI) from the server's operating system command-line.
- The Oracle server processes which perform the actual work of reading datafiles, control files, and both on-line and archived redo log files during a *backup* operation, sending the information as either *backup sets* or *image copies* to the media-management layer. These same Oracle server processes also request those *backup sets* or *image copies* from the media-management layer and write information down to datafiles, control files, and redo log files during a *restore* or *recovery* operation.
- The Recovery Catalog, a limited version of which is stored locally in the *control files* of the database being backed-up, restored, or recovered. A more extensive version of the Recovery Catalog can optionally also be created in a remote Oracle database; it is highly recommended to use this functionality in order to exploit RMAN's capabilities to their fullest. The Recovery Catalog is used to record all *backup* activities and *structural information* about the database in order to automate *restore* and *recovery* activities.

The other half of Recovery Manager, the *media-management* layer, is either platform-specific or, in some cases, simply based on preferences in the event that there is a choice of media-management products. On

proprietary platforms such as mainframes or Windows, there is generally only one choice for software to operate the backup media, which generally consists of tape drives. On most UNIX variants (including Linux), there is often a wide-variety of possibilities, due to healthy competition between vendors.

The upshot is that Oracle has been forced to abstract a *generic interface* between the generic half of RMAN and the actual tape drives upon which backups will ultimately reside. It would be virtually impossible for Oracle Corporation to even attempt to provide its own media-management software for the ever-expanding universe of backup media. It would also be exceedingly foolhardy. Another alternative approach might have been for Oracle to take on the responsibility of implementing interfaces from RMAN to all of the media-management software packages in existence, or at least a select group of the most popular, but Oracle has chosen not to do that. For one thing, it would be a massive undertaking, which Oracle as a development organization might be ill-equipped to perform in a timely fashion. They also don't want to do it; it's not in their main line of business, after all.

Therefore, they have defined a generic interface for RMAN to talk to, and they have made the *media-management vendors* (MMVs) responsible for meeting that interface. Thus, each MMV can optimize their interface, which is in their own best interest, instead of trusting Oracle developers to do so. Also, as each MMV's own software undergoes its own evolutionary improvements and revolutionary leaps and bounds, each MMV can ensure that their Oracle-based customers can take advantage of those changes according to their own timing and priorities, not Oracle Corporation's.

However, it due to this mutual flux on both sides of the interface wherein lies the rub. Often, the two sides of the interface do not work together. The interface itself is being updated continually. Thus, where documentation or procedures published a month ago may have worked, the same is not necessarily true tomorrow because either the Oracle version has changed, the interface definition has changed, or the MMV version is now incompatible.

Both sides, Oracle and the MMVs, have experienced bugs and glitches. Both sides have been issuing new versions of software. The end result is that the poor bewildered customers have been caught in a game of software musical chairs.

And so, RMAN has acquired a reputation as being difficult to install. This is, unfortunately, true. The fault lies not with the product or with the MMVs. Still, when you are seeking results, blame doesn't matter. And so, RMAN has been relatively slow to catch on.

### **Stability**

Oracle database administrators (DBAs) are also skeptical of a *point-zero* release of a new product from Oracle. Their skepticism of the stability of RMAN has proven well justified. In versions 8.0.3 and 8.0.4, RMAN stability was a wild roller-coaster ride. Over the past two years, bugs introduced within the Oracle RDBMS from patch-set to patch-set and from version to version have made it virtually unusable for mission-critical systems. As of version 8.0.5.1.0, most of these problems appear to have been worked out, but I've heard reports of disabling internal bugs even at this late version, requiring yet more emergency patches.

### **Look and feel**

Another issue is that RMAN is relatively late to the game, compared to other products that have been occupying this market for years. As the latecomer, Oracle should expect fierce resistance from people who have resolved a problem using 3<sup>rd</sup> party vendors in an area which Oracle neglected for too long.

Also, many Oracle database administrators (DBAs) have been performing the backup-and-recovery function for years with their own custom-built scripts and tools, and they are understandably reluctant to uproot this crucial part of their infrastructure in favor of a new product with its own scripting language. Why fix or change that which is not broken? Why make a mystery out of something that is reliable, known, and comfortable?

### **Without a compelling reason, why bother to switch to RMAN?**

Despite all of these drawbacks and well-founded reasons for not using RMAN, the product does provide three important high-end features that no other solution can provide. Because of its unique integration with

the Oracle RDBMS, no other product, whether vendor-provided or custom-built, can provide these capabilities. As you will see, these features each address problems introduced by extremely high-end environments and their requirements, so the importance of these features varies according to the business- and system-requirements of the each database.

The three important features are:

The ability to perform *hot* or *online* backups without the ALTER TABLESPACE ... BEGIN BACKUP or ALTER TABLESPACE ... END BACKUP commands

The ability to perform incremental backups and restores

The ability to detect database block corruption

On the surface, as stated here, you might say to yourself *Big deal!*

*Removing the ALTER TABLESPACE commands do not seem so important; after all, they complete in fractions of seconds anyway. Removing them is not some kind of wonderful feature, is it?*

*As far as incremental backups are concerned, don't the Oracle utilities EXP and IMP do that, too? Why would it be important for RMAN to do that also, especially if EXP is already part of my backup strategy? And, if I'm not mistaken, there are several 3<sup>rd</sup> party products that do that as well, are there not? Isn't it overstating to say that only RMAN does incremental backups?*

*Last, what's this talk of detecting database block corruption? Are you just trying to scare me? I spoke to my disk drive vendor sales people about this, and they just laughed. I asked my Oracle sales rep about it, and they just took me to lunch at a great place. Come to think of it, we never did finish that discussion! I called Oracle Support and logged a TAR asking about database block corruption, and sent me a bulletin on the ORA-01578 error message. I've even heard about the new DBMS\_REPAIR package in Oracle8i that fixes any corruption that might occur. So, how could this ability to detect corruption be important?*

These are all reasonable questions, at one level or another. There is a fair amount of explanation in the answer. So, let's take them one at a time...

## **Hot backups without ALTER TABLESPACE ... BEGIN/END BACKUP**

The advantage to eliminating these commands from the process of taking "hot" or "online" backups is not in simply eliminating the commands themselves. Rather, the advantage lies in understanding exactly what is happening while a tablespace is in *backup mode*, and in understanding how RMAN's architecture makes that unnecessary.

### **Hot versus Cold backups**

There are two and only two ways to take valid backups in Oracle: *cold* and *hot*.

*Cold* backups are taken when the database instance is entirely shut down, and is completely inactive. For you trivia buffs, you can also safely take a cold backup if the database is started in NOMOUNT mode. Since this mode means that only the System Global Area (SGA) has been allocated in shared memory and that the background server processes (such as DBWR and SMON) are running, but no files have been opened, you can still get a valid cold backup. But this is a digression, as the NOMOUNT mode is used only by the CREATE DATABASE and CREATE CONTROLFILE commands, for starting STANDBY databases, and by RMAN for taking cold backups.

Anyway, cold backups involve copying all (or at least one) of the control files, all of the online redo log files, and all of the datafiles. This is the simplest form of backup, since it only involves shutting down the database and taking the backup.

*Hot* backups mean that the database is up and running, on-line, and completely available for all activities while the backup is occurring. Without RMAN, this means that each tablespace must be placed in *backup mode* in order to be copied, using the ALTER TABLESPACE ... BEGIN BACKUP to begin backup mode and ALTER TABLESPACE ... END BACKUP to end backup mode. Also, the control files themselves are not copied in a hot backup. Instead, the ALTER DATABASE BACKUP CONTROLFILE command must be used to create a backup copy of the control files. Last, the online redo log files themselves cannot be backed up directly. Instead, only the archived redo logfiles should be backed up. Since the rate at

which archived redo log files are generated may cause them to overflow the file-system in which they are located, the schedule for backing up archived redo log files is often much more frequent than the backups of the datafiles.

### **What happens while a tablespace is in *backup mode*?**

When the ALTER TABLESPACE ... BEGIN BACKUP command is issued, a *global database checkpoint* occurs. This means that the current *system change number* (SCN) is logged to the redo log stream in a *checkpoint record*, and all buffers in the Buffer Cache modified prior to that SCN must be flushed down to the datafiles. Once the checkpoint completes, then the header block of each datafile in the tablespace is *frozen*, meaning that the header will not be updated to record the SCN of future checkpoints.

However, the writing of database blocks to the *body* of the datafiles is not changed in any way. During future checkpoints, while the tablespace remains in backup mode, blocks continue to be written to the datafiles, just like normal. When the DBWR process flushes blocks to the datafiles outside of checkpoint processing, this continues in the datafiles belonging to the tablespace being backed up, just like normal.

Meanwhile, while this normal I/O occurs to and from the tablespace in backup mode, some type of operating-system utility is copying the datafiles, backing them up. This utility, which on UNIX might be the *dd*, *cp*, *tar*, *cpio*, or *dump* commands, is actually backing up datafiles that are *in flux*. The files are being changed, as they are being saved to backup media. To put it another way, the datafiles that are being backed up are *not consistent*, and in one sense of the word, are *corrupted*.

*And we don't care!*

Because, should a recovery situation occur, the datafiles will be recovered from the point-in-time specified by the SCN logged in the file header. That SCN represents the point-in-time of the checkpoint that was triggered by the ALTER TABLESPACE ... BEGIN BACKUP command. Thus, should the database crash before the ALTER TABLESPACE ... END BACKUP command, all changes to the datafile since the SCN in the header would be re-applied. In the event that the datafile being backed up was itself lost, it could be restored from a previous backup copy, and all changes from that point-in-time would be re-applied.

The point is, freezing the SCN timestamp in the datafile header keeps the datafile safe while backups are occurring. Also, should the backed up datafile be restored in a recovery situation, all the changes from the point-in-time of BEGIN BACKUP would be applied from the redo logs. The so-called *inconsistent* or *corrupted* datafile, backed up while it was *in flux*, provides just a baseline from which redo log information can be applied. Thus, it permits the datafile to be recovered safely, regardless of the changes made while it was being backed up.

*Except for one situation...*

### **Block Fracture**

Let's suppose that the DB\_BLOCK\_SIZE of our database is 8 Kbytes or 8,192 bytes. Now, let's suppose that the operating-system utility we are using for backups performs I/O in 512 byte blocks; in fact, this is the default for the UNIX *dd* utility.

So, while *dd* is copying some blocks from disk, it pauses to write those blocks to tape. While it is writing to tape, suppose that the DBWR process comes along and writes down an 8 Kbyte database block, exactly over the blocks that *dd* had been reading! Now, given the disparity in I/O block sizes, it is possible, even likely, that *dd* was only partway through one of the 8 Kbyte database blocks. When it finishes writing the blocks it has already read to tape, and resumes reading from disk, it will now resume copy the rest of the same database block, *which has now been changed*. In essence, one half of the database block was backed up from one point-in-time image, while the rest of the database block was backed using another point-in-time image. This is known as *block fracture*.

Normally, the only information sent to the redo logs are the precise bytes that were changed in the database block. For example, if one column was updated in a row, and the database blocks stored hundreds of such rows, only the bytes changed are logged to the redo logs.

If a restored datafile contained fractured blocks, then the recovery process would be writing these *redo vectors* to database blocks which are inconsistent, which simply would not work.

To resolve this situation, Oracle has changed the algorithm for redo logging. If a database block is being changed for the first time after the tablespace has been placed into backup mode, then the *entire database block image* is logged to the redo log stream, not just the bytes being changed. This means that the volume of redo logging can increase dramatically while a tablespace is in backup mode.

In some situations, this increase in volume might make a bad situation much worse. Some systems may already be generating such huge volumes of redo logging that any addition volume is a matter of grave concern. In fact, this additional volume might be the straw that breaks the camel's back; the system may be unable to keep up with the additional volume.

### ***How does RMAN resolve this?***

Since Oracle server processes perform the reading of database blocks from the datafiles, they are able to use the same *read-consistency* mechanisms that SQL statements use. Fractured blocks are not a problem for SQL statements, and neither are they a problem for Oracle server processes used by the Recovery Manager during a backup.

Recovery Manager records the SCN at the point-in-time the backup of the datafile started, in its own Recovery Catalog in the control files, so there is no need to freeze the datafile headers. Without freezing the datafile headers and without the need for putting the tablespaces into backup mode, there is no longer any need for the ALTER TABLESPACE commands.

Thus, RMAN permits hot backups of extremely busy tablespaces without any adverse impact on the volume of redo logging.

### ***A small added bonus***

Further, since the datafile header blocks are not frozen, the need for the ALTER DATABASE DATAFILE ... END BACKUP command also disappears. This command was introduced in version 7.3 of Oracle7 to resolve a vexing situation that would result if a database instance crashed while tablespaces were in backup mode. Upon startup, Oracle would detect the out-of-sync datafile headers and attempt to perform media recovery, to bring them up to date. This was not strictly necessary, since the bodies of the datafiles were, in fact, completely up to date; it was only the datafile headers that needed to be synchronized. So, prior to Oracle7 version 7.3, this situation always required a useless media recovery. After version 7.3, you could simply issue the ALTER DATABASE DATAFILE ... END BACKUP command to avoid the media recovery. If you are using RMAN, even this is unnecessary.

## **Incremental backups**

Without RMAN, the only types of backups that are possible are *full image copies* of the datafiles. This means that every single database is copied, regardless of whether it has ever been used.

RMAN includes the concept of *incremental backups*. A *level 0 backup* is the lowest level of backup, and this level of backup will copy all database blocks containing data. In other words, a *level 0 backup* is not the same as a *full image copy*; the latter copies all blocks, whether they contain data or not, while the former differentiates amongst blocks containing data and those which do not. So, right from the beginning, *incremental backups* copy less information to the backup media than do *full image copies*.

Then, *level 1 backups* only copy database blocks that have changed since the most recent *level 0 backup*. This reduces much further the volume of data sent to backup media, without any loss of protection. *Level 2 backups* only copy database blocks that have changed since the most recent *level 1* or *level 0 backups*, and *level 3 backups* only copy database blocks that have changed since the most recent *level 2, 1, or 0 backups*. And so on.

So, depending on the nature of the application, incremental backups can reduce the sheer volume of data sent to backup media dramatically, by 25%, 50% or even higher. For larger databases, measuring hundreds of gigabytes or several terabytes, this can be crucial. Just the sheer cost savings on the number of tapes consumed on a monthly basis could make a serious dent in somebody's budget. The timesavings from smaller, faster backups and restores could make for swifter recovery times.

## Incremental backup capabilities from other vendors

Products from other vendors long ago reverse-engineered the block formats in Oracle and have touted the ability to perform incremental backups as well. While this is true, Oracle Worldwide Support has stated on numerous occasions that they will not provide support for problems arising from the use of these incremental backups in a restore and/or recovery situation. While the use of these features from non-Oracle products does not nullify your Support with Oracle, it could make that support much less useful.

## Detecting corrupted blocks

Just like a bumper sticker would say it: *CORRUPTION HAPPENS!* Why it happens is simply due to either some form of failure or some form of bug. Nobody likes to talk about it, because data corruption is inherently unpredictable and results from failure.

Nevertheless, it occurs. Oracle's recommended method for correcting it is to simply treat corruption much the same as you would treat media failure:

- Fix the problem that caused the corruption in the first place
- Restore from a backup taken prior to the occurrence of the corruption
- Recovery to the desired point-in-time

Now, as you may have commented to yourself, each of these steps has its special difficulties.

Fixing the problem that caused the corruption involves first knowing the problem. Sometimes, it is not possible to know the nature of the problem; at least not without (possibly lengthy) investigation. The scope of this paper does not include answering this question, which is, after all, the crux of the whole issue of data corruption. Suffice it to say that the root cause is either self-evident, or it requires some investigation, or it requires a lot of investigation. Due to the complexity of today's systems, also prepare yourself for the eventuality that the root cause may never be found.

The next step, to restore from a backup taken prior to the corruption, is next to impossible without RMAN. Since RMAN uses the same *read-consistency* mechanisms utilized by SQL statements, a backup performed using RMAN is able to raise the same error messages that a SELECT statement would raise upon detection of something amiss inside a database block. Even a backup program which is aware of the detailed format of Oracle database blocks cannot accurately detect corruption, as such a process would lack the *read-consistency* mechanism available only to Oracle server processes. Such a program, like Oracle's own DBVERIFY program, would still be subject to *false alarms* on corruption due to the fractured database block problem described earlier.

RMAN has a command, SET MAXCORRUPT, which can be used to set the *tolerance* for database blocks that are detected to be corrupt. By default, SET MAXCORRUPT is 0, meaning that zero blocks will be tolerated. If a corrupt block is detected, then the backup will terminate with an error. Setting MAXCORRUPT is on a datafile basis; if a particular datafile has corrupt blocks of which you are aware (but cannot do anything about), you might want to SET MAXCORRUPT to the number of known corrupt blocks, so that backups can complete successfully. Then, if any new corrupt blocks are detected, you will still be informed.

Additionally, RMAN will attempt to insert the location of the database into a list in either V\$COPY\_CORRUPTION or V\$BACKUP\_CORRUPTION views. If the database block is already in this list, then another insert is not attempted; each database block is only entered once. Also, when a block is inserted either of these views, when it is detected as corrupt for the first time, an ORA-01578 error message is logged in the "alert.log" file, regardless of the setting of MAXCORRUPT.

So, being able to restore a datafile from a backup prior to the corruption, while impossible without RMAN, is quite easy with RMAN.

The new DBMS\_REPAIR package does not, despite the names of some of its procedures, actually repair or fix corrupt blocks, in the sense of restoring lost data or somehow fixing them to be usable again. In fact, the procedure DBMS\_REPAIR.FIX\_CORRUPT\_BLOCKS "fixes" corrupted database blocks in much the same way that a veterinarian "fixes" your pet so you don't have to worry about procreation issues. It simply marks the block as "corrupt", so that it can be skipped over and no longer accessed.

So, the only way to truly *repair* or *fix* a database block after it has been corrupted is to restore it from a clean backup and recover. That's it. There is no magic...

### **Alternative methods of detecting corrupt blocks**

Outside of using the RMAN product, is there any alternative method for detecting corrupted database blocks in a database?

Yes, there is. But you won't like it!

Quite simply, what is necessary is to periodically query every single database block belonging to tables, throughout the entire database. Most often, the EXP utility was used for this. A FULL export to the *null-device*, also known as the *bit-bucket*, was one way to accomplish this:

```
$ exp userid=system full=y file=/dev/null
```

What can this possibly accomplish, you might ask? In essence, you are querying every table in the database and simply throwing the information away. In this situation, you do not care to save the data, you simply want to have the Oracle I/O mechanism touch each table block. Throwing the retrieved data away tends to improve performance.

There are several problems with this method:

- It consumes massive amounts of resources
- It takes a long time on large databases
- It does not check the data stored in the SYS schema (i.e. the data dictionary)

There is no way around the first problem; just part of the nature of the beast, if you don't use RMAN. As for the third problem, you can create a SQL script that queries the data in the SYS schema, and throws it away. As a matter of fact, to resolve the second problem, you could perhaps run multiple EXP processes in parallel, each checking a different part of the database. Or, you could do the same job with SQL scripts, "C" programs, PL/SQL or Java stored procedures; you name it.

Or, you can just do your backups with RMAN.

Or, just ignore it.

It's your choice.

### **Conclusion: Is RMAN Really Worth the Trouble?**

So, deciding whether RMAN is worth the trouble comes down to answering the following questions:

- Must you perform hot backups? If so, is there any problem with the volume of redo logging increasing?
- Would you like to decrease the volume of data you are backing up?
- Would you like to increase the speed or decrease the duration of your backups?
- Would you like to increase the speed of your restores and recoveries?
- Would you like to decrease the duration of your restores and recoveries?
- Must you prevent any data loss, under any circumstances?

If you answered "yes" to one or more of these questions, then it would be worthwhile to consider implementing Oracle's Recovery Manager product. Based on what has been discussed in this paper, you certainly understand that these questions are not just marketing hype. RMAN can resolve many of these problems in a truly efficient, sensible way. No gimmicks.

It's your choice.

### **About The Author**

Tim Gorman is VP of server technologies for SageLogix, a consulting and education services firm based in Colorado, which exploits Oracle and Java technologies for web-based and decision-support. He has been a "C" programmer for over 16 years and has worked with Oracle for 10 years. Tim is co-author of "Oracle8

Data Warehousing" and "Essential Oracle8i Data Warehousing" published in 1998 and 2000, respectively,  
by John Wiley & Sons,

Tim can be reached at:

SageLogix, Inc.

<http://www.sagelogix.com/>

tim@sagelogix.com