Backup & Recovery ...

by
Howard J. Rogers

Contents

Backup and Recovery: Chapter 1 – General Considerations

If you had to define what we mean by 'Backup and Recovery', and what we'll be looking at on this course, you'd have to say that it's basically about protecting the database from failure in the first place, and (I suppose) increasing the time between failures by various means… . You would already have met a lot of these concepts on the basic DBA course, but they include things like mirroring your control files, mirroring your Redo logs, how you distribute your data files across physical discs, and so on… . All ways of making sure that the database is kept available for as long as possible.

But then you have to consider, 'well, what if the database DOES fail after all' then your job as a DBA becomes one of getting the thing back up and running as quickly as possible –and, probably most importantly, doing so whilst minimising the amount of data loss.

In fact, as you'll learn on this course, if you do things the right way, I can guarantee that you will NOT lose any committed transactions in Oracle. I guess that this course is really about learning what all the 'right things' are.

So I guess we should start by asking: who already has a backup and recovery strategy? And if you do, what is it? What are its key features? What factors determine a backup and recovery strategy?

Now, as it happens, the course material tries to pretend these factors break down into neat little categories. They don't: the categories and factors overlap greatly, and influence each other.

So, for example: you might take the general category "Business Requirements'. What do we mean by that? Well, there may be a business requirement to keep the database operating uninterruptedly for 24 hours a day. That, in fact, turns into an "Operational Requirement", according to the course material! The point for us is that a requirement to run the database in one particular way turns into a simple Backup and Recovery principle: we can't do "Cold" or "Offline" backups: we have to do "Hot" backups –and that means we have to throw various other switches in the database to make that a practical proposition.

Take another example. There may be a business rule: "In the event that there is database failure, the thing must be back up and running in (say) 30 minutes or less". That in fact turns into a technical requirement to run the database with lots of small Online Redo Logs –because, although we'll be switching between them very frequently, we'll only have to play back a small amount of Redo in the event of an Instance Failure.

It works the other way round, of course. For example, I suppose the best way of backing up a database is to take simple copies of all the database

files –either onto another disk, or onto tape somewhere. But that means you have to have plenty of disk space handy –or an impressive collection of tape backup hardware and software. There is thus a technical need for heaps of disk space –that turns into a need for Management to *supply* you with that amount of disk space! They can't demand 'No Data Loss' and then not properly resource a backup strategy that can achieve that! So there's a technical requirement impinging on business and management issues.

Or again: you could adopt the highly technical approach of taking *logical* database backups. That basically means you use Export to create files which store the data structures of the database –and the data, too, of course. Much smaller than your physical backups… and guaranteed not to be corrupt, too. But absolutely hopeless for protecting you against actual data loss. Good for the inadvertent dropping of an important table, certainly. And most DBAs would use Export as an important part of a good Backup strategy, even where physical backups were being taken as well. But again you see the intertwining of technical and business issues –that a technical decision has profound implications for what the business (and its management) can realistically expect in terms of protection from failure.

So, there you have it: a bit of scene setting for what this course is all about: understanding the interrelatedness of matters technical, operational and managerial. A decision taken as to how you structure and create your tablespaces because (say) your operating system can only support 2Gb-sized files can have significant impact much later on what can be recovered, how, and how quickly. Much of the next three days is hopefully going to be spent unravelling such interconnections and dependencies.

A very important point to consider, by the way, but one which we don't spend any time on in this course is the matter of disaster recovery. The loss of a data file, or of an entire disk is no problem! However, if your office happens one morning to be blasted into a meteorite crater… well, there's not a great deal of advice I can offer to protect against that sort of thing! So, disaster recovery planning is important, but a bit outside the scope of this course.

However, just also consider that your DBA team all being poached by more generous employers at the same time is just as much a disaster as being struck by a meteorite. Loss of staff through natural attrition –or unnatural entanglements with the proverbial bus- is a disaster waiting to happen, and the only thing you can do to protect yourself against *that* sort of disaster is to do three things: Document, Document and Document. Document your procedures, your database structures, your technical infrastructure. And keep your documentation up-to-date, otherwise you might just as well not bother.

Backup and Recovery:
Chapter 2 – Oracle Architecture

This is a bit of revision of the basic Oracle architecture – with the idea that we'll look particularly at those aspects of the architecture that have an impact on the way we do Backup and Recovery operations.

First things first: Why don't I get you to just name all the components of the Oracle architecture that you can remember from the DBA course? You can just name them in any old order! We'll sort out what each component does at the end, but for now just tell me what you can remember!

[As items of the architecture are named, put them up on the Whiteboard]
[Discuss functions of each item in general terms]

2.1 Processing DML

Now, we ought to just remind ourselves how Oracle handles transaction processing using these components. So, suppose I issue this statement:

```
Update EMP
Set sal=750 where name='Bob';
```

How does that get processed? First: we read the SQL statement into the Library Cache of the Shared Pool. There it gets parsed, and we check that there really is a table called EMP, that there really is a column in it called 'Sal' and that you, the User, really do have rights to update the data in that column. Assuming it passes all those checks, we construct an Execution Plan and store that, along with the original SQL statement, in the Library Cache. That's the PARSE phase of the execution of a piece of DML. Now we have to execute the thing!

First, we check to see if the data that we want to update is already in memory – in the Database Buffer Cache. If it is, fine – we'll use it. If not, our Server Process will go and read it off the disk, and store it in the Buffer Cache. We now do a series of 4 things, as follows:

- Take a lock on the record, so no-one else can alter it whilst we are working on it
- Write the Before and After images into the Redo Log Buffer
- Prepare a Rollback block, and write the Before image into it, with the lock information on the original record pointing at the Rollback Block
- Alter the original record to reflect the new value

2.2 Commit Processing

Now, when we commit this change, what happens? "Not Much" is the short answer! We write a System Change Number (a "SCN") into the Log Buffer. This is the identifier that tells the system that this a commit was issued for this transaction. Log Writer (LGWR) then springs into life, and flushes the contents of the Log Buffer down to the Online Redo Logs. As soon as the write to the Log Files is finished, you are told your transaction has committed. The lock is thereafter removed from the record. A commit has, you may remember, absolutely nothing to do with writing the data from the Buffer Cache back down onto our data files –writing out large 16K blacks would take much too long. Redo, being simply the actual piece of data being changed –not even the entire record- is much smaller, and hence much faster, to write out.

What that means is that the Redo Logs are crucial to correctly interpreting the contents of our data files. It's apparent from even this brief discussion that the data files will ordinarily contain data that is both committed and uncommitted –it's the presence in the Redo Files of the before and after images of the data, together with all relevant SCNs, that allow us to look at the data in the data file and say, for example, 'Now this is the before image, and the change to it was committed. This is the after image, but the change that caused it to be this value hasn't been committed yet'.

In the event of Instance Failure, therefore, we will rely on our Redo Logs to tell us what to do to the data in our data files: if it is the before image, roll it forward. If it has been committed, leave it in that rolled forward state. Otherwise, roll it back again. And so on.

2.3 Archiving Redo

Given that we rely on our Redo Logs like this, it should be something of a worry to see that as we fill up one Redo Log file, we switch to another, and eventually come back to the first, ***and start over-writing its contents.*** If we've over-written a Log File, we lose all that essential information that allows us to understand what on earth is in our data files.

So, that's why we can (optionally) turn on Archiving. Archiving means that, when we fill up the first Log file, and move to the next, we take a **copy** of that just-completed Log. Actually, one of the background processes takes that copy for us: ARCH. With a copy safely taken, we can of course now overwrite the ONLINE version of the Logs: the key information they contain is secured somewhere else.

You wouldn't bother taking archived copies of your Redo Logs if you didn't care about losing transactions. In a test database (or a development

environment), for example, who cares if you can't tell whether your data files contain committed or uncommitted transactions? You can just trash the lot by restoring from an earlier backup, and losing committed transactions is hardly traumatic news! Similarly, if you work with data warehousing applications, you may not bother taking archives: if the data is just itself an archive of various on-line systems, if anything went wrong, you can safely go back to a previous backup –there won't be any transactions to lose!!

So, taking archives is one thing that can happen when we do a log switch. What else happens?

2.4 Checkpointing

Well, we perform a "checkpoint" at each log switch. In Oracle 8, the checkpoint process (CKPT) is entirely responsible for these (in Oracle 7 you get LGWR to do double-duty as the checkpointing **and** log-writing process). What happens at a checkpoint? Just three things, as follows:

- Every dirty buffer in the Buffer Cache is written back down to the data files by DBWR
- Every data file header is marked with the latest SCN (System Change Number)
- Every copy of the Control File is updated with the latest SCN

Checkpoints are fairly I/O intensive operations, then. That's one good reason to try and avoid them as much as possible –your database slows down enormously if you're checkpointing too often. On the other hand, lots of checkpoints are good for instance recovery: since **all** changed data was flushed down to disk at the last checkpoint, we only need to apply any redo that we find *after* the last checkpoint.

The other key thing to note about checkpoints is that they are the 'pulse' of the Oracle system: they mark the passage of time, without actually having to worry about whether the clock on your server has been set correctly, or adjusted for Daylight Savings. We can say of a database that it was 'time-stamped' at time '15346', or that it failed after time '17865'. It doesn't matter what actual time relates to '17865' –if the database is brought completely back to '17865', we know it is internally consistent at that point (because as that checkpoint number was issued, all data was flushed down to disk).

The SCN is therefore what SMON actually checks when you start up the database, when it is checking for entire database consistency. If the SCN in ALL the data file headers is the same, **and** matches that stored in the Control File, then we know the database is consistent. If just one file has an old SCN number, then we know something funny has gone on with it (data files don't just suddenly take it upon themselves to walk backwards in time!).

2.5 Miscellaneous

Now, we ought finally just to mention a couple of other architectural bits and pieces that have relevance for Backup and Recovery.

2.5.1 The Control File

Remember what the Control File does for us? Basically, it tells us the physical location of every data file in the database. Without a valid Control File, therefore, you won't be able to get at **any** of your actual data. It is also, as we've just discussed, closely involved in working out whether your database is healthy and consistent –because it stores the latest SCN.

Basically, therefore, we ought to *mirror* our Control Files –because losing them is serious. Mirroring means simply taking a copy of a clean Control File onto separate physical drives (so if one hard disk fails, you don't lose all of them). Mirroring onto the same disk is more common than it should be, and is a complete and utter waste of time!! Your Parameter File then simply lists all copies of the Control File, each one separated from the other like this:

```
CONTROL_FILES=(C:\ORACLE\CONTROL01.CON, D:\ORACLE\CONTROL02.CON)
```

2.5.2 The Large Pool

Now, you won't actually have heard of the Large Pool before –certainly not on the standard DBA course, anyway. It's actually yet another part of the SGA, and it's Oracle 8 specific. There is no such structure in an Oracle 7 database. It's also optional in Oracle 8!

What it's for is simply to provide working space in memory for the new Oracle 8 Recovery Manager (RMAN) program (which we'll look at briefly on the last day of this course). Recovery Manager is a tool that can perform backup and recovery operations for you, and automatically work out which files are needed for any particular recovery operations. If you want to dedicate memory specifically to RMAN, then you create a Large Pool simply by setting a parameter "LARGE_POOL_SIZE" in the Parameter File. If you

aren't going to use RMAN (and most people don't), then don't bother with a Large Pool.

Incidentally, RMAN will work perfectly well without a dedicated Large Pool: if you use RMAN, but didn't reserve a Large Pool, RMAN just pinches what memory it needs from the basic Shared Pool.

Backup and Recovery:
Chapter 3 – Configuring for Backup

3.1 To Archive or to Not Archive?

Now we've already talked about *why* we may need to Archive our Redo Files: as a quick reminder, our Redo Logs are the things we need to have in order to repeat transactions, and to interpret what is in our data files correctly (whether they contain committed data, whether they contain the before or after images of the data and so on). Therefore, merrily over-writing your Redo Logs (which is Oracle's default behaviour) is pretty serious stuff… . you lose all ability to recover the database completely in the event that trouble strikes.

The decision as to whether or not to take archive copies of your Online Redo logs as each of them fills up appears therefore to be a bit of a no-brainer. There's actually more to it than this, though. Here's the full story behind NOT taking archives:

- You must backup the ***entire*** database (redo logs, control file and all data files) at each backup operation
- The entire database must be shutdown when you backup -otherwise the backup is useless
- You can only restore the ***entire*** database from the last backup taken
- You lose all transactions since the time of the last backup, committed or not
- If a tablespace (i.e., a ***part*** of the database) goes berserk, you either have to drop it, or restore the entire database –you can't do backups and recovery at the tablespace level meaningfully

So, if none of these points is of worry to you, by all means don't bother taking archives. Life is much simpler without them! You'll note that you simply work with the database on a **whole database** basis, and you can practically guarantee that committed transactions (if there are any) will be lost. You also have the added benefit of not having to worry about the rest of this chapter!

The reverse points apply to when you *do* decide to take archives:

- You can backup parts of the database (entire tablespaces, or individual data files)
- The database can be either open or closed when you take backups –it really doesn't matter

- You can restore just the bits of the database that are causing the problem
- You can be sure that every single committed transaction will always be recoverable

3.2 How to switch on Archiving

OK, so now you know whether you want to archive or not. The **default** behaviour for Oracle is ***not to archive.*** So, if you don't want to archive, *do nothing! (Skip to Chapter 4 instead).*

If you want the greater flexibility and security that results from archiving, you'll have to switch on archiving. This is not quite the 'throw a switch and relax' process that perhaps it should be.

3.2.1 Step 1: Set Archivelog Mode

First, set the database into **ARCHIVELOG mode.** When the database is in the **MOUNT** state, issue the following command:

```
ALTER DATABASE ARCHIVELOG
```

This is simply an instruction to the database *not to allow an Online Redo Log to be over-written before a copy has been taken.* It is NOT any form of instruction to Oracle to do anything about actually taking the copies! Once issued, the database remains in ARCHIVELOG mode permanently, through all subsequent shutdowns and start-ups. (To stop being in ARCHIVELOG mode, simply issue the command 'ALTER DATABASE NOARCHIVELOG' –again, in the MOUNT stage). After you've switched into ARCHIVELOG mode, you can open the database with the 'ALTER DATABASE OPEN' command.

NB: You won't be able to successfully issue this command if the database was earlier shut down in an 'unclean' fashion. In other words, if the last thing you did was a SHUTDOWN ABORT then it's no good trying to open in the MOUNT state and issuing the ARCHIVELOG command. You'll get an error message saying:

ORA-00265: instance recovery required, cannot set ARCHIVELOG mode

3.2.2 Step 2: Switch on Archiver

Now, if you simply switched the database into ARCHIVELOG mode and did nothing else, you'd be setting yourself up for a guaranteed system hang: you've told Oracle not to overwrite the Redo Logs until they've been

copied, but you've done nothing about actually copying them. When Oracle comes back to a used Redo Log, therefore, it will simply sit there, waiting. Not good!

So, you need to switch on the ARCH process ('Archiver'). This is the background process responsible for actually taking copies of each of the Redo Logs as they fill up.

Unfortunately, there are two ways to do this. The first, assuming the database is already open, is to issue this command:

ALTER SYSTEM ARCHIVE LOG START

This switches ARCH on, and it will immediately start to archive Redo Logs as they fill up. However: if you shut the database down, and re-start it, ARCH will revert to being in the 'Switched Off' mode. This again means you are riding for a system hang (because the database will *remain in ARCHIVELOG mode through a shutdown and re-start).*

So, to ensure that ARCH is always started whenever the database is re-started, enter the following parameter in your Parameter File:

LOG_ARCHIVE_START=TRUE

3.3 Where are my Archives being created?

The short answer to that question is: wherever you want them created… only you'll have to tell Oracle about where you want them put. First, open up your Parameter File once more. Add the following line:

LOG_ARCHIVE_DEST= *path*

… where 'path' refers to any appropriate drive and directory designator. For example, 'LOG_ARCHIVE_DEST=E:\ARCHIVES\'.

Whilst you're at it, you may as well tell Oracle what you want each of the archive files to be called. The default naming convention is 'arch%s.arc' – and the '%s' there is an instruction to include the log sequence number in the file name. So you end up with a bunch of files called 'arch103.arc', 'arch104.arc' and so on. The default strikes me as having too many references to 'arc' and 'arch', so I'd suggest, instead, a name of 'arch_%s.rdo'. But it's up to you.

If you want to specify your own naming convention, enter the following line into your Parameter File:

LOG_ARCHIVE_FORMAT=name

… where 'name' is whatever naming convention you fancy. Just make sure that there is a reference to '%s' in the name you choose, otherwise all your archives will have the same filename –which means that Oracle will refuse to overwrite the first with the second, and you'll be left with the prospect of a guaranteed system hang when the second archive can't be created.

3.4 For the Truly Paranoid… .

You may well want to consider multiplexing your archives! In Oracle 8, you can actually only *duplex* your archives (i.e., there can only be two copies of each file made). Oracle 8i lets you have up to 5 copies of each Archived Redo Log file.

If you want to duplex your archives, enter the following line in the Parameter File:

LOG_ARCHIVE_DUPLEX_DEST=path

… and once again, the 'path' there should be replaced by something like '/disk3/archivedest/' –or whatever is appropriate. Just make sure you duplex onto a separate **physical** device from the original archive destination; otherwise you might just as well not bother duplexing in the first place! Also, you'd potentially cripple your system, because it would have to wait for two lots of conflicting I/O on the same device to finish before an Online Redo Log could be overwritten.

For the same reason, it is **not** a good idea to suggest a duplexing destination down a network wire somewhere… you would then be depending on your network to run flawlessly, and at speed –otherwise, your database will hang.

(Incidentally, you can specify the 'LOG_ARCHIVE_MIN_SUCCEED_DEST' parameter, too. Set it to '1', and only one of the archives needs to succeed. Set it to '2', however, and both copies must successfully be made before Oracle deems the Redo Log to have been archived.

3.5 Doing Other Things With ARCH

3.5.1 Stopping Archiving

Having switched ARCH on, you may decide to switch it off (perhaps your archiving hard disk is full, and you want to do some maintenance there – though this would be a sign of exceedingly *bad* DBA-ing!)

To switch ARCH off, issue the command:

ALTER SYSTEM ARCHIVE LOG STOP

… but be aware that issuing this command whilst still in ARCHIVELOG mode will render you liable for a system hang (because switching ARCH off does **NOT** take the database out of ARCHIVELOG mode).

3.5.2 Manually Archiving

You'd be mad to do it on a regular basis, but there are times when you may want to archive a specific Redo Log, and not let ARCH decide for itself what to archive. The command to do that is simply:

ALTER SYSTEM ARCHIVE LOG SEQUENCE X

… where "X" is the number of the Online log you wish archived. You can replace to words 'SEQUENCE X' with various other options –the most common is with the word 'CURRENT'. Hence 'ALTER SYSTEM ARCHIVE LOG CURRENT' guarantees that the current Online Redo Log is archived.

Backup and Recovery: Chapter 4 – Doing the Deed: Taking Backups

So you need to decide whether to take archives of your Redo Logs or not. See Chapter 3!

How you actually perform backups depends almost entirely on the nature of that decision. If you chose **NOT** to take archives (i.e., you are running the database in NOARCHIVELOG mode, you will ***have*** to perform COLD backups. If you decided you *would* archive your Redo Logs, you will therefore be running in ARCHIVELOG mode – and you accordingly have the choice to backup either cold or HOT (most likely, operational requirements will dictate that you perform hot backups).

4.1 Cold Backups ("Offline Backups")

Conceptually, these are extremely straightforward:

- Shut down the database (do NOT 'SHUTDOWN ABORT')
- Copy everything onto appropriate backup media (tape etc)
- Start the database up again

What exactly does "copy everything" mean? Just that the backup set should include:

- All data files
- All copies of the Control File
- All online Redo Logs
- A copy of the operational Parameter File
- A copy of the operational Password File (if there is one)

Watch out when you copy these files when your Server is used to host more than one database – SYSTEM01.DBF could be the name of the System data file for ALL your databases, for example. You'll need to be certain that, when it comes time to restore, you can guarantee to restore the right file to the right database (it's been known to go wrong!).

Despite being extremely simple to perform, it's nevertheless a good idea to develop scripts that do all of this automatically. You can then schedule the script to run in the middle of the night, or whenever is appropriate, with minimal user intervention. Don't forget to keep the scripts up-to-date: if you add new tablespaces, there are additional data files to be backed up.

Use your V$ data dictionary views to get bang-up-to-date information about what files are needed to be backed up (for example, V$DATAFILE, V$CONTROLFILE, and V$LOGFILE).

Incidentally, when it comes time to shut the database down, avoid SHUTDOWN ABORT. That causes an 'unclean' shutdown, in the sense that the database is left in a state requiring Instance Recovery. This is not necessarily disastrous, but it means that subsequent recovery will take longer than it should do. If you can't issue the plain ol' SHUTDOWN NORMAL command with any degree of certainty of timely shutdown, at least use the SHUTDOWN IMMEDIATE version. (Quite why a database which doesn't run in ARCHIVELOG mode still has Users logged in at 3.00am in the morning, however, is a significant question on its own –sounds like Users needs educating to me).

4.2 Hot Backups ("Online Backups")

Rather more awkward to perform, the beauty of hot backups is that you backup a piece of the database at a time, whilst ordinary Users are still accessing (and updating) the entire database (including the bit you're backing up).

The basic steps to take are easy enough to grasp:

- Put the appropriate tablespace into 'backup mode':

  ALTER TABLEPACE *name* BEGIN BACKUP

- Take a copy of the data file(s) associated with that tablespace onto appropriate backup media (eg, a tape)

- Take the tablespace out of 'backup mode':

  ALTER TABLESPACE *name* END BACKUP

- Force a checkpoint so that all the data file headers are updated with a new SCN:

  ALTER SYSTEM SWITCH LOGFILE

You repeat this cycle for every tablespace in the database, one at a time. Don't issue a whole heap of 'BEGIN BACKUP' commands, do a whole batch of copying, and then a mass of 'END BACKUP' commands –the amount of Redo generated whilst a tablespace is in backup mode is (relatively speaking) huge, and puts your Redo system under a great deal of strain.

It doesn't matter how long it takes you to cycle through every tablespace in your database like this… provided you keep ALL the archived Redo from the time you started backing up the FIRST tablespace. Often, you'll come

across sites where it takes an entire week before all tablespaces are backed up. Not a problem.

This sequence of events takes care of backing up your data files. It does nothing for you Control Files, Password Files, Parameter Files and so on. These still need to be backed up as well, of course. In fact, the range of files you need to ensure are backed up for Hot backups is precisely the same as for Cold backups –***except that you should NOT backup the Online Redo Logs.***

There's no real harm done if you *do* decide to backup the Online Redo Logs – it's just that it's a complete waste of time to do so. Because the database remains open throughout the backup process, the Online Logs are continually being written to, as Users perform work in the system. The second you take a backup of such a file, it is out of date –and out-of-date Redo Logs are rather less than worthless.

4.3 Backing up the Control File

All of the above, in one manner or another, ensures that your data files are backed up. You've also seen that your Online Redo Logs are either being backed up (in Cold mode), or being ignored (in Hot mode) because backing them up would be a waste of time.

You need to consider backing up the third member of the Oracle database triumvirate: the Control File.

If you're taking closed database backups, you can do this just as you would for the rest of the database: with the database closed, you can simply perform a normal operating system-style copy of the file onto tape, or any other appropriate backup medium.

If you're taking open database backups, it's a bit trickier. You can still take binary copies of the file, but you probably will need to get Oracle to make the copy for you: issue the command

```
ALTER DATABASE BACKUP CONTROLFILE TO 'filename'
```

… and 'filename' there should be replaced by a fully-qualified path and filename combination that's suitable for your operating system.

On the other hand, working with binary backups of a Control File can get pretty tricky (as you will find out later in this course!), so my fervent recommendation is that you issue, instead, the command:

```
ALTER DATABASE BACKUP CONTROLFILE TO TRACE
```


This creates a User trace file (i.e., it's created in whatever directory your USER_DUMP_DEST parameter points to), which is actually a script that can re-create ***all*** your Control Files in one easy process. To convert the trace file into a useable script file, you just have to do these three things to it:

- Remove all the gumph at the start of the file (the comments, the authorship drivel and so forth)
- Add one line right at the top: CONNECT / AS SYSDBA (or whatever gets you on as a Privileged User in your environment)
- Add a "PFILE=initxxxxx.ora" to the existing line that reads 'STARTUP' –and make sure it points to the right Parameter File for your database

The reason that this trace file is not the exclusively-recommended method of protecting your Control Files is that the script will be out-of-date the minute you make a physical alteration to your database. If you add a new tablespace, add a datafile to an existing tablespace, drop a tablespace, or change your Redo Log group/member configurations… the script file is immediately rendered worthless.

You must therefore remember to take a fresh trace file dump every time you make a change to the physical configuration of your database.

For this purpose, remember that changing tablespace into read-only, or changing it back into read-write, results in the SCN being updated in the data files associated with that tablespace. Since the Control File knows about what SCN to expect to find in the headers of each data file, you need to regard altering tablespace between Read-Write and Read-Only statuses as a physical change to the database –so re-issue the 'backup to trace' command each time you do it.

Personally, I would schedule a simple CHRON or AT job to take a backup to trace every evening. It takes all of about 5 seconds to perform, and it protects against forgetful DBAs who make changes to the structure of their databases in 'fire fighting' mode, and neglect thereafter to update their Control File backup scripts.

4.4 NOLOGGING and Backups

You can flip tables between LOGGING and NOLOGGING modes. Typically, you flip into NOLOGGING mode just before a bulk data load from an external source (because it makes it run much quicker, and the load is repeatable anyway). You should always be careful to switch BACK into LOGGING mode once the load has been performed –otherwise all your subsequent transactions on that table will be unrecoverable.

But watch out: If you switch into NOLOGGING, do a data load, then resume LOGGING, the base data that was bulk-loaded is **un**recoverable. Yet you'll

have transactions recorded *after* the load that Oracle will attempt to repeat during recovery (because they *were* logged). If those transactions relate to data which was only loaded as part of the (un-logged) bulk load, you are going to get yourself into a right old mess.

Short answer: every time you finish a bulk load, and you switch a table back into LOGGING mode, do a backup of the entire tablespace involved. That way, your baseline for recovering the tablespace *includes* the otherwise-unrecoverable data.

Backup and Recovery: Chapter 5 – What Can Go Wrong?

With your database now safely backed up, either Hot, Cold or vaguely lukewarm, you can rest assured that you have some form of protection against disaster. But what sort of disasters precisely are you protected against? What, in other words, can go wrong with our databases?

5.1 Statement Failure

Not your problem! "Statement Failure" is a posh way of saying 'bug'. Time to employ some new developers maybe… but otherwise, the DBA is not really involved here.

A statement failure is when the application issues a statement that results in scrambled data, or causes an endless loop –or otherwise wreaks havoc. It could also be that the application is attempting a perfectly good statement –but simply has insufficient privileges actually to execute what it is trying to do (that implies the DBA needs to look at privileges and roles).

One other aspect of statement failure *does* clearly involve the DBA. If the statement is trying to insert a new record, for example, that might mean the relevant table needs to acquire an extra extent. The whole thing can then fall over if there is a quota on the tablespace –or if the tablespace has genuine run out of room (or the disk has!).

Apart from fixing up the application logic, checking privileges, or allocating extra space to a quota or to a tablespace (and maybe buying an extra hard disk or three), therefore, the DBA doesn't get too concerned about statement failures. They are problems, but with no horrible side effects or unknown 'gotchas'.

5.2 User Process Failure

Again, not really your problem. If Users insist on doing stupid things like switching off their PCs whilst connected to the Oracle database, then they must expect trouble. The best you can expect to do is to gain some vicarious satisfaction from giving them a stern talking to for behaving like idiots in the first place.

'User Process Failure' is another fancy-schmanzy way of saying that the User Process that was doing some work for (amazingly enough) a real-life User suddenly had the rug pulled out from it. Maybe the User just switched off his PC; maybe (but don't believe it straight off) the User's PC simply crashed (they were running Microsoft Windows after all). In either case, the poor old User Process that was in their control is left high and dry, and whatever they were doing terminates abnormally.

If they were in the middle of a ten-million-record update, tough luck: 5 million updates are automatically rolled back, by PMON as it happens.

And that goes for User Process failures generally. They are not your problem, because PMON does all the spotting of the problem, and all the fixing of it up too –including releasing record locks owned by an ex-User, and rolling back all and every uncommitted transaction that User had pending. The User will no doubt moan at you, and harass you interminably: tell them to get a life, and a better operating system for their PC.

5.3 User Errors

"User Errors". Yet another euphemism… this time for 'User Stuff-Ups'.

```
DROP TABLE ORDERS
DELETE FROM CUSTOMERS
TRUNCATE TABLE ACCOUNTS
DROP TABLESPACE ALL_DATA INCLUDING CONTENTS
```

… all perfectly good statements (no 'Statement Failures' there, then). Equally, it is true to say that the User Process didn't fail before carrying out those instructions (more's the pity). No, here we have a class of error that is just genuinely better know as 'stuff up'… the "er, I just did something stupid boss… " school of stuff-up.

There is much gnashing of teeth and weeping and wailing amongst DBAs when this sort of error occurs. Even though it wasn't your fault, the DBA is now embarked on a lengthy and usually troubled program of resolution of the error that tends to leave everyone unhappy.

You can always try and educate your Users such that these problems are minimised, but personally, as a professional trainer, I rather think you're wasting your time. Users are, well… Users, after all. I'd hit them first, to make you feel better. And **then** you have to start trying to fix the problem.

Unfortunately, there are no really nice fixes.

If a table has been dropped, you can't roll that statement back. (DDL doesn't have rollback). The best that you can hope for is to restore from a previous version of the database, where the table was still present, and roll the database forward to just before the time when the User tells you they issued the command to drop the table (known in the trade as a 'Point in Time' recovery). This approach has two problems. First, it means that all transactions entered in the database after the 'drop table' command are lost. Second, you inevitably find that the shamed (and beaten) User has lost all sense of timing –he tells you the thing was dropped at 10.54. So you restore to 10.52. You then find that his watch is 10 minutes fast…. The restored database is still missing the required table (another hefty wacking session is then in order… but you might consider buying your User a new watch, as a form of future insurance).

Time to hit the User just one more time, and repeat the entire process.

You could attempt to recover the lost table from an Export file. That has the benefit of not requiring fancy point-in-time recovery methods. It also has the distinct disadvantage of bringing back a table without ANY of the transactions that affected it from the time the last export was taken –*and with absolutely no chance of ever applying Redo to the restored table* (that little sting in the tail is a feature of exports).

Humour (or attempted humour) aside, therefore: User errors ***do*** require inordinate amounts of DBA intervention, for recovery results that are compromises at best.

5.4 Instance Failure

Instance failure is serious, but not particularly nasty –and the DBA doesn't really have to worry about doing much to recover from it. Oracle does all the work for you.

"Instance failure" means that the structures in memory that are managing your Oracle database simply die. Classic causes include friendly road workers slicing through power cables to your office (your instructor has had this happen 3 –yes, three!- times), power supplies blowing up, hard disks (containing the OS) dying or (perish the thought) operating systems deciding they've had enough for the day (Unix is, of course, totally immune from this last case!!)

You really don't have to worry about Instance Failure, however.

Re-start your Server (assuming new power supplies or OS-containing hard disks have been installed). When you start, SMON will compare the SCN stored in the Control File with that stored in each of the data files. There

will be no discrepancy noted, because an Instance Failure **does not** mean that your database is in an inconsistent state.

However, SMON will then notice that there are transactions in the Online Redo Logs ***after*** the last SCN was issued. Now that simply can't happen under normal circumstances (if you shut down cleanly, the very *last* thing that happens is that an SCN is issued). So at that point, SMON knows that it has to read the Redo Logs for the last, post-SCN, transactions –and thus to repeat them.

Instance Recovery is therefore initiated, and the transactions after the last SCN are repeated –whether committed or not (it doesn't matter at this stage). When all the transactions after the last SCN have been replayed, from the Online Redo Logs), the message is passed to Users that the database is open for business. The rollback of transactions that were actually not committed is then conducted as an on-going process: as data is requested, the system rolls back the relevant transactions, so that the agreed version of the data is presented to Users. This basically means that for a good while after the database is open for 'normal' use, response times are generally sluggish, until all pending transactions are rolled back.

5.5 Media Failure

The nastiest of the lot, and the one that really involves us on this course, Media Failure basically means that a hard disk has gone AWOL, a disk file has become irreparably corrupted, or a trainee DBA (the Lord Preserve Us!) has deleted a file from a hard disk (Note: Trainee DBAs are **dangerous.** Get them out of training as soon as possible. The dangers of partial ignorance should never be underestimated).

Media failure is much more serious than Instance Failure. The real point is this: a data file is detected as either not being there at all, or as being there but with an SCN timestamp that is out of synch with the rest of the database (because you chose to restore a file from an earlier backup).

How you recover depends intimately on whether you decided to archive or not. If you don't have archives, life is simple, but limited: you restore the ***entire*** database from the last backup. You **definitely** lose any committed transactions after the last backup.

If you *have* been taking archives, then –in principle- you restore the duff file from backup, and play it forward in time to the SCN of the rest of the database by applying the redo contained within the Archived Redo Logs. Committed transactions up to the point of media incompetence are therefore recovered in their entirety.

This is basically what the rest of the course is all about.

The fundamental truth is that unless all data files and online Redo Logs share the same SCN as that stored in the Control File, the database will be spotted (by SMON) as being inconsistent at startup –and SMON will then request the application of whatever archived Redo is available that will *make* the database consistent. If the archived Redo is not available (because you aren't running in ARCHIVELOG mode, or because something awful has happened to your archives (e.g., trainee DBA let loose), then the database is only recoverable from the last complete backup.

5.6 The Alert Log

Always check your Alert Log. It will tell you the system time for a lot of system events (such as dropping of a tablespace, or creation of a new one). If you need to do a 'Point in Time' recovery, here's where you find the relevant times to recover until.

Similarly, the Alert Log tells you anything that went wrong with the database: if there's corruption in the data files, for example, the Alert Log will be the place where you'll find that mentioned.

5.7 Preventative Maintenance

There are three methods that you can use to try and minimise the risk the risk of data file failure in the first place. Be aware of them –and then be aware that they are crippling for an ordinary production database.

5.7.1 Data File Checksums

You can set the DB_BLOCK_CHECKSUM parameter to be TRUE. Every block then has a checksum calculated and stored in the block header. Every subsequent block access involves comparing a freshly-computed checksum with that stored in the header. If corruption occurs, the discrepancies in the checksums will reveal it. Disastrous performance implications, however.

5.7.2 Log File Checksums

Same deal, but related to Log Files, rather than data files. To use these, set the LOG_BLOCK_CHECKSUM parameter to be TRUE. The nice thing about this parameter is that it means that ARCH reads and compares the log block

checksums before it copies the file: if there is any discrepancy, it moves to the next log file in sequence. This is therefore a handy tool when trying to ensure that your Archived Redo files are 'clean'.

On the other hand, it's such a performance dog that it barks. Not something you'd switch on without due consideration.

5.7.3 DBVERIFY checking

Instead of computing checksums on the fly, and thereby knackering your system, you can manually run the DBVERIFY utility to check the integrity of your data files periodically.

On UNIX, you run dbv. On the rather wordier NT, you run dbverf80. For example:

```
DBV FILE=C:\THIS_ONE_OVER_HERE.DBF LOGFILE=D:\WRITE_RESULTS_HERE.LOG
```

This is a useful utility for ensuring that the file you are about to restore is actually clean and useful… it prevents that awful sinking feeling you get when you spend half and hour restoring a file which turns out to be just as corrupt as the one you are trying to recover.

Backup and Recovery:
Chapter 6 – Complete Recoveries

The term "Complete Recovery" is used whenever we're talking about a recovery process that finishes up recovering **all** committed transactions – as opposed to an 'Incomplete Recovery', where we (more or less reluctantly) end up with some committed transactions having been lost.

Now, there are various ways of achieving a complete recovery… it all depends on exactly what has gone wrong in the first place. But all have one thing in common: they are ways to recover from **Media Failure**. As we saw in the last chapter, that's the *only* sort of failure that really requires the DBA to do some work.

6.1 Recovery without Archives

Nothing terribly clever goes on here. You have media failure of one sort or another (a hard disk dies, your trainee DBA deletes a data file, or maybe the Alert Log simply tells you you've a severe case of corruption in one of your files). Recovery consists simply of restoring the **entire** database from the last backup, and then manually re-keying all the transactions that hit the database after the time of the last backup.

You therefore must restore:

- All Control Files
- All Data Files
- All Online Redo Logs

You'd also restore the Password and Parameter Files if they'd gone walkabout, but not otherwise.

How long does this form of recovery take? Simply the time taken to copy back all the relevant files. Recovery time is a function of hardware capabilities.

6.2 Subtle Plans without Archives

You can do better than the basic plan outlined above ***if the needed Online Redo Logs haven't been over-written.*** For example:

- You have three Redo Logs – 150, 151 and 152.
- You took a closed database backup at 150.
- A hard disk containing data files dies whilst Oracle was busy writing to log 152

You can just about get away with restoring the missing data files from the previous backup, rather than the entire database, because all the Redo that we need to apply to the restored data file is available in the Online Logs. In other words, we restore the data files at time 150, and then apply all the redo from Logs 150, 151 and 152 – and Lo! The database is made consistent, with not a lost transaction in site.

Provided all the required Redo is available from the Online Redo Logs, you can get away with just restoring the missing parts of your database, and rolling them forward.

To apply the required Redo, just type the command:

```
RECOVER DATABASE
```

… when you are left sitting in the MOUNT stage after a failed attempt to start up the database (the startup fails because Oracle spots something has happened to your data files).

6.3 Moving Files before Recovery

Given that we are recovering from media failure, it is highly likely that you won't be able just to restore files back to the original homes – if one of your hard disks dies, you don't want to have to wait for replacement hardware before attempting a recovery. Chances are, then, that as well as restoring files from backup, **you'll have to move them too.**

6.3.1 Control Files

If your control files need to be moved onto new disks, just edit your Parameter File to point to the new Control File locations, before trying to start up the database. That means editing the CONTROL_FILES=(X,Y,Z) parameter: just make sure X, Y and Z really do point to functioning Control Files.

6.3.2 Data Files

Moving Data Files is a bit more involved. You have to be able to talk to the Control File, and re-set the pointers it contains to the new locations of the Data Files. So, having restored the missing Data Files to their new locations, you then start the database in the MOUNT state:

```
STARTUP MOUNT PFILE=initSID.ora
```

You then issue as many 'Rename File' commands as you need to re-set all the required pointers:

```
ALTER DATABASE RENAME FILE
'E:\ORACLEDATA\DATA01.DBF' TO
'C:\TEMPORADATA\DATA01.DBF'
```

… keep going until all the affected Data Files have been re-pointed.

If you are able to do the subtle recovery described earlier (because all required Redo is in the Online Redo Logs), you'd try that now (by issuing the RECOVER DATABASE command). Otherwise, you'd simply be able to issue the ALTER DATABASE OPEN command to get the database back open for business.

6.4 Recovery With Archives

The basic principle to observe when recovering a database that has archived Redo available is: you DON"T restore the entire database, just the bits of it that have a problem. You certainly **don't** start restoring the Control File, or the Online Redo Logs. Having restored the faulty Data File from an earlier backup, your database is left in an inconsistent state, so you then apply all the Redo generated from the time of the last backup (archived as well as online) to that file to bring it back up to date.

The time taken to perform such recoveries is therefore a function both of the time taken to restore the faulty files from backup, **and** the time required to apply all required Redo to those files.

The specifics of how you actually do a recovery, and what syntax you use, depends entirely on the state of your database at the point of media failure, and what operational constraints you are running under. We'll go through the various scenarios one by one.

6.4.1 Media Failure Shuts the Database Down

Suppose that the hard disk that decides to blow up is the one that is used to host the SYSTEM tablespace. Its failure will therefore inevitably mean that your database dies horribly, crashing and taking all the Users with it. For

our purposes, the key thing is that **the database is closed prior to recovery.**

This is the simplest scenario. You simply:

- Restore the lost Data Files from the last backup, to new hard disk locations if needed
- Start the database in MOUNT state
- Use the 'Alter Database Rename File' command to relocate the Data Files if needed (see above, Section 6.3)
- Issue the command to recover the database:

```
RECOVER DATABASE
```

That command will cause Oracle to prompt you to supply all the required Redo –and that means you need to make sure you have all the necessary *archived* Redo already available before issuing it.

6.4.2 Media Failure but the Database Remains Open

Suppose now that your trainee DBA reports that one of the Data Files that comprises the DATA01 tablespace was, er, 'accidentally' removed. First, you sack him of course. Then you get to thinking that whilst the loss of a data file like this is not pleasant, it's not terminal either –whatever your Users were in the middle of doing would most likely be trashed, but it's highly likely that the database would remain functioning, and Users accessing the DATA02 tablespace would still happily be going about their business.

The key point here, then, is that recovery is required to part of the database, **but the rest of the database remains up and running.**

Recovery is still fairly easy:

- Check if the Data File has already been taken offline by Oracle:

```
SELECT FILE#, NAME, STATUS FROM V$DATAFILE
```

(The 'Status' field may say 'RECOVERY' or 'OFFLINE' –both mean the file is offline)

- If the file is not already offline, take it offline yourself:

```
ALTER DATABASE DATAFILE 'C:\Name_of_File' OFFLINE
```

- Restore the file from the latest backup set
- Recover the individual Data File using the command:

```
RECOVER DATAFILE 'C:\Name_of_File'
```


- Bring the recovered Data File back online:

  ALTER DATABASE DATAFILE 'C:\Name_of_File' ONLINE

Once again, before issuing the "Recover Datafile" command, make sure all the necessary archived Redo is ready to hand.

You'll notice here that we're working with individual Data Files. You could equally well with the entire tablespace –in other words, take the whole tablespace offline, restore the relevant files (maybe just one or two that make up a tablespace of many), then recover the entire tablespace. The command to do that would be: RECOVER TABLESPACE DATA01. You'd then bring the whole tablespace back online, of course.

6.4.3 Getting the Database open as quickly as possible

This scenario is really just a clever combination of the previous two. Suppose media failure has resulted in the crash of the database: the database is thus closed (just like in our first scenario at 6.4.1 above). However, you need to get Users back onto the system as quickly as possible. You can't just open the database (because one of its files is missing!). But what you *can* do is to use the techniques from our second scenario (6.4.2 above) to offline the damaged part of the database, allowing the rest of it to be opened whilst you get on with recovering the damaged bit.

There is just one proviso here, though: **You can't do what follows if the damaged part of the database contains the SYSTEM tablespace** (hopefully for obvious reasons) **or if it contains ALL of your Rollback Segments** (because there must be some non-System Rollback Segments available if useful work is to be carried out on the database).

The steps involved are:

- Start up the database in the MOUNT state
- Check if the Data File has already been taken offline by Oracle:

  SELECT FILE#, NAME, STATUS FROM V$DATAFILE

  (The 'Status' field may say 'RECOVERY' or 'OFFLINE' –both mean the file is offline)

- If the file is not already offline, take it offline yourself:

  ALTER DATABASE DATAFILE 'C:\Name_of_File' OFFLINE

- Now open the database:

  ALTER DATABASE OPEN

(Taking the dodgy Data File offline means that the database sort of 'forgets' that it ever existed –so what is left is nice and consistent, and capable of opening without a problem)

- Finally, restore and recover the dodgy Data File(s) as normal. Restore the file from tape (and 'alter database rename file' if it's restored to a different location), then use the RECOVER DATA FILE 'C:\OVER_HERE' command to apply archived redo to the restored file. When recovery is complete, you can bring the Data File back on-line.

The point here is that you get most of the database up and running before recovering the broken parts. Note that if an entire tablespace needs recovery, you **cannot** issue a 'alter tablespace offline' command at step 3 above (because that command can only be used when the database is fully open). You are reduced, therefore, to issuing as many 'alter database Datafile 'X' offline' commands as are needed to cover all the relevant Data Files in a tablespace.

However, once the restore from backup has been done, you *can* issue commands to recover and to online the entire tablespace –because by that stage the database has been opened. The relevant syntax would be:

- **RECOVER TABLESPACE DATA01**
- **ALTER TABLESPACE DATA01 ONLINE**

6.4.4 Recovering a File without Backup

This gets seriously clever!

Imagine one or more Data Files disappear, and that when you start the usual recovery processes described earlier, you discover you fall at the first hurdle: you don't have a backup of the data files in question. This is more common than you might think: backups fail, tapes have physical failures, or (very commonly) the harassed DBA forgot to include a new Data File in the normal backup scripts.

Whatever: the point is, how can you recover the file when you can't supply a baseline backup to which Redo can be applied? Amazingly, perhaps, you **can** actually do it, provided you have available EVERY piece of Archived Redo ever generated since the file was first created.

(In fact, this is such an onerous requirement that this technique really only works for Data Files which have been created fairly recently –say, within the past week or three).

The basic recovery strategy is as follows: Offline the Data File; Create a new Data File with some fancy syntax; Apply Redo to the new file. Clearly, the heart of the matter is to be found in that 'create a new Data File' bit. Here's the clever syntax to do that:

ALTER DATABASE CREATE DATAFILE 'E:\OLDFILE.DBF' AS 'C:\NEWFILE.DBF'

The key thing here to understand is that this causes the creation of a brand new operating system file, *with all the characteristics of the original,* ***including its Oracle identifiers.*** That means that Redo can be applied to the new file because, to Oracle, it looks and feels like the original file. It's no good, in other words, just creating a new file in the usual way –Oracle will see it as a different file, and will refuse to apply 'old-file' Redo to it.

Once you've recreated the file, you can recover it as you would any other Data File, always assuming, of course, that you have ALL the Redo generated since the file was first created.

One **extremely nasty gotcha** with this syntax: to all right-thinking people, the syntax reads 'create a new Data File as a copy of the old one', and therefore you specify the new filename first, and the old one (that has been damaged or lost) second.

However, this is **completely the wrong way round.** The syntax is *supposed* to be read like this: 'create me my old file once more, only this time call it something else'.

In other words, the syntax runs 'alter database create datafile X as Y' – *where X is the OLD filename and Y is the NEW.*

6.5 Bits and Pieces

6.5.1 Helpful Data Dictionary Views

Some data dictionary views are particularly useful when trying to recover.

V$RECOVER_FILE is useful for seeing which Data Files actually need recovery. You'll get output similar to this:

```
FILE#      ONLINE  ERROR              CHANGE#    TIME
---------- ------- ------------------ ---------- ---------
         1 ONLINE  FILE NOT FOUND              0
         2 ONLINE  FILE NOT FOUND              0
         3 ONLINE  FILE NOT FOUND              0
         4 ONLINE  FILE NOT FOUND              0
         5 ONLINE  FILE NOT FOUND              0
         6 ONLINE  FILE NOT FOUND              0
6 rows selected.
```


(Incidentally, do you realise how incompetent you have to be to get such a report?! It took me ages to cripple my database like this!)

This is all very well for a grand overview of the scale of the problem, but doesn't help much if you actually want to know physically what files to restore. For that, you'll have to query V$DATAFILE:

```
FILE#      NAME
---------- ----------------------------------------
         1 C:\ORACLE\ORADATA\HJR1\SYSTEM01.DBF
         2 C:\ORACLE\ORADATA\HJR1\RBS01.DBF
         3 C:\ORACLE\ORADATA\HJR1\USERS01.DBF
         4 C:\ORACLE\ORADATA\HJR1\TEMP01.DBF
         5 C:\ORACLE\ORADATA\HJR1\INDX01.DBF
         6 C:\ORACLE\ORADATA\HJR1\OEMREP01.DBF
```

… and now we can see physical filenames associated with each file number obtained from the earlier report.

6.5.2 Redo Log Issues

If an ***inactive*** Redo Log is lost, it doesn't really matter: the fact that it is inactive means that it has been archived – so its contents are safe. You don't even need to bother with recovering the errant Redo Log file: just drop it (actually, its entire group) and create a new one.

You may not always be able to drop the relevant log group however: if you've only got 2 groups, the system won't let you drop one of them. The basic syntax to deal with this situation is:

ALTER DATABASE CLEAR LOGFILE GROUP *x*

(… and you replace the X there with whatever group number is appropriate).

Similarly, you cannot drop the CURRENT redo log group, so you might choose to CLEAR that group instead.

At least, that's what the Oracle documentation tells you to do. Personally, I'd be thinking about temporarily adding a third redo group, dropping the corrupted one, and then re-creating it from scratch yourself. Similarly, if the problem is that the corrupted group is the current one, make it NOT the current one by issuing an ALTER SYSTEM SWITCH LOGFILE command, and then dropping the group isn't a problem. Why Oracle wants you to learn a whole bunch of fancy syntax I can't imagine.

Backup and Recovery:
Chapter 7 – Incomplete Recoveries

"Incomplete Recovery" is sometimes called "Point in Time" recovery. It's a general term referring to any form of database recovery where committed transactions **will** be lost. You classically do one of these to recover from a User error ("Ooops! Just dropped the ACCOUNTS table. Sorry.") The other main reason you are forced to do an incomplete recovery is when there is a gap or hole in your sequence of Archived Redo Logs –you can never recover beyond the point of the gap.

NB. The Oracle documentation for this chapter is seriously misleading, in that it talks about a third reason for doing an incomplete recovery –when you use a backup copy of your Control File to re-open a database. This is simply not true. If you use a backup Control File, it is most certainly the case that you have afterwards to issue the RESETLOGS option, which is usually associated with incomplete recoveries. But that's where the relationship ends. It is perfectly possible to recover a database, complete with ALL its committed transactions, using a backup copy of the Control File –and that would constitute a COMPLETE recovery.

The basic principles behind incomplete recoveries remain mainly the same as for their complete cousins –it's just that at some point in the recovery process, you stop further application of archived Redo to the restored Data Files. You then issue the '**ALTER DATABASE OPEN RESETLOGS**' command to allow the database to be opened with only a partial recovery having been performed.

However, there is one crucial difference between complete and incomplete recoveries: you don't just restore the damaged Data File(s), you restore ALL of them.

Note that if you are not running in ARCHIVELOG mode, this chapter is of absolutely zero significance to you! Since you don't have archives, your recovery options are so limited that none of the points raised here will ever be applicable.

7.1 Two Types of Incomplete Recovery

There are two main ways to halt the application of archived Redo to a restored data file. Either tell the database to apply redo **'Until Cancel'** (i.e., you type in the word 'CANCEL' half way through the recovery process); or tell the database to apply redo '**Until Time**' (and then you specify a specific time to roll the database forward to, and then stop).

(NB. There's actually a third way to do this job, too, but it's not covered on the Oracle training course –you recover 'Until Change' –and you

specify a specific SCN at which the system should stop the recovery process).

7.1.1 Until Time

Say, for instance, that the trainee DBA "accidentally" dropped the ACCOUNTS table. It contains crucial information, fairly obviously. It needs to be recovered –but there's no 'rollback' command for DDL commands. This is a job for an Incomplete Recovery!

This is why you don't *immediately* strangle the trainee. First you ask him very politely at what time he thinks he dropped the table. When he tells you, 'around 11.00am', it is **then** safe to strangle him: you've got all the information that you need.

The steps to recover are then these:

- Shutdown the database (this is a bit of a pain… but you cannot do incomplete recoveries by restoring bits of the database. You have to restore ALL the Data Files, and that therefore means that none of them can be in use. Hence the need to shut down.)
- Take a good closed backup of the database as an insurance policy (this is equally a pain, and can be skipped **if you really know what you are doing.** However, to do so would indicate that you are certifiably insane, since you risk rendering your database totally unusable, without a fallback option. It's your choice.)
- Restore ALL the Data Files from last night's backup
- Open the database in the MOUNT state
- Issue the following command:

```
RECOVER DATABASE UNTIL TIME '2000-04-23:10:58:00'
```

- Apply all the Redo that the system asks for
- Open the database using the crucial command:

```
ALTER DATABASE OPEN RESETLOGS
```

- Check that the missing table is back

There are a couple of things to note here. First, ***restore all the Data Files.*** The entire database complement of Data Files must be restored from the previous good backup –the whole database needs to be rolled forward to a time just prior to the dropping of the crucial table.

Secondly, the syntax of the time statement is a bit tricky to type in: make sure you get all the colons and dashes in the right place. Also beware of the twenty-four hour clock. If the trainee dropped a table in the afternoon, don't recover until, say, '3:00:00', but until '15:00:00'.

Thirdly, always give yourself a bit of elbowroom, and allow for human error. The trainee reported dropping the table 'around 11.00am' –don't try

recovering until 10:59:59 and expect the problem to be fixed. (Also, check his watch against the server's time. If it's fast or slow, adjust you 'recover until' time accordingly'. If, upon opening the database, you discover that the crucial table is STILL missing, you simply have to repeat the whole recovery process again, using an earlier time at which to stop).

Perhaps most importantly, the ***resetlogs*** option means precisely what it says: it resets the Log File sequence numbers, so that it appears as if your database has just been freshly created –the Logs are at sequence number 0, 1 and 2 (and so on). **That means that all prior backups and archive Logs are now useless, and you should therefore take a brand new CLOSED database backup immediately after the recovery process is complete, to provide you with a new baseline from which future recoveries can be undertaken.**

(Incidentally, you'll notice at least two closed backups are therefore involved with every incomplete recovery. Incomplete recoveries are not, therefore, the DBA's best friend, and not exactly desirable in a 7x24 environment.)

Finally, you will hopefully be aware of the fact that **all transactions committed after the time specified in your RECOVER DATABASE UNTIL… statement will have been lost, and will therefore need to be re-input.**

7.1.2 Until Cancel

The steps involved in doing an incomplete recovery 'until cancel' are pretty much the same as those described for the time-based recovery above. You would classically use this method of recovery when your trainee DBA drops a crucial table*, but you also know beforehand that one of your archived redo logs are missing*, or that the current Online Redo Log group has disappeared.

Since you cannot roll the system past the point of the Redo 'gap', or past the point represented by the missing Online Redo Log group, you have to accept that at the point of reaching the 'gap', you will have to cancel the recovery process yourself. Committed transactions in that gap, or after it, **will be lost.**

The steps to recover are then these (largely duplicated from above, it must be said!):

- Shutdown the database (this is a bit of a pain… but you cannot do incomplete recoveries by restoring bits of the database. You have to restore ALL the Data Files, and that therefore means that none of them can be in use. Hence the need to shut down.)
- Take a good closed backup of the database as an insurance policy (this is equally a pain, and can be skipped **if you really know what**

you are doing. However, to do so would indicate that you are certifiably insane, since you risk rendering your database totally unusable, without a fallback option. It's your choice.)

- Restore ALL the Data Files from last night's backup
- Open the database in the MOUNT state
- Issue the following command:

```
RECOVER DATABASE UNTIL CANCEL
```

- Apply all the Redo that the system asks for, keeping a careful watch on the actual archive files it is asking for each time. When it asks for the archive that you know is missing, instead of hitting the [Enter] key as normal, type the word **cancel.** That halts the recovery process dead in its tracks.
- Open the database using the crucial command:

```
ALTER DATABASE OPEN RESETLOGS
```

- Check that the missing table is back

As before, note that we ***restore all the Data Files*** –even though the problem of the missing table only affects one of them.

Secondly, you might consider multiplexing your archives to prevent this sort of problem arising in the first place.

Third, and most importantly, the ***resetlogs*** option means precisely what it says: it resets the Log File sequence numbers, so that it appears as if your database has just been freshly created –the Logs are at sequence number 0, 1 and 2 (and so on). **That means that all prior backups and archive Logs are now useless, and you should therefore take a brand new CLOSED database backup immediately after the recovery process is complete, to provide you with a new baseline from which future recoveries can be undertaken.**

7.2 Recovery to a different Database structure

Suppose that instead of dropping a single table, the trainee DBA drops an entire **tablespace.** Now, if that happens, you can't use either of the techniques we've just discussed.

Why not? Because in both of them, we restore **Data Files** and roll the whole database forward to a point just before the disaster happened. In neither case, however, did we restore or roll forward the Control Files or the Online Redo Logs. *But the dropping of a tablespace is a physical change to the structure of the database, and is recorded as such in the Control File.* You could accordingly recover your Data Files to just before the DROP TABLESPACE command was issued, but you still wouldn't see the tablespace back in

action –because you didn't do anything about the Control File, and it still thinks the tablespace was dropped.

What you really need to do, of course, is to roll the Data Files forward in precisely the way we discussed earlier, but also ***restore an old copy of the Control File –one that is aware of the structure of the database as you need it to be, rather than as it currently is.***

Here are the recovery steps in full:

- Decide whether the Data Files are being recovered 'until time' or 'until cancel' (assuming you need to perform recovery on the Data Files at all)
- Shutdown the database
- Take a closed backup of the database (we're going to be using the RESETLOGS option, so it is again advisable to have a workable database backup, just in case)
- Restore ALL the Data Files from last night's backup ***and all copies of the Control File.***
- Open the database in the MOUNT state
- Issue one of the following commands to initiate recovery:

  RECOVER DATABASE UNTIL TIME 'xxxx' USING BACKUP CONTROLFILE or

  RECOVER DATABASE UNTIL CANCEL USING BACKUP CONTROLFILE or

  RECOVER DATABASE USING BACKUP CONTROLFILE

That last one is for the rare situation where no actual data recovery is required –it just happens that you've lost all copies of your Control File and have to work with an old one. You aren't terminating the 'recovery' early, therefore, but you are using a backup of the Control File, so you tell Oracle about it.

There is at least one nasty 'gotcha' using this technique. If any of your Data Files are offline when you try and recover the database in this way, they are likely to be totally unrecoverable after the process has finished. **Therefore, always make sure that ALL your Data Files are ONLINE before issuing the 'RECOVER DATABASE' command.**

7.3 Recovery through a 'RESETLOGS'

This is tricky. Imagine you've done an incomplete recovery (because a table was dropped). You successfully recovered the table, and the Users were OK about re-entering their transactions after the time of failure. However, because you are a 24x7 shop, you decided to skip the recommended closed database backup after recovery. You just thought you'd perform the 'hot

backup' routine you generally employ (and which takes three days to cycle through).

Now imagine that two days into your hot backup cycle, one of your hard disks blows up. You can't ordinarily recover the Data Files affected – because your last good backup of them was when the Online Redo Logs were at, say, sequence number 208 –and now, thanks to the incomplete recovery and its use of the RESETLOGS option, your database is at sequence number 18.

What can you do?

In fact, you **can** recover from this situation, but it isn't easy. These are the steps to take:

- Shutdown the database
- Copy the existing Control Files to somewhere safe (the 'sequence 18' copies)
- Restore ALL the Data Files and ALL the Control File copies from the last complete backup (the 'sequence 208' copies)
- Read your Alert Log, and find the SCN associated with the previous incomplete recovery
- Open the database in the MOUNT state
- Issue the following command:

  RECOVER DATABASE UNTIL CHANGE xxxx USING BACKUP CONTROLFILE

  … where you replace the 'xxxx' in that statement with the SCN determined earlier from the Alert Log

- Allow recovery to proceed as normal, but then do a clean shutdown of the database, instead of opening it.
- Restore the 'sequence 18' versions of the Control Files
- Open the database again in the MOUNT state
- Issue whatever RECOVER DATABASE command is appropriate **(in the event of a hard disk blowing up, you'd probably try for a complete recovery, so you wouldn't qualify the syntax in some way. If the trainee DBA had been deleting tablespaces before your backup cycle was complete, however, you'd be performing a normal 'recover until' sort of recovery).**

In other words, conceptually you are doing two recovery operations. The first is an incomplete recovery to a point just prior to the original RESETLOGS operation. The second is a form of recovery after that point.

If you think about it, it isn't very hard. But there are a lot of fiddly operations going on, any one of which could be stuffed up. So take it slowly, and do each and every step rather carefully. Frankly, I wouldn't just take a backup of the Control Files as stated above –I'd be taking a complete closed backup of the database, of which the Control Files are just a part. At least you've the makings of a working database to fall back on if anything goes wrong!

7.4 Common Features of Incomplete Recoveries

In all that we've discussed about incomplete recoveries, a few common features stand out.

- **ALL DATA FILES ARE RESTORED FROM BACKUP.**
  This might seem obvious, but it is the thing that distinguishes incomplete recoveries from incomplete ones –where only the faulty Data File(s) is brought back from backup. The point is, in incomplete recoveries, the entire database is brought back to a consistent point in the past (the time of the last backup) and then rolled forward to a point just short of whatever calamity has struck the database.

- **THE DATABASE IS ALWAYS RE-OPENED WITH A RESETLOGS OPTION**
  This means that prior backups and archive Redo Logs are essentially useless (except in the unusual circumstance described earlier in section 7.3). That in turn means that you should **always** take a complete, closed backup of your database immediately after the recovery is finished. If operational requirements are such that you just have to get the database up and running as quickly as possible, be aware of your vulnerability to the sort of disaster outlined in section 7.3 above until you *do* have a complete backup.

  The RESETLOGS option also means that you need at some point to clear out your old archived Redo Logs. Otherwise, the database will start wanting to write out a new archive with a name of, say, 'arch208.rdo' –only to discover that there's one with that name already there (i.e., the one generated before the RESETLOGS). At best you'll have a system hang on your hands, because ARCH will refuse to over-write the old version. At worst, you will get yourself mightily confused about what files are actually applicable to your database, and what ones are essentially redundant.

- **ALWAYS TAKE A BACKUP BEFORE STARTING AN INCOMPLETE RECOVERY!**
  This advice is often ignored in the panic of the recovery moment, but you do so at your peril. Unless you take an 'insurance' backup (database closed, backup Data Files AND Online Redo Logs and Control File) before you attempt an incomplete recovery, you will NOT be able to repeat the recovery process… so if you carefully recover until what you think is three minutes before the dropping of a crucial table, only to discover that it *still* hasn't been recovered, you are completely, utterly and irretrievably stuffed. Assuming you still need the table back, the best you can do at that point is to restore the complete database from last night's usual backup, and take the enormous data loss that goes with it.

  Had you taken the 'insurance' backup as suggested here, you can repeat the incomplete recovery process as often as you need until the table is correctly recovered.

Backup and Recovery: Chapter 8 – Export and Import

Export is an Oracle tool that is often used by DBAs to create *logical* backups of their data – in contrast to the *physical* backups obtained by simply taking binary copies of the Data Files, Control Files and the online Redo Logs.

Backups created by the export utility are 'logical' in the sense that, if you open an export file in, say, Notepad or vi you'll see that it consists of a whole heap of 'Create Table' statements, followed by a stack of compressed data to insert into each of the tables. In other words, recovery with the use of an export file results in the creation of **brand new tables**. That has a significant implication: you can't apply old Redo to these new tables, despite the new versions sharing the same name as the old ones

Backup & Recovery:
Chapter 9 - Introduction to Recovery Manager (RMAN)

9.1 What is it?

It's an Oracle-supplied utility that performs and manages the backup (and restoration) of Oracle databases (well, the important bits of them, anyway). It has a horrible –but highly flexible- command line language that can be used to create backup scripts, or initiate backups interactively; but it can also be accessed via a GUI –the OEM Backup Manager. And as we all know, GUIs are intuitive, easy to use and … etc etc etc!

In order to achieve its tricks, RMAN stores backup information –in either the Control File of the database it's backing up, or in a special 'Recovery Catalogue'. A Recovery Catalogue is just a fancy name for a separate set of tables that is specially created for RMAN, and dedicated to its sole use. I suppose it's a bit like the OEM Repository in that respect –a bunch of tables that an Oracle utility will populate in its own good time. It is usual, though not essential, for the Recovery Catalogue to be housed in *its own dedicated database.* On the other hand, you can house it in any existing database, provided that it is contained within an RMAN schema.

Whilst you can choose to run RMAN without a Recovery Catalogue, it seems madness to do so (for reasons which shall become apparent). If you do, though, stress is placed on the Control File of the database you're backing up. It will grow big, and *must never be allowed to be totally lost* –which means multiplexing it at least once, and preferably twice.

Being an integral part of Oracle, RMAN understands the logical constructs in an Oracle database. In other words, it can do things like backing up a **tablespace,** which something like Backup Exec could never understand. It can also understand commands like 'backup files that have got too much un-backed-up redo information associated with them' (shame you can't use this sort of natural language approach, however!) … it understands what 'Redo' is, and is able to access and interpret that sort of information.

RMAN can backup to disk, or to tape (though it needs the services of a **Media Management Layer** (MML) to do this last one –something like the Legato products, for example).

When it comes time to recover a database that has gone horribly wrong, RMAN knows what has been backed up and when, and is able to pull all the

strings together: it will initiate the supply of (for example) backup tapes A, F, and M –and Lo! The database will work again.

In short, it's seriously clever, highly flexible and configurable, and a boon to the seriously-large-database administrator. However, if your database is relatively small (say a few Gigabytes or so), I would have to suggest that it *may* be more trouble than it's worth. It depends on how you take to learning convoluted syntax. On the other hand, anyone and his dog can learn how to use a GUI, so maybe it's still an option for you.

9.2 Limitations

There are a couple of limitations to RMAN of which you should be aware before we go any further.

First, it only backs up Archived Redo Logs *separately.* In other words, for any reasonably normal database, RMAN is going to have to be invoked twice –once to get the Control Files, Data Files and Online Redo Logs, and once more for the Archived Redo Logs on their own.

Second, it *never* backs up Password Files or Parameter Files. You'll have to take care of these files yourself, using some other mechanism.

Although not strictly a limitation, you might also want to think about this: if you choose to run with a Recovery Catalogue, and the Recovery Catalogue is housed in its own database, what backs up the Recovery Catalogue database? The Oracle recommended answer is that *another* database is used to house a **second** Recovery Catalogue, and each Catalogue backs up the other. Sounds a bit clunky to be honest, but it's actually fairly logical.

Finally, if you choose to run without a Recovery Catalogue (which is frankly a bit perverse –since you are choosing to run with RMAN in the first place, you might as well run it *properly*), then you cannot use RMAN to do an **Incomplete Recovery**, or **store backup scripts**. You are also in deep do-do if the Control File in which all the RMAN stuff is stored is lost or damaged: if that happens, you cannot do restores and recoveries using RMAN, full stop. Of course, if you were going to place such heavy responsibilities on the Control File, you'd be making sure the thing was multiplexed a couple of times first, wouldn't you?!

9.3 Good Things about RMAN

The fact that I listed the limitations of RMAN before its good points might tell you something about my attitude to RMAN! However, it is genuinely a powerful tool. It has a number of key plusses that should be taken into consideration:

First, it's an Oracle program that understands the contents of Oracle blocks, so *it can detect block corruption*. Taking O/S copies of your files is all very well (and nice and simple) –but any corruption within those files goes along for the copy. RMAN will detect corruption, stores information about it, and can thus navigate around the problems in the event of a restore being required. It's not perfect, but it's a lot better than nothing.

Incidentally, the fact that it understands Oracle blocks means that RMAN is able to skip over blocks that are empty when performing a backup: this is something an OS physical copy of a Data File obviously can't do. It makes for smaller, more practicable backups.

Second, because RMAN is doing the backing up (instead of an O/S utility), and because it can understand the concept of an Oracle block, it can read those blocks in an intelligent way. So when you are doing a **hot backup without RMAN,** you have to put the tablespace into backup mode ("… BEGIN BACKUP") to ensure complete blocks are backed up in a read-consistent fashion –and that places a great deal of strain on your Redo Logs. But a **hot backup *with* RMAN** requires no such Redo generation. RMAN understands blocks, so just keeps re-reading them until it is sure of a consistent backup image for each block. No Redo, no strain.

Third, you can parallelise your backups (well, you could if you can spell 'parallelise'). RMAN uses Server Processes to actually do the backing up, and is easily able to spawn multiple server processes to do the same backup simultaneously. I don't think ArqServe or Backup Exec have quite mastered this trick yet. What this means is that for large databases, RMAN might be the only practical tool available to you to ensure the backup actually gets done in a reasonable time.

Finally, you can do Incremental and Cumulative backups with RMAN, which means that only those **blocks** that have changed since the previous backup are included in the new one. (OS-based backups can only detect difference at the Data File level, of course). This makes for fairly compact backups, and again may be the difference between managing to complete a backup in a given maintenance window, or not.

9.4 Setting it Up

Note: Most of what follows is going to assume that you want to run RMAN with a proper Recovery Catalogue: it's really a waste of time to be running it any other way.

9.4.1 Creating the Recovery Catalogue

First, catch your database! Either create a brand new database, or select one that already exists, as the home of the Recovery Catalogue.

Next, you need to create the *owner* of the Recovery Catalogue. That means creating a new User in the host database (usually, though not inevitably, called 'RMAN'. Funny about that). The RMAN User needs to have a couple of specific settings, so here's the complete syntax:

```
CREATE USER RMAN IDENTIFIED BY RMAN
TEMPORARY TABLESPACE TEMP
DEFAULT TABLESPACE RCVCAT
QUOTA UNLIMITED ON RCVCAT
```

Now, you will also note that this means that you should **previously** have created a new tablespace in the host database: one called 'RCVCAT'. When sizing the Data File for this tablespace, allow 10Mb per year per backed up database (i.e., if you're going to use RMAN to back up 5 databases, make it 50Mb big, and anticipate allocating the same sort of space each year thereafter).

The RMAN User needs some basic privileges before it can actually do anything, of course, so:

```
GRANT RECOVERY_CATALOGUE_OWNER TO RMAN
```

Now, finally, you can actually generate the Catalogue –which is just a bunch of tables owned by the User RMAN (or whatever name you decided to run with). You do this by running an Oracle-supplied script, called "catrman.sql" –which always makes me think of South Park, and hence has the most memorable name of any of the Oracle-supplied scripts!

```
@$ORACLE_HOME/rdbms/admin/catrman.sql
```

This completes the creation of a new, empty Recovery Catalogue. You can now start RMAN, by typing the command 'rman' at the command prompt (on NT, the executable is called 'rman80').

9.4.2 Connecting to the Target Database

Once RMAN is up and running, you need to tell it to connect on the one hand to the database that contains the Recovery Catalogue you've just created, and on the other to whatever database it is that you want to back up (we generally refer to the database being backed up as the **target** database).

The syntax is basically this:

rman rcvcat rman/rman@rcat target scott/tiger

but it could be this:

rman target scott/tiger@DB00 rcvcat rman/rman

… the difference between the two statements is simply whether ORACLE_SID is set to the name of the Catalogue database (a 'remote connection') or to the name of the target database (a 'local' connection).

In both cases, you use two keywords, 'rcvcat' and 'target'. One is an announcement that what follows is the connect string for the database hosting the Recovery Catalogue (in the example above, I've called it 'rcat', but it could have been called anything). The other announces that what follows is the connect string for the target database.

If your ORACLE_SID is set to be the name of the target, you don't need to name it – RMAN will make a connection to the Instance with the same name as the ORACLE_SID by default. However, you'll then have to tell RMAN manually the name of the Catalogue database. This is what is happening in the first of the examples above.

If your ORACLE_SID belongs to the Catalogue-hosting database, though, the situation is reversed: you don't need to tell RMAN what it is, but you **do** need to then name, explicitly, the target database.

In either case, note that Oracle thinks of 'local' or 'remote' in terms of proximity to the *target* database, not the Catalogue.

9.4.3 Registering a Target Database

Once you've established a connection both to the Catalogue host, and to the target database, you need to initialise the Catalogue with basic information *about* the target. This process is called **registering** the database

This is probably the last intuitive bit of RMAN syntax you're going to meet: the command to register a database is simply:

REGISTER DATABASE

This command causes RMAN to read the contents of the target database's control file (so the target has to be at least in the MOUNT stage), and to extract key details about the target for inclusion in the Catalogue.

One of the key details is the unique **database identifier,** which is stored in the Control File when you first create the database. Because it's unique, you can't register the same database twice. The command then causes the structure of the target database to be transferred to the Catalogue host – what tablespaces exist, what Data Files and so on.

Whilst it remains true that you can only register a database once, you will nevertheless need to refresh the Catalogue from time to time –whenever the Control File itself changes, or if the physical structure of the database changes. We don't call these 'refreshers' registering the database, however. Instead, they are called **resetting** and **resynchronising** the database –and I'll talk about these two concepts in detail later on.

For now, with a Recovery Catalogue created, and a target database registered, you're able to get on and actually use RMAN.

Backup & Recovery:
Chapter 10 – Working with RMAN

Initial Note: all the syntax described here is typed within RMAN, and correspondingly requires termination with a semicolon. I've omitted the terminators here to try and keep things clean and simple.

10.1 Registering, Resetting and Resynchronising

You already know how to register a database (see 9.4.3). You do it once, the very first time you connect to it using RMAN. To refresh your memory, the command –staggeringly enough- goes like this:

REGISTER DATABASE

This causes RMAN to take careful note of the contents of the target database's Control File –so anything that causes changes to the target Control File should also cause you to initiate a –re-registration with the Recovery Catalogue.

However, we don't call this refreshing process a 're-registration'. Instead we go for a number of *other* words that start with 're' –resetting and resynchronising.

10.1.1 Resetting a Database

If you think about what is in the Control File, you'll readily understand when we need to give RMAN a second look at the target Control File. First off, recall that the Control File tells us what Log Sequence Number we're up to… so if you ever have a need to reset those sequence numbers, you'll need to **reset** the database in the Recovery Catalogue. Otherwise, it will still expect to be talking about 'log sequence 12,231' whilst you've caused the target to start talking about 'log sequence 3'.

In short, RMAN rule 1 goes as follows: ***every time you open the target with a 'RESETLOGS' option, you must reset the Recovery Catalogue.***

The command to do this, amazingly enough, is:

RESET DATABASE

Since this causes RMAN to take a second look at the Control File of the target database, you must have connected to the target, and to the Catalogue host database, and the target must be at least in the MOUNT stage.

A new entry is added to the Catalogue for the reset target. The target's database identifier won't have changed (it's still the same database after all). But the database will have a new **incarnation**. You can see all the incarnations of a database by typing the command:

LIST INCARNATION OF DATABASE

You'd then see a report that looks a bit like this:

```
DB Key     Inc Key    DB Name    DB ID      CUR  Reset SCN
------     -------    -------    ---------  ---  ---------
1          2          DB00       162582899  YES  23876
1          431        DB00       162582899  NO   189556
```

The top line is the current incarnation.

10.1.2 Resynchronising a Database

What else is stored inside a Control File? Only the entire logical and physical description of the database –what Data Files exist, and where, for example, together with all the tablespace names. Anytime this information changes, therefore, you again must refresh the Catalogue, only this time you use the 'resynchronise' command (the theory being, I suspect, that an alteration of the logical or physical description of the database doesn't mean we've actually changed *incarnations* –all the sequence numbers still keep ticking along as ever they did, for example).

So, another RMAN rule presents itself: ***every time you add (or drop) a new tablespace, or alter the contents of existing tablespaces, or add (or drop) a new rollback segment, you must resynchronise the Recovery Catalogue.***

(That last item in the list is a bit out of left field, I grant you. Why should new Rollback Segments require a resynchronisation? Good question. It just does… live with it!)

The command to actually initiate resynchronisation is straightforward enough:

RESYNC CATALOG

10.1.3 Now for the Bad News

The bad news is simply this: what other physical bits of the database does the Control File know about? Each and every Archive Redo Log file, that's all. So you might reasonably expect that each time a *new* Archived Redo

Log file is generated, the Recovery Catalogue would have to be told about it –and you'd be right.

You might also reasonably expect some kind of automatic notification to take place, since log switching (which causes new Archives to be written) is a fairly common operation. And you'd be as wrong as it's possible to be!

No automatic resynchronisation takes place after a log switch, so the third rule of RMAN is that ***you need to resynchronise the Catalogue yourself after a reasonable number of Log Switches.***

So what's a 'reasonable number' of log switches? Who can say?

One consideration is that the information the Recovery Catalogue needs is stored temporarily in the Control File. This makes the target's Control File start to grow –and you might want to resynchronise before it grows too big.

At a **minimum** you must resynchronise the Catalogue at intervals *shorter* than whatever is specified for the parameter '**CONTROL_FILE_RECORD_KEEP_TIME**', because otherwise the required information simply drops out of the Control File –which renders the target unrecoverable using RMAN.

If you want a ball-park guestimate for how often you should resynchronise, I'd be suggesting once a day… on the understanding that if your huge database is switching logs very frequently, that might not be often enough.

10.2 Prior Backups

It's not an unreasonable proposition to imagine that you've been merrily backing up your database for a while ***before*** deciding to use RMAN. The only problem would be that RMAN would be totally oblivious to the existence of those prior backups, which renders them slightly less than useless.

So, there must be a way of informing the Catalogue about the existence of those previous backups –and indeed there is. It's all done with the use of the 'catalog' command. For example:

```
CATALOG DATAFILECOPY 'C:\OVERHERE\SYSTEM01.BAK'
```

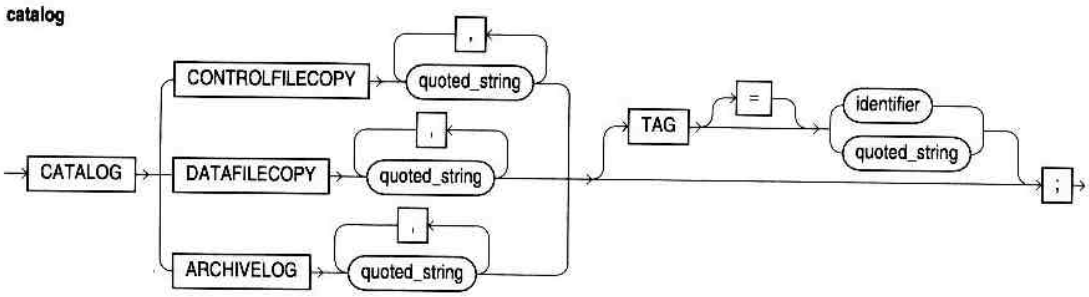
There are a couple of restrictions to note with this command. First, the file you point to has actually got to exist. Secondly, the file you point to must be either a set of Data Files or Control Files, or an Archived Redo Log –but not a combination of the different types.

When you're cataloguing backups of Data Files, then it doesn't matter if they are internally consistent or inconsistent –RMAN can sort all that out for

you (in other words, it doesn't make any difference whether the files were obtained via a hot or a cold backup).

Finally, all the files that you manually add to the Catalogue like this must relate to the ***current database incarnation only.***

The actual syntax for the CATALOG command has a number of variations, which this diagram attempts to explain:



In other words, an acceptable command would be:

```
CATALOG CONTROLFILECOPY 'C:\HERE\CONTROL01.BAK'
```

or

```
CATALOG DATAFILECOPY 'D:\THERE\FULLDATABASE.BAK' TAG='OLDONE';
```

(A tag is a friendly name by which to refer to a backed up file –it saves you constantly having to refer to it by full file and path references. Note you can only use the TAG keyword when dealing with the 'datafilecopy' keyword –that's how these diagrams are supposed to be read.)

Incidentally, this and similar pictures are available in the back of the "Oracle 8 Server Backup and Recovery Guide" user manual, supplied with Oracle 8. They are, frankly, a bloody nightmare to interpret correctly, but it's the best source of syntax information there is. A frightening thought, I agree.

10.3 Manual Changes to the Catalogue

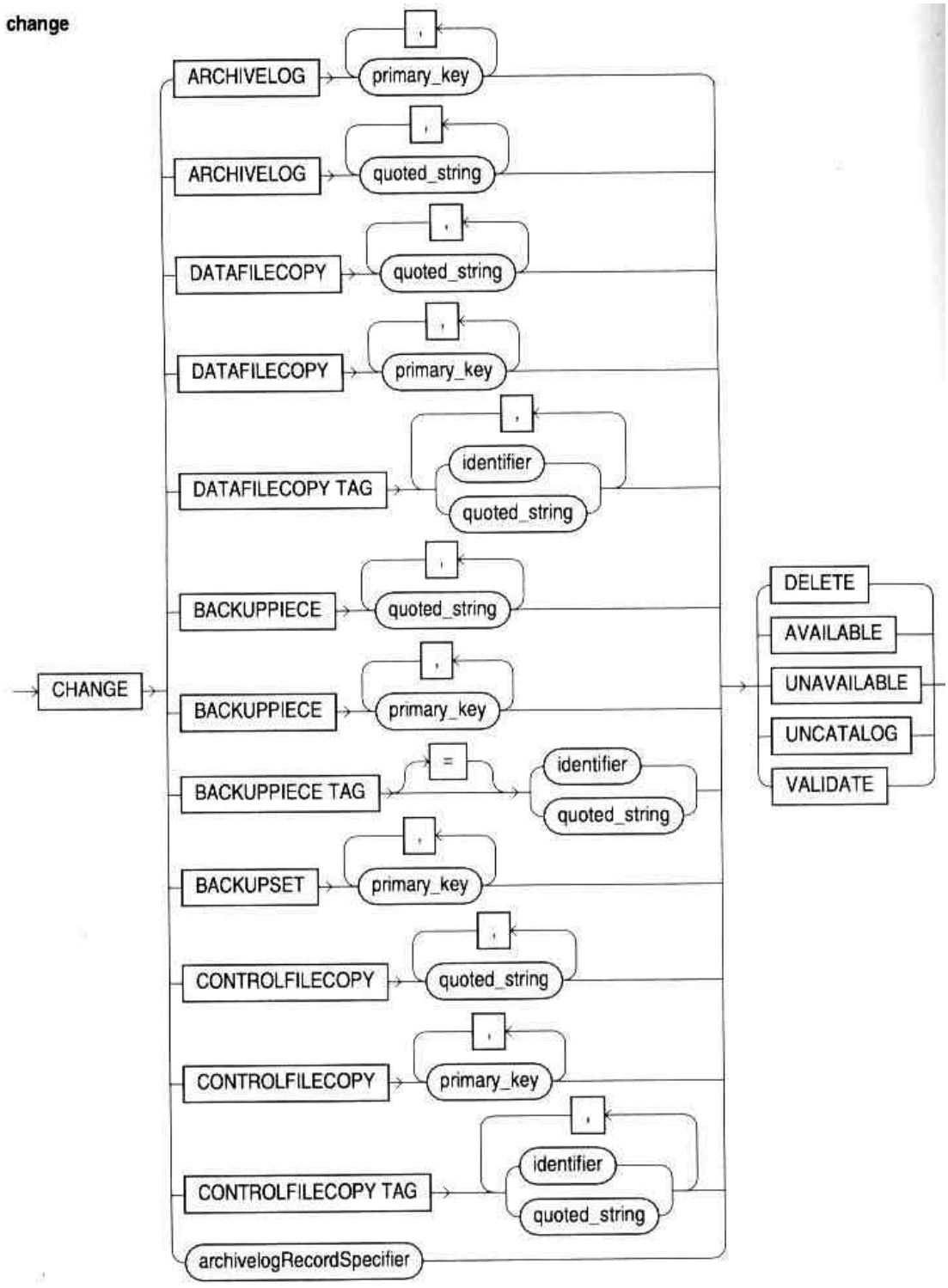
Another reasonable consideration is this: the Catalogue thinks it can rely on the existence of a backup file that you manually told it about, except that your trainee DBA accidentally deleted it.

Clearly, you need to **remove** reference to that file from the Catalogue – otherwise, in an emergency, RMAN will quite probably start asking you to supply it with the relevant file.

Well, you can manually alter the Catalogue contents by using the **CHANGE** command, like this:

CHANGE DATAFILECOPY 'C:\THATONETHERE\SYSTEM01.BAK' UNCATALOG

There are heaps of variations on this syntax, so rather than trying to wade through them all, try and follow this diagram:



You'll note that not only can you 'uncatalog' a file, but you can actually get RMAN to physically delete one off the disk, as well. A file might also go temporarily walkabout, in which case it can be marked as 'unavailable' – and when it comes back, you can mark it as 'available once more'.

Finally, the 'validate' option can be used to check whether the quoted file really is available after all, and if it isn't, to remove the corresponding catalogue entries automatically. For example, if you've cleared a lot of files off a disk, try this piece of syntax:

```
CHANGE DATAFILECOPY ALL VALIDATE
```

(and, yes, I know that the Oracle-produced diagram is mysteriously silent on the use of the ALL keyword, but that's documentation for you. It isn't described, but it *can* be used).

10.4 Reporting

You can use RMAN to generate reports, but you'll need to do heavy-duty battle with the syntax first.

There are four types of report you can produce, each with their own syntax, as follows:

- Report unrecoverable
- Report need backup
- Report obsolete
- Report schema

Each type of report is based on the contents of the Recovery Catalogue – so unless you are using a Recovery Catalogue, you can forget these options.

10.4.1 Report Unrecoverable

This is used to list all Data Files that cannot be recovered, perhaps because you've used the 'unrecoverable' key word back in the target database, or because you've lost Archived Redo Logs.

This is the simplest of the Report options, because the syntax is strictly limited to this:

```
REPORT UNRECOVERABLE
```


10.4.2 Report Need Backup

Cunningly enough, this report tells you which files are most in need of a new backup, assuming that the most recent backup would be used in the event of a need to recover a database. It would be useful, for example, to determine which Data Files would need the application of more than 3 incremental backups to be fully recovered –that would clearly be a Mean Time to Recover issue. The syntax for such a report would run something like this:

```
REPORT NEED BACKUP INCREMENTAL 3 DATABASE
```

Catchy, huh? Try this one: which files in the DATA01 tablespace haven't had a backup of any sort for at least 5 days? Answer:

```
REPORT NEED BACKUP DAYS 5 TABLESPACE DATA01
```

10.4.3 Report Obsolete

If you backup regularly, earlier backups are rapidly going to become obsolete, and can therefore (if you're feeling lucky) be deleted –both physically, and from the Recovery Catalogue.

For example, which Data File copies are no longer required because at least 3 additional copies have subsequently been taken? Answer:

```
REPORT OBSOLETE REDUNDANCY=3
```

If you are in the habit of opening your target database with the **RESETLOGS** option, then you'll have multiple incarnations of your database. If your Catalogue is aware of incarnations 1, 2 and 4 (with 4 being the current one), then backup files associated with incarnations 1 and 2 are technically orphans –they can never be used to recover the database to the current time. So you can report on these sorts of files with this piece of syntax:

```
REPORT OBSOLETE ORPHAN
```

10.4.4 Report Schema

This is a pretty boring option really. It simply reports on what Data Files and tablespaces existed at a specified time (or the current time). Quite why this information would be of especial use to the budding DBA, I have no idea, but here goes anyway:

```
REPORT SCHEMA AT SCN 15678
REPORT SCHEMA AT LOGSEQ 109
REPORT SCHEMA AT '25-MAY-2000'
```


In the first case, we are specifying a particular SCN, in the second a specific Log Sequence number, and in the third, a simple date. You can also simply ask to 'report schema', and you'll get a report of what the database contents are like right now (you could get much the same information by doing selects from dba_tablespaces and v$datafile).

10.5 Listing

The distinction between lists and reports in RMAN is –to be honest- not entirely clear to your instructor. But that there is a different keyword is (unfortunately) beyond dispute.

Suppose you want to list all the known backups of the system Data File. Then you might type something like this:

```
LIST BACKUPSET OF DATAFILE 'C:\ORADATA\SYSTEM01.DBF'
```

Or, suppose you wanted a list of all the image copies (i.e., physical copies of a file taken at the O/S, and then catalogued as described earlier) of the system tablespace:

```
LIST COPY OF TABLESPACE SYSTEM
```

You might use such lists to ensure, for example, that the files that RMAN *thinks* are available are, indeed, still there on disk.

Remember, too, the **LIST INCARNATION OF DATABASE** command that we saw earlier (see 10.1.1).

Obviously the LIST syntax has heaps more variations than this, but this gives you some idea of what commands are available.

10.6 Scripting

The final thing to be aware of about RMAN is that you can use it to store, in the Recovery Catalogue, any number of backup scripts. Scripts are named, and can be run by using the syntax **EXECUTE SCRIPT *name.***

There are four things you can do to scripts:

- Create one
- Replace one
- Delete one and
- Print one

10.6.1 Creating and Replacing Scripts

You create a new script with this sort of syntax:

CREATE SCRIPT NORMALBACKUP {
…*WHOLE BUNCH OF CONVOLUTED SYNTAX*…}

That is, you give your script some meaningful name, open a pair of curly braces, type in heaps of clever stuff, and close the whole thing off with another curly brace.

If you make a typing error when you're typing in the clever stuff (and believe me, you will), then you can correct your error with this sort of syntax:

REPLACE SCRIPT NORMALBACKUP {
…*WHOLE BUNCH OF CORRECTED, CONVOLUTED SYNTAX*…}

10.6.2 Deleting Scripts

Nothing too tricky about this one! Just type:

DELETE SCRIPT *name_of_script*

10.6.3 Printing Scripts

If you forget what's actually in a script, you can 'print' it –which means either display it on the screen, or send it to the RMAN log file, where you can inspect it with a text editor.

Oddly, therefore, the Print Script syntax ***doesn't*** actually result in a nice, hardcopy printout. No surprises there, then.

You can probably guess the syntax: **PRINT SCRIPT *name_of_script***

All of which is fascinating, but what exactly can you put inside a script, and what sort of syntactical rules must a script follow? Good questions both, and answered (sort of) in the next Chapter.

Backup & Recovery: Chapter 11 – Scripting an RMAN Backup

11.1 Types of Backup

RMAN can perform two basic types of backup –either an Image Copy, or a Backup Set.

11.1.1 Image Copies

An **Image Copy** is a copy of a single file, made by RMAN, but resulting in a physical image of the file, almost as if you'd used O/S commands to take a copy. The other big difference between an Image Copy and a normal O/S copy of a file is that RMAN can validate the contents of the file as it copies it –being an Oracle application, it can look 'inside' the file it's copying, and check for block corruption.

Image copies can ONLY be written to disk, never to tape. They are like O/S copies in that RMAN writes ALL data blocks to disk –it doesn't ignore empty ones.

The following script takes a copy of some Data Files:

```
RUN {
ALLOCATE CHANNEL D1 TYPE DISK;
COPY DATAFILE 1 TO 'D:\BACKUPAREA\SYSTEM01.BAK',
DATAFILE 2 TO 'D:\BACKUPAREA\DATA01.BAK';}
```

This piece of syntax introduces us to a couple of basic RMAN rules:

- First, all RMAN commands need to be contained within a pair of curly braces.
- Second, all backup operations (whether Image Copy or Backup Set) need to have allocated to them a **'channel'** –which is a fancy RMAN term for a server process that is going to do the actual copying, deletion or whatever job you are asking it to do.
- Third, every channel must be defined as being either type 'DISK' or a named string relating to a useable TAPE device.
- Commands are terminated with semicolons. Multiple commands can be contained within a set of braces –each must be terminated with a semicolon.

These basic rules ALWAYS apply.

11.1.2 Backup Sets

A **Backup Set** contains one **or more** files, and the files can be either Data Files, or Archived Redo Logs –but never both at the same time (on the grounds that it would be a bit risky to have both the files you are trying to protect, and the protection mechanism, included together on the one tape or disk).

Backup sets are comprised of 1 or more **Backup Pieces,** each piece resulting in a single output file. If you are backing up a huge database, for example, a single backup file might well exceed the O/S-limit for the size of a file (2Gb for many operating systems). You'd therefore do the same backup, but chop it up into many 2Gb *pieces.*

When creating a Backup Set, RMAN **DOES** ignore empty data blocks, provided they have *never* been used. This means the resulting Backup Set ends up being considerably smaller than the actual size of the database involved.

Backup sets may be **full** or **incremental.** A full backup takes copies of all the used blocks of the Data Files requested; an incremental backup only takes copies of those blocks of the requested Data File that have changed since the last backup of any kind. (Note, therefore, that the word 'full' doesn't mean the entire database is being backed up. You can have a 'full' backup of a single Data File). I'll discuss these different types of backups in more detail later on.

The commands to perform a simple backup might look like this:

```
RUN {
ALLOCATE CHANNEL DEV1 TYPE DISK;
BACKUP
(TABLESPACE SYSTEM, DATA, INDEX, RBS
FORMAT 'C:\BACKUPS\BP_%s_%p');}
```

Note the continued use of the braces to encompass all commands; the presence of semicolons to terminate each command within the braces; the allocation of a channel, and the specification of which type of device we're writing to. All the rules we met before are thus still being observed.

Note, too, the file name for the backup pieces: it uses a number of variables, so that each piece is uniquely named. The piece number is given by %p and the set number is given by %s. There are a number of other variables you can use, such as %d to give the database name.

11.2 Creating and Running Backup Scripts

The examples described above are what might be termed 'standalone backup jobs' –you basically have to type them into RMAN in 'interactive mode' –and RMAN immediately begins execution of the requested tasks. Of course, if you wanted to repeat the same tasks on the next evening, you'd have to type the same thing all over again.

This is clearly no way to run a proper backup process! What we need instead is for these sorts of things to be **stored** as scripts that can simply be called, over and over again.

An example of such a script (it happens to back up an entire database, and its Archived Redo Logs) is shown below:

```
CREATE SCRIPT NORMALBACKUP {
ALLOCATE CHANNEL D1 TYPE DISK;
ALLOCATE CHANNEL D2 TYPE DISK;
ALLOCATE CHANNEL D3 TYPE DISK;
BACKUP
        INCREMENTAL LEVEL 0
        TAG NORMALBACKUP
        FILESPERSET 6
        FORMAT '/BACKUP/PROD1/DF_%S_SP.DBF' (DATABASE);
        SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT';
BACKUP
        FILESPERSET 20
        FORMAT '/BACKUP/PROD1/ARCH/ARC_%S_%T.RDO'
        (ARCHIVELOG ALL DELETE INPUT);
}
```

Now, although this looks horrible, there's not an awful lot here that's completely new. We still have the braces, the allocation of channels of a particular type, the termination of commands with the semicolon, and the backup command.

For the purposes of this section, the really important line is that first one: "Create Script" means that what follows is stored in the Recovery Catalogue with the name as supplied, and we can repeatedly run this script by having the following command scheduled to run (as a chron job, or an AT command in NT):

```
RUN {EXECUTE SCRIPT NORMALBACKUP;}
```

However, this script does also have a couple of extra interesting things going on, so I'll go through some of them now, pulling out the lines of particular relevance.

11.2.1 Parallelism

Note the use of THREE 'allocate channel' commands simultaneously. What this means is that multiple server processes are let loose on the target database at once. If you're going to do this, it is then important to have those **FILESPERSET 6** and **FILESPERSET 20** lines included. What these do is to ensure that each channel only processes the specified number of files, leaving extra files for additional channels. If you didn't use this line, then in fact the entire backup would be undertaken by channel 1, and the other two would sit idle.

What this means in the end is that 3 different backup sets (i.e., collections of Data Files) are going to be produced by the three different server processes.

11.2.2 Tags

Tags are a way of naming a backup set instead of having always to refer to it by file and path names. **TAG NORMALBACKUP** means that all the backup sets produced by this script can be referenced by the name 'Normalbackup'. When you come to delete files, for example, from the Recovery Catalogue, this is a handy way to refer to lots of them in one go.

11.2.3 External Commands

The line **SQL 'ALTER SYSTEM ARCHIVE LOG CURRENT'** means that we can get RMAN to issue normal SQL statements –in this case, we are about to backup the Archived Redo Logs, so we want to make sure that the current Online Redo Log is switched from, so that it can be included in the backup.

You can do this for any SQL statement. If you want RMAN to create a table for you (bizarre, but possible), you simply follow the format of preceding the required SQL command with the keyword **SQL:**

```
SQL 'CREATE TABLE TEST TABLESPACE DATA';
```

Just note that the semicolon here is RMAN's –it belongs *outside* the inverted commas, because it is terminating an RMAN command. There is no need for one inside the commas, as there would be if you were typing this yourself in SQL*Plus.

You can also issue normal operating system commands, as though you were at the command line, by preceding them with the keyword **HOST.** For example:

```
HOST 'COPY D:\TEXT.TXT E:\NEWTEXT.TXT';
```


11.2.4 Archive Log backups

The line **ARCHIVELOG ALL DELETE INPUT** is interesting because it is doing two, independent things: the first part is how you tell RMAN to copy Archived Redo Log files. The second part tells RMAN to delete the originals once the backup has taken place.

Instead of saying 'all' Archived Redo Logs should be backed up, you can specify a specific log sequence number range:

```
… ARCHIVELOG FROM LOGSEQ=1098 UNTIL LOGSEQ=2021…
```

Deleting the source Archives is a bit dodgy, if you ask me. However, RMAN does provide some reassurance in this regard: if the backup is not completed, the deletions don't take place. Also, if RMAN encounters any corruption within the Archives, the backup abruptly terminates, so if the Archives *do* get deleted, you can at least be sure they were clean, and that the backup completed successfully.

Finally, you should note the general point that the backup set for the Archived Redo Logs is entirely separate from that for the Data Files. However, the job of creating both types of backup set can be included within the one backup script.

11.3 Backup Pieces

If your backup sets threaten to be huge, you may want to chop them up into smaller pieces. You do this by including the following syntax somewhere in your script:

```
SET LIMIT CHANNEL D1 KBYTES 2048000;
```

Note that this line must come ***after*** the allocation of the channels, and you'd have to specify an equivalent line for each and every channel previously allocated (so if we were doing this for the script included in 11.2 above, we'd have thee of these lines, one each for channels D1, D2 and D3).

Note, too, that the measurement is **always** in Kilobytes. That means the above line refers to 2Gb (last time I checked, 2 million Kilobytes is the same as 2 Gigabytes, but I allow that my maths may be wonky).

So what all this boils down to is that as each of our channels is busy creating its backup set, if it reaches a 2Gb file size, it stops, and writes out what it has done so far. It then resumes, and keeps repeating that process until it has finished its work. We end up with multiple 2Gb *pieces* comprising each backup set.

11.4 Incremental and Cumulative Backups

The very first backup you create of a database should be (as it was in the script at 11.2 above) at **INCREMENTAL LEVEL 0.** This means that all the database blocks (provided they've actually been used at some point) are written out to the backup set(s).

Subsequent backups can then be made at incremental levels lower than 0, and only blocks changed since the last backup at the same, or at a higher level, get written to the backup set(s).

Now, just so we are clear (as the Oracle documentation **isn't**), Level 0 is thought of as the ***highest*** level. Level 2, for example, is ***lower*** than Level 0, and Level 4 is the lowest level that you can specify.

So, now for the mental gymnastics.

If on Monday you perform a LEVEL 0 backup of the entire database, you get ALL the database blocks.

On Tuesday, you perform a LEVEL 2 backup … You get all blocks that have changed since Monday

On Wednesday, you perform a LEVEL 2 backup … You get all blocks that have changed since Tuesday.

On Thursday, madness possesses you, and you perform a LEVEL 1 backup. You get ***all blocks that have changed since MONDAY.*** Why? Because Level 1 is *higher* than Level 2, and incremental backups always contain blocks changed since the same, or higher, backup level –so a Level 1 backup will always backup changes since a previous Level 1 or Level 0 backup. We haven't got any Level 1 backups, so RMAN goes to the previous Level 0 backup –and hence we get all blocks changed since Monday.

One particularly horrible twist to this saga is that ***a FULL backup does NOT affect the behaviour of incremental backups at all.***

For example, you may have thought that if on Friday you were to perform a FULL backup, then the backup at Level 1 on Saturday would contain blocks modified since Friday. Wrong. It will actually contain the blocks changed since Thursday –Friday's full backup is considered an entirely separate affair outside the normal scope of things, and doesn't get considered when working out what blocks to include in a backup set.

In short, whilst a Level 0 incremental and a Full backup are identical physically (they both contain all database blocks), they are completely different from the RMAN logic point of view.

One other point: you may be wondering how you specify that a full backup should take place. The syntax would be: … BACKUP FULL… and since FULL is the default anyway, if you just have the command to 'BACKUP' in your script, you're doing a full backup. In other words, if you want to do incremental backups, you have to say so explicitly with the INCREMENTAL LEVEL keywords.

(Note: This is too painful to discuss at any length, but this doesn't take into account the "CUMULATIVE" keyword, which just makes things even more complicated! A cumulative backup at, say, Level 2 would include all blocks changed since any higher backup –say, a Level 0 or a Level 1. Compare this with a 'normal' Level 2 backup, which would include only those blocks changed since a previous Level 2 backup (if there was one).

Nasty, huh? I suggest if your brain has as much difficulty encompassing all these levels and schedules as mine, study the slides on Pages 11-40 and 11-42 *very* carefully, because they make perfect graphical sense. Just don't then try and explain it to anyone else.)

Backup & Recovery: Chapter 12 – Performing Recovery with RMAN

12.1 Complete Recovery

Complete recoveries can be performed relatively painlessly by RMAN. The precise operations depend on whether the target database is open or closed (i.e., whether the loss of a Data File has resulted in a shutdown – as it would if the SYSTEM tablespace was involved- or not.)

There is really only one golden rule when recovering with RMAN: **the files to be recovered must be offline.**

Now, if the database is closed, you can simply proceed with recovery procedures, because all the Data Files are obviously offline already.

If, however, the database is open, you must ensure that the bit that you want to recover is offline before attempting recovery – in other words, you must yourself ensure the troublesome tablespace (or Data File) is manually taken offline.

Suppose, for example, that one of the Data Files associated with the DATA tablespace has been accidentally deleted, but the rest of the database remains up and running. The following RMAN code would do the job:

```
RUN {
   SQL 'ALTER TABLESPACE DATA OFFLINE IMMEDIATE';
   ALLOCATE CHANNEL CH1 TYPE DISK;
   RESTORE TABLESPACE DATA;
   RECOVER TABLESPACE DATA;
   SQL 'ALTER TABLESPACE DATA ONLINE';
}
```

The key parts of the syntax to note are:

- The use of the RUN keyword, with all subsequent commands contained within a pair of braces
- The termination of each command with a semicolon
- The use of an external SQL command to take tablespace offline and online
- The allocation of channels, of a named type, to perform actual I/O operations

The two lines that really do all the clever stuff are the 'Restore tablespace… ' and 'Recover tablespace… ' commands, of course. The first one causes RMAN to physically retrieve the relevant Data Files from the most appropriate backup set (it works out which backup set can be used to achieve the speediest recovery). The second one applies Archived Redo to the files so that it is brought into synchronisation with the rest of the database.

12.2 Incomplete Recovery

Suppose now that you are told the EMP table has been dropped, and it happened "at about 12.30pm". If you were doing an incomplete recovery manually, one of the first things you would do is to shut the database down and take a backup. Some of this still applies if you're using RMAN: the database MUST be in the MOUNT state, so a shutdown and then a 'startup mount' is the minimum requirement. Because you are using RMAN you can skip the usual backup in between. Because the entire database is only in the mount stage, you can also skip having to offline any Data Files or tablespaces.

The sequence of events then runs, logically, like this:

- Specify the time you wish to recover until
- Allocate channels
- Restore the entire database complement of Data Files
- Recover the database
- Open the database with the RESETLOGS option

The syntax to do all these things might look like this:

```
RUN {
    SET UNTIL TIME MAY 15 2000 12:25:00';
    ALLOCATE CHANNEL CH1 TYPE 'SBT_TAPE';
    RESTORE DATABASE;
    RECOVER DATABASE;
    SQL 'ALTER DATABASE OPEN RESETLOGS';
}
```

Note that the exact format of the time string may vary, depending on your date format environment variables.

Remember that after a 'resetlogs' operation, you MUST reset the Recovery Catalogue, so you would then issue this command:

```
RESET DATABASE;
```


If you were doing this sort of recovery manually, you would then be expected to shut down the database and take a clean backup. The same applies here, so your next command should be:

```
RUN {
    ALLOCATE CHANNEL CH1 TYPE 'SBT_TAPE';
    BACKUP DATABASE;
}
```

You could now open the target database, and carry on as before (taking well-earned vengeance on the person that deleted the EMP table in the first place, of course).

Incidentally, the syntax and the procedures outlined here remain practically the same if you were recovering until a particular SCN or log sequence number (this last one is the equivalent of the 'recover until cancel' command you might use if you were doing the job manually), rather than the 'recover until time' option outlined here. The specific bits of syntax you'd use are:

```
RUN {
    SET UNTIL SCN 1000;
    ALLOCATE CHANNEL CH1 TYPE DISK;
    RESTORE DATABASE;
    RECOVER DATABASE;
    SQL 'ALTER DATABASE OPEN RESETLOGS';
}
```

Or…

```
RUN {
    SET UNTIL LOGSEQ 6;
    ALLOCATE CHANNEL CH1 TYPE 'SBT_TAPE';
    RESTORE DATABASE;
    RECOVER DATABASE;
    SQL 'ALTER DATABASE OPEN RESETLOGS';
}
```

In all cases, the key bit of syntax is the 'SET UNTIL… ' command.

Note: it is possible to perform an incomplete recovery without using a Recovery Catalogue, but it is seriously scary stuff. Frankly, don't go there. Use a Catalogue and make using RMAN worthwhile.

12.3 Restoring to a Different Location

All the above examples, for incomplete and complete recoveries, assume that the files contained in the backup sets can be restored to their original locations. If the source of your trouble is a blown hard disk, this may not be possible. You are likely, therefore, to have to restore a lost Data File to a new location, and then perform recovery.

The syntax for this sort of thing might look a bit like this:

```
RUN {
    ALLOCATE CHANNEL CH1 TYPE DISK;
    SET NEWNAME FOR DATAFILE '/DISK1/DATA01.DBF' TO '/DISK2/DATA01.DBF';
    RESTORE TABLESPACE DATA;
    SWITCH DATAFILE ALL;
    RECOVER TABLESPACE DATA;
}
```

The key things to note here are the 'Set newname' and 'Switch Datafile all' commands.

'Set newname' is hopefully obvious in its implications. RMAN knows the original destination (and name) of the file, and you are over-riding that and forcing it to restore the file to the new location (and, potentially, a new file name).

However, on its own, the command results in RMAN simply thinking that it has retrieved a *copy* of the relevant Data File. What you need to do is to make sure that RMAN believes that copy should actually be considered the *real* version of the Data File. That's precisely what the 'Switch datafile all' command achieves.

12.4 General Considerations

Whenever you are performing recovery with RMAN, remember that RMAN does all the hard stuff, like working out which backup sets or image copies it is going to use. RMAN applies the following rules when sorting all this out for you:

- It prefers image copies to backup sets
- It prefers to use the most recent version of otherwise identical image copies or backup sets
- It prefers files on disk to files on tape, assuming there's a choice

RMAN will prompt for the supply of whatever it decides is the most appropriate source of required Data Files: if the tape it needs, for example, is not already available, it will ask you to insert it into the relevant device.

In each and every case, you have to at least be able to connect to the target database's Instance –so the target must be at least in the NOMOUNT stage. If you can't get the database to NOMOUNT, you can't recover it.

Unless it's your Control File that is causing the problem, though, I'd recommend being in the MOUNT stage every time.

