Database Availability

10

What the reader will learn:

- that we should not expect our databases to be always available
- that the more available a system is, generally the more expensive it is
- that there are several approaches to making a system as available as is appropriate to suit the business need
- that Cloud computing changes the way we think about availability
- In some circumstances the expense of a Disaster Recovery strategy is irrelevant as client expectation will drive the decisions

10.1 What Do We Mean by Availability?

As with most chapters in this book we will start by defining what we mean by availability. And, as with some other chapters there are several possible answers.

For an online business, availability is measured mostly by uptime. Because every second the database is unavailable (or "down") to potential customers could cost thousands of dollars in lost income, having the database usable for the magic five nines (99.999 % up-time) is a sensible, if difficult to achieve, target. This is around 5 minutes downtime in a year.

As we shall see, the move from three nines (99.9 %) to five nines can be very expensive. Some organisations will not need that level of availability. Three nines is under nine hours per year. If you need to spend an extra $100000 to get to five nines, those will be a very costly few hours of computing you have bought!

To be more accurate, database up time is not the same as availability. It is possible for a database to be open, but for network problems to prevent access to the database.

But there is more to availability than merely keeping a database open and accessible. What happens if your data centre is struck by a meteor? Or flood? If you are a bank, for example, your customers will probably expect you to have an alternative server somewhere else that can kick in to (almost) seamlessly replace the destroyed one. Disaster recovery (DR) is part of the availability mix. A DR strategy like that of the bank would be very, very expensive and there would really need to be a good

P. Lake, P. Crowther, *Concise Guide to Databases*,
Undergraduate Topics in Computer Science, DOI 10.1007/978-1-4471-5601-7_10,

business case for having an expensive server (or servers) sitting doing nothing "just in case".

And the other important element to availability is recovering from smaller problems cleanly. If a user mistakenly deletes some data and wants to retrieve it, the DBA will have a number of tools available to address the problem. This could even be something the user can effect themselves, like Oracle's Flashback technology. Part of the standard toolkit here is Backup and Recovery. Taking a full copy of the database once every now and then (full backup), and then recording the changes made on a regular basis is one very common strategy that we will examine in this chapter.

In a traditional Client/Server environment a fail-over strategy may also include the use of redundant disks (RAID). Again the downside is the expense is the extra disk and processing required to make the system more robust, and a potential effect on performance.

Cloud has allowed us to change the way we think about availability. Disk Storage as a Service is now commonplace, meaning that you do not need to worry about maintaining an array of expensive disks, but rather you rent the space you need, when you need it. But what Cloud changes is more to do with costs and flexibility than the overall principles involved ensuring optimum availability.

So, what is our definition of Database Availability? How about this:

> "Ensuring that the database is open and accessible for as long as possible, by planning for as many eventualities as is reasonable, given the constraint of cost."

In this chapter we will explore three key areas in database availability: Keeping the system running—immediate solutions to short term problems; back-up and recovery; Disaster Recovery (DR).

10.2 Keeping the System Running—Immediate Solutions to Short Term Problems

High Availability (HA) is not a new concept and has been part of every leading database vendor's sales pitches for many years. But before we look at some approaches to ensuring high availability, we need to explore some key concepts.

As we discussed in Chap. 3, databases are primarily about making data permanent. From the earliest days of relational databases that has meant storing ons and offs on some form of magnetic medium, such as a floppy, or hard disk.

Although, in the author's experience, it is true that computers have become generally more reliable over the decades, it is also true that they are far from perfect. Disks can fail, as can CPUs, motherboard elements and a whole host of communications interfaces. Availability is about recognising the less than perfect environment and putting in place processes for coping with such disturbances with minimum effect on the users.

10.2.1 What Can Go Wrong?

There is an apocryphal story that does the rounds with seasoned database professionals. It happens that the DBAs would regularly come in on a Monday morning and find that the database had been down for about half an hour at 4pm on Sunday afternoon. None of them came in to work on a Sunday, and there was no remote access. The server brought itself up, but nonetheless, this was a worrying event.

One week the lead DBA decided to go in and watch the server for himself, and couldn't believe his eyes when at 4pm the server room door was opened by a cleaner who, after smiling and waving to our DBA, proceeded to unplug the server and plug in the vacuum cleaner.

What this story highlights is that one of the major points of failure in any database system is the humans who interact with it in any way. These events may never have happened, but the database professional should attempt to think about all possible occurrences that might in any way affect the database and work out ways of preventing the event, or at least limiting the damage caused by it.

The other significant problem caused by humans is in the writing of the code which makes up the database management system, and any database systems using that database. Any database management system is itself a piece of software that has been written by software engineers. A product like Oracle will have had many software engineers produce many thousands of lines of code to bring us the product we now use. And they still beaver away, generating new releases with improvements (or corrections) to previous versions.

10.2.1.1 Database Software Related Problems

The problem is, humans make mistakes. And software engineers are no different, regardless of the quality assurance mechanisms that are put in place. When products evolve, as Oracle has from Version 6 in 1988 to its current Version 12c, there is potential for mistakes to be built on mistakes. This isn't to say Oracle is an unsafe product—far from it, Oracle's reputation is very strong in the area of reliability. However, if you count the patches that are released every year you can't help but notice that somebody, somewhere, gets things wrong!

Often such problems are caused by software engineers making assumptions about customer's operating environments and data distributions, and so it isn't necessarily an error that is being corrected so much as an omission. But this nicety of definition doesn't help the DBA affected by it.

So potential fault number one is a problem introduced by a change to database software release, or its configuration. As a DBA looking to keep the database available at all times, how can we mitigate the effect of these potential problems?

Most of the approaches we cover in this chapter start with the age-old dilemma: how much are we willing to spend to make the system more robust? One solution to the dangers of errors introduced by upgrades is to run two parallel systems for a period, with the upgraded version running besides the older version. Should anything bad happen the new version can be switched off and the old version can continue.

This approach sounds like a sensible one. But then the question is, how long do we run in parallel? A week; a month; a year? Much will depend upon the type of system this is (OLTP or Decision Support, for example) and the volatility of the data. It will also depend upon how critical not losing the database is to the organisation. Whatever the decision, an accountant may see this as a very expensive safety net.

If you are a bank, however, you will probably be willing to pay very considerable amounts to ensure that there is no outage as a result of an upgrade or change to a system. Your customers would, quite rightly, be very angry if they could not access their cash whenever they wanted to. In these circumstances the servers involved would probably already be mirrored (see later in the chapter about mirroring) with DR stand-by servers involved. These will all probably need to be upgraded at the same time too, and will need protecting as they are upgraded. This makes it a very complicated and expensive process with the risk of significant financial damage if it were to go wrong.

10.2.1.2 Systems Design Problems and User Mistakes

RDBMSs are at the heart of many applications and e-commerce systems. Vendors design their applications to interact with the database in many different ways and it is possible for there to be mistakes in the coding. Whilst we should not get involved in the wider discussion of systems testing, DBAs may need to be able to change the values stored because a badly designed system element has caused some values to be mistakenly altered. It might even be as simple as users misunderstanding the input screens and typing wrong numbers into a dialogue box.

DBAs themselves are not infallible. There may be issues like there not being enough disk storage space available to write some inserts, or inappropriate settings on the UNDO which cause processes to fail due to the "snapshot too old" error.

Whatever the cause, it is not unusual for the DBA to be asked to return the row or rows affected to the way they were at a point in time, or a point in a transaction process to allow the correct values or operations to be applied.

This process of reverting to previous values was made easier in Oracle with the development of flashback technology. This was first released in Version 9i and has been built on in subsequent versions. Oracle are not forthcoming about how this technology works, but it is evident that the use of UNDO data, stored in the UNDO TABLESPACE is important. Undo was around before 9i. It was sometime referred to as Rollback segments as it serves as a place to store the values before any change is made so that, should the ROLLBACK command be issued, changes can be undone. It is also used to allow read consistency (see Chap. 4).

Let us take an example of a user who has just accidentally deleted a customer, ID201213, from the CUSTOMER table. They contact the DBA and they can retrieve the missing row by issuing the following command:

```
INSERT INTO CUSTOMER
   (SELECT * FROM CUSTOMER AS OF TIMESTAMP
      TO_TIMESTAMP('2013-04-04 09:30:00', 'YYYY-MM-DD HH:MI:SS')
      WHERE cust_id = 'ID201213');
```

The subquery (within the brackets) retrieves the row for ID201213 as it was at a point of time, as described by the timestamp. This row is then inserted to the CUSTOMER table as it is now.

Even careless dropping of tables can be undone using flashback. Here we retrieve a table called PERSONS that has been accidentally dropped:

```
FLASHBACK TABLE persons TO BEFORE DROP;
```

Perhaps the most powerful is the FLASHBACK DATABASE command which allows you to revert the database to the way it was at a particular point in time, or at a particular System Change Number (SCN). Naturally the database has to be down for this to happen, and you will lose all the data between the point in time and shutting the database down. Replaying data from the Redo logs or archive files would allow some recovery.

10.2.1.3 Technology Failures

As well as human-initiated problems, databases exist in a world of fallible technology. The Hard disks (HDD) that many database management systems use to provide permanency to their data are, at their core, very old technology. They record their data by magnetising a very thin film of ferromagnetic material and this can become corrupted, losing or altering the data as it does. That would result in the partial loss of some data. The mechanisms which allow the disk to be spun and the disk head reader moved to the appropriate place are also vulnerable to failure, and any such failure would result in the loss of all the disk content.

The measure used to describe disk reliability is Mean Time Between Failure (MTBF). The fact that there is even a measurement for this should worry us! Some HDD manufacturers claim 300000 hours between failures. But since this is a mean, many devices could fail well before that time has elapsed. The point is they DO fail, however occasionally. Even the most expensive disks fail; you pay the extra because they fail less frequently. And if they fail in the middle of your company payroll run, you can bet you will have many unhappy employees!

The HDD is the weakest link in a database management system. As Sam Alapati (2009) says in the *Expert Oracle Database 11g Administration*:

> The most common errors on a day-to-day basis are hardware related. Disks or the controllers that manage them fail on a regular basis. The larger the database (and the larger the number of disk drives), the greater the likelihood that on any given day a service person is fixing or replacing a faulty part. Because it is well known that the entire disk system is a vulnerable point, you must provide redundancy in your database. Either a mirrored system or a RAID 5 disk system, or a combination of both, will give you the redundancy you need.

We will review RAID later in this chapter.

If you refer back to Chap. 3 you will see that Oracle keeps its data in Tablespaces which are themselves supported by one or more Operating System files. If the area of disk which stores the data gets corrupted, the DBA will need to recover that file, potentially taking the tablespace offline (making it unusable) whilst the copy is restored from a back-up. Naturally, whilst recovery can be relatively trivial in terms of tasks to undertake, the loss of a part of the database can have expensive business repercussions.

Once the backed up replacement file is in place the DBA will need to "replay" the changes made between the time of back-up and the failure. They do that using the Redo Log files. More on this process follows in the Recovery section of this chapter. But the fact that Redo is required for any recovery, and that Redo is also stored on vulnerable HDDs means that we have to be extra careful with Redo files. Critical elements of the database architecture, such as Redo and Control Files, are stored multiple times across multiple disks to ensure that recovery for a disk failure can be both rapid and, where possible, automatic.

Redo Logs which consist of two or more files that store all the changes made to the database as they occur, are so critical to the recovery task that they are multiplexed. This means at least two identical copies of the redo log are automatically maintained, but on separate locations, ideally on separate disks.

10.2.1.4 CPU, RAM and Motherboards

Other parts of the server architecture are also vulnerable to failure. CPUs are made up from many transistors and failure of one can cause the failure of the processor itself. The failure can be triggered by excesses in temperature, for example.

If your database is running in a single server, as it will be for most smaller scale applications, then failure of CPU or motherboard can mean your database is down. As we keep saying, that outage can be expensive to the company.

Larger systems may well consist of more than one physical server. Oracle Version 10 gained a "g" suffix, becoming Oracle 10g. The "g" stands for grid. In essence grid, in Oracle terms, is about using multiple, relative cheap "commodity servers", all connected together working on the same database to meet an organisation's performance and availability requirements. Oracle have two types of grid, as explained in their online references:

- A Database Server Grid is a collection of commodity servers connected to run one or more databases.
- A Database Storage Grid is a collection of low-cost modular storage arrays combined together and accessed by the computers in the Database Server Grid.

They go on to say:

The same grid computing concepts can be used to create a standby database hub that provides data protection, minimizes planned downtime, and provides ideal test systems for quality assurance testing and all for multiple primary databases

Having multiple servers and redundant disks available was, and still can be, an ideal way of managing availability. Some forms of Cloud computing offer this sort of advantage without having to pay for and maintain multiple servers yourself, thus replacing the high capital expenditure costs involved with hardware purchase with a longer term spread of revenue costs as processing is "rented" from cloud providers.

Storage as a Service is fast becoming a popular offering by providers. As this allows an organisation to have redundant data spread across the world the service offers potentially more security than having servers in one geographical location.

10.2.1.5 Using RAID to Improve Availability

Some organisations will not be looking to the Cloud for some time to come. Those that worry about security, or which have recently invested heavily in internal infrastructure, will still want to ensure their databases are as available as possible.

A Redundant Array of Independent Disks (RAID) is one way of enabling higher availability. Most external storage devices provide support for RAID. There are several types of RAID types, known as *levels*. RAID can either improve performance or provide higher availability, dependant upon the level used. As with many aspects of database management there are trade-offs between the different desired outcomes. Replicating data to more than one disk will add redundancy and thus improve availability, but, even if the writes are made in parallel, there will be an overhead involved in ensuring the success of the write that will slow the write process down.

Here we are concentrating on availability, and the RAID levels most often used are RAID 1, also known as Disk Mirroring, and RAID 5. RAID 1 is the simplest form of high availability implementation, and needs two disks. If the pair of disks are available for parallel reading, queries can perform more quickly than against a single disk. However, since it needs double the amount of disk space to store the data it is not the most efficient mechanism.

RAID 5 is a popular RAID level for enterprise systems since it offers better performance than RAID 1 as well as high availability, and although it needs the use of at least 3 disks, it is potentially less space hungry since not all data is replicated. It manages this by striping both data and parity information across the three (or more) disks. (See Chap. 3 for a discussion of Striping and Parity.)

Parity arrays work well because they are built on the assumption that a HDD will rarely fail entirely, but may on occasion fail in a sector of the disk. A recovery of that lost sector is made possible by using the parity value, with the data you do have, applying an *exclusive or* (XOR) and regenerating the missing data.

If you have a system with n disks, you will need $n \times 2$ disks for RAID 1, and $n + 1$ for RAID 5. In addition to using less disk resource, RAID 5 can be better performing than RAID 1 since your data is striped across more disks and therefore,

with parallel reading enabled, potentially able to return data faster. The downside however is that writes will be slower than RAID 1 since the parity calculation will need to occur for all data being written.

As usual there is no simple answer for the DBA. They need to know the systems they are supporting well to be able to select the most appropriate availability mechanisms. Write-heavy OLTP systems, for example, might find RAID 5 too slow, whereas it may be very suitable for a non-volatile Decision Support System.

10.2.1.6 Recovery at Startup

Failures to write data can happen at anytime. They can even be as a result of a DBA's actions if they issue the SHUTDOWN ABORT command, for example. This might be the only way they can get the database to close, but it has to be a last resort since any live SQL statements are terminated immediately.

If the database has terminated uncleanly, either because of an ABORT, or some system generated fault, Oracle's STARTUP command will try and carry out an automatic recovery. The process which looks after this is called SMON (System Monitor) and this is a core process, which will cause the database to terminate if it were to fail.

The DBA may be lucky and the system may be able to get back to the last consistent state, or they may have to recover files manually before opening the database for wider access. They would do the latter by issuing the STARTUP NOMOUNT command. This starts the database and allocates memory structures, but does not connect to the disks, allowing changes to be made at the operating system level. Once the recovery has happened the DBA can make the database attached to disks and available to all users by issuing the ALTER DATABASE OPEN command;

10.2.1.7 Proactive Management

You could, of course, greatly reduce the risk of disk failure on your system by simply having a rolling replacement program that replaces your HDD resources every three months, or perhaps six. Using the MTBF measure, at least statistically, you will have a more robust system. However, against that possible improvement in availability you have to recognise the actual cost of excess HDD purchasing.

Silly though the above example may seem, if your data is core to your business's success, you may well be willing to pay good money to reduce the risks of failure. System maintenance should indeed be an important aspect of keeping a system available. Replacement of servers and disks should be part of the strategy, with replacements being timed to cause least disruption.

10.2.1.8 Using Enterprise Manager

Most of the main RDBMS vendors have some form of management console. Oracle's is called Enterprise Manager. It has a dashboard home screen on which appears warnings of various types, some of which the DBAs will have set up for themselves. You can configure it, for example, to warn you when a particular tablespace is getting to 75 % full. That may give you time to go and buy some more disks to allow for more expansion later. You can also have warnings emailed to the DBA team to ensure that they are not overlooked.

Since the database will, at the very least, reject inserted rows if the file supporting the tablespace is full, and at worst, abort, taking a proactive approach to disk usage is very much a key part of availability management.

10.3 Back-up and Recovery

As we have seen, we can't rely on the database platform being error free. To safeguard against unexpected data loss we need to take copies of the data known as Back-ups. If you should lose any data, then you use the copies stored as a backup to replace the lost data.

Before we begin looking at backing up, we should just remind ourselves that the logical data we see in our RDBMS is made permanent using physical media, such as HDDs. There are, therefore, two types of back-up: Physical, which is the primary concern of this chapter and which will be dealing with operating system level files, and Logical which is more often to do with taking temporary copies of data, often of tables, as a precaution when applying changes to the original data.

Dealing with Logical first, let us follow an example. The user wishes to apply a 5 % payrise to all employees with a Grade of 'B'. Because things can go wrong our cautious user wants to take a copy of the Employee table before applying the changes. They are, in effect, backing up the table. This code might be what they use:

```
Create Table oldEmployee as select * from Employee;
    .... do the processing
    .... When happy that you no longer need the Table back-up:
    Drop Table oldEmplyee;
```

As we have seen with the Flashback command, there is more than one way of getting to the way the data was at a point in time, but this could be one solution.

All the logical application data in the database, and all of the data dictionary data, will actually be stored in operating system files. These files are precious and regular copying will allow the database to be restored in the event of corruption to the files used by the database. In Oracle these files can be either copied (backed up) directly from the operating system, known as User-Managed back-up, or through an Oracle process called RMAN which helps with the management of the back-up process by collecting meta-data about what is being copied.

Some important back-up concepts and terms used need exploring first:

1. **Consistent Back-up**

   A consistent backup is one where datafiles and control files are in agreement with the system change number (SCN). The only way to ensure a consistent Back-up is to issue a SHUTDOWN command other than SHUTDOWN ABORT, and then take the back-up. An open database may well have important data in memory rather than written to disk, so all files have to be closed. This

of course means the database is unusable whilst it in down and the back-ups are being taken. This is also referred to as a cold back-up.

2. **Hot Back-up**

   Shutting down the database is clearly not a good thing to do if users are being refused access as a result. The irony is that you are, albeit temporarily, making the database unavailable to help with future availability.

   Hot back-ups are taken when the database is still open. If it is supporting a 24/7 operation you may have very little choice but to use online backups. However, the risk is that a datafile is being written to whilst the back-up is being taken and this will result in an inconsistent back-up. This means that using a Hot Back-up to restore from always runs the risk of needing to apply redo data to remove the inconsistency.

   If you are in ARCHIVELOGS mode (that is, storing the redo logs so they do not get overwritten) then Oracle can help manage the problem by placing a tablespace being hot-backed up in a special status whilst the data is being copied, in which no users can write data to the tablespace. Assuming the copy is quite rapid users will see this as little more than a glitch in system performance.

   However, there is always a slight risk with hot backups that you do not account for changes being made at the moment of back-up. These risks can be limited, and so many DBAs will take the risk rather than bring the whole database down to enable a consistent cold back-up. Backing up tablespaces one at a time, rather than backing up all datafiles at once also spreads the overhead of back-up processing, often allowing the user to see no adverse affect from the back-up process.

3. **Whole Database Back-up or Partial Back-up and Incremental Back-ups**

   If we take a backup of the whole database whilst it is down we will have a consistent and immediately usable copy of the data. If the back-up finishes and then one second later, once the database is restarted, it collapses, the DBA will simply need to restore the back-up they have just taken.

   Unfortunately that sort of luck rarely happens! Full back-ups can take hours, depending upon how much data is being stored. And unless you are fortunate enough to work in an environment which doesn't mind the database being down for a day, full back-ups are not going to be a popular move.

   So, one option is that you take a full back-up on one day, and then every day thereafter you store only the changes that have been made to the data. These are called incremental back-ups and they save only data blocks that have changed since the previous back-up.

   The good thing about incremental is they are so much faster to store the back-up. The bad thing is that, should you need to recover the database, it will be so much slower since you will need to restore the last full back-up and then apply the changes as stored in the incremental back-ups.

The trade off between time to back-up and time to restore is again typical of the DBAs balancing act. For different organisations, different back-up strategies will be appropriate. In terms of the lifespan of the backed up data, the final decision is to have a retention policy which specifies which backups need to be retained to meet

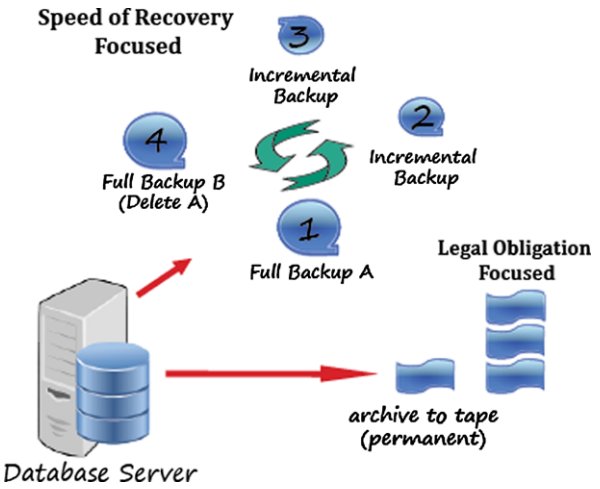

Fig. 10.1 Generalised view of back-up process

the organisation's recovery requirements. Probably the minimum would be the last full back-up plus subsequent incremental ones, but there may be legal obligations on the company to keep data much longer than this would enable.

A generalised view of the back-up process can be seen in Fig. 10.1.

10.3.1 MTBF and MTTR

Two key metrics when deciding on availability strategies are:

- Mean Time Between Failures (MTBF) which is a measurement of how long elapses between failure. This will be influenced by both the hardware in use (which may fail) and the software systems and human interaction with it and so will be different for every organisation. Historic failure data is often used to estimate this by dividing a given period (say a month) by the number of times the system was unavailable.
- Mean Time To Recover (MTTR) which is a measure of how quickly your database will be available in the event of a failure.

Oracle allows you to set a target time for MTTR and will automatically calculate what it needs to do to ensure that this happens. The FAST_START_MTTR_TARGET initialization parameter controls the duration of recovery and startup after instance failure.

However, as always, there is a trade off. Oracle speeds up the MTTR by forcing the system to checkpoint more often. This ensures that the consistent data (which can therefore be used immediately) is as big a proportion of the total data as possible, and that there is very little Redo data (with SCN greater than that at the checkpoint) to apply to bring the database back to its position at failure. The increased checkpoint activity is, however, detrimental to performance as it forces more write activity, and if your system is an active OLTP system this could be quite a problem.

10.3.1.1 Recovery

The other half of Back-up and Recovery is just as important as taking the back-ups. A DBA needs to know where the most current back-ups are and how to apply them. Typically this will mean the use of tape storage devices and the retrieval of tape cartridges from a fire-proof safe. The cloud now offers a cleaner alternative, with the back-up being retrieved from a Storage-as-a-Service provider.

Two key terms are used in bringing a database back to as close to its state at failure as possible;

- Restore: Copying datafiles from a back-up to the server
- Recovery: "replaying" the changes since the last back-up by applying the redo logs.

The first, and most critical part of any recovery is the notification of failure. Modern dashboard systems like Oracle's Enterprise Manager will automatically notify the DBA, sometimes by sending emails, when there is some error or warning from the database. If the DBA is out on a picnic in an area with no telecoms available, the recovery may take longer than it could, so having policies in place to ensure DBA access between team members is an important starting point!

Assuming the DBA gains actual, or remote, access to the database terminal the next task is to establish the type of problem. SQL errors may require no intervention other than talking to the user responsible. An error with one of the key processes may require the database to be "bumped"—shutdown and restarted.

But in terms of restoring data it is more likely that the DBA will be responding to a disk failure or similar media related problem. If Oracle tries unsuccessfully to read a data file on a disk it will flag the problem and the DBA can restore from a back-up and replay any redo information to bring the file up-to-date. The database will continue to run and be accessible to any processes not accessing that file.

If Oracle can't write to a data file it takes it off line It is the datafile that is taken offline—the tablespace remains online and can be used. If Oracle can't write to the SYSTEM Tablespace (which contains the data dictionary) however, the database will shut down automatically and a restore will be required from the last available back-up.

10.3.1.2 RMAN and Enterprise Manager

As usual we will use an Oracle database to provide examples of the tasks that make up back-up and recovery. And again, as usual, Oracle provides two key approaches: the GUI Enterprise Manager with built-in wizards; and the command line utility called RMAN (**R**ecovery **Man**ager). You will need special privileges to issue these commands, either granted to a specially created Backup Manager user, or using SYS or SYSTEM DBA privileges.

Newcomers may wonder why DBAs don't just use the far simpler interface. But the fact is that many Oracle systems have grown from earlier versions of Oracle, and many DBAs have years of experience of writing scripts to do the important maintenance tasks in the database, and so the command line tool remains very popular.

Let us start by comparing the approaches to resetting the MTTR. The Enterprise Manager approach is shown below, and is followed by the command line equivalent (Fig. 10.2).

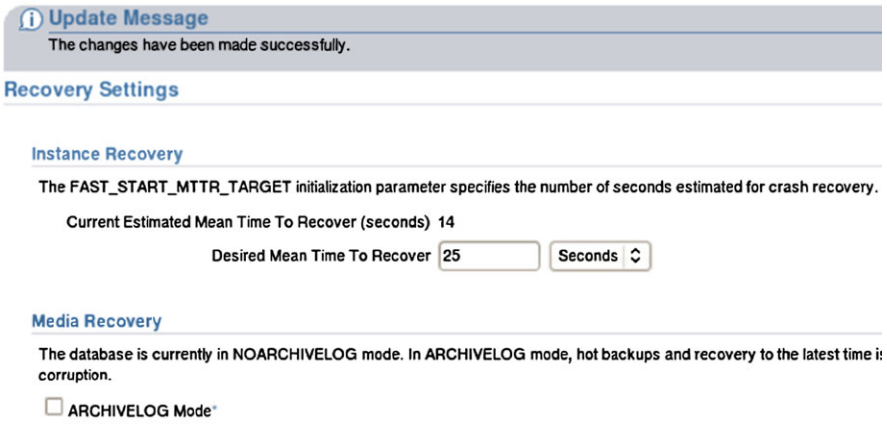

Fig. 10.2 MTTR setting in Enterprise Manager

At the SQL prompt the equivalent is:

```
SQL> ALTER SYSTEM SET fast_start_mttr_target = 25 SCOPE=BOTH
```

The Availability management page of Enterprise manager (11g) is shown in Fig. 10.3, showing the sorts of tasks that can be controlled.

And these are the commands required to do a full backup of a database called ***orcl*** using RMAN (Fig. 10.4).

10.3.1.3 Storing the Back-Ups

Whatever your back-up strategy you will need to decide on which medium you will select to store the data. As usual there are good and bad points about most of the alternatives.

Solid State Drives (SSD) are becoming relatively cheap and can store large volumes of data and because back-ups write few times, it could be argued this makes SSD a good medium for back-ups. HDD are also dropping significantly in price/Gb and are therefore a reasonable suggestion for storing back-up.

Both of these media would be far quicker at the restore stage than would the more traditional medium of magnetic tape, but, even at today's prices, they would also be more expensive. As tape is a sequential read medium access time to any specific section of a back-up is likely to be far slower to the HDD which allows random access.

Tape drives are relatively simple and not over-used and so they tend to last a long time. Many organisations who invested in databases more than a few years ago will therefore already have invested in a (tape oriented) back-up system which they will not need to replace for a long time. The cost of the alternatives, therefore, is only likely to be borne should there be a good business case for speeding recovery times.

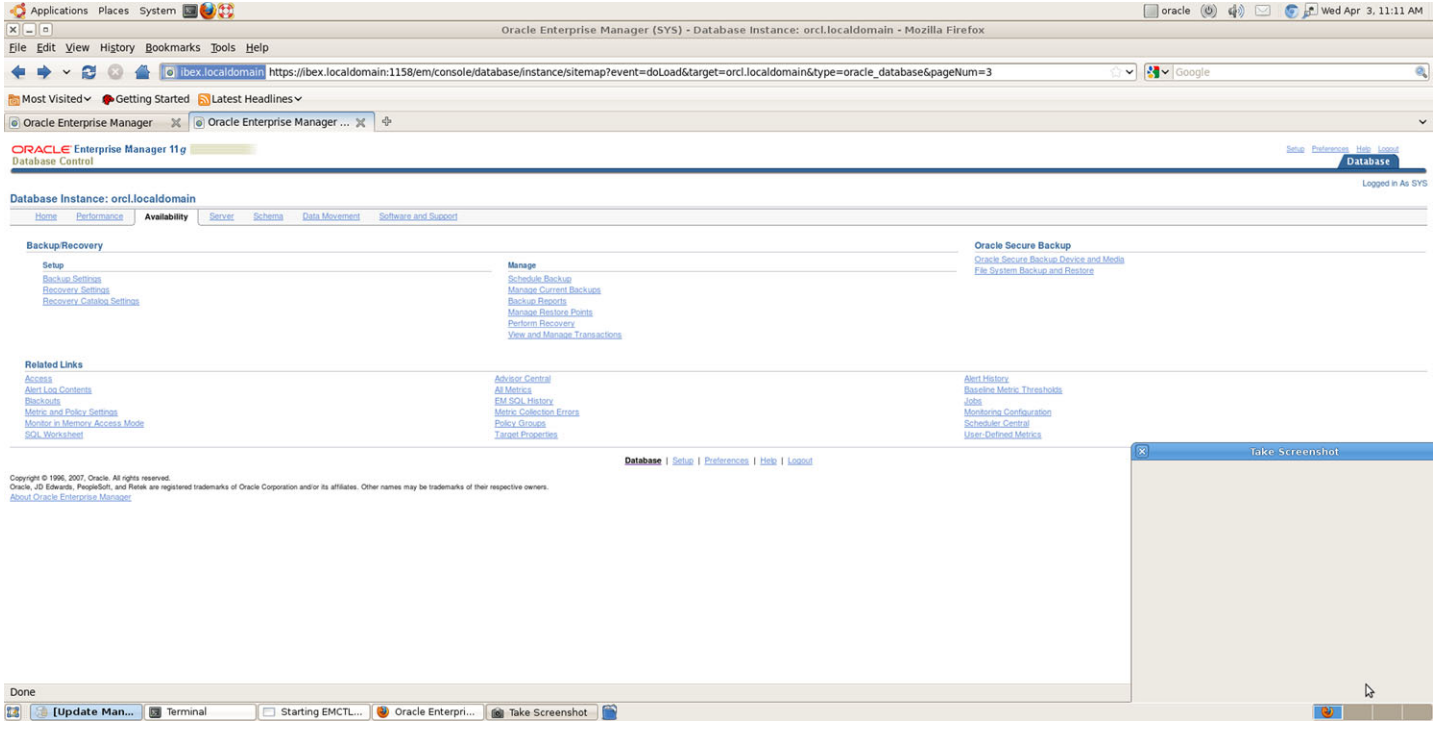

Fig. 10.3 Using Enterprise Manager to control Back-up tasks

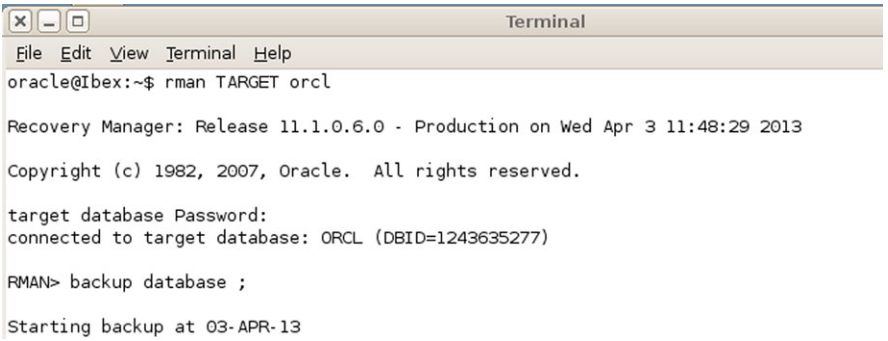

Fig. 10.4 Using RMAN from the command prompt

In the last couple of years Storage as a Service has meant that HDD storage has become more popular by default as Cloud users access disk farms made available by the service providers. As the service is rented there is no up-front capital investment needed to provide back-up facilities, but the revenue cost will eventually accumulate over time and may well overtake any capital cost that might have been incurred instead.

10.4 Disaster Recovery (DR)

Whilst Back-ups ensure the longevity of an organisation's data, they are not really a means of keeping a database readily available. Should a major event, such as an earthquake or tsunami destroy your company's data-centre, the presence of a set of back-up tapes stored off-site will allow you to get your database back—but only eventually, and only to the point of the last back-up!

Finding and procuring replacement IT infrastructure will take time, maybe even weeks. And then the process of rebuilding the database from the back-ups could be lengthy too, especially on a large database which has been backed-up using incremental back-ups. The back-up medium can also slow things down as tape is not the fastest (but is among cheapest) storage method.

In our often quoted example of a bank, having your system unavailable for weeks is going to do you a lot of damage—maybe even enough to put you out of business. When the disaster happens you need to be able to, as seamlessly as possible, get the system up and working again.

An example of the sort of damage that happens to businesses the Bank of New York (BONY) immediately post 9/11. The oldest bank in the US had a DR strategy but found that because most operations were heavily concentrated around Manhattan, and because of the scale of the disaster, they were unable to keep in reliable contact with customers.

As the Wall Street and Technology online journal (2001) said (Guerra 2001):

> But the temporary loss of an entire data center and a damaged telecommunications network left BONY struggling to connect with its customers and industry utilities like the DTC.
>
> BONY's difficulties during the hours after the attack gain gravity when its vital role in world of securities processing is understood, making the fact that it could regroup from the disaster—and do it quickly—of intense interest to more than just the firm's shareholders.

Demonstrating how short-lived can be feelings of sympathy where money is concerned, the article goes on to say:

> Starting just days after the attack, BONY began to be pilloried in the press for systems malfunctions and other technological failures which had purportedly left billions of dollars in securities transactions unsettled. Articles cited angry customers who were misled about the status of BONY systems, demanding the facts on unsettled orders.

And yet it is clear that the Bank did have a disaster recovery strategy and that it worked. However, it did not work well enough to meet expectations of some customers and commentators.

10.4.1 Business Impact

It is assumed that some level of difficulty would follow the loss of an organisation's database. After all, why incur the expense of having one if it isn't that vital? However, it may be that your business will not suffer unduly if you lose access to your database for a few days. In other words, not everyone needs to spend lots of money on HA.

So one of the first steps required when considering your DR strategy is to measure the potential impact of the loss of your database systems. This will involve tangible (for example, lost production) and intangible costs (for example, loss of goodwill) and discovering just which records are vital and time-critical. As usual, the more broadly robust you make your systems, the more expense you will incur, so you will need to balance the potential losses against real costs.

One key variable in the size of the costs is the time you are willing to be without your database. This is referred to as the Recovery Time Objective (RTO). In general, the shorter the RTO the more expensive the solution. So the decisions about DR tend to be business driven, not technology led, although there are alternative approaches available as we shall see in the next section.

10.4.2 High Availability for Critical Systems

We have already discussed a couple of key methods of maintaining availability. In some circumstances—let's stick with banks as a good example—Availability is of critical importance to the business and expense is much less of a barrier.

These are the circumstances in which the term High Availability (HA) tends to get used. There are several approaches the database architect can take:

- Stand-by databases which are copies of the Master database and which kick-in in the event of failure of the Master
- Cloud-based stand-by services
- Clustering the database, using several servers to provide some redundancy. Different vendors use different approaches to clustering:
  - Shared Disk (for example, Oracle) where separate nodes each run an instance of Oracle but share the same database (that is data, redo logs, control files). The database must sit on a Storage Area Network (SAN), which provides the capability to connect to many servers, usually through high-speed fibre connections. Having multiple servers allows for automatic fail-over to happen and connections seem to persist, even if, in the background, another instance has taken over. Disk redundancy is managed by the SAN.
  - Shared nothing (for example MySQL) where multiple servers and their disks save copies of the same database. Data within each node is copied to at least one other data node. If a node fails, the same information is always to be found in at least one other server.
- Distributed databases, providing the copies of the same database on multiple sites, allowing continuous operations even if a catastrophic event happens in one geographic location.

A standby system is the traditional form of providing DR. Having one, or several identical servers running an exact copy of your live system allows the organisation to switch between servers in the event of a failure to one.

The most significant issue with stand-by servers is when and how you do the copy process. If money is no object and you can manage your connections in-house, the speediest way to bring a standby server up is to use an approach similar to RAID 1, with two servers mirrored. This solution requires good network connectivity and the required infrastructure can be expensive to provide. The standby server is on and running all required applications. This is often referred to as Hot Standby.

A less costly solution is to send packets, for example a day's worth, of archived redo logs to the standbys and then apply the changes. The downside to this solution is that its can take time to replay the redo information so fail-over, whilst it can be automated, is not instant. This is known as Warm Standby.

A Cold Standby is the least complex solution which differs little from a Back-up and Recovery strategy except that the recovery is to a different server in a different location. The standby server, when needed, is started. A valid backup may need to be applied and as much redo history as can be replayed is used to bring the database to as close to current as possible. This approach will be much slower that the other two approaches.

Having a standby system available is not a new idea by any means. But using the cloud as a DR solution is. Specialist services providers are beginning to appear in this area. OpSource, for example (http://www.opsource.net/Services/Cloud-Hosting/Managed-Services/Cloud-Based-Disaster-Recovery) are suggesting a 4 hours restore time for a standby server is much cheaper in the cloud than on in-house owned hardware. The basis of the saving is that the database server is built and then taken off-line. As with most cloud solutions, the highest rental cost is for CPU and RAM usage, which are clearly zero whilst the database is offline. Should a disaster happen the server can be restarted and restored and connected to by users.

Since Storage As A Service is becoming relatively affordable and convenient even smaller organisations can provision their own DR using a provider's disk space and just start-up a server and restore the data for themselves.

10.4.3 Trade-offs and Balances

As we saw in Chap. 5, CAP theory has it that we can't have everything: out of Consistency, Availability, and Partition Tolerance, we can only have two. NoSQL databases, and Big Data systems like Hadoop, are built to be massively distributed. Availability comes as a result of the built-in redundancy from replicating the data across many nodes.

If always being able to access the data, regardless of network breakages is the most important thing to your organisation the theory tells us we can't guarantee anything other than eventual consistency. If both availability and consistency is critical (that is all the data across nodes agrees) then you have to live with the fact that your systems may be vulnerable to network problems to the degree that a network outage means your database is unavailable. This approach is what happens in many traditional RDBMS systems. Replication is used for availability and protocols such as the two-phase commit are used for consistency.

10.4.4 The Challenge or Opportunity of Mobile

As more organisations encourage employees to use their own devices such as smart phones and tablets, there is a danger of critical information being spread out in an unmanageable way. Data collected on a tablet but not synchronised with a server is exceptionally vulnerable, and yet could be business critical in nature.

This is a new challenge for information professionals. Adopting appropriate Bring Your Own Device policies which include synchronisation and back-up is clearly one useful step.

But the fact that some of these devices have relatively large storage available on-board, and the very fact that they are mobile means they are likely to be in a different geographical location to the central servers, or even other devices. A natural distributed system which perhaps could be used as part of a DR strategy? At the time of writing this technology is still new and this may end up being the section of this book that future readers laugh at… but who knows!

10.5 Summary

This chapter has reviewed the need to keep databases available for as long as possible. We discussed the trade-off between seeking Five Nines availability and the expense that that sort of robustness can incur. We then went on to talk about Back-up and Recovery types, and finished by looking at alternatives for limiting the downtime a database may suffer because of some sort of disaster.

10.6 Review Questions

The answers to these questions can be found in the text of this chapter.

- What is meant by "Five Nines" in terms of database availability?
- What element in a typical database server is most vulnerable to failure?
- What is meant by MTTR? And MTBF?
- Describe how RAID 1 and RAID 5 can be part of an availability strategy
- How does Shared Nothing differ from Shared Disk?

10.7 Group Work Research Activities

These activities require you to research beyond the contents of the book and can be tackled individually or as a discussion group.

Discussion Topic 1 You are asked to consider what can be done to mitigate against a potential disaster, such as an earthquake or flood, hitting your organisation's data centre.. Discuss some different approaches there are to Disaster Recovery whilst relating your discussion to the business requirements being addressed by each approach. Consider drivers like stakeholder expectation, cost, technical capability.

Discussion Topic 2 Keeping copies of the organisation's database is obviously a sensible precaution. There are, however, pluses and minuses to each potential approach. Imagine you are a DBA faced with implementing a back-up strategy. What options will you have before you, and what factors would impact your final strategy selection?

References

Alapati S (2009) Expert Oracle database 11g database. Apress, New York

Guerra A (Nov 2001) The buck stopped here: BONY's disaster recovery comes under attack. http://www.wallstreetandtech.com/operations/the-buck-stopped-here-bonys-disaster-rec/14703629