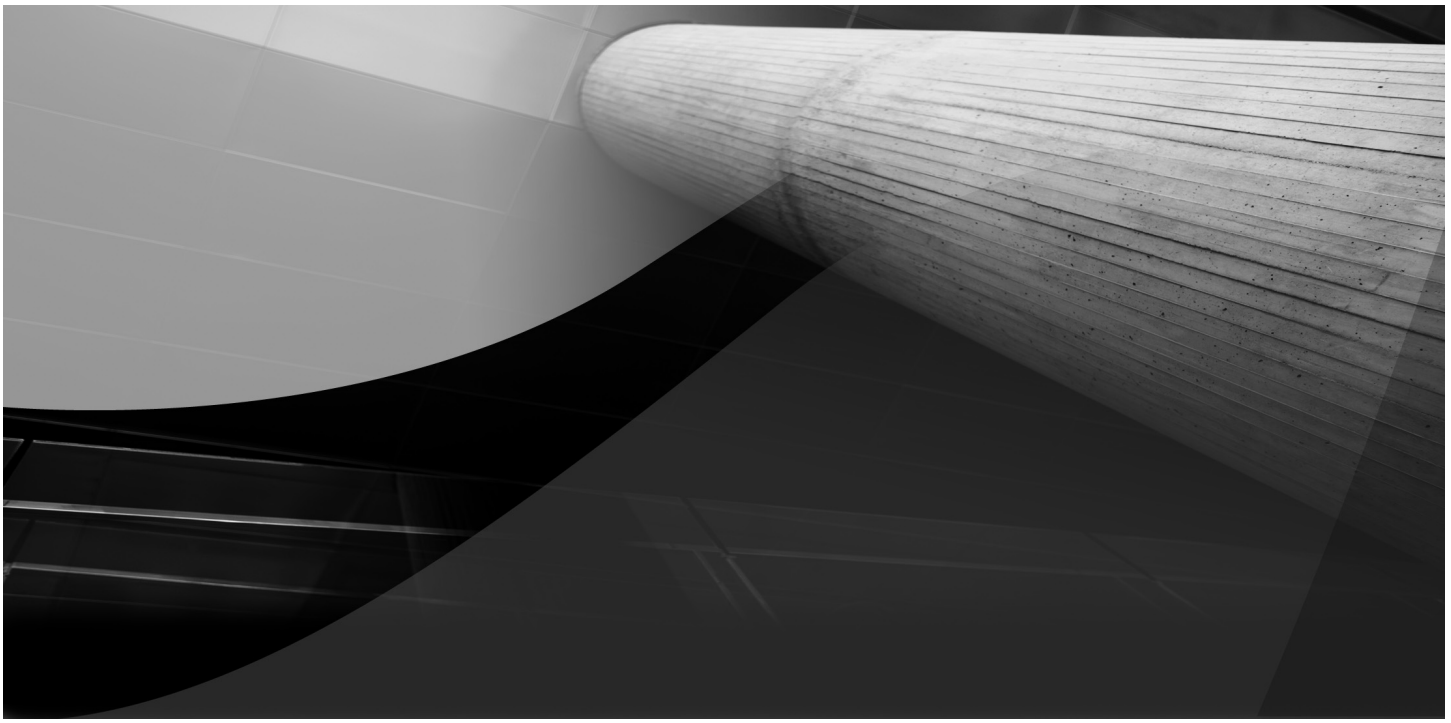
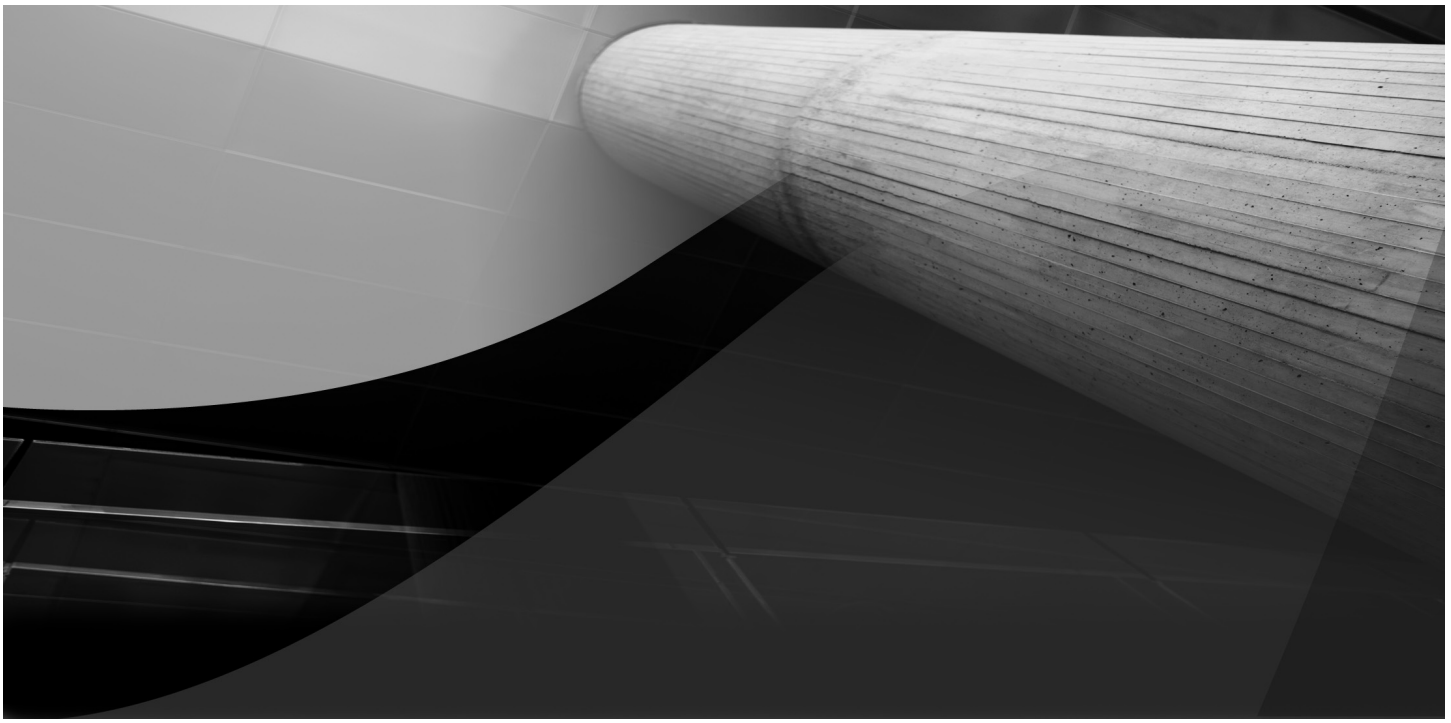


PART I

Oracle's Grid Infrastructure



CHAPTER 1

Architecting the Oracle Database Grid

As you have already discovered, or perhaps are about to discover, maintaining the Maximum Availability database is a tricky prospect. Hundreds of possible sources of downtime are hiding in the nooks and crannies of your enterprise. Outage situations cascade into catastrophes, as an attempt to resolve a problem creates another outage and quickly spirals out of control. But with careful planning or swift corrective action, almost all downtime scenarios can be prevented.

This book discusses numerous technologies that can prevent or drastically reduce a variety of database outages. What are these downtime scenarios? Which possible outages can you protect yourself against? The fact is that the Maximum Availability DBA lives by Murphy's Law: "Anything that can go wrong, will go wrong." Pure and simple. So to survive as a DBA, you must live by this maxim. Not only do you have to prepare for every eventuality—you have to expect it. Simply hoping that bad things or extreme scenarios will not happen is courting disaster. Putting things off with the mindset that things are working well so far is courting disaster. It is okay for you to hope for the best, but it is imperative that you plan for the worst, and this is never more true than in your role as a Maximum Availability DBA.

As discussed in the introduction to this tome, database availability is not defined by the database administrator; it is defined by the perception of the business—the users who expect the data they need to be accessible at the touch of a button. To illustrate the various types of situations that threaten the availability of your database, this chapter will walk through the planning stages of setting up a database grid meant to insulate the business and these end users from various types of problems that can ultimately cause data to be inaccessible, for whatever reasons. We will discuss different types of outages and the particular technologies you can use to protect against these problems, and we'll point you to the specific chapter in this book where the implementation of a particular technology is covered in detail.

LunarTrax: To the Moon and Beyond!

For the situations in this chapter, and for the workshops and examples throughout the book, we will use the database from the company LunarTrax, Inc., a fictitious startup company whose ambitious goal is to provide tourism to the moon. This company's primary databases hold several terabytes of data—everything from flight data, to solar flare forecast information, to marketing data for potential target customers, in addition to data used for internal accounting and human resources (HR).

The lead DBA and architect, Maximus Aurelias Anthony (known as Max to his friends), is a veteran of 1000 psychic wars and is taking no chances with the uptime and availability of his system. To ensure that the environment is as easily managed as possible, Max has implemented a consolidation strategy that reduces the original number of planned databases to a finite number. Once this strategy is implemented,

the primary production databases will include one database for customer-facing data (external customers) and one for internal data/internal customers. This consolidation strategy greatly reduces costs, both from an initial investment standpoint and from a maintenance standpoint. However, from a database standpoint, it essentially puts all of the company's eggs into just two baskets. Therefore, along with this consolidation strategy, it is essential that Max put in place an infrastructure that will guard these "eggs" according to their true value.

As a veteran of many years in the DBA world, including some time spent in the technical support arena, Max is well aware of the potential for disaster from all corners. To combat this, he is determined to implement a database grid with complete redundancy and the ability either to avoid or quickly recover from any of the common (and less common) pitfalls that can afflict a database and ultimately impact the business.

Planning the Grid

In planning for his database grid, Max must account for several areas to ensure that the business continues to run smoothly and is not impacted by database operations or failures. Max believes it is imperative that the environment be set up so that planned maintenance activities occur on an ongoing basis, while minimizing the impact to the business. Planned maintenance is crucial, because technology never stands still. In particular, in this competitive industry, the ability to stay one step ahead of the competition often depends on the best use of the latest technology. Therefore, having the flexibility to upgrade systems, both from a hardware and software perspective, is a key component to Max's database grid.

Max is also keenly aware of the fact that component failures occur. This is just a fact of life. Hardware in particular has a finite lifespan—moving parts can continue moving for only so long before they give out. Whether a lower-cost commodity hardware solution or a so-called high-end hardware solution is used, the reality is that at some point, a component is going to fail.

In addition to hardware failures are the inevitable software failures that must be anticipated. Whether a software failure occurs because of a design limitation or some unintentional flaw (aka a software bug), the chances of encountering a software failure of some type are a practical guarantee. Failure to plan for this inevitability is nothing short of a plan to fail.

Perhaps more concerning to Max than the inevitable component or software failure is the unpredictability of his users. As any good DBA will attest, Max would feel most secure if no users were allowed into the database at all. Unfortunately, that would defeat its purpose. So, like a father turning over the car keys to his teenage daughter for the first time, Max has to accept the fact that at some point, he will need to turn over at least some of the keys to the database to his end user population. When that happens, any number of things could potentially go wrong. He knows that the database grid at LunarTrax must have resiliency to withstand and recover from user errors.

Another priority for Max is having the ability to expand in the future. As a startup in a new industry, LunarTrax has high hopes for a bright future, but a shoestring budget in the present. Therefore, Max must build in the ability to expand his database grid in the future to meet the anticipated increase in demand that will arrive with the success of this endeavor—but this future expansion cannot be allowed to impact the current systems.

Finally, Max realizes that his primary responsibility is to prepare for the worst-case scenario—total disaster. Whether something as simple as a cut utility line causing loss of power to the data center, or something as catastrophic as a flood that leaves your data center under three feet of water, disasters can break a business. Protecting against disasters of this type is vital, and Max knows that he must ensure that his database grid can withstand the extended loss of the primary data center. His plans must account for both failover and failback strategies in such a situation.

This chapter discusses various aspects of the grid from a theoretical perspective and lays the groundwork for the rest of the book, which delves into the specifics of Oracle technology and how it addresses many important concerns.

The Grid and Planned Maintenance

Max is keen on ensuring that his company, and specifically his corporate database infrastructure, can stay up-to-date with the latest hardware and software maintenance releases. From a database perspective, this means he must develop a strategy for maintaining the software.

His first step is regular, proactive application of patchsets and patchset updates (PSUs), but in the longer term, he must plan for database upgrades in a timely fashion. But he has other aspects to think about as well. Operating system maintenance must be considered—including both the patching and upgrading of the operating system, plus any third-party software and applications. Max knows from experience (particularly from his days in technical support), that these activities are too often avoided and postponed rather than embraced. In many IT departments, patching and other types of maintenance are not always viewed as part of the expected routine for a DBA, but something to be avoided at all costs until it is forced on the DBA by a crisis. In these situations, the lack of system upkeep is either the direct cause of the crisis or, at a minimum, it can become an insurmountable hurdle in diagnosing and resolving the issue. What subsequently occurs is a "fire drill," in which the DBA is forced to perform necessary maintenance in the middle of a crisis, without proper planning and preparation. This lack of planning and preparation can easily make things worse, even though the measures are required to resolve the primary issue. Avoiding these "fire drills" is a crucial component of Max's MAA strategy.

In addition to the software maintenance, the DBA must be able to update system hardware—whether that means adding components such as CPU or memory

or implementing firmware or BIOS changes. In extreme cases, entire servers may need to be switched out, whether due to failure or simple obsolescence.

Based on past experience, Max is determined to embrace the maintenance of his environment aggressively. His plans are to incorporate the upkeep of all components of his database grid into the daily routine of his DBA staff. Even though many of these activities may not occur until well into the future, he knows that proper planning in the early stages can ensure that these future activities can be undertaken with minimal impact on the business, and the end result of this is an environment that can be sustained and healthy for the long term.

Oracle Technologies: Reducing the Impact of Maintenance

In terms of long-term planning, you have a few choices from an Oracle technology perspective that will help you keep up with the technology with minimal impact to your operations. This book focuses on two of the major technologies in this respect—Oracle Real Application Clusters (Oracle RAC) and Oracle Data Guard.

Oracle RAC and Planned Maintenance

We begin with a discussion of Oracle RAC. Oracle RAC is an Oracle feature based on clustering technology which provides simultaneous access to the database from every active node. This simultaneous access from live nodes is known as a "shared everything" architecture. Having "shared everything" access to your database from multiple nodes (servers), which is what Oracle RAC provides, is in many ways a boon to your database maintenance. You can apply many individual patches to the Oracle Database itself, including PSUs, in a rolling fashion by applying each patch to one node at a time. This lets you take advantage of the most critical short-term fixes without users losing access to the database.

In addition to database patches, Oracle RAC also lets you patch and even upgrade your operating system in a rolling fashion without incurring downtime. Whether you are applying a service pack (from RH4 U6 to RH4 U7, for example) or performing an actual operating system upgrade (such as an upgrade from Oracle Enterprise Linux, OEL 4 to OEL 5), it is possible to do this on a single node at a time in your Oracle RAC Cluster, without ever having to incur a full outage. Because of this (among other reasons), Max has chosen Oracle RAC to be a central component of the database grid at LunarTrax.

From a hardware perspective, implementing Oracle RAC also allows hardware maintenance to occur in a rolling fashion. Changes to BIOS or firmware can usually occur one node at a time, as well as replacement of individual components, either because of failure or due to an upgrade. Oracle RAC and patching specifics are covered in Chapters 6 and 7.

Oracle Data Guard and Planned Maintenance

Although Oracle RAC is clearly a central tenet of the database grid, it is not the be-all, end-all solution to every potential need. Applying patchsets to the Oracle Database or upgrading the database from one major release to another cannot be done in a rolling fashion with Oracle RAC alone. However, by combining Oracle RAC with Oracle Data Guard, you can achieve these upgrades in a rolling fashion with minimal downtime.

Converting a physical standby database to a logical standby allows you to perform the upgrade on the logical standby while logs are still being applied from the primary (where the business is still happily chugging away using the older release). After the upgrade of the logical standby, users can be migrated to the newly upgraded system with minimal impact, and the original primary can then be upgraded—again, while log files are moving between the two sites and everything is still synchronized. After users have migrated, the original primary system can be upgraded, and users (the business) can be switched back following the same process—all with minimal, if any, impact.

Therefore, Oracle Data Guard is another key component in Max's database grid strategy for LunarTrax. Oracle RAC and Oracle Data Guard combined allow Max the flexibility to perform ongoing maintenance of his environment, with minimal impact to the business. Specifics of implementing Oracle Data Guard are discussed in more detail in Chapters 9 and 12.

Additional Oracle Technologies

In addition to Oracle RAC and Oracle Data Guard, Max has evaluated technologies such as Oracle Streams and Oracle GoldenGate to solidify his database grid. Because Max has chosen to defer implementation of these technologies for later, we will not discuss them in this book but will include them in future editions as appropriate.

Recovering Quickly from Failures

Max's strategy of aggressively embracing and being proactive with maintenance will undoubtedly pay dividends by avoiding problems in the future. However, no one at LunarTrax is naive enough to assume that all problems will be avoided. No matter how diligent the LunarTrax DBA team is with patching, the possibility of encountering a bug in the software stack always exists, particularly because LunarTrax is intentionally pushing the boundaries of technology. And of course, as mentioned earlier, hardware component failures are a fact of life that cannot be avoided no matter how much you patch the software stack. Max hopes to eliminate downtime due to hardware component failures by ensuring redundancy throughout the system. This includes redundant network cards (network interface card [NIC] bonding), redundant disk access cards (multipathing), redundant storage (RAID), and even redundant servers (Oracle RAC).

In addition to this redundancy, a solid backup (and restore) strategy is imperative for any DBA. We'd like to say that this goes without saying, but Murphy's Law tells us to take nothing for granted. No matter how much redundancy exists in the system, data corruption or data loss can occur—so backups, as always, are a crucial component to any database grid.

Oracle Technologies at Play

To achieve the desired redundancy necessary to recover quickly from software failures or component failures, Max wants to ensure that he is using the best technology for his specific purposes. He is well aware that when it comes to backups, he has a plethora of options from which to choose. As an Oracle veteran, Max harkens back to the days of a database backup being as simple as an OS copy of a few gigabytes' worth of files while the database was shut down. But in the real world, with terabytes' worth of data, OS file copies no longer cut the mustard. Let's explore the Oracle technologies Max needs to achieve his goals.

Again with Oracle RAC

Clearly, Oracle RAC, with its shared everything architecture, is a key component to a redundant architecture. Any failure affecting an individual server's ability to function properly will not impact the entire database, because other nodes in the cluster will be able to continue the work seamlessly, even if an entire node is lost. Although other technologies such as Oracle Data Guard or Streams can achieve the same goal, they do so by keeping copies of the data in separate and distinct databases. This, however, implies a time delay in reacting to and recovering from relatively simple failures. Only Oracle RAC can provide this redundant access to the same database, so that even in an extreme case such as the complete failure of a node in your cluster, other nodes are up and running and actively accessing the same database even in the midst of the server failure. The remaining instances will automatically perform instance recovery for the instance that crashed, and any sessions that were connected to the downed server will be able to reconnect immediately to another instance that is already actively accessing the database.

Oracle Clusterware

The Oracle Clusterware component of Oracle's grid infrastructure is a necessary underpinning of an Oracle RAC Database. In addition to facilitating shared access to the actual database, Oracle Clusterware offers the benefit of monitoring for failures of critical processes such as instances, listeners, virtual IP addresses and the like, as well as monitoring node membership and responsiveness. When the Oracle Clusterware stack detects a failure of a critical component of the cluster, corrective action is taken automatically to restart the failed resource, up to and including the node itself. This architecture allows Max to meet his goal of recovering quickly from localized and relatively minor failures without impacting the business.

Oracle Recovery Manager (RMAN)

With respect to backups, too much is never enough. Max has determined that regular data pump exports will occur, but since a data pump export is only a point-in-time backup, his Oracle RMAN backup strategy is key to success. Because he will also be using Automatic Storage Management (ASM), the RMAN strategy is doubly important. RMAN allows you to take full or incremental backups, and lets you restore and recover as little as a single database block. Max knows that this type of flexibility in terms of backup and recovery is integral to maximizing the availability of the LunarTrax databases.

Flashback Database

Finally, Max decided early on to enable Flashback Database in all the environments. In days gone by, Max often found that he needed to perform an entire database restore to recover up to the time just prior to the occurrence of some critical error (such as an inadvertent deletion of data, some other logical corruption, or corruption of an online redo log). However, with the advent of Flashback Database, he knows now that he can alleviate the need to do this by essentially storing all of the blocks necessary both to redo and undo transactions for a specified period of time. This means that Max can do a "rewind" of the database without first doing a full restore, potentially saving immeasurable time in a crisis. Flashback Database and RMAN are discussed in more detail in Chapters 10 and 11.

Protecting Against and Recovering from User Errors

Speaking of inadvertently deleting data, the thing that most often keeps Max awake at night is the possibility of user errors causing data loss. Perhaps the most difficult outage situations are those tricky logical errors introduced by the users themselves—a user updates the wrong table or updates the wrong values, a developer thinks she is logged into the test system but is actually logged into the production system, or a user omits the where clause on a delete or update statement and 100,000 rows vanish or are logically corrupted. Max has been around long enough to see these problems happen over and over again—when months of work go down the drain.

This is a concern for Max particularly in the LunarTrax environment, where everyone works at a frenetic pace, and the company is not yet large enough to have clear lines of demarcation between production, development, and test environments. Max wants to be able to restrict access altogether to the production environments, but with the company's small development staff, and the rate at which changes are pushed through, this is just not practical.

Oracle Technology Checkpoint: Flashback Query and Flashback Table

Fortunately, the flashback features allowing recovery from serious errors are built into the database. The Flashback Query feature allows undo data stored in the Undo Tablespace to be read so that the DBA or anyone with appropriate access can "go back in time" and query the data as it existed before the incident. The Flashback Table feature also allows quick recovery if a table is inadvertently dropped, by maintaining a recycle bin, whereby dropped objects are essentially renamed and stored until space is needed. Max needs to ensure that enough storage exists to retain a sufficient amount of available undo, and a sufficient amount of free space must be available in user tablespaces to allow for upkeep of the recycle bin.

Beyond that, it is a matter of educating the developers and other users about the capabilities of Oracle Flashback technologies. The key is catching errors quickly—so full disclosure is important. Flashback technologies rely on data storage, and storage resources are not infinite. As long as errors are uncovered within a reasonable time period, Max and his DBA staff should be able to quickly recapture this type of data, and in some cases, the developers or users may be able to correct their own mistakes.

Again with the Flashback Database

Sometimes, however, serious database errors are tough to overcome. Typically, user errors do not occur in a vacuum, and an erroneous update can occur alongside hundreds of correct updates. Pretty soon, the bad data is buried by thousands of additional updates. How can you find just one of those transactions among thousands? Can you "rewind" the entire database back to a previous point in time? The answer is *yes*. Flashback Database is an important feature in Max's Maximum Availability Architecture (MAA) plans for the LunarTrax environment, and it can in fact "rewind" the database to allow for "do-overs." So in extreme cases, when Flashback Query is not enough, Flashback Database can help the MAA DBA retain his or her sanity in a world gone mad. Oracle Flashback features are covered in detail in Chapter 11.

Planning for Expansion and Future Growth

A key component in Max's Oracle database grid is the ability to grow the grid as LunarTrax grows. Max intends to budget as much as possible toward ensuring that a solid foundation is in place for development, testing, and quality assurance (QA) environments, as well as the production grid—but he knows his budget can't accommodate all of the planned capacity immediately. He hopes that, once the first

launches get off the ground, more money will be available for hardware, just in time for the expansion that LunarTrax will need in its database grid. Max knows that he still needs to plan now, however, for that future growth. And as the requirements for the grid expand and the grid grows, that growth cannot be allowed to have a detrimental impact on the current operations—it must be as seamless and transparent as possible.

Oracle Technology: Automatic Storage Management

Technologies such as ASM and Oracle RAC play pivotal roles. By implementing ASM from the outset, Max will have the flexibility to grow his storage easily in the future, without the need to bring down operations. Because ASM allows both the addition and removal of disks as needed, Max can even remove/replace storage that becomes outdated in the future, again without impacting the day-to-day operations of the database, since this can all be accomplished online.

ASM also allows striping across all available disks; this means Max does not have to invest costly resources in determining manual file system layouts. In the past, Max had to spend time determining where the I/O hot spots were as part of his regular tuning; then he had to move files to different file systems on different disks to achieve the I/O performance necessary for his operations. With ASM, this is not necessary, because not only are all files automatically striped across all available disks, but "hot blocks" are also automatically moved to the "sweet spot" on the disk, to improve seek times. ASM is discussed in more detail in Chapter 5.

Again with the Oracle Clusterware

Beyond storage, capacity at LunarTrax may need to grow as well, with the addition of servers to provide more CPU, memory resources, or networking resources to the business. Business needs often cycle over time: users may find the database completely accessible, but when activity in some areas increases, resources may become restricted. Max anticipates that the database for internal customers will experience problems at the beginning of each month, when accounts receivable (AR) must close out the preceding month's accounts and run massive reports on the status of all opening and closing accounts. This month-end requirement sends the database processing needs of the AR department through the roof—but only for a week or so, and then it settles back into a more routine usage pattern. At the same time, HR typically finds its peaks near the middle and end of the month, as it processes employee hours, salaries, and payments.

By implementing Oracle Clusterware, which is part of Oracle's 11*g* Release 2 grid infrastructure stack, LunarTrax will be able to create *server pools*, which allow for the transition of servers between different grids depending on demand. In addition, instances from different databases can be allocated to run on more or fewer servers within the grid, as needed. In the future, if the total number of servers needs to be

expanded due to an overall increase in demand, a new server can be added and the overall capacity of the grid can be expanded easily. By the same token, if older servers need to be retired or reallocated to new environments, this can be just as easily accomplished without impacting end users. This can all occur without taking the database offline, by adding nodes to or removing nodes from a cluster while the business continues to function as usual. Oracle's grid infrastructure and server pools are discussed in detail in Chapters 3 and 4.

Disaster Recovery

The next item on the agenda for Max and the LunarTrax staff is to plan for the ultimate: some type of disaster that renders the data center inoperable. During his time as a DBA and as a support tech, Max was exposed to all manners of issues that rendered an entire system (and in some cases, an entire data center) inoperable. These issues ranged from a simple power outage that extended beyond the capacity of the UPS, to full-blown flooding of the data center. Of course, hurricanes, earthquakes, and other natural disasters do occur, and the potential for terrorist attacks or sabotage is omnipresent. But a catastrophe in the data center itself is not always the cause of these disasters. Max recalls a scenario in which an inadvertent overwrite by the systems administrator of all the shared disks in a system rendered it completely useless. Although this did not impact the data center itself, the fact that Max's company had a disaster recovery site available for switchover, meant they were able to quickly recover from this type of error and keep the business going, saving several hours of downtime and countless thousands of dollars.

Oracle Technology: Oracle Data Guard

Based on their experience, Max and the LunarTrax crack staff have determined that an Oracle Data Guard environment should exist in two geographically separate locations. Given LunarTrax's plans to promote space tourism, Max intends to be the first person to implement a lunar Oracle Data Guard environment, but that is a long-term plan, dependent on some networking improvements. Near-term, the intent is to achieve geographic separation of at least 500 miles.

The LunarTrax DBA and networking teams have scouted appropriate terrestrial sites for a secondary data center to house their Oracle Data Guard environment. Given that their headquarters and primary data center are located in areas susceptible to hurricanes, they ultimately settled on the foothills of Colorado as an ideal secondary site, realizing that hurricanes, earthquakes, and other such natural phenomenon are relatively rare in Colorado. Nevertheless, the cautious nature of the DBA has induced the team to scout out tertiary sites as well, not necessarily to create cascaded standby sites, but to take advantage of the ability to sync up multiple standbys to a single primary. One site will have a delay implemented in the application of the logs to

provide some lag time in synchronization. This will allow Max and the team time to intercept any unwanted changes in the redo stream before they are applied to the tertiary site, preventing the propagation of any logical corruptions to all sites, should it occur.

Not wanting to tie up all this hardware in waiting for the unlikely (but inevitable) need to use it, Max wants to maximize utility as well as availability. Therefore, the intent of the LunarTrax secondary and tertiary DR sites will be to run a combination of Oracle Active Data Guard and Logical Standby to allow access to these systems for reporting purposes. This additional capacity will allow Max to justify the expense of maintaining the disaster recovery systems beyond simple standby resources. Oracle Data Guard, Oracle Active Data Guard, and Oracle Logical Data Guard are discussed in more detail in Chapters 9 and 12.

What Next?

"What next?" is a question often asked by DBAs in the midst of a crisis. In this context, though, this is a good thing. Here it means: What's next in terms of the proactive work that must be undertaken to keep data available to the business? In this case, you are in control of the next step, rather than having events control you. After the architectural foundation is created for the database grid, testing and implementation can begin.

Test, Test, and Test Some More

Crucial to your success in planning for the worst is your ability to react quickly to different types of problems. In many cases, Oracle has automated the responses—but sometimes, even when automation is possible, the DBA prefers to leave those decisions in the hands of humans as opposed to machines.

Testing should not only encompass functional testing, but it should cover processes and procedures for reacting to certain situations. This ensures that you and your staff have the practice and expertise necessary not only to know what the correct decisions are, but also to be able to execute those decisions flawlessly and efficiently. In addition, it is crucial that test environments mimic production as closely as possible. A three-node Oracle RAC cluster in your production environment with a single-instance test environment is not going to allow you to test all aspects of the production environment. By the same token, having a production environment with Oracle Data Guard, and a testing or development environment without Oracle Data Guard, will also leave you wanting in terms of the ability to test real-life scenarios that could impact your production environment.

A DBA with the best of plans and a perfect understanding of the theory of a technology can still lack practical experience, that only testing and trial by fire can provide. Even if decisions are automated to a certain extent, testing is still crucial at all stages. Automation relies on software written by humans, and because humans

have flaws, software has flaws. Couple this with a lack of practical experience and even the best theoretical systems can fail to live up to their expectations.

Test the Redundancy of Your Environment by Inducing Failures

As you know, when running Oracle's grid infrastructure and Oracle RAC, one of the primary goals is to achieve redundancy of various different hardware and software components. In many cases, recovery from certain failures is automated. But to be sure that redundancy works when needed and as expected, testing is a must. Testing uncovers configuration issues and defects that may cause the stack to fail to work as expected. If you have redundant network cards (aka bonding, or IP network multipathing [IPMP]), you can expect that if one card fails, the bonded pair will continue to act as one. By the same token, if you have enabled multipathing to your storage, how do you know that if one path fails, the other path will continue to provide unhindered access to your storage? You cannot just take this on faith. The answer is to test the environment. If you expect the system to recover from a failure automatically, testing by inducing failures prior to going into production will ensure that the system will perform as expected, or else allow you the opportunity to uncover configuration issues or software defects that prevent the stack from working as expected.

Test Your Backups by Restoring

In addition to testing the automated features of a clustered environment, you need to test the manual processes as well. In Max's time as a DBA and technical support specialist, he often encountered other DBAs who were adept at backing up their databases but had never practiced a restore or a recover operation. When something went wrong, and a restore was needed, they'd call Max. Even though he was happy to assist, the fact is that this cost time.

But what if this lack of experience costs you more than just time? If a restore and recovery have never been tested, how do you know it will work as expected? The middle of a crisis is not the time to find out that your backups were incorrectly configured, and now you need to do a full restore with only half of your files in the backup set. The middle of a crisis is not the time to learn that you don't have access to the tapes where your backups are stored. The middle of a crisis is not the time to find out that your database exports will actually take seven hours to import back in.

Max knows that the only way to ensure that a backup is successful is to restore that backup regularly to a test location. And the only way to know how long it will take to recover is to test it and measure it. It is not enough just to test prior to going into production, either. Things change in the production environment: the amount of data increases, the environment changes, and the situation changes. Even the people change. Regular testing not only validates the backup itself, but also the process.

Testing ensures that the DBAs on Max's team are skilled and practiced at restore and recovery operations, and when a crisis does occur, he can be confident that the necessary data will be available and the reaction to the crisis will be second nature. Furthermore, multiple solutions to a particular problem are often available. Some situations may call for using Flashback Database, while other situations call for a partial restore or perhaps a full restore. In other scenarios, Data Pump may be the most desirable solution. Knowing how long certain operations will take will allow you to make the correct decision as to which operation should be performed to keep the business up and running or to get it back online as quickly as possible.

Test Your Disaster Recovery by Switching to the DR Site

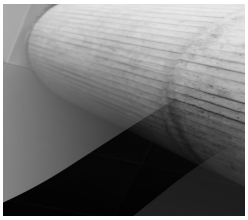
Beyond testing of backups, you need to test your disaster recovery scenario. The middle of a crisis is not the time to realize that you are unsure of the steps necessary to perform a switchover or failover. The middle of a crisis is not the time to find out that firewall issues exist between your application servers and the DR site on which the database is running. The middle of a crisis is not the time to realize that your network dropped a log file three weeks ago, and now your Oracle Data Guard environment is out of sync!

Max's database grid architecture is accompanied by test plans that include regular monitoring, regular testing of database backups by restoring to test systems from the production backup sets, and also regular switchovers of the production environment. Max intends to perform a planned switchover and switchback of the production environment between the primary site and the DR site at least twice per year, even after the production phase begins, to ensure that the necessary functionality is available when it is needed.

Sandbox Environments

So what exactly is meant by the "production system" anyway? "Production down" no longer means only the production system is affected. If a staff of testers or developers are idle while waiting on a "development" environment, then whenever that environment is down, it is costing the business money. Therefore, caution needs to be used even in these "lower environments" when introducing changes that might impact the productivity of any of your staff. A sandbox environment can be created to test basic patches and other scenarios that might impact any user base. A sandbox system does not necessarily need to be an identical copy of the production environment, but a system that can be used at will by the DBA staff is something that many businesses cannot afford to do without. It allows quick and immediate access to a production-like environment, where some basic sanity testing—such as checking for patch conflicts—can occur without impacting any users in the environment.

Go Forth and Conquer

The downtime scenarios illustrated in this chapter are the tip of the iceberg. You can see that many common, everyday situations can be fixed using the functionality provided by the full Oracle stack. In most cases, with Oracle Database 11*g* Release 2, this functionality is at your fingertips and you can leverage it immediately, with just a few steps outlined in the chapters that follow. Remember that planning, preparation, and testing are important to ensure that you are prepared to take advantage of the Oracle database grid to the full extent of its capabilities.

Now that we've explored a few of the situations that might cause angst for the MAA DBA, let's roll up our sleeves and dig into the technologies available to prevent (or significantly reduce) these disruptions. The ensuing chapters will provide the technical details you need to build your own database grid from the ground up. Go forth and conquer!