Backup and Recovery

Backup and Recovery Motives

The DBA must remember that the data in a table cannot be recovered without backups. This is the critical nature of backups for recovery purposes. When determining the setup for a backup schedule, the DBA should keep several things in mind. First, the nature of business for this application should be remembered. Is the application meant for online transaction processing situations, where several thousand users are entering data daily? If so, then the backups taken on that type of system should reflect the volatility of the data being stored. On other systems, where data may be fed to the database on a batch schedule such as in the situation of a data warehouse, backups may be required only after said batch processes **update** data. This may true because after that time, the database is primarily available on a read-only basis to its users. The point being made here is that the backup and recovery strategy must be tied very closely to the overall objectives of the database being backed up.

Disaster can strike any computer system at any time. Some of the implications of disaster recovery for any system include data loss, reduced productivity for users while the computer system is recovered, and hardware costs for replacing missing or destroyed server and/or client components. Other costs to the computer system include the time it may take to manually reenter data that could not be recovered automatically due to the absence of backup components. Any or all of these issues may be deemed acceptable by the organization, depending on whether or not the organization is risk-taking or risk-averse. However, as with most decisions in organizational settings, management should be involved and ultimately responsible for making the hard decisions about backup and recovery. Oracle supports many options that can provide a high level of recoverability for database data. However, many of the options, such as backup archiving offsite or a standby database, have premium costs attached to them. Having as many options as possible at the DBA's disposal is important, but so is the cost of providing those options.

One final area of importance when considering the motives for backup and recovery is to test the backup strategy to ensure the success or failure of the strategy before a problem occurs. Testing backup and recovery in the organization accomplishes several goals. The first goal accomplished is that the overall strategy can be observed in several different scenarios to determine if there are flaws or weaknesses in the plan. The bonus of testing the plan is also that the plan can be remodeled in a noncritical atmosphere, with little or no consequence regarding the loss of data and subsequent frustration of the organization with its DBA. The second important consequence of periodic testing of the database backups relates to the use of physical backups. Testing backups allows the DBA to determine if there is some sort of subtle data corruption in the physical database files that is being copied into all the backups, rendering them as defective as the production database. Finally, testing the backup and recovery strategy allows the DBA to develop expertise in that area, which offers the benefit of the DBA developing well-tuned backup and recovery.

Overall, the success of a recovery is not only dependent on whether or not all data can be restored successfully. The true victory in database recovery is won when the DBA can restore all data *successfully and quickly*, so as to minimize the overall strain a database failure poses to the organization.

Backup Methods

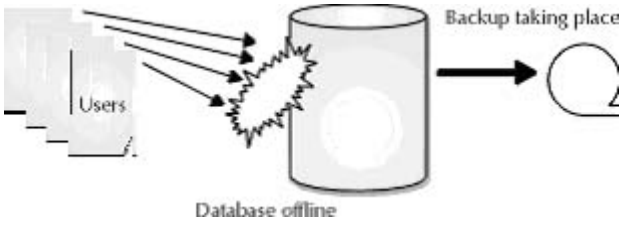
The first difference between physical and logical backups is based on the two fundamental viewpoints a DBA has on the information stored for the Oracle database on disk. From the logical perspective, the disk contains tables, tablespaces, indexes, sequences, and the like. The logical backup supports this viewpoint by allowing the DBA to handle the backup entirely within Oracle, at the level of tables, indexes, sequences, users, and the like. The tool in Oracle

used to support logical backups is the EXPORT tool. The other perspective of Oracle disk usage is the perspective of the operating system. Oracle stores data in files that look the same from the operating system as other files. This is the physical view of the database. Backups from this perspective consist of copies of the files that comprise the Oracle database.

Logical database backups can be taken with the EXPORT utility provided with Oracle. Good exports start with the identification of low usage times on the database, times that are suitable for exporting data. Ideally, the users are locked out of the database by the DBA enabling the **restricted session** mode. With the users locked out of the database, the DBA can ensure that the backup made is a read-consistent view of the database at the point in time the backup is made. After the logical backup is complete, the database should be opened for general use by disabling the **restricted session** mode.

Cold Backup

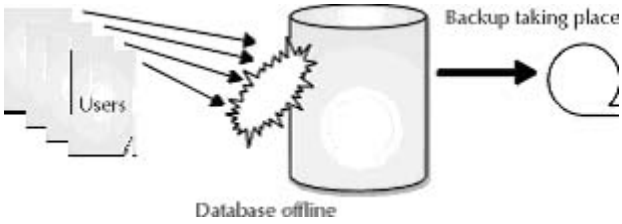
Physical database backups are divided into two categories-offline, or "cold" backups, and online, or "hot" backups. Cold backups may only be accomplished by the DBA when the database is closed, or unavailable to users. The database must be shut down using the **shutdown normal** or **shutdown immediate** statements. If the **shutdown abort** options are used to shut down the database, the next time the database is opened, Oracle will attempt instance or media recovery, respectively. Shutting the database down in **normal** mode is the method used to avoid this problem. Only when the database is closed normally is the DBA assured that all changes made to data in the database that may still be in memory have been written to disk. Once the database is closed normally, the DBA can begin the process of taking a complete copy of the physical database files in the database. These files are generally of four types. The four types are datafiles, control files, redo log files, and parameter files. Only full backups of the database are available with offline backups; it is inappropriate to back up only a few of the datafiles in this situation. To obtain a list of all the files that should be backed up in the database, the DBA can look in the following views: V$DATAFILE for datafiles, and V$LOGFILE for the redo log files. To obtain the names of the control files for the Oracle database, the DBA should execute the **show parameters control_files** command from Server Manager. Once all the database files have been backed up, the DBA can then restart the database. Diagrammatic representation of offline backup:



Online Backup

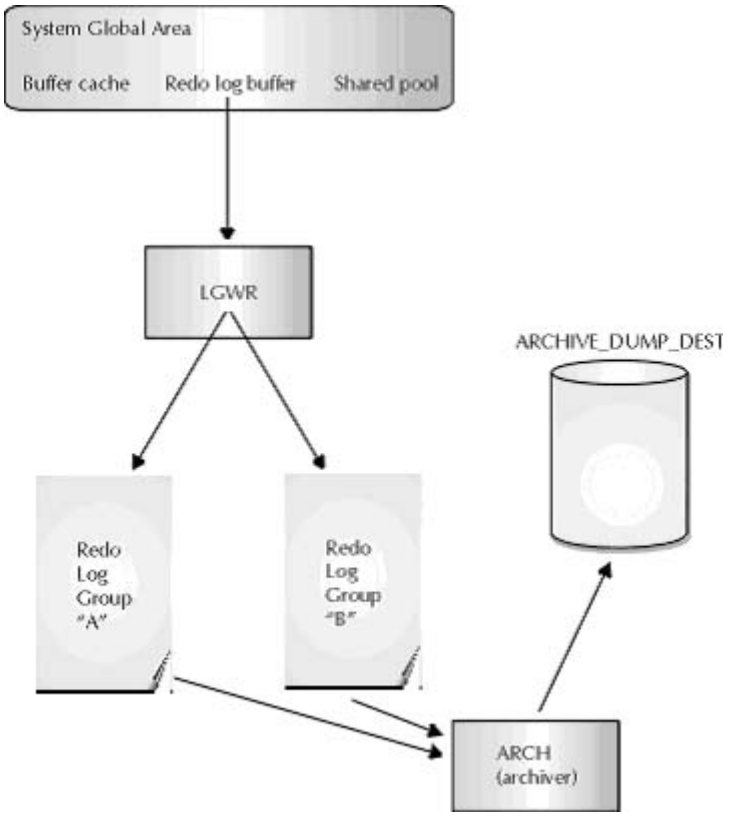
The final type of backup considered is the online backup. These types of backups allow the database to be up and running during the entire time the backup is taking place. Hot backups require archiving of redo logs in order to recover the backups. Hot backups require the DBA to adopt a hybrid logical-physical view of the database. Hot backups are taken in an iterative process, one tablespace at a time. The first step in a hot backup is to place the tablespace into **backup** mode with the **alter tablespace** *name* **begin backup** statement. Once executed, the DBA can make a copy of the datafiles that comprise a tablespace to an alternate disk, or off the machine entirely to tape or other external media. Once the backup is complete, the **alter tablespace** *name* **end backup** statement is issued. Since the database was open for activity the entire time the backup took place, the DBA must archive the redo logs that were collected during the time of the backup. In order to know which redo logs were used, the DBA should execute the **archive log list** statement within Server Manager and take note of the oldest online redo log sequence. After the backup is complete, the DBA can force archival of the redo

log entries saved written during the backup with the **alter system switch logfile** statement. Online backup illustration:



Archiving Logs

The importance and value of archiving redo logs in the various backup situations. Archiving redo logs allows the DBA to save all redo information generated by the database. In the event of a database failure that destroys some data, the changes that were written to the redo logs can be reapplied during database recovery to allow for full re-creation of all data changes committed right up to the point the database failed. If the DBA is using the logical export options provided with the Oracle EXPORT and IMPORT utilities, then archiving is of limited value because archiving cannot be applied to recovery using logical backups. Only if the database is using physical backups can the use of archiving be applied. For any database that requires recovery of data to the point of database failure, Oracle recommends the archiving of redo logs and the use of physical database backups. Oracle redo log architecture is shown below:



Availability

Some operations have special database availability considerations that will influence the backup options that are required for those systems. This statement is particularly true for databases that are required to be available at all times, or 24? 7 databases. There are two types of situations where 24-hour availability may be required for a database. One is where the database is deployed globally to users around the world who work at different times. The other is for databases that are deployed to users on a limited range of time zones, but the application relies on periodic refreshing of data as provided by batch processing that runs during off-peak usage time periods in the day. The 24-hour database cannot be brought down or have **restricted session** enabled on it frequently enough to allow for offline or logical backup. Furthermore, their recovery requirements include the ability to recover all data up to the point of time the database failed. As such, these systems generally require online backups in conjunction with archiving of online redo logs. The online backup allows users to access the database at all times, even while the backup takes place, and the archival of redo logs allows the DBA to apply all database changes made during and after the backup up to the moment in time the database failed.

Recovery

Database recovery has several implications related to time. Database recovery is a two-step process. The first step is the application of the backup to the database to restore lost or damaged information. The second part is the application of archived redo logs in order to allow for recovery to a point after the most recent backup, usually to the point in time the database failed. Depending on the type of recovery plan pursued and the number of archived redo logs that need to be applied, the database recovery can be time-consuming. Another time consideration that should be weighed into database recovery is the amount of time required by users to recover their own data changes by reentering the data lost.

Implications of Backup Time

The backup options pursued by the DBA have time implications of their own. Depending on the option chosen for use in the database, the DBA may require time where the users are not permitted to access the system. This requirement may be fine for systems that are expected to be available during normal business hours. In this case, logical or offline backups may be an option. However, there are situations where the database must always be available. In this case, the DBA should choose online backups in conjunction with the archiving of redo logs. The choice of backup method used to support recoverability on the database will impact the time required for recovery on the database.

Levels of Transactions

Transaction volume will ultimately affect the backup and recovery of a database as well. This fact is especially true for databases that archive their redo log information. Since high transaction volumes produce large amounts of redo log information, the number of archived redo logs has a tendency to increase quickly. In order to recover all database changes made in the event of a database failure, all archived redo log entries made during and after an online backup, or after the offline backup, will need to be applied after the backup is applied. This effort of applying the redo information takes more or less time depending on the number or size of the redo logs to be applied. The more redo logs, the longer the recovery. In order to combat the length of time it takes for database recovery with archived redo logs, the DBA should save tablespaces that experience large transaction volumes frequently. Even in environments where archiving is not used, the overall recovery time for a database, including the time it takes for users to reenter the data lost from the period of time following the most recent backup to the database failure, is improved if backups occur frequently. However, obtaining downtime on the database for an offline backup or a logical backup may not be possible more often than once a day. This fact means that the system may always be susceptible to a day's worth of downtime in the event of a database failure, plus the time it takes to perform the recovery that is possible using the offline or logical backups available.

Read-Only tablespaces

The other situation is that involving read-only tablespaces. Since the data in read-only tablespaces cannot be changed, the backup requirements for these tablespaces are minimal. Whenever the data does change, the tablespace must be backed up. After the tablespace is set into **read only** mode, the data should be backed up once and then left alone, as the DBA can rest assured that the tablespace will not change.

Function and Purpose of logical Backup and Recovery

Failure Scenarios

The first area of discussion involves descriptions of the various failure scenarios a DBA may encounter in the course of administering an Oracle database.

Statement failure

The first failure scenario is that of statement failure. Statements fail for many reasons. For one thing, the user issuing the statement may have spelled a table name incorrectly, causing Oracle to return a "table does not exist" error. Another reason for statement failure is the inability of the user to see the database object he or she is trying to access. Often, the cause for this problem is the fact that the user is missing or denied an object privilege, or that the object truly does not exist in the database.

User process failure

Another type of failure scenario that the DBA may encounter is user process failure. In this situation, an entire process executing on the database fails. Oracle handles many aspects of process recovery after statement failure with the use of the PMON process. PMON is process monitor, a process that monitors the activities of user processes to clean up after processes that fail. Some of the things PMON handles include release of locks the statement may have had, rollback of any transaction activity the process may have generated, and removal of that process from the list of active processes maintained in the V$ dynamic performance views.

User Error Failure

The next failure scenario considered is the user error. This type of failure is generally caused when someone inadvertently deletes or changes data in the database and commits the change. Alternately, the situation may arise when the user or developer truncates or drops the table in error. Data definition language statements such as truncation and table drops are not recoverable with the use of the rollback segment in the same way that an **update, insert** and **delete** are. The DBA may need to intervene in this situation by providing data or object recovery. Although logical database backups for the supplemental support of user error is the ideal approach to this failure scenario, it is possible to make a point-in-time recovery using physical backup followed by an export of the database object that was changed or dropped.

Instance failure

Another scenario explored is that of instance failure. This scenario occurs when there is some problem with the hardware of the host machine running Oracle that causes the database to shut down unexpectedly. Additionally, instance failure occurs when the **shutdown abort** command is used to shut down the database. Finally, instance failure can occur when there is some power failure on the host machine running Oracle that causes the instance to terminate abnormally. The SMON (system monitor) background process handles instance recovery the next time the Oracle instance is started if the database shows signs of instance failure. The

most that is expected of the DBA in this situation is to identify the cause of the instance failure and resolve it. At any rate, if the DBA handles the hardware situation that creates instance failure, Oracle will handle the rest automatically once the instance is restarted.

Media failure

The final situation that a DBA may encounter is media failure. Media failure, also known as disk failure, is the result of a problem with the disk drive that stores Oracle data. If there is a situation where the disk is irreparably damaged, or something happens that renders Oracle's access to its physical database files impossible, then the DBA will have to obtain new hardware for the database to use to store information and also recover the files that were lost by using the backups the DBA keeps in the event of this type of an emergency. This situation is generally the most manual intensive for DBAs to handle out of all the failure scenarios. It is also the most destructive. In a situation of media failure, there is always the chance for permanent loss of data. The DBA must institute an effective strategy for database backup and recovery in order to guarantee that the users of the database do not experience loss of their data.

Logical backup

Using Export

The utility used for logical database backup is called EXPORT. This utility is included in the distribution of Oracle database software, and can be executed by the DBA or another user who has the EXP_FULL_DATABASE role granted to their username. This role is created by the **catexp.sql** script, which is run automatically at Oracle software installation time.

In order to manage the execution of EXPORT, the DBA passes a variety of parameters to the tool at the time the export is taken.

USERID (*name/pass*)	The user and password under which EXPORT will execute.
BUFFER (*number*)	Defines a buffer size EXPORT will use to fetch rows for the export file.
FILE (*filename*)	Identifies the name and location of the export file produced.
GRANTS (Y/N)	Indicates whether table grants should be included in the export.
INDEXES (Y/N)	Indicates whether table indexes should be included in the export.
ROWS (Y/N)	Indicates whether table rows should be included in the export.
CONSTRAINTS (Y/N)	Indicates whether table constraints should be included in the export.
COMPRESS (Y/N)	Indicates whether EXPORT will place all rows of the table into one initial extent in the export file.
FULL (Y/N)	Indicates whether EXPORT should export the entire database.
OWNER (*name*)	Indicates a list of users whose database objects should be exported (**user** mode).
TABLES (*list*)	Indicates a list of tables that should be exported. (**table** mode)
RECORDLENGTH (*number*)	Lists the size of each data record in the export file. If the DBA wants to import the exported file onto a database in another operating system, this value must be modified to the proper value for that operating system.
INCTYPE (*keyword*)	Accepts the keywords **complete, cumulative**, or **incremental** to indicate the type of EXPORT executed.
HELP (Y/N)	Displays a help message with all features of EXPORT described.

RECORD (Y/N)	Will specify information about the export in one of the following SYS tables used to track export information: INCVID, INCFIL, INCEXP.
LOG (*filename*)	Specifies the name of a file containing runtime details and error messages for the given export.
CONSISTENT (Y/N)	Allows the cumulative or full export to obtain a read-consistent view of all data exported. Requires large rollback segment. Effect can be duplicated by the DBA enabling **restricted session** mode before beginning export.
FEEDBACK *(number)*	When set, displays a dot to indicate progress on rows exported per table.
STATISTICS (*keyword*)	Accepts the **estimate, compute**, or **none** keywords. Used to generate statistics for cost-based optimization on the database.
MLS (Y/N)	Used for Trusted Oracle. Stores the multilayer security label for tables and rows in the export.
MLS_LABEL_FORMAT	Used for Trusted Oracle. Redefines the default format for multilayer security labels.
DIRECT (Y/N)	Allows the DBA to run faster exports using the direct path. Similar in function to direct path in SQL*Loader.

These parameters control virtually every aspect of the export, from the name of the export dump file produced to the objects it will contain. Some of the parameter names that are crucial are FILE, USERID, OWNER, TABLES, and FULL. In order to execute, EXPORT must be able to log onto Oracle with a username and password.

The final three parameters mentioned above have the purpose of determining the mode that EXPORT will run in.

Export Modes

The three modes available to EXPORT are **user, table**, and **full**.

<B.USER< b>mode is where EXPORT accepts the name of a user in the database who owns some database objects. The objects owned by that user schema are then the only objects that will be exported. Using the export tool in user mode:

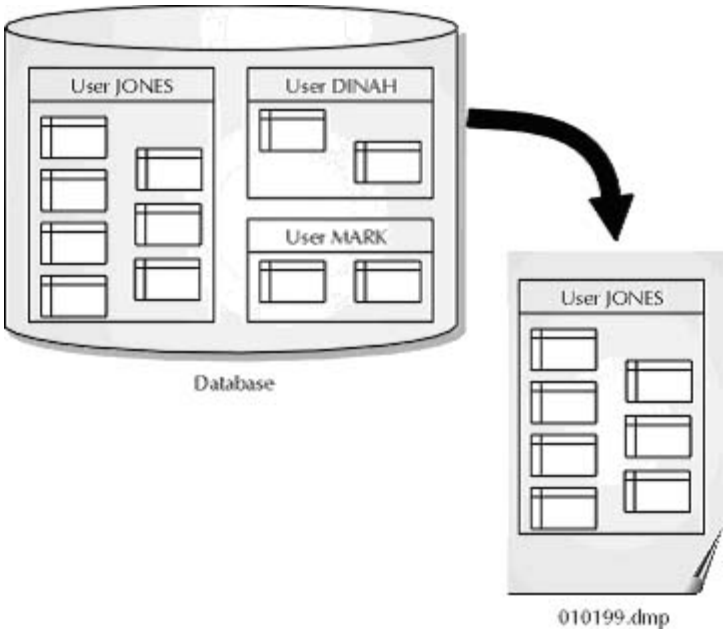


Table mode is another mode where EXPORT is passed a list of specific tables that it will store in the export file. In both of these cases, the DBA can further limit the database objects that are taken for backup with the use of parameters. These parameters specify whether or not the classes of objects they represent will be exported into the dump file. These object parameters include INDEXES, CONSTRAINTS, TRIGGERS, and ROWS.

Full mode is specified with the FULL parameter set to Y. In order to run EXPORT in **full** mode without conflict, it is important that the DBA not set the OWNER or TABLES parameters in conjunction with setting the FULL parameter. There are three types of full exports available to the DBA. They are *complete, incremental* and *cumulative*.

The type of EXPORT made in **full** mode depends on the setting the INCTYPE parameter to either **complete, cumulative**, or **incremental.**

Complete exports save the information one would expect they would save-the entire database, minus any database objects specifically omitted by the INDEXES, CONSTRAINTS, and other parameters identified above.

Incremental exports store only the database objects that have experienced changes since the last export *of any type* was taken. Note that the incremental export stores a copy of the entire object, not simply the data that was changed. In other words, if one column in one row in a million-row table was altered, the whole table is exported.

Cumulative exports store all database objects that have changed since the last full or cumulative export was taken. In a sense, cumulative exports are somewhat redundant because they export the same data that is stored on incremental exports, which may have been taken prior to the cumulative export but after a complete or cumulative export. However, in this situation, cumulative exports provide the value of consolidating the data backed up on several incremental exports into one file.

Using Import

Once exported, the data in an export file can be imported to the same database or to another database. This "recovery" of backup data is handled with the IMPORT utility. This utility

complements EXPORT. IMPORT accepts several different parameters that modify how the IMPORT will operate. Like EXPORT, IMPORT runs in three modes. Those modes are user, table, and full. An important difference to note here is that IMPORT can run in user or table mode using an export dump file, regardless of the mode EXPORT ran in to produce the export dump. As long as the database object is there for IMPORT to use, IMPORT can specify it to be imported any way the DBA sees fit. List of parameters supported by import:

USERID *(user/pass)*	The username and password used to run the IMPORT.
BUFFER *(number)*	Parameter that defines the number of rows inserted into a database at one time.
FILE *(filename)*	Determines the name of the export file to use for the input
SHOW (Y/N)	Displays the contents of the export file but doesn't actually cause IMPORT to import anything.
IGNORE (Y/N)	Specifies whether to ignore errors that occur during import.
GRANTS (Y/N)	Specifies whether grants in the export file should be imported.
INDEXES (Y/N)	Specifies whether indexes in the export file should be imported.
ROWS (Y/N)	Specifies whether rows in the export file should be imported.
FULL (Y/N)	Determines whether the import will be in **full** mode.
FROMUSER *(name)*	The names of schema user database object owners for the objects in the export file that should be imported.
TOUSER *(name)*	Identifies the user schema into which database objects should be placed if the IMPORT is running in **user** mode.
TABLES (Y/N)	Specifies whether tables in the export file should be imported.
RECORDLENGTH *(number)*	Identifies the length in bytes of the each record in the export dump. Must be specified properly.
INCTYPE *(keyword)*	Defines the type of import that will occur. Valid values are **system** and **restore**.
COMMIT (Y/N)	Specifies whether IMPORT should **commit** after each time a buffer's worth of data is written to the database
HELP (Y/N)	Indicates whether IMPORT should display help information about the parameters and their meanings.
LOG (filename)	Indicates the name of a file into which all IMPORT runtime information and errors will be stored.
DESTROY (Y/N)	Indicates whether IMPORT should reuse the datafiles that exist in the database for storage of imported objects.
INDEXFILE (Y/N)	Indicates whether IMPORT should create a file that contains a script to create the index for a table rather than creating the index itself.
FEEDBACK (Y/N)	IMPORT gives the same dot notation to indicate progress in the importation of data.
MLS	Used in conjunction with importing data into Trusted Oracle.
MLS_LISTLABELS	Used in conjunction with importing data into Trusted Oracle.
MLS_MAPFILE	Used in conjunction with importing data into Trusted Oracle.

User mode for IMPORT is slightly different from the user mode for EXPORT. There are two parameters that manage the use of IMPORT in user mode. Those parameters are FROMUSER and TOUSER. The FROMUSER parameter corresponds to OWNER in EXPORT insofar as the database objects in the export dump have a schema owner. If the DBA wants to import the objects owned by a particular user stored in the export file, the DBA specifies that user

schema in the FROMUSER parameter. However, the DBA must also specify the TOUSER parameter for the execution of IMPORT as well. This fact is due to the situation where the user on one database from which the database objects were extracted does not exist on the database to which the database objects will be imported.

There is a difference between the options used to run full imports as well. First, there are only two different types of full imports specified as values for the INCTYPE parameter, **system** and **restore**. The import must be run with FULL=Y and INCTYPE=**system** first using the last export created on the database before a media failure, and is critical to the successful recovery of the database. This run of IMPORT re-creates vital data dictionary and other SYSTEM tablespace information. Then IMPORT must be run to recover the database objects stored in all the different types of exports, namely complete, cumulative, and incremental. The proper order for applying exports to the database for recovery purposes after the most recent export is applied in the SYSTEM import, is listed as follows. First, apply the most recent complete export. Next, apply all cumulative exports since the complete one in least- to most-recent order. Finally, apply all incremental exports taken since the most recent cumulative export.

Bear in mind that the logical backup and recovery services provided by EXPORT and IMPORT have the limitation of only being able to provide recovery to the point in time of the most recent database export. In other words, if there have been several database changes since the most recent export, those changes will be lost. To recover those changes, the users will have to reenter the data.

Dictionary Tables

In order to determine which exports contain which database objects, the DBA has several dictionary tables at his or her disposal. The three tables in the data dictionary that are used to determine the contents of export files are called INCEXP, INCVID, and INCFIL, and all are owned by SYS.

INCEXP contains a listing of all the database objects that are stored in exports and the schema containing the object.

INCFIL is a catalog of all database exports, their ID numbers, and all information pertaining to the creation of the export, such as time and date and the user who created it.

INCVID contains the information for the last export that was created. INCVID is used for the purpose of creating an ID for the next export that is created on the database.

Read Consistency and Database Export

Read consistency has been defined so far at the statement and transaction levels as the consistency of data in the database during the period of time the statement or transaction executes. It is specified with the CONSISTENT parameter. When specified, it allows the complete or cumulative export occurring in the database to have read consistency during the period of time of the export. Depending on the number of objects in the database, the time it takes the export to complete, and the number of transactions that are changing data in the database, the rollback segment space that is required to sustain the export can be quite large. In addition, read consistency is not possible for use with incremental exports.

Read consistency with the CONSISTENT parameter promises a lot, but in reality does not always measure up. Since the parameter cannot be used with incremental exports, the DBA must secure other methods for ensuring that the read consistency of the database is not compromised during the export. One popular method for doing so involves the use of Oracle's **restricted session** mode. When the DBA issues the **alter system enable restricted**

session, only the users that have the **restricted session** privilege granted to them may access the database. Ideally, this privilege is only granted to the DBA. Since no other users can access the database during this time, there are two factors that should be considered. First, the export should take place when there are few users on the database, considering the full-service backup and recovery approach offered in Figure 12-2. Second, the restricted session will prevent users from making database changes while the export runs, effectively creating read consistency.

As mentioned on a few different occasions, the goal of a database recovery is not only to restore lost data, but to do so quickly. A parallel objective involves the fast backup of data as well. Due to the fact that the export is generally conducted on a database in restricted mode in order to avoid the read inconsistency issues raised by leaving the database open to user changes and messing with the CONSISTENT option, the backup of a database using EXPORT should happen as quickly as possible. This can be assured when the DBA incorporates the DIRECT parameter into the usage of EXPORT. EXPORT uses the same SQL processing mechanism used by regular user queries to obtain data from the database. The direct path eliminates some of the processing that regular queries and the conventional export path incorporate. The direct path optimizes the queries used to obtain data from the database as well.

One of the biggest drawbacks of the logical backup and recovery options provided by IMPORT and EXPORT include their inability to restore data changes that were made after the most recent backup was taken. Generally, this function is provided with the application of archived redo logs. However, the export files that are created with EXPORT are incompatible with archived redo logs. As a result, it adds no value to database backup and recovery to archive redo log information. Logical backup strategy is most effective in situations where the database user population can withstand the inevitable data loss.

Character Set

The use of national language support. When an export takes place, EXPORT uses the character set for the database to store the exported data for that database. The IMPORT tool can import data from that character set even if the target database uses a different character set, due to the fact that the IMPORT tool can perform a character set conversion on the fly. However, this conversion will increase the amount of time required for the import to finish. If the DBA tries to export the data from the database in a character set other than the character set for that database, the export will fail.

Role of Archiving

The importance of archiving was again highlighted as an integral part of database recovery. The architecture for redo log archiving consists of many different components. The first component for redo log archiving is the redo log buffer, a memory area designed to capture the redo information produced by statements changing data in the database. This redo information is written to a disk file, called the online redo log, by a special background process called LGWR. There must be at least two of these online redo log groups in order for Oracle to function properly. LGWR writes data to the redo log group, switching back and forth between them. When one log group fills, LGWR writes to the other log group. This design forces LGWR to overwrite redo information unless the DBA chooses to archive redo log information. When archiving is enabled, then Oracle allows for the archiving, or saving, of redo information. The movement of this data is handled either manually by the DBA or automatically by the ARCH background process.

The DBA should strongly consider archiving redo log entries because it allows for many different options when the need arises to make database recovery happen with the backups available. When archiving is used and recovery is required, Oracle makes the recovery an interactive process with the DBA. Along the way, Oracle suggests archived redo logs to apply

based on the contents of its V$ performance views for redo logs, namely V$LOG and V$LOG_HISTORY. The DBA can confirm these suggestions or supply her own recommendation, which Oracle will then use to execute the recovery. In addition, the DBA can use automatic recovery, whereby Oracle will simply apply its suggestions for recovery, bypassing the interactive portion of the recovery.

Types of recovery

There are several different types of recovery available to the DBA. One division of recovery options into categories is complete and incomplete recovery.

Complete recovery

Within complete recovery, there is generally only one type of recovery that is complete. That is the complete recovery made possible with archived redo logs to recover database changes to the moment of the database failure.

Incomplete recovery

Forms of incomplete recovery include any type of recovery that takes place by applying archived redo logs to a point in time in the past. There are three types of incomplete recovery: *change-based, time-based, and cancel-based*. These three types of database recovery are designed to allow the DBA to recover the database to a point in time in the past, specified by a variety of different methods.

Cancel-based recovery

A cancel-based recovery is one that runs based on the interactive part of database recovery with archived redo logs. The recovery will continue to apply archives until Oracle makes a suggestion for the next redo log and the DBA issues the **cancel** command.

Changed-based recovery

This recovery is based on the system change numbers (SCN) that Oracle applies to every transaction that runs on the database. When issuing the command for change-based recovery, the DBA must supply a system change number that Oracle will use to determine the stopping point for the recovery. When Oracle completes application of the redo log information for the SCN for the log containing that SCN, the recovery will end automatically. The result is a database that is recovered to a point in time in the past at which the final transaction is the one whose SCN is the one defined at recovery time by the DBA.

Timed-based recovery

Perhaps the most straightforward of all incomplete recoveries, the time-based recovery requires the DBA to specify a time to which the database should be recovered. When Oracle has applied enough archived redo information to recover the committed transactions to the point in time specified for time-based recovery, Oracle will automatically cease the recovery effort.

Although there are only the three incomplete recoveries described that work in conjunction with archived redo logs, other types of database recovery that do not work with archived redo also may be considered "incomplete" because they are only capable of recovering data to a point in time in the past. These types of recovery are database imports and physical recovery based on the full physical offline backups.

Full recovery

A full recovery from complete backup is handled in the following way. The database must first be shut down. The DBA replaces all files in the Oracle database with the backup versions. If need be, the name and/or location of the control file may be changed in the **init.ora** file before mounting the database. At this point, the database can be mounted, but if there was some movement of datafiles or redo log files from one disk to another, the DBA should use the **alter database** statement at this time to ensure Oracle's awareness of the change. Finally, the DBA should open the database using the **resetlogs** option to discard all online redo information.

Recovery of Read only tablespaces

There are special considerations the DBA must make for recovery of **read only** tablespaces. The golden rule for facilitating recovery on read-only tablespaces is to always make a backup of the control file after the status of a tablespace changes from **read only** to read-write, or vice versa. Without this backup of the control file, there are always extra steps to manage the recovery of **read only** tablespaces.

Enabling Archiving

There are several aspects to setting up the database to handle archived redo log information. The first of these items covered is determining whether or not archiving is enabled on the database. One method for handling this step is to look in the V$DATABASE dynamic performance view. Another is to issue the **archive log list** command from within Server Manager. The output from this command not only gives the DBA information about the archiving status, but also gives a great deal of information about the archived redo logs themselves. Finally, if archiving is enabled, the V$LOG view gives information about whether the individual online redo logs have been archived.

Putting the database into **archivelog** mode is perhaps the greatest and simplest step a DBA can take to ensure the recovery of the Oracle database. Archiving is enabled in two ways. First, at the creation of a database with the **create database** statement, the DBA can include the **archivelog** option to begin archiving of redo log entries right away. Alternately, the DBA can issue the **alter database archivelog** statement to change a previously nonarchiving database into one that archives its redo logs. To change the database into **noarchivelog** mode, the DBA can issue the **alter database noarchivelog** statement. It is important that the DBA take a complete backup of the entire database after changing the archiving status of the database to **archivelog** mode in order to ensure that the database archived redo logs are useful in database recovery. Archived redo logs cannot be applied to a database that has been recovered from a backup taken of the database when it was not in **archivelog** mode.

Once the DBA has set the database to **archivelog** mode, she can either handle the archiving of redo logs manually with the **alter system archive log all** statement or by setting Oracle up to handle automatic archiving with the use of the ARCH background process. Automatic archiving is started using the **alter system archive log start** statement. When enabled, the ARCH background process archives redo information automatically every time a log switch occurs, placing them into a destination specified by the value for the LOG_ARCHIVE_DEST in the **init.ora** file. The name ARCH will file the archive under is contingent on the value specified for the LOG_ARCHIVE_FORMAT parameter, also specified in the **init.ora** file.

Viewing logs

To view the archived log list, the DBA may look in the V$LOG_HISTORY performance view. The ARCHIVE_NAME column contains the name assigned to the archived redo log sequence and thread, based on automatic archiving using the LOG_ARCHIVE_FORMAT parameter.

Finally, the DBA can selectively archive redo log information by manually archiving redo with the **alter system archive log** statement. There are several manual options available for selective archiving. They are **seq, change, current, group, logfile**, and **next**. The **seq** option allows the DBA to archive redo logs according to sequence number. Each redo log is assigned a sequence number as LGWR fills the online redo log. The **change** option can be used to archive a redo log that contains a certain SCN. The **current** option archives the redo log that is currently being written by LGWR, which forces Oracle to perform a log switch. The **group** option allows the DBA to specify a redo log group for archiving. The **logfile** option allows the DBA to archive redo logs by named redo log member files. The **next** option allows the DBA to archive redo information based on which redo log is next to be archived. Also, it should be noted that **thread** is also an option for archiving redo logs. A **thread** is a number representing the redo information for a single instance in a multi-instance parallel server setup using Oracle's Parallel Server Option. The **thread** option can be set for any of the options for manually or automatically archiving redo log information using the **alter system archive log** statement.

Supporting 24-hour operation

Today's corporations need more database availability for a global user base and/or a 24-hour workforce. They need databases that are available on a 24-hour basis as well. Supporting organizations with high availability requirements puts intense pressure on the DBA in two ways. First, many of the options presented, such as database export or offline full backups, are not viable strategies for database recovery. The only viable option is the one that takes advantage of archiving--online hot backups. The second factor is the additional pressure on the DBA to recover the database as quickly as possible. Since the database has users at all hours of the day, the DBA has pressure to restore the 24-hour database at all times, day or night.

Online vs. Offline Backups

As a point of fact, an offline database backup is usually just that--a backup of the database. Thus, when the DBA is examining a full database backup, it is usually one that contains all datafiles, redo log files, control files, and parameter files. A full backup is usually taken while the database is offline so that the backup is read consistent to a single point in time.

Online database backups, however, are slightly different. An online backup is usually taken tablespace by tablespace. As such, the online backup usually only consists of the datafiles for each tablespace, rather than all datafiles for all tablespaces. Of course, there is nothing that says the DBA cannot take a backup of all tablespaces one at a time or in parallel on each occasion that online backups are performed.

Online backup methods

In order to take online backups of the database, archiving of redo logs must be used. The database is open, available, and accepting changes during the entire time a hot backup is taking place. In order to guarantee that the changes made to the tablespace while the backup took place are kept, it is required that the DBA archive redo logs that were taken during the operation of hot backups. Prior to taking the hot backup, the DBA should issue the **archive log list** command from Server Manager in order to determine the oldest online redo log sequence that should be saved in conjunction with the online backups being taken. Once the tablespace backups are complete, the **archive log list** command should be issued again, followed by a log switch to ensure that the current redo log entries made for the tablespaces backed up are archived properly for use by the backups should recovery be necessary.

The steps to the process are as follows. Execute **archive log list** from Server Manager. Note the value for "Oldest online log sequence." This is the oldest redo log required for using the online backup. Execute **alter tablespace** name **begin backup** from Server Manager. This

step prepares the tablespace for online backup. Copy the datafiles for that tablespace using operating system commands or third-party products. Be sure the copy resides on a disk other than the production datafiles themselves. Execute **alter tablespace** *name* **end backup** from Server Manager. This step completes the online backup process for that tablespace. Repeat for all tablespaces to be backed up. Execute **archive log list** again from Server Manager. Note the value for "Current log sequence" this time. This is the last redo log required for using the online backup. Issue an **alter system switch logfile** to cause Oracle to create an archive of the current redo log. This archive should then be stored in the LOG_ARCHIVE_DEST area. If desired, copy the archives associated with the backup to tape. Create a copy of the control file. This is done with the **alter database backup controlfile** statement.

Online Backups of the Control file

The final step in the online backup of a database is an important one that merits special consideration. There are two different types of backups of database control files. The first is a backup created of the actual control file, ready for use on the Oracle database. Should the production version of the control file be destroyed, the DBA can simply put the backup in place and move forward. The **alter database backup controlfile to** *filename* statement can be used in this case to ensure that a usable backup copy of the control file is available in the event of an emergency. The other option is to use a special keyword in place of *filename* in the **alter database backup controlfile** statement. This keyword is called **trace**. The use of this keyword allows the DBA to obtain a special backup of the control file. Rather than backing up the control file itself, this statement backs up a copy of the script that can be used to create the control file. If the DBA has space considerations or wishes to create the control file on a database running under a different operating system, this option will provide the backup required.

Complete Recovery with archiving

Complete recovery can be accomplished in two different ways: with offline backups or with online tablespace backups.

Complete Recovery Methodology

To perform complete recovery, there are two things required. The first thing required is the backup. There are two different types of database backups that a DBA might use to execute a database recovery: the offline backup and the online backup. Complete recovery also requires a complete set of archived redo logs from the period of time the backup was taken to the present. In addition, the DBA may require the archived redo log information taken while the online backup was taking place if online backups are used on the database.

The first step required in complete database recovery using offline backups and archived redo logs is to make the database unavailable to the users. This step can be accomplished with the **shutdown abort** statement executed within Server Manager. The database cannot be available to the users during recovery from offline backups because a full database recovery is required to bring the contents of the database up to the point in time the media failure occurred, which requires Oracle to write a lot of redo information. The same condition is not required for database recovery from online backups, however, because database recovery from online backups need only consist of tablespace recovery, which means the rest of the database can be available for use during the period of time the recovery is taking place.

Complete Recovery with Disk Replacement

After making any hardware repairs or replacements necessary, the following steps are necessary make complete recovery. These steps are accomplished only when the disk media that failed is replaced before recovery begins.

1. Restore all files from the complete backup taken, including datafiles, redo log files, and control files. All files should be restored--not just damaged ones--in order for the entire database to be read consistent to a single point in time, unless complete recovery using archive logs will be performed. In this case, only lost files should be recovered. Operating system copy commands are used to handle this step.
2. Recover the database using archived redo log information. The command syntax for this process is the **alter database recover database** statement. This statement will begin the interactive process of applying redo log information:

   ALTER DATABASE RECOVER DATABASE;

3. Take a full backup of the database. Use operating system copy commands for this step.
4. Open the database for use with the **resetlogs** option.

   ALTER DATABASE OPEN RESETLOGS;

Complete Recovery Without Disk Replacement

It may not be possible to secure a repair or replacement for the disk that failed. In these situations, the DBA may need to recover the database by placing files that were on the damaged disk onto other disks. The database must be closed for this process, and the other steps are listed.

1. Restore all files from the complete backup taken, including datafiles, redo log files, and control files. It is important that all files be restored--not just damaged ones--in order for the entire database to be read consistent to a single point in time (the point in time of the last database backup), unless complete recovery using archive logs will be performed. In this case, only lost files should be recovered. Operating system copy commands are used to handle this step for database recovery from a full backup.
2. Move control files to other disks if they were lost in media failure. Change the location of control files as specified in the CONTROL_FILES parameter of the **init.ora** file to reflect a new location for the control file of the Oracle database.
3. Mount the database in **exclusive** mode.
4. Move datafiles and redo log files onto other disks with the **alter database rename file** statement from Server Manager. The full pathnames for the datafile at the old and the new locations should be specified in the statement.

   ALTER DATABASE orgdb01
   RENAME FILE '/u01/oracle/data/data_301a.dbf'
   TO '/u02/oracle/data/data_301b.dbf';

   ALTER DATABASE orgdb01
   RENAME FILE '/u01/oracle/ctl/rdorgdb01'
   TO '/u02/oracle/ctl/rdorgdb01';

5. Back up the database in the same manner as for offline backups.
6. Recover the database using archived redo log information. The command syntax for this process is the **alter database recover database** statement. This statement will begin the interactive process of applying redo log information:

   ALTER DATABASE RECOVER DATABASE;

7. Back up the database.
8. Open the database using the **resetlogs** option. ALTER DATABASE OPEN RESETLOGS;

Tablespace Recovery

Recovery can be performed on individual tablespaces in addition to the entire database. The DBA can execute a tablespace recovery using the **alter database recover tablespace** statement. The database must be open in order to accomplish a tablespace recovery, so that Oracle can view the contents of the database while the tablespace recovery occurs. However, the tablespace itself must be offline. A benefit to tablespace recovery is that the DBA can allow users to access other tablespaces while the offline tablespace is restored. In this example, assume that the USERS01 tablespace needs to be recovered.

1. Open the database if it is not already open.

   ALTER DATABASE OPEN;

2. Take the tablespace on the disk that failed offline.

   ALTER TABLESPACE users01 OFFLINE;

3. Restore damaged datafiles with their respective backup copies using operating system commands, assuming hardware problems have been rectified. If the hardware problem has not been rectified, or if the DBA decides to move the files of the tablespace to another disk, the DBA should issue the appropriate **alter database rename file** command.

   ALTER DATABASE
   RENAME FILE '/u01/oracle/home/data01.dbf'
   TO '/u02/oracle/home/data01.dbf';

4. Recover the tablespace. To minimize the amount of interaction required between DBA and Oracle, the DBA can include the **automatic** keyword in the **alter database** statement, allowing Oracle to automatically apply its own suggestions for redo logs.

   ALTER DATABASE RECOVER AUTOMATIC TABLESPACE users01;

5. Back up the database.
6. Bring the tablespace online using the **alter tablespace online** statement.

   ALTER TABLESPACE users01 ONLINE;

Datafile Recovery

Recovery can be performed on individual datafiles in the tablespace with the **alter database recover datafile** statement. The database and tablespace must be open in order to accomplish datafile recovery so that Oracle can view the contents of the database while the datafile recovery occurs. However, the datafile itself must be offline. A benefit to datafile recovery is that the DBA can allow users to access other tablespaces while the offline datafile is restored. In this example, assume that the **users01a.dbf** datafile needs to be recovered.

1. Open the database if it is not already open.

   ALTER DATABASE OPEN;

2. Take the datafile on the disk that failed offline.

```
ALTER DATABASE DATAFILE 'users01a.dbf' OFFLINE;
```

3. Restore damaged datafiles with their respective backup copies using operating system commands, assuming hardware problems have been rectified. If the hardware problem has not been rectified, or if the DBA decides to move the file to another disk, the DBA should issue the appropriate **alter database rename file** command.

```
ALTER DATABASE
RENAME FILE '/u01/oracle/home/data01a.dbf'
TO '/u02/oracle/home/data01a.dbf';
```

4. Recover the datafile. To minimize the amount of interaction required between DBA and Oracle, the DBA can include the **automatic** keyword in the **alter database** statement, allowing Oracle to automatically apply its own suggestions for redo logs.

```
ALTER DATABASE RECOVER AUTOMATIC DATAFILE 'users01a.dbf';
```

5. Back up the database.
6. Bring the tablespace online using the **alter database datafile online** statement.

```
ALTER DATABASE DATAFILE 'users01a.dbf' ONLINE;
```

Incomplete Recovery Process

Application of archived redo logs is handled interactively by Oracle. The database prompts the DBA to supply the name of the next archive to apply, while also issuing a suggestion for which redo log to apply. This suggestion is based on the V$LOG_HISTORY dynamic performance view, the LOG_ARCHIVE_FORMAT parameter, and the LOG_ARCHIVE_DEST parameter. If the DBA desires, he or she can specify Oracle to apply its own suggestions automatically during database recovery by including the **automatic** keyword in the **alter database recover** statement.

All archived redo information will be applied by Oracle to execute a complete recovery. When Oracle is complete, the DBA can open the database for usage by issuing the **alter database open resetlogs** statement. The **resetlogs** option indicates that the DBA wishes to reset the log sequence number for the database online redo logs to 1. It is generally recommended that prior to opening a newly recovered database, the DBA take a full backup of the recovered database in the event of an emergency.

As a point of reference, to minimize downtime during database recovery, it is faster to use online tablespace backups. This is because tablespace recovery runs faster than full database recovery required for offline backups, and also allows the DBA to leave the unaffected parts of the database available to users while fixing the damaged portion.

Why incomplete recovery may be necessary (Using Archive redo information)

There are several situations that may force the DBA to recover the database to a point in time in the past using incomplete database recovery methods. Some of these reasons include an errant batch process or feed that contains bad data, which causes some improper information to enter the database. Another situation where incomplete recovery may occur simply involves bad luck, where the DBA loses an archived redo log file or finds that the file has been damaged. In any event, incomplete recovery is the recovery of a database to some point in the past.

One of the things that makes incomplete recovery involving archived redo logs better than incomplete recovery involving backups that do not work in conjunction with archiving, or backups on a database that does not use archiving, is choice. The DBA can choose the point in time to which incomplete recovery will occur (barring loss of archived redo logs) when archiving is used. In contrast, the only "point in time" to which a database can be restored is the point in time when the last backup completed.

Requirements for an incomplete recovery

To accomplish incomplete recovery, the DBA requires two things. The first is a complete backup of the system. Incomplete recovery is a complete database recovery operation. Incomplete recovery from online tablespace information, and the tablespace recovery operation, is not allowed. This is because incomplete recovery of an individual tablespace would put the entire database into a read-inconsistent state.

The second item required is archived redo log information. As stated, incomplete recovery may be the result of lost archived redo logs. If, for example, consider that there are three archived redo logs for a database, numbered 1, 2, and 3, and each archive contains information for 30 transactions, whose SCNs are from 0-9, 10-19, and 20-29. If archive sequence 3 is lost, the DBA can only recover the database through SCN 19, or archive sequence 2. If 2 is lost, then the DBA can only recover the database through SCN 9, and if archive sequence 1 is lost, then no archived redo log information can be applied.

Types of incomplete recovery

Those three types are **change-based, time-based, and cancel-based.**

The change-based incomplete recovery allows the DBA to specify a system change number (SCN) of a transaction at which, once its full set of committed operations are applied, Oracle will stop running the database recovery automatically.

The time-based incomplete recovery allows the DBA to specify a date and time later in the past than the database backup, to which the database will be recovered. When Oracle applies redo information to that point in time specified by the time-based recovery, Oracle will automatically stop the recovery process.

The final type of database incomplete recovery is the cancel-based recovery. This recovery allows the DBA to arbitrarily decide when to end the recovery by issuing a **cancel** command when Oracle prompts the DBA to enter the name of the next archived redo log entry to be applied. This option gives the DBA complete control over when to end a database backup, but requires the DBA to interact with Oracle during the recovery at all times.

Steps for Incomplete Recovery

As with complete recovery using offline backups, the DBA cannot allow users to have access to the database while the incomplete recovery is taking place. As such, the database should be mounted in **exclusive** mode to prevent other instances from opening it, but not opened by that instance either. The DBA can then place all datafiles in place on their respective disks from backup and issue the **alter database recover database until** statement. Following the **until** clause, the DBA can specify the exact form of incomplete recovery that will be employed, either using **change** followed by the SCN, a date and time, or simply the **cancel** keyword. If the DBA wants to limit the interactive portion of recovery where Oracle prompts the DBA for names of archived redo log files to apply, the DBA can use the **automatic** keyword in the **alter database recover** statement, except in the case of using the cancel-based recovery option. When using cancel-based recovery, the DBA must interact with Oracle during application of redo logs in order to issue the **cancel** command.

After recovery is complete, it is recommended that the DBA perform a full database backup before allowing the users to access the database. After that, the DBA can allow the users access to the database by executing the **alter database open resetlogs** statement. Using the **resetlogs** keyword means that Oracle will reset the log sequence number to 1, effectively discarding all archived redo information taken before the recovery and underscoring the importance of taking a backup before allowing user access to the database again.

Incomplete Recovery and Re-Creating the Control File

Of particular importance in running incomplete database recovery is the manipulation or creation of a new control file in the event that a new one is required. If the control file is damaged, and there is no backup to it that will allow the DBA to perform complete recovery on the Oracle database, the DBA may be forced to create a new control file as part of the recovery process. This process is accomplished with the use of the **create controlfile** statement. In it, the DBA will identify the location of all redo logs and datafiles, as well as specifying the **resetlogs** option to discard any online redo information (also to make the control file usable for recovery). The DBA should also remember to specify the **archivelog** option because, after all, the database is archiving redo. If the DBA ever used the **alter database backup controlfile to trace** option for control file backup, the script backed up could be modified (if necessary) and executed in lieu of formulating a new one.

Database Recovery Using a Backup Control File

Finally, in the event that the DBA is using an actual backup of the control file and needs to execute recovery from it, the DBA can issue a the **alter database recover** statement using a special clause that tells Oracle to use the backup control file. The full statement would be **alter database** *dbname* **recover database using backup controlfile** *ctlname*.

Database startup after system failure

The database startup after system failure will be faster than a typical database startup. This is due to the use of fast transaction rollback when opening the database in media recovery situations. Database recovery consists of two general steps: rolling forward and rolling back.

The roll-forward process consists of applying all transactions, committed and uncommitted, to the database, while rollback consists of discarding those transactions that were not committed to the database at the time of the failure. Opening the database with fast transaction rollback consists of several items. First, Oracle will not open the database until the roll-forward process completes. However, the database will not wait until the rollback completes. Instead, Oracle will open the database for regular use after the roll-forward takes place, rolling back uncommitted transactions at the same time users are accessing the database objects. A couple of situations may arise from this.

First, if a user process attempts to change a row or table that is involved in the rollback process, the transaction that failed may still hold a lock to that object, forcing the user to wait.

Second, there will be fewer rollback segments available to user processes while the fast transaction rollback takes place, due to the fact that the rollback segments that were being used by transactions that failed in the database failure will be involved in the fast transaction rollback effort. However, fast transaction rollback does allow the DBA to open the database sooner that would otherwise be permitted in a database recovery situation. Thus, downtime can be minimized.

Starting the Database when Missing Database Files

Another area of minimizing downtime comes with the ability to open the database for use when datafiles are damaged or missing as a result of a media failure. In order to do this, the DBA must first mount but not open the database. At this stage, the DBA can take the tablespace containing lost datafiles offline. Then, the DBA can open the database. By opening the database even while parts of it are damaged, the DBA allows users of the database to access undamaged parts of the database while damaged parts are being fixed.

The DBA can then initiate a complete tablespace recovery on the damaged tablespaces with the use of online tablespace backups and a full set of archived redo logs. The types of organizations that benefit most from this sort of recovery are those with multiple applications running on the same database. A media failure in this situation may damage the datafiles associated with only one of the applications, implying that other applications should not have to suffer downtime because of damage to another application. This operation is usually accomplished with the **recover tablespace** option. Recall that the **recover tablespace** option requires a complete recovery to be performed to the point in time of the database failure.

Parallel Recovery

Parallel recovery can improve the performance of a database recovery when two or more disks have been damaged, or a great deal of redo must be applied as part of the recovery.

The parallel feature is incorporated into recovery with the use of the Server Manager **recover database parallel** command. The DBA can issue this statement from Server Manager line mode or using the graphical interface. Parallel recovery requires two clauses to be specified as part of the recovery.

The first is called **degree**. The integer specified for this parameter represents the degree of parallelism for the database recovery, or the number of processes the database will have actively attempting to recover the database.

The second clause is called **instances**. The integer specified for this parameter indicates the number of instances that will accomplish the recovery. Each instance involved in the parallel recovery can have the number of processes dedicated to recovery indicated by degree, so in a sense the **instances** parameter is a multiplier for the **degree** clause.

Because of the multiplier effect, it is important to remember that the number of processes that can be used to handle parallel recovery may not exceed the value set for RECOVERY_PARALLELISM, an initialization parameter set at instance startup in the **init.ora** file. Further, database recovery is more effective when the operating system of the machine hosting the Oracle database supports *synchronous I/O*. Otherwise, there will be limited gains in recovery performance using **recover database parallel.**

Troubleshooting the Oracle Database

Using Trace Files and the Alert Log

The first technique is mastering the use of trace files and the **alert** log for identifying problems with the operation of the database. The Oracle database has several background processes handling various functions for the database, such as the operation of moving redo information or database blocks from and to the disks, various recovery activities, and other operations.

Each of these background processes writes a log of its activities, detailing any errors it may have encountered, when it started running, and other things. These logs are called trace files, and they are stored in a location identified to Oracle by the use of the initialization parameter in **init.ora** called BACKGROUND_DUMP_DEST.

Each of the user processes connecting to the database requires a server process to run Oracle database transactions on its behalf. These processes also create an activity tracing log. This log is stored in another location, specified with the use of the initialization parameter in **init.ora** called USER_DUMP_DEST. A final, and perhaps most important, log mechanism is the overall log of activity for the Oracle database, called the **alert** log. This file contains all error messages encountered by the database in its normal activity, along with startup and shutdown times, archiving information, and information about the startup and shutdown of background processes. If the DBA suspects there is a problem with the operation of the database, the **alert** log should be the first place she looks.

Using the V$ Performance Views to Identify Problems

Another method the DBA can incorporate into the detective work required for the identification of database problems is the use of the dynamic performance views in the Oracle data dictionary. These views track performance information for the database, and their names are usually prefixed with either V$ or X$. Several of these views identify the status for various components of the database, such as the datafiles and redo log file of that database. If there is a problem with the status of a datafile or redo log file arising from the failure of a disk, Oracle will mark the file with the appropriate status in the appropriate dynamic performance view.

There are several performance views involved in the task of backing up and recovering the database. Some of these have already been identified. The two views emphasized in this discussion are the V$DATAFILE and the V$LOGFILE views. Both of these dynamic performance views contain status information for each of the files they represent. These files are datafiles and redo log files, respectively. If the DBA should find a datafile with a status of RECOVER or a redo log file with a status of INVALID, the DBA may want to investigate a problem with accessing the disk containing the datafile and/or log file using operating system means. These methods may be used to check the status of the database for recovery issues.

Detecting Corruption in Online Redo Logs

One particular issue for DBAs that may arise in the need to identify damage to the database is the need for verification mechanisms of the database. There are a few mechanisms available for the DBA to do just that. The first pertains to the verification of operation on the online redo logs. There is a feature that will verify the blocks of an online redo log before archiving, or before applying an archived redo log as part of database recovery, to prevent the propagation of corrupted data blocks in the backup and recovery of an Oracle database. To use this feature, the DBA needs to set the LOG_ARCHIVE_CHECKSUM initialization parameter to TRUE.

Clearing Corruption in Online Redo Logs

If the DBA is using archive checksums to confirm the integrity of the blocks in an online redo log, the following process will occur. When Oracle reaches a log switch, it will check the online redo log for corruption in the data blocks of the online redo log to archive. If Oracle finds corruption in an online redo log, it will try to write archive information using a different redo log file. If all members contain the corruption in the same data block, Oracle will not be able to archive that redo log and archiving will stop. At this point, the DBA must intervene in the archiving process. The redo log containing the corrupt data block will need to be cleared using the **alter database clear unarchived logfile group** statement. Since the log has not been archived, the DBA will also need to include the **unarchived** option as described above. Any redo log group can be cleared, depending on the desires of the DBA.

For example, a redo log can be cleared after archiving by using the statement identified above for clearing online redo logs, but since the statement has been archived, the **unarchived** option can be eliminated. However, once a redo log is cleared, the DBA no longer has a complete set of archived redo logs for database recovery. Unless the DBA backs up the database at this point, the DBA will only be able to execute incomplete recovery from database failure. The DBA should be sure to back up the database in the event of clearing online redo logs.

The DBVERIFY Utility

Another tool available for the DBA to use in verification of other types of files such as datafiles is the DBVERIFY utility. This utility will take an offline datafile and inspect it for block corruption. DBVERIFY is a stand-alone utility that operates in the same way as other utilities discussed in Oracle. To run it, the DBA supplies a set of parameters. Some parameters the DBA can identify for running DBVERIFY are listed: FILE, START, END, BLOCKSIZE, LOGFILE, FEEDBACK, HELP, and PARFILE.

The FILE parameter is used by the DBA to name the file that DBVERIFY will operate on, while START and END are used to tell DBVERIFY where in the file to start and end the verification. The defaults are the beginning and end of the file, respectively.

The BLOCKSIZE parameter specifies the size of Oracle blocks in the database, in bytes.

The LOGFILE parameter names a file to which the output of DBVERIFY will be written. If no log file is specified, DBVERIFY writes its output to the screen.

The FEEDBACK parameter can be assigned an integer that specifies how many pages of the datafile will be read before DBVERIFY puts some notification of its progress on the screen.

Setting the HELP parameter to 'Y' will cause DBVERIFY to display information the DBA can use for setting other parameters.

Finally, the PARFILE parameter can be used to name a parameter file containing values for other parameters used in DBVERIFY. Due to the fact that DBVERIFY identifies problems involving database corruption, use of it may be best undertaken with the guidance of Oracle Worldwide Support.

Standby Database Feature

DBAs who need to provide a high degree of availability for the users of a database may incorporate a standby database into the overall backup and recovery strategy. A standby database is an identical twin database for some other database, and it can be used in the event of a disaster. In order to use the standby database feature of Oracle, the DBA must use archiving in both databases. The reason archiving is necessary is because the standby database is updated with data changes on the primary database by applying the archive logs generated by the primary database.

Standby Database Configuration

To create a standby database, the DBA must execute the following steps. First, the DBA should acquire a machine to host the standby database that is identical to the machine hosting the primary database. Next, the DBA needs to take a complete backup of the primary database. After that, the DBA must create a special control file for the standby database using the **alter database create standby controlfile as** *'filename'.* Finally, the DBA should archive the current set of redo logs using the **alter system archive log current** statement.

With the archived redo logs, datafile backups, and standby control file, the DBA can create a baseline standby database. After creating the standby database, it must be perpetually standing by for database recovery in order to keep the standby database current with the changes made in the primary database. To do so, the DBA should use the **startup nomount** statement in Server Manager, followed by the **recover standby database** statement. Archive logs generated by the principle database should be placed in the location specified by LOG_ARCHIVE_DEST on the standby database, or, alternately, the **from** clause can be used in conjunction with the **recover standby database** statement. Ideally, the process of moving archive logs from the primary database to the standby will be automated in order to keep the standby database as current with changes in the primary database as possible.

Deployment of the Standby Database & Switching Back to the Principle Database

When disaster strikes the production database, the DBA must do the following to get users on the standby database as quickly as possible. First, the application of all archived redo logs on the standby database must complete as quickly as possible. Next, the DBA should shut down the standby database and restart it using **startup mount**. From there, the DBA should execute a **recover database** statement, being sure to omit the **standby** clause. As part of this recovery, the DBA should try to apply the current online redo logs on the production database to the standby database in order to capture all transaction information up to the moment of failure on the other database. After recovery of the standby database is complete, the DBA can execute the **alter database activate standby database** statement. From this point on, the standby database is the production database. Switching back to the original production database, then, will not happen unless the production database is made into the standby of the new production database and the new production database fails.

Standby databases can offer enormous benefit to the organization requiring 24?7 database availability with little or no option for downtime. However, standby databases are a costly solution to database recovery. The costs of a standby database are usually twice the price of the production database hardware and software, plus added expenses for maintaining both databases. Of course, the price of downtime per minute on a mission-critical database system can be far, far higher, thereby justifying the additional costs.

Good luck….