ORACLE **Technical Report**

Prevention, Detection and Repair of Database Corruption

Janani Mahalingam

Center of Expertise
Oracle Support Services
Oracle Corporation

Prevention, Detection and Repair of Database Corruption

Introduction

This paper is targeted towards customers who want to get more information about corruption issues in an Oracle database environment.

Corruption is a generic term that covers many types of problems and is used in different contexts by different people. To help you understand Oracle's concept of corruption, we should first present our definition of the term. **Corruption can be defined as an inconsistency in the data block structures or memory structures as a result of disparate problems.** Corruption can be the result of human error (including software, firmware, and hardware bugs) or the environment (component failure). The goal of this paper is to give readers an understanding of the common causes of corruption and assist them in developing better methods of prevention, detection and repair.

This paper is organized along the following lines. First we will define the terms prevention, detection, and repair. We then provide details about three common types of corruption and present detailed methods to prevent, detect, and repair them.

We will also briefly discuss the infamous internal ORA-600 errors. The misconception of equating ORA-600s to corruption will be clarified.

At the end of the paper is a table summarizing the places where corruption can occur and the best practices that can be performed to prevent it from happening, detect it quickly if it does occur, and repair or recover from it in minimal time. We also provide a glossary for some of the terminology used.

Defining Prevention, Detection, and Repair

Prevention

Prevention can be defined as a method to anticipate a problem and stop it from happening. In the context of this paper, prevention can be defined as a set of steps that can be taken to anticipate and stop corruption from happening in a production environment. With recent releases of Oracle, we believe(?) the most common cause of corruption is user error. While we cannot completely prevent corruption, the risk of corrupting the production environments can be greatly reduced by improving the operational infrastructure.

Examples of prevention by improving operational infrastructure include the following:

- Having redundant hardware such as mirrored disks.
- Defining and enforcing strict security procedures.
- Strict change control with proper testing practices.

A meticulous operational infrastructure should be able to catch most software, hardware, and business process bugs before changes are made to the production environment. Other than that, you can also design applications with automatic error detection and correction. We will discuss this in more detail later.

Detection

Detection can be defined as a method to determine the existence of a problem. In the context of our paper, detection can be defined as the ways to discover the presence of a corruption in the database. In particular, detecting a corruption must encompass detecting anomalies in the hardware, disks, memory, database and application data. The following are some examples of detecting corruption which will be discussed in detail in later sections of the paper:

- Monitoring the System logs
- Monitoring the Oracle trace files
- Monitoring any application logs
- Monitor audit trails for any security violations

Repair

Repair can be defined as a method of restoring the system to a healthy state. In the context of this paper, repair can be defined as the steps taken to bring a database back to a consistent state. Interestingly, the way Oracle software defines a consistent state may differ from what the business needs are. For example, in some environments, some data loss is acceptable. The method of repair will vary due to the extent of the corruption and the business needs. For this reason, the scope of this paper will focus on quickly repairing damage to Oracle database objects.

Types of Corruption

In this section, we will further define corruption by discussing the various types that can occur in a an Oracle environment. Physical or structural corruption can be defined as damage to internal data structures which do not allow Oracle software to find user data within the database. Logical corruption involves Oracle being able to find the data, but the data values are incorrect as far as the end user is concerned.

Physical corruption due to hardware or software can occur in two general places -- in memory (including various IO buffers and the Oracle buffer cache) or on disk. Operator error such as overwriting a file can also be defined as a physical corruption. Logical corruption on the other hand is usually due to end-user error or non-robust(?) application design. A small physical corruption such as a single bit flip may be mistaken for a logical error. For purposes of the paper we will categorize corruption under three general areas and give best practices for prevention, detection and repair for each:

- Memory corruption
- Media corruption
- Logical corruption

Memory Corruption

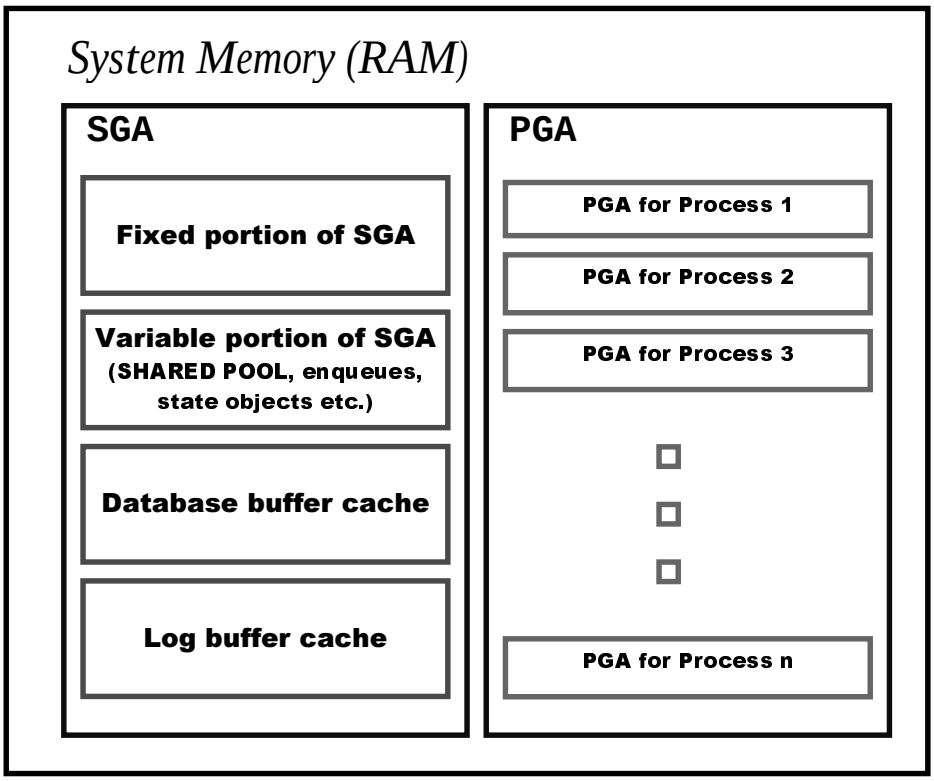
Background

This section starts by describing the Oracle RDBMS memory structures and how they are managed.

Oracle allocates both shared and private memory. Shared memory is allocated when an Oracle instance starts and all processes (or threads) connecting to an Oracle database can access it. Oracle software defines how this shared resource is accessed to prevent multiple processes from simultaneously writing to the same address. It also has to recover any incomplete changes made to memory by a process that dies abnormally. The amount of shared memory allocated is static in size and is only freed when the instance is shut down. Private memory is allocated and freed as needed by each process (or thread) at the OS level.

Corruption is more likely to occur within shared memory than private memory so we focus attention to the structures and algorithms used within shared memory (also known as the Oracle SGA).

The SGA is divided into four portions - fixed, variable, Database Buffer cache, and Redo log buffer. A diagram appears below.



Definition

Memory corruption can be defined as inconsistencies in the data structures that are related to handling memory. This inconsistency could appear in any of the different parts of memory discussed above. As discussed in the section above, corruption in memory can be caused either in the SGA or the PGA.

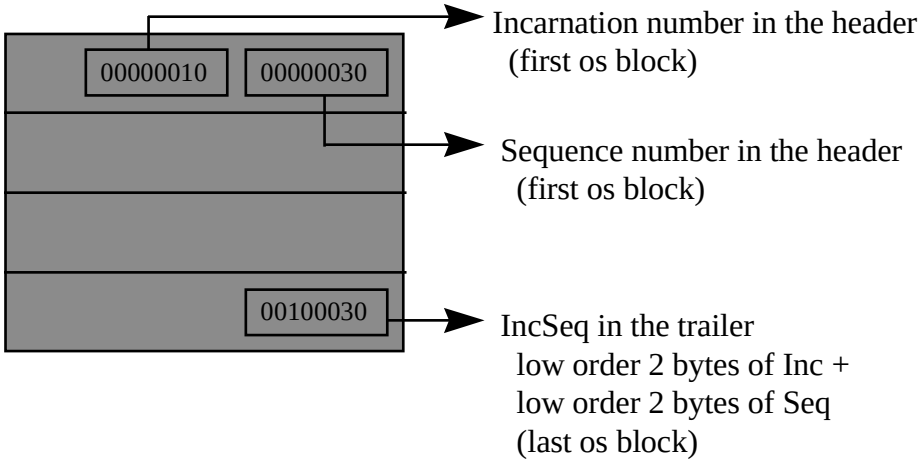
Only corruption in the database buffer cache portion of the SGA can potentially lead to data loss. This is termed as 'Cache corruption' and is discussed in detail below. Corruption in the other parts of SGA do not result in loss of data, but can still cause the instance to crash. On the other hand, a corruption in the PGA causes only the corresponding process to crash. If this process is updating a block in the buffer cache when this happens, then the background process, PMON does the necessary recovery on the block being changed by this process. SMON and other processes will rollback any other uncommitted data.

Cache corruption

The Oracle buffer cache is a mechanism where frequently accessed blocks are stored in memory for quicker access. The cache also maintains older versions of blocks for consistent read purposes. If there is a corruption in this part of memory then there is a possibility of loss of data.

The foreground processes read Oracle blocks from the disk into the buffer cache. There are certain checks done on the data block when it is read from the disk. For example, one of the checks is to compare the Incarnation Number (INC) and Sequence Number (SEQ) data structures from the header of the data block with the INCSEQ structure in the footer to make sure

that the block versions match. This is done to avoid reading a block from disk whose header is corrupt.



The structures discussed above is specific to version 7.

Causes

In general, memory corruption is usually identified by a background or shadow process whenever it tries to access the part of memory that is corrupted. Cache corruption, which might cause a loss of data, is usually caught by (1) the shadow process when trying to read or update a block in memory, (2) by the background process, DBWR, when trying to write a dirtied block from memory to disk, or (3) by the background process, PMON, while performing block recovery. Common causes include:

- Operating system bugs causing bad reads or bad writes
- Hardware issues
- Oracle bug
- Non-Oracle program attaching and illegally writing to the same shared memory address

Prevention

There is very little that can be done from the user's perspective to prevent memory corruption. An INIT.ORA parameter, DB_BLOCK_CHECKSUM can be set to true. This enables another check where the block checksum is calculated and compared with the checksum stored in the block header. If they are different then the block is considered corrupt on disk, and the block is not read into memory. This prevents corrupt blocks from entering the cache. When a block is changed and being written to disk from memory, DBWR calculates a new checksum for the block by summing up its contents and writes it in the block header. There is a CPU overhead associated with this parameter since the checksum is calculated every time a block is read and written.

A similar parameter, LOG_BLOCK_CHECKSUM, can be set to true in the INIT.ORA for verifying the records written to the redo log buffer. This extra test prevents bad redo from being applied to a block in cache during recovery. There is CPU overhead for reasons similar to the DB_BLOCK_CHECKSUM.

The best form of corruption prevention from occurring in a production environment is to test the hardware, operating system, database, application and so on for bugs before rolling them into production. This is also true when introducing new hardware, patches and upgrades.

Detection

Memory corruption can be detected from errors in the operating system logs indicating any kind of memory problems. These can also be found from the Oracle ALERT.LOG by certain ORA-600 errors. The first argument of the ORA-600 could be from 17000 through 17999 in case of memory corruption. Some cache corruption can be detected by ORA-600s in the range from 2000 through 8000. One thing to remember is that not all ORA-600s imply memory corruption.

Repair

Location	Data loss	Repair
SGA - Buffer cache	Probable data loss	⇒ If the corrupt block has been written to disk, then the object to which the block belongs to has to be repaired by methods discussed under Media corruption. ⇒ SHUTDOWN and STARTUP the instance ⇒ To diagnose the causes for the corruption, call Oracle support with appropriate files listed below the table.
SGA - Redo buffer	No data loss	⇒ SHUTDOWN and STARTUP the instance. ⇒ To diagnose the causes for the corruption, call Oracle support with appropriate files listed below the table
SGA - Shared SQL Area	No data loss	⇒ SHUTDOWN and STARTUP the instance. ⇒ To diagnose the causes for the corruption, call Oracle support with appropriate files listed below the table
PGA	No data loss	⇒ ⇒ ⇒ To diagnose the causes for the corruption, call Oracle support with appropriate files listed below the table

To diagnose the cause of the corruption, call Oracle support with the following information:

- ◊ INIT.ORA file
- ◊ ALERT.LOG
- ◊ Trace files for any ORA-600s found in the directory specified by the INIT.ORA parameter, USER_DUMP_DEST
- ◊ Heap dump : this can be obtained by executing the following command in SVRMGR or SQLDBA:
  ALTER SESSION SET EVENTS ‘IMMEDIATE TRACE NAME HEAPDUMP, LEVEL 10’;
  This creates a trace file in the USER_DUMP_DEST.
- ◊ Reproducible test case

- ◊ Thorough history of events that led to the corruption
- ◊ Record any noticeable changes to the environment such as new INIT.ORA parameters, new code, patches, and upgrades.

Best Practices

To sum up this section, some best practices can be listed as:

- ♦ Monitoring scripts written to monitor system logs for any kind of problem with the hardware, disks, controllers, firmware, etc.. If a problem is detected early enough, there is possibility of avoiding corruption from occurring by offlining the files that will be affected.
- ♦ Test any new application or new code in a test environment before introducing into production
- ♦ Capacity planning (sometimes bugs manifest themselves only beyondcertain thresholds).

Logical Corruption

Logical Corruption can be defined as a situation where the actual data is not corrupted in a data block but a query results in a wrong set of data due to a problem in the way data was loaded into the database or due to a misexecution of the optimizer path. A logical corruption is not an inconsistency in a data block but an inconsistency in the result of a query.

For example, a query that is expected to fetch 5 rows might result in 10 rows because the data in the table was duplicated due to the lack of a primary or unique key.

Causes

- Bad application design that lacks validation or proper integrity checks
- Optimizer Bugs (in rare cases)

Prevention

The only kind of prevention methodology for avoiding user errors is to test that applications return valid results thoroughly before implementing them in production environments. There cannot be a way to prevent logical corruption caused due to optimizer problems unless the appropriate patches are applied or the database is upgraded to the most recent version.

Detection

When logical corruption is caused by user errors, they are more difficult to detect than those caused by optimizer problems. In case of user errors, the user should have a good knowledge of the application that is being run to identify inconsistencies in the query results. In the case of optimizer problems, inconsistencies may be accompanied by changes in the query execution plan leading to a different response time. Also one may notice, invalid results only when using the cost based optimizer rather than the rule based optimizer. . Other changes in the query path can be caused by changing the optimizer mode, specifying a different optimizer hint,, dropping or creating indexes, or analyzing objects to generate new statistics.

Repair

When these problems are caused by user errors, they can be fixed by making the application more robust. For example, if invalid data is successfully inserted into a table because of a lack of integrity checking , then constraints should be created and the invalid data will have to be found and deleted. In case of optimizer problems, contact Worldwide Customer Support Services to determine if this is a bug with an available patch. They will need to be provided with the query, EXPLAIN PLAN and potentially the export dump of the tables involved.

Best Practices

To sum up the section, some best practices can be defined as:

- Test new applications in a test environment before introducing into production
- Capacity planning
- Run the query with different optimizer modes set
- Use different indexes

Media Corruption

Media corruption can be defined as a situation where an inconsistency has occurred in the data block structures in the physical disk as a result of media failures. Media failures are failures that are caused due to hardware problems, operating system problems, controller problems, logical volume problems, and so on. As a result of a media corruption, the data in the corrupted block is lost.

Media Corruption could occur in different parts of the database and the detection, prevention and repair are different depending on the object that is corrupted. The following are different objects that could be corrupted:

- Control file
- Redo log file
- Data file
  The block could belong to one of the following categories in case of data file corruption:
  - ⇒ File header block
  - ⇒ Data dictionary object (SYSTEM tablespace)
  - ⇒ Undo header and Undo blocks (ROLLBACK tablespace)
  - ⇒ Sort blocks (TEMP tablespace)
  - ⇒ Data/Index blocks (DATA/INDEX tablespace)
    - ◊ Tables
    - ◊ Clusters
    - ◊ Indexes

Let us look into the causes, detection, prevention and repair for each of the above corruption problems.

Control file corruption

The Control file is the file that has the structural information of the database. The control file has information such as the database name, names and locations of the data files and the redo log files belonging to the database, the creation timestamp of the database, log sequence information, checkpoint information and so on. The control file can be dumped in ascii format by executing the command:

```
ALTER SESSION SET EVENTS 'IMMEDIATE TRACE NAME CONTROLF LEVEL 10';
```

This command creates a trace file in the location defined by the INIT.ORA parameter, USER_DUMP_DEST. When any of the mirrors (Oracle's multiplexing) of the control file is corrupted, critical information about the database cannot be accessed which will result in a database crash. The following tables explain the potential causes, detection, prevention and repair of a corruption in the current control file.

Causes and Prevention

Potential causes	***Prevention***
• User errors - users overwriting current control files by mistake • Hardware problems - disk problems • Operating system bugs • Controller issues • Oracle bugs	• Security - files belonging to the database should not be accessible by users • Hardware mirroring • Multiple controllers • Backups of control file taken frequently • Control file create scripts created with ALTER DATABASE BACKUP CONTROLFILE TO TRACE command taken frequently and after any structural changes done to the database • Automated monitoring scripts written and maintained properly to monitor the appropriate log files (system logs, network logs and so) and to detect any hardware and operating system problems early enough to halt any corruption that could occur in the database

Detection and Repair

Detection	***Repair***
• Database crash - usually, the database crashes when it detects a corruption or loss in any of the mirrors (Oracle's multiplexing) of the control file	• If the control file has been mirrored then: ⇒ SHUTDOWN ABORT ⇒ delete reference to the damaged or missing control file in he CONTROL_FILE init.ora parameter ⇒ STARTUP NORMAL ⇒ If the instance does not come up, try removing another controlfile reference until only good controlfile(s) remain • If there are no good mirrors available and a CREATE CONTROLFILE script is available: ⇒ edit the CREATE CONTROLFILE script and make sure that it contains all the structural changes done to the database (add/drop tablespace, add/drop data file and so on) ⇒ run the CREATE CONTROLFILE script and create a new control file • If the script is not available: ⇒ Gather the names of all the data files and log files in the database with the complete path and create a new control file with the CREATE CONTROLFILE command documented in the Server SQL Reference

Redo log file corruption

Redo log files are critical in protecting a database from failures. The changes made to data in a database are recorded in the redo log files. When there is a failure which prevents data from being written to the data files on disk from memory, the changes can be obtained from the redo log files. When a data file is corrupted, a backup from an earlier day can be restored and the changes from that day onwards can be applied to the data file from the redo log files. The redo log files are used only when a database is recovered from a failure. There are two kinds of redo log files: Online redo log files and Archived redo log files. Archived redo log files are the spooled copies of the online redo log files. Corruption could occur in an online redo log file or an archived redo log file. If a corruption occurs on an online redo log file, the instance crashes if there are no mirrors of the redo log file or if the mirrors are corrupted as well. If the corruption occurs in an archived redo log file, the database is not affected unless a backup is restored and recovery is being done for which it is needed.

Causes and Prevention

Potential causes	***Prevention***
• User errors - users overwriting online redo log files by mistake • Copying or Backing up an incomplete archive log • Backup errors (ftp-ing in ASCII mode and so on) • Hardware problems - disk problems • Operating system bugs • Controller issues • Media backup errors • Tape corruption • Oracle bugs	• Security - files belonging to the database should not be accessible by users • Oracle's multiplexing of online redo logs across different disks • Hardware mirroring • Redundant controllers • Automated monitoring scripts written and maintained properly to monitor the appropriate log files (system logs, network logs and so on) and to detect any hardware and operating system problems early enough to halt any corruption that could occur in the database

Detection and Repair

Detection	***Repair***
• ALERT.LOG should be monitored frequently to detect any kind of ORA errors • The BACKGROUND_DUMP_DEST and the USER_DUMP_DEST should be monitored frequently for any trace files that are created • The database crashes if there is corruption in the current online redo log • Recovery fails with errors	• If the corruption is on an online redo log file, then the mirrored copy can be used if it exists. • If the mirror does not exist or is corrupted, then the file can be cleared by issuing an ALTER DATABASE CLEAR LOGFILE command if it is not the current online redo log • A good backup can be restored and recovery done until a point in time before the corrupted log • If all else fails, an undocumented parameter, _ALLOW_RESETLOGS_CORRUPTION can be set to TRUE in the INIT.ORA using which the database can be forced to be started without applying changes from the online redo logs. Once the database is up with this parameter set, consistency of the data and data dictionary is no longer guaranteed. The database must be rebuilt by exporting the full database, creating a new database and importing into it. (Read paragraph below about the advantages and disadvantages of this parameter).

Forcing the database to startup using _ALLOW_RESETLOGS_CORRUPTION:

This parameter is undocumented and unsupported. The_allow_resetlogs_corruption should only be done as a last resort. Usually when a database is opened with either the RESETLOGS or NORESETLOGS option, the status and checkpoint structures in all the file headers of all data files are checked to make sure that they are consistent. Once this is checked, the redo logs are zeroed out in case of RESETLOGS. When the _ALLOW_RESETLOGS_CORRUPTION parameter is set, the file header checks are bypassed. This means that we do not make sure that the files are consistent and open the database. It will potentially cause some lost data and lost data integrity. The database should be rebuilt since data and the data dictionary could be corrupt in ways that are not immediately detectable. This could lead to future outages or put the database in a state where it cannot be salvaged at all. There is no guarantee that this will work.

Advantages

- In some circumstances, database can be brought up even when no valid backups are available

Disadvantages

- The database will not be in a consistent state once the undocumented parameter is used and so it has to be rebuilt by doing a full database export, recreate the database and a full database import.
- Not guaranteed to work

Data file corruption

The data files are the physical storage unit of data stored in a database. Each data file is made up of data blocks which can be divided into 5 different types (in the context of media corruption):

- File header blocks : File header block is the first Oracle block in every data file in an Oracle database. This block keeps track of a lot of information about the data file that it belongs to.
- Data Dictionary blocks : SYSTEM tablespace consists of data dictionary objects. Data dictionary objects are objects that keep track of information stored in the database such as the information about the tablespaces, information about the data files, information of amount of free space in each tablespace.
- Undo header and Undo blocks: ROLLBACK tablespace consists of rollback segments that are made up of undo header blocks and undo blocks. These blocks are used to undo a transaction when it fails or when the user executes a ROLLBACK command.
- Sort blocks: TEMP tablespace consists of temp segments that are made up of sort blocks. Sorts are usually done in memory where they are allocated a size specified by the INIT.ORA parameter, SORT_AREA_SIZE amount of space. If the sort is so huge that it cannot fit in the allocated space in memory, then temp segments are created in the user's temporary tablespace for doing the sort on disk.
- Data blocks: DATA tablespace consists of tables, clusters and indexes.

The following tables explain the potential causes of data file corruption and also the different ways to prevent data file corruption.

Causes and Prevention

Potential causes	***Prevention***
• User errors - a user simply overwrote a file or removed a file • Hardware issues - Disk problems • Controller issues • Operating system bugs • LVM (Logical Volume Manager) bugs • Oracle bugs • Issues with upgrades of hardware, operating system, database, application, etc.	• Security - Proper privileges have to be maintained to make sure that users can copy files belonging to a live database. • DBAs should make sure that the backup set that they are trying to restore is valid • Monitoring scripts should be written to automate the process of monitoring system logs and network logs for any kind of hardware or disk problems early enough so that the database could be saved from being corrupted by offlining the appropriate data files or shutting down the instance if needed • Any upgrades of hardware, operating system or database should be done on a test environment before upgrading the production. Any corruption due to upgrades can thus be detected before it is seen in the production database.

Now that we have discussed the causes and the prevention, let us discuss the detection and repair which is different for corruption in different blocks.

File header block

File header block is the first Oracle block in a data file. This block keeps track of information about the data file itself (i.e., file metadata) including different checkpoint structures (explained below), status of the file (e.g., hot backup in progress, hot backup ended, media recovery required, instance recovery required), and resetlogs information (information on the time when the database has been brought with resetlogs option most recently).

Checkpointing is the process of writing the blocks that have been changed in memory to disk. The control file and the data file headers are updated after every checkpoint is done in the database. This is an automatic action executed by the background process called the Database Writer (DBWR). This can also be forced by certain user commands such as a normal SHUTDOWN, a normal OFFLINE of a tablespace and so on. The data file header block keeps track of the most recent checkpoint information which denotes that all blocks in this data file that has been changed before this checkpoint has been written to disk from memory and so a failure in the memory will not affect the data in this data file before the checkpoint.

When a file header block is corrupted, the information stored in this block cannot be accessed which means that the objects in the data file cannot be accessed either. The following tables give an idea of the potential causes for a corruption in a file header block and the prevention, detection and repair of the same.

Detection and Repair

Detection	***Repair***
• DBVERIFY can be run to detect any inconsistency due to lack of integrity of file header blocks • ALERT.LOG should be monitored frequently to detect any kind of ORA errors • The BACKGROUND_DUMP_DEST and the USER_DUMP_DEST should be monitored frequently for any trace files that are created • ALTER SYSTEM CHECK DATAFILES can be run to verify access to data files. If verification fails, a message is written in the ALERT.LOG. Any hardware problems should be fixed and the data files should be verified again before the instance could be brought up. • When a file header block is corrupted, the file is offlined. The appropriate V$ views can be queried to detect the corrupted header such as V$RECOVER_FILE • Usually, a corruption in the file header block results in an ORA-1578 which gives the file number and block number of the corrupted block. Since it is the file header, it is the first block of the file.	• Restore from a valid backup and recover

Data dictionary object (SYSTEM tablespace)

Data dictionary objects are objects that reside in the SYSTEM tablespace which has critical information about the objects in the database and their relationships and attributes.

Data dictionary objects found in the system tablespace are described below:

1) Objects created by the script SQL.BSQ run by the CREATE DATABASE command found under $ORACLE_HOME/dbs directory. There are certain tables, indexes and clusters that cannot be dropped and recreated in the SYSTEM tablespace. These are the tables that are used when bringing up the database. They are called the bootstrap objects and are found in the SQL.BSQ. It is not an easy task to locate data dictionary objects that can be dropped and recreated since the relations between the different objects could result in an inconsistent database if we drop the wrong object.
2) Views created on the fixed data structures (V$ views).
3) SYSTEM rollback segment created after the database creation. If the corruption is in this segment, then the most recent backup should be restored and a point in time recovery should be done on the database up to the time when the corruption could have occurred.
4) Compatibility segment (this is the only segment of type 'CACHE' in the SYSTEM tablespace). The Compatibility segment is a segment that keeps track of the features being used in the database which will be used when the database is being downgraded to an earlier version. This segment is used to make sure that the features being used in the current version are disabled before being downgraded to the earlier version. If the compatibility segment has a

corruption then, the database can be brought up by shutting down and starting it up. If the problem is still not fixed call Oracle support with appropriate trace files and the alert.log.

The supported way of fixing data dictionary corruption is to restore from a backup and roll forward using the archived redo logs.

Detection and Repair

Detection	***Repair***
• DBVERIFY can be run to detect any inconsistency due to lack of integrity of data blocks in the system data files • ALERT.LOG should be monitored frequently to detect any kind of ORA errors • The BACKGROUND_DUMP_DEST and the USER_DUMP_DEST should be monitored frequently for any trace files that are created • An ORA-1578 on the data files belonging to the SYSTEM tablespace	• Restore from a valid backup and recover • Disaster recovery should begin

Undo header and Undo blocks (ROLLBACK tablespace)

Rollback segments are undo segments that have information about the transaction that has been executed so that it can be rolled back in case of failure of the transaction or when the user asks for the transaction to be rolled back explicitly. They are made up of undo header blocks and undo blocks which are required to access undo information to provide for consistent reads and transaction consistency. If the rollback segment is corrupted, the transaction consistency of the data blocks (including data dictionary objects) can be jeopardized.

Detection and Repair

Detection	***Repair***
• ORA-1545 on the corrupted rollback segment if it is not offline • ORA-1578 on a query that needs an older version of the block from the rollback segment • ORA-600 with first argument in the range between 4000 and 5000 • DBVERIFY errors on the data file belonging to the ROLLBACK tablespace • Errors in ALERT.LOG • Trace files in BACKGROUND_DUMP_DEST or USER_DUMP_DEST • Check V$ROLLSTAT - the rollback segment has a status of NEEDS RECOVERY.	• Try to offline the rollback segment and drop it. • If it is not possible, restore a valid backup and roll forward • If the window to recover is too small, there are some unsupported ways to fix the problem which may work. These are explained in the following paragraphs

The corruption found when undoing a transaction could fall under three categories

- ⇒ belongs to the object (table/index/cluster) that has the data on which the transaction was executed (data block)
- ⇒ belongs to the undo block that is being used to undo the transaction (undo header)
- ⇒ belongs to the undo segment header block of the segment where the undo block is found (undo block)

Case 1:

The first case where the corruption is in the **object to which the active transaction belongs** to, we have to identify the object first. This can be done by setting an event in the INIT.ORA as follows:

event = “10015 trace name context forever, level 10”

This event traces the undo segment recovery when the database is started. It fails to bring up the database in pre-7.3 and succeeds to bring up the database in 7.3 since we do not do the transaction recovery during instance recovery in 7.3. In either case, the event puts out a trace in the directory specified by the INIT.ORA parameter, USER_DUMP_DEST. This trace file contains a transaction table for each of the rollback segments that are onlined in the database. The trace file has a message that says ‘error recovering tx(#, #) object #’. Tx(#, #) refers to the transaction information and the object # is the object id of the object that has a corruption. The following query gives the name of the object that is corrupted:

```
SELECT OWNER, OBJECT_NAME, OBJECT_TYPE, STATUS
  FROM SYS.DBA_OBJECTS WHERE OBJECT_ID = <object #_from_tracefile>;
```

The database has to be brought up to be able to query for the object. In 7.3, the database can be brought up without going through the transaction recovery. But for databases that have a version earlier than 7.3, the rollback segments have to be forcefully offlined. This introduces an **undocumented** and **unsupported** parameter, _OFFLINE_ROLLBACK_SEGMENTS, in the INIT.ORA file. Once the rollback segment is specified in this parameter, the database can be

brought up. We should not drop the rollback segment once the database is up. If we do so, we are forcing the active transactions in the rollback segment to be committed and so we might have to rebuild the database. Instead, once the database is brought up, the query is run on the DBA_OBJECTS to find the object name. The only way to fix the problem now is to drop this object. Once the object is dropped, the rollback segment is released and can be dropped.

One thing to remember now is that we should remove the undocumented parameters from the INIT.ORA file.

Advantages	***Disadvantages***
• Better than restoring a complete backup and recover • Better than recreating the whole database • No loss of data if the dropped object can be easily recovered or recreated	• Not supported • Need to recreate object and any dependencies • Difficult to identify all objects • Not guaranteed to work

Case 2 and 3: If the corruption is in the undo header block or the undo block then the object # from the trace file created with the 10015 event (explained in Case 1) points to the same rollback segment. This could be possible when the transaction is actually following a chain of undo to get a consistent read version of a block. In this case, the only option is to list the rollback segment in another **undocumented** and **unsupported** parameter, _CORRUPTED_ROLLBACK_SEGMENTS and **rebuild** the database.

One thing to remember now is that we should remove the undocumented parameters from the INIT.ORA file.

Advantages	***Disadvantages***
• If valid backups are not available, then this is a good workaround	• Not supported. • Future corruption is likely to occur • Not guaranteed to work at all times • Need to rebuild the database ⇒ full database export ⇒ CREATE DATABASE ⇒ full database import

In any of the cases mentioned above, if a valid backup is available and the database is in ARCHIVELOG mode then it is always better to go to the backup than trying to bring up the database in the unsupported ways.

Sort blocks (TEMP tablespace)

Sorts are usually done in the part of memory allocated from the SGA. This is defined by the INIT.ORA parameter SORT_AREA_SIZE. If the sort space needed for a sort is so big that it cannot fit in the sort area defined in memory, then it is done on disk by creating segments called

the Temporary segments. It is advisable to create a separate tablespace called the TEMP tablespace. After creating this tablespace, alter the users to use this as their temporary tablespace by executing the following command:

ALTER USER user_name TEMPORARY TABLESPACE TEMP;

This way, the temporary segments created by any user will be in the TEMP tablespace and it provides easy manageability.

Detection	***Repair***
• Usually this segment is never corrupted since they are reformatted every time they get used	• Temp segments are reused frequently • If problem persists, either move or drop and recreate the temp tablespace

Data/Index blocks (DATA/INDEX tablespace)

When a data block is corrupted, when it belongs to a table segment, cluster segment or an index segment, the detection mechanisms are the same:

Detection

- DBVERIFY can be used to detect the corrupted blocks in the data file
- ANALYZE command run on the objects give errors
  (ANALYZE <table/cluster/index> <table/cluster/index_name> VALIDATE STRUCTURE;)
  When an ANALYZE (with CASCADE option) is run on a table or cluster, it cross verifies the index and data/cluster blocks along with the integrity checks done for the block.
- DB_BLOCK_CHECKSUM can be set to TRUE in the INIT.ORA file.
  When a block is changed and being written to disk from memory, DBWR calculates a checksum for the block by summing up its contents and writes it in the block along with it on disk. The next time when the block is being read by the foreground process, it calculates the checksum again for the block that is being read and compared with the checksum already written in the block on disk. If both are different then the block has been corrupted on disk and so the block is not read into memory so that it prevents cache corruption. There is an overhead associated with this parameter since the checksum is calculated each time it is read and written.
- Events 10210, 10211, 10212 can be set in the INIT.ORA file to detect software corrupt blocks. When there is a corruption in a block, it is not detected until the block is being updated. So any SELECTs on a corrupted block is executed until it is marked as software corrupt. When the events are set in the INIT.ORA, the blocks are checked for integrity by comparing certain data structures and once there is an inconsistency found, the seq or the sequence of the block is set to 0 in the block header representing that the block is software corrupt. The events can be set as follows:
  event = “10210 trace name context forever, level 10” (for data blocks)
  event = “10211 trace name context forever, level 10” (for index blocks)
  event = “10212 trace name context forever, level 10” (for cluster blocks)
- Users receiving ORA-1578 when trying to access an object. The query from DBA_EXTENTS given in pg# 30 shows that the error is on a table, cluster or index in the data tablespace
- ALERT.LOG shows ORA-1578 or ORA-600s with the first argument in the range of 2000 to 8000

Let us discuss the ways to repair the object when the corruption is thus detected:

Tables

Repair

When a data block is corrupted in a table, it should be understood that the data in the corrupted block is lost. The only way to not lose any data from the table is to restore from a valid backup and recover until a point in time before the corruption occurred.

If the data in the corrupted block can be recreated, then the following methods might be useful

- **Event method**: Event 10231 can be set to skip corrupted blocks on full table scans in the INIT.ORA file. The object can be exported after setting this event. This is not guaranteed to work for every kind of corruption. This works only when the block is soft corrupted, sequence is set to 0. The event can be set as follows:

event = "10231 trace name context forever, level 10"

- **ROWID method**: Extract the data that does not belong to the corrupted block using ROWIDs. Every row in every table in an Oracle database has a ROWID column which is usually not displayed unless SELECTed explicitly. The ROWID format used below pertains to version 7 of the database.

  SELECT ROWID, DEPTNO, DNAME, LOC FROM DEPT;

ROWID	DEPTNO	DNAME	LOC
00000440.0018.0004	30	ACCOUNTS	BOSTON
00000440.0019.0004	55	HR	SAN JOSE
00000440.001A.0004	33	OPERATIONS	BOSTON
00000440.001B.0004	65	APPS	SF
00000440.001C.0004	33	OPERATIONS	BOSTON

  Every ROWID has three parts to it.
  - ⇒ BLOCK ID in hex
  - ⇒ row offset or record index
  - ⇒ FILE ID

  If we have an ORA-1578 with file id being 3 and block id being 10, then converting the file id and block id to hex numbers, the ROWIDs for the rows in this block looks like: 0000000A.xxxx.0003.
  Now, a new table can be created with the data in this table can be selected around the corrupted block as follows:

```
CREATE TABLE NEW_TABLE AS SELECT * FROM DEPT
  WHERE ROWID NOT LIKE '0000000A.%.0003';
```

  The DEPT can then be dropped and recreated from the NEW_TABLE as:

```
CREATE TABLE DEPT AS SELECT * FROM NEW_TABLE;
```

- **Index method**: Extract the data that does not belong to the corrupted block using an index on the table.

If the table with the corrupted block has a unique index then the data around this block can be extracted using the index. For example, let us take the DEPT table with deptno, dname and location columns. A unique index exist on the deptno column. If there is an ORA-1578, the data block corruption error on file id 3 and block id 10, then the following query gives the deptno values for the rows in the corrupted block:

```
SELECT DEPTNO, ROWID FROM DEPT
  WHERE DEPTNO > 0
  AND ROWID LIKE '0000000A.%.0003';
```

Notice that the where clause on the DEPTNO forces the index on the deptno column to be used to give the result of the query and the where clause on the ROWID gives the rows in the corrupted block. This would be helpful if the index is in a different tablespace from the table itself. If the table and the index are both in the same tablespace, then there is more probability for the index to be corrupted along with the table.

If the deptno had not been a numeric column but is a character column then the query can be rewritten as:

```
SELECT DEPTNO FROM DEPT
  WHERE DEPTNO > ' '
  AND ROWIDTOCHAR(ROWID) LIKE '0000000A.%.0003';


DEPTNO   ROWID
-------------  -----------------
500      0000000A.%.0003
501      0000000A.%.0003
```

A new table can be created with rows around the corrupted block now as follows:

```
CREATE TABLE DEPT_NEW AS SELECT * FROM DEPT
  WHERE 1 = 2;
```

This command creates a table called DEPT_NEW with the same structure as the corrupted table, DEPT.

```
INSERT INTO DEPT_NEW SELECT * FROM DEPT
  WHERE DEPTNO < 500;
INSERT INTO DEPT_NEW SELECT * FROM DEPT
  WHERE DEPTNO > 501;
```

These two commands inserts the rows above and below the corrupted block into the new table. The corrupted DEPT table can then be dropped and then recreated from the new_table as:

```
CREATE TABLE DEPT AS SELECT * FROM NEW_TABLE;
```

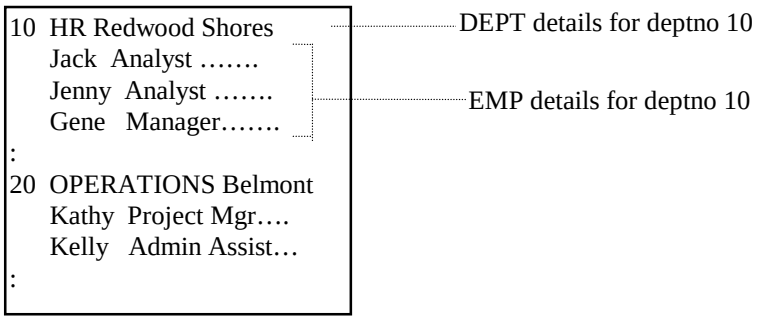
There is only one datatype that might give us some trouble when fixing corruption in the ways explained above. It is the (LONG) RAW datatype.

The only problem in the last two ways of extracting the data is that, it is not possible to run a SELECT statement on a table with a (LONG) RAW datatype. In case of this situation, a Pro*C program can be written to extract the long raw columns. A sample of this is given in the Pro*C users guide as well as a soft copy exists under $ORACLE_HOME/proc/demo.

Method	Advantages	Disadvantages
Event method	Easiest of the three should be used if they cannot use ROWID method. Probably used with export.	Not guaranteed to work
ROWID method	Since ROWID is unique, all the data around the corrupted block can be salvaged. Less IO than the index method.	Cannot identify the rows that are lost in the corrupted block since we have only the ROWIDs of the rows that we lost
Index method	If the index is not unique, we might lose more data than the ones in the corrupted block. More IO since we need to look up the rows in the index segment and then get the rows from the table.	We can identify the rows that we lose in the corrupted block using the index key which can be used to recreate the rows

Clusters

Clusters can be defined as a way to store more than one tables physically together since the tables have some common columns. By storing the related rows from multiple tables together, the access time can be reduced. For example, if a cluster is formed with the EMP and DEPT tables, the cluster block looks like:



A query that runs across EMP and DEPT will have to access just one block to get both tables' rows. The IO is less and the access time is less too.

The detection and repair for a cluster block corruption is similar to that of a table block corruption but the only difference is that when there is a corruption in a cluster block, all the objects that make up the cluster has to be fixed.

Indexes

Detection and Repair

Detection	***Repair***
• DBVERIFY gives the corrupted block information • ALERT.LOG shows corruption errors • Users running queries against the index get ORA-1578 on the index • ORA-600 with first argument in the range between 2000 and 8000	• Drop and recreate the index segment • Restore from a valid backup and recover

Best Practices

To sum up the section, let us discuss some of the best practices that can be followed to handle media corruption:

- Monitoring scripts written to monitor system logs for any kind of problem with the hardware, disk, controller, firmware and so on. If a problem is detected early enough, there is possibility of avoiding corruption from occurring by offlining the files that will be affected.
- Redundant hardware and software such as mirroring the database
- Rehearse automation of backup, restore and recovery scripts
- Security which include system, database and data privileges
- Monitoring the Oracle trace files such as the ALERT.LOG for any errors (in particular ORA-1578 or ORA-600)
- Monitoring the directories specified by the INIT.ORA parameters, BACKGROUND_DUMP_DEST and USER_DUMP_DEST for any trace files that are produced
- Monitoring any application logs to find anomalies in your application data.
- Monitor audit trails for any breach in security.
- Oracle utilities such as DBVERIFY, EXPORT and Oracle trace to be run frequently
- Oracle database and session level events
- Mirror control file and redo log files
- Mirrored copy should always be in a different disk if possible and should also be handled by different controllers
- Valid backups taken frequently
- ARCHIVELOG mode of database is always preferable
- Effective recovery scripts

Internal errors

Errors of the form ORA-600 are called internal errors. This section clarifies the misunderstanding of ORA-600s being synonymous with corruption.

An ORA-600 usually has up to five arguments associated with it which gives Oracle support and developers more information about the error. The first argument of an ORA-600 indicates where in the source code an unexpected condition was encountered. The remaining arguments give information about the variables and data being examined. An analyst can then determine if the error was the result of some type of corruption.

Oracle server source code is subdivided into about 13 different layers or modules, each of which is assigned a range of ORA-600 errors. Of these, only some ranges pertain directly to the types of corruption described in this paper.

Examples of ORA-600 ranges where corruption is a possible cause:

Code layer	Range of errors	Example
Cache layer	2000 - 4000	ORA-600[3398] Block corruption in memory ORA-600[2103] Control file enqueue timeout - this could be due to hardware problems where there is possibility of a slow/hung system but no corruption
Transaction layer	4000 - 6000	ORA-600[4146] undo/undo header block corruption
Data layer	6000 - 8000	ORA-600[6590] Row length inconsistency - block corrupt
Generic layer	17000 - 18000	ORA-600[17182] heap corruption - one of the pointers to a memory location does not have a valid address

Final Summary

The table below …

Type of Corruption	Definition	Potential Causes
Memory corruption	Inconsistency in data structures in memory	User errors Hardware problems Controller issues
Logical corruption	Inconsistency with the data	Bad application logic Lack of integrity constraints Application software bug Software bug
Media corruption	Inconsistency of data structures on disks	User errors Media problems Hardware/Disk problems Controller issues

Memory Corruption

Prevention	Detection	Repair
Monitor system logs and detect any system problems early enough so that the data files in the problematic section can be offlined or the instance can be shutdown to avoid corruption	ALERT.LOG and trace files reporting corruption	SHUTDOWN and STARTUP Trace files, reproducible case, dump memory trace files and so on can be used by support or development groups for further analysis.

Logical Corruption

Prevention	Detection	Repair
Cannot prevent Logical corruption since they are caused by software and application bugs	Queries returning inconsistent rows with different optimizer modes and with different execution plans	Use different optimizer mode Fix the problem by applying appropriate patches or by upgrading to the most recent version

Media Corruption

Prevention	Detection	Repair
Monitor system logs and detect any system problems	Utilities ⇒ DBVERIFY	Control file: ⇒ Use a mirror copy

early enough so that the data files in the problematic section can be offlined or the instance can be shutdown to avoid corruption	⇒ EXPORT ⇒ ANALYZE Traces ⇒ System logs ⇒ ALERT.LOG ⇒ Trace files in USER_DUMP_DEST and BACKGROUND_DUMP_DEST ⇒ Events can be set in the INIT.ORA to check for block integrity	⇒ Create new control file <u>Redo log file</u>: ⇒ Use a mirror copy ⇒ Clear log file if not current ⇒ Point in time recovery done using a valid backup until the corrupted log is hit <u>Data file</u>: File Header block ⇒ Restore and Recover Data Dictionary block ⇒ Restore and Recover Undo/Undo header block ⇒ Drop rollback segment if no active txn ⇒ Restore and Recover if there is active transaction ⇒ Unsupported methods discussed in the paper Sort block ⇒ Temp segments are reused frequently ⇒ If problem persists, drop and recreate temp tablespace Table/Cluster block ⇒ Restore and Recover ⇒ event 10231 can be set in INIT.ORA and the corrupted rows can be skipped ⇒ ROWID method of salvaging the data that is not corrupted ⇒ Index method of salvaging the data that is not corrupted Index block ⇒ Restore and recover ⇒ Drop and recreate index

Appendix

ALERT.LOG

Oracle provides a log file called the ALERT.LOG to record any DDL (Data Definition Language) commands such as STARTUP, SHUTDOWN, CREATE TABLESPACE, ADD DATAFILE, ALTER DATABASE, DROP ROLLBACK SEGMENT, etc. that were executed in the database. This file also logs many ORA errors that users might encounter. This is very useful in diagnosing problems in the database.

ANALYZE

This command is used to analyze a table, index or cluster object in a database. This command actually checks for any inconsistencies in the data structures within each block belonging to the object and also cross verifies the index and the table/cluster blocks along with the integrity checks. The command looks like:

```
ANALYZE TABLE/INDEX/CLUSTER <table/index/cluster name> VALIDATE
STRUCTURE;
```

More information on the command syntax is in Server SQL Reference Manual. This can also be run on a schema by running the procedure DBMS_DDL.ANALYZE_SCHEMA. This procedure can be created by running the script, dbmsutil.sql found under $ORACLE_HOME/rdbms/admin.

BACKGROUND_DUMP_DEST

This is an initialization parameter found in the INIT.ORA file. This denotes the directory location where the background processes such as PMON, SMON, DBWR and so on should write their trace files in case of an error or when they are forced to write trace files by adding events in the INIT.ORA file.

CREATE CONTROLFILE SCRIPT

The CREATE CONTROLFILE script can be created before hand which will be helpful when all the copies of the control file are either corrupted or deleted. The following command executed in SVRMGR or SQLDBA creates a trace file in the USER_DUMP_DEST:

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

The trace file looks like ORA_#####.TRC. This can be edited and copied as a .SQL file and can be run as a script in SVRMGR or SQLDBA to create a new control file. A sample of the CREATE CONTROLFILE looks like:

```
CREATE CONTROLFILE REUSE DATABASE "V723" NORESETLOGS
NOARCHIVELOG
   MAXLOGFILES 32
   MAXLOGMEMBERS 2
   MAXDATAFILES 62
   MAXINSTANCES 8
   MAXLOGHISTORY 800
LOGFILE
  GROUP 1 '/u05/oracle/7.2.3/dbs/log1V723.dbf'  SIZE 500K,
  GROUP 2 '/u05/oracle/7.2.3/dbs/log2V723.dbf'  SIZE 500K,
  GROUP 3 '/u05/oracle/7.2.3/dbs/log3V723.dbf'  SIZE 500K
DATAFILE
  '/u05/oracle/7.2.3/dbs/systV723.dbf',
  '/u05/oracle/7.2.3/dbs/rbsV723.dbf',
  '/u05/oracle/7.2.3/dbs/tempV723.dbf',
  '/u05/oracle/7.2.3/dbs/toolV723.dbf',
  '/u05/oracle/7.2.3/dbs/usrV723.dbf';
```

DBVERIFY

This utility is available from 7.3.2 version of Oracle database. It can be run on data files to verify each block in the data file for consistency within itself. The command to run the utility is :

dbv FILE filename_tobe_verified [options]

Options:

FILE - Name of the file to be verified
START - Block number from which data file should be verified
END - Block number to which data file should be verified
BLOCKSIZE - Oracle block size
LOGFILE - Output log file name
FEEDBACK - Display progress

Output:

```
DBVERIFY: Release 7.3.2.1.0 - Production on Fri May 30 13:24:17 1997

Copyright © Oracle Corporation 1979, 1994.  All rights reserved.

DBVERIFY - Verification starting : FILE = jan1.dbf
Page 1 is marked software corrupt

DBVERIFY - Verification complete

Total Pages Examined             : 1
Total Pages Processed    (Data)  : 0
Total Pages Failing      (Data)  : 0
Total Pages Processed    (Index) : 0
Total Pages Failing      (Index) : 0
Total Pages Empty                : 0
Total Pages Marked Corrupt       : 1
Total Pages Influx               : 0
```

The output gives the number of corrupted blocks in the data file that has been verified. This also specifies the number of index blocks corrupted and the number of data blocks corrupted.

More information on the utility is documented in the Server Utilities User's Guide.

INIT.ORA

This is the initialization file that the instance looks up when it is brought up. The parameters listed in this file are initialized at the start of the instance which will be used during its lifetime.

ORA-1545

```
01545, 00000, "rollback segment '%s' specified not available"
// *Cause: Either:
//        1) An attempt was made to bring a rollback segment online that is
//           unavailable during startup; for example, the rollback segment
//           is in an offline tablespace.
//        2) An attempt was made to bring a rollback segment online that is
//           already online.  This is because the rollback segment is
//           specified twice in the ROLLBACK_SEGMENTS parameter in the
//           initialization parameter file or the rollback segment is already
//           online by another instance.
//        3) An attempt was made to drop a rollback segment that is
//           currently online.
// *Action: Either:
```

// 1) Make the rollback segment available; for example, bring an
// offline tablespace online.
// 2) Remove the name from the ROLLBACK_SEGMENTS parameter if the name
// is a duplicate or if another instance has already acquired the
// rollback segment.
// 3) Bring the rollback segment offline first this may involve
// waiting for the active transactions to finish, or, if the
// rollback segment
// needs recovery, discover which errors are holding up the rolling
// back of the transactions and take appropriate actions.

ORA-1578

01578, 00000, "ORACLE data block corrupted (file # %s, block # %s)"
// *Cause: The data block indicated was corrupted, mostly due to software
// errors.
// *Action: Try to restore the segment containing the block indicated. This
// may involve dropping the segment and recreating it. If there
// is a trace file, report the errors in it to your ORACLE
// representative.

Query from DBA_EXTENTS to locate the corrupted object

ORA-1578 usually gives the file # and block # of the corrupted block. The following query can be used to locate the object to which the corrupted block belongs to:

```
SELECT SEGMENT_TYPE, SEGMENT_NAME FROM SYS.DBA_EXTENTS
WHERE FILE_ID = <file_id_from_ORA-1578>
AND <block_id_from_ORA-1578> BETWEEN BLOCK_ID AND (BLOCK_ID+BLOCKS-1);
```

UNDOCUMENTED AND UNSUPPORTED

These describe INIT.ORA parameters which are not documented in any official Oracle manuals. These parameters are to be used only as a last resort with supervision from Worldwide Support if there is no other supported way of restoring the database. Actions done by setting these parameters are not rigorously tested or guaranteed to work and can cause further damage. One thing to remember is to remove these parameters immediately after they have served their purpose.

USER_DUMP_DEST

This is an initialization parameter found in the INIT.ORA file. This denotes the directory location where the user should write their trace files in case of an error or when they are forced to write trace files by adding events in the INIT.ORA file.

V$ views

There are a number of data dictionary views that start with ‘V$’ which extracts the information about the database from the control file. Queries from these views are always helpful to diagnose any problems with the database. Some of the useful V$ views are:

- ⇒ V$ROLLSTAT - To query about rollback segments
- ⇒ V$SYSSTAT - To query on hanging problems with the system
- ⇒ V$SESSTAT - To query on sessions
- ⇒ V$DATAFILES - To query on data files
- ⇒ V$LOGFILE - To query on log files