DSI301: Advanced Server Support Skills

Instructor Guide

90008GC10
Production 1.0
December 1999
M09534

ORACLE®

Authors

Jim Womack
Ulrike Schwinn

Technical Contributors and Reviewers

Richard Exley
Ashish Prabhu
Khaled Kassis
Tammy Bednar
Wim Coekaerts
Jim Barlow
Erhan Odok
Vijay Lunawat
Christine Jeal
Michael Möller
Jose Marco-Dominguez
Peter Gram
Lex de Haan
Troy Anthony
David Austin

Publisher

Tommy Cheung

Contents

1

Introduction

Schedule:	Timing	Topic
	20 minutes	Lecture
	20 minutes	Total

Course Objectives

After completing this course, you should be able to do the following:

- **Convey the benefits of reading and interpreting the Oracle Server source code**
- **Explain in detail how events work and how to use them in diagnosing Oracle problems**
- **Describe the hang, loop, and crash diagnostic process**
- **Understand, identify, and diagnose heap corruption**

The three days DSI301 course is the first in a series of courses that targets Oracle Support Services (OSS) technical analysts and customer support. The primary purpose is to provide participants with the tools and utilities to analyze and troubleshoot server-related failures. It is also intended as a preparatory course for courses DSI302 to DSI308.

Course Objectives

- Dump data blocks, and interpret Oracle7 and Oracle8*i* block dumps
- Handle block corruption problems
- Handle rollback segment corruption issues
- Explain how to use the Data Unloader Utility (DUL)

DSI Curriculum

The Data Server Internals (DSI) curriculum consists of the following eight courses:

- **DSI301: Advanced Server Support Skills**
- **DSI302: Data Management**
- **DSI303: Database Backup and Recovery**
- **DSI304: Query Management**

DSI Curriculum

The DSI curriculum provides internal and confidential information for OSS technical analysts and customer support staff with a minimum of 12 months experience supporting the RDBMS. Each of the following courses lasts four days:

- DSI302 provides insight into data management within the Oracle server and uses demonstrations and lab exercises to show how rollback operations are performed.
- DSI303 teaches participants about operating system- and server-managed (Oracle8*i*) backup and restore; the internals of redo generation, application, and archiving; block, crash, instance, and media recovery; the resetlogs operation; the implementation of read-only and offline tablespaces; object (TSPITR) recovery; parallel recovery; and how to gather and analyze recovery diagnostics.
- DSI304 provides insight into query operations within Oracle and how to interpret and modify their execution plans.

DSI Curriculum

- **DSI305: Database Tuning**
- **DSI306: Very Large Databases**
- **DSI307: Distribution and Replication**
- **DSI308: Parallel Server**

DSI Curriculum (continued)

- DSI305 provides detailed insight into database tuning, enabling students to interpret and evaluate a database profile and to gain an understanding of the internal layers and mechanisms that give rise to the common database tuning profiles.
- DSI306 teaches participants about transaction monitors, features, and relevance in a VLDB environment. Participants also learn about techniques like MTS, partitioning (manual and automatic), and parallel DML. Other topics covered include advanced queuing, sort enhancements, deferred transaction recovery, and tuning data loads.
- In DSI307, participants explore the troubleshooting and design of advanced replication architectures.
- DSI308 discusses Oracle7 and Oracle8*i* Parallel Server advanced topics.

Suggested Course Schedule

Day	Unit	Lessons	Labs
1	Introduction	1	1
	Source Code, The Ultimate Oracle	2	2
	Using Diagnostic Events	3	3
	Hang, Loop, and Crash Diagnostics	4	
2			4
	Heap Corruption Diagnostics	5	5
	Block Dump Analysis	6	6
	Block Corruption	7	
3			7
	RBS Corruption Recovery	8	8
	Data Salvage Using Data Unloader	9	9

Note: This is a suggested course schedule only. The amount of time spent on each subject area will depend on the interest shown by the participants in the particular class.

2

Source Code:
The Ultimate Oracle

ORACLE

Schedule:	Timing	Topic
	60 minutes	Lecture
	20 minutes	Examples
	60 minutes	Lab Exercises
	140 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- Explain the benefits of reading the Oracle source code
- Use the source code repository
- Describe how the source code is organized
- Search and navigate through the source code and identify and interpret common structures

Introduction

- The source code is the ultimate definition of the Oracle server.
  - Provides documentation where manuals and papers are lacking
  - Makes the reader an indisputable authority
- The bottom line: You can use the source code to provide a higher level of support.

Introduction

The ability to read and understand the source code will make you an indisputable authority on the Oracle server and enable you to resolve issues that would otherwise be beyond your scope. It is a skill that takes a significant amount of time and effort to acquire, and you may often be discouraged by the cryptic and sometimes confusing syntax and nomenclature. If you persevere, however, you will find that things become more clear and familiar over time, and your level of confidence will grow.

C Language

- The source code is written in the C programming language so knowledge of C is useful.
- C can be rather cryptic, but does not take long to learn.
- You only have to be able to read C, you do not have to write it.

C Language

All Oracle source code has a similar look and feel. This is because the developers follow an Oracle C Coding Standard (OCCS).

Getting Started

- To gain access to the source code, use the email public template Source Userid Request.
  - Eligibility requires six months or more of continuous employment with Oracle, and director's approval.
- The source code is located on a UNIX Solaris machine called tao.
- Account holders use a restricted shell. Access to standard UNIX commands is severely restricted.
- Users can only search and navigate the code trees using cscope.

Getting Started: Example from tao

```
login: jwomack
password: *****

Last login:  Mon Nov 1 08:19:58 from whq4op3x38-rtr-o
**************************************************
This system is now running under increased security.
ONLY the following commands are available for analyst use:
*         view     *
*         more     *
*         pg       *
*         cdb      *
*         findora  *
*         cscope   *
Only files under this relative path, /export/home/restrict,
are available for viewing.
```

Navigation

- On tao, you can choose between many Oracle versions. Currently, analysts can choose any version between 6.0.37 and 8.1.5.
- The cdb command is used to switch you to the root directory of the source code version you have selected when using utilities such as findora, find600, and cscope.

Navigation: Example from tao

The following source trees are available:

```
 Version           SRC_ID          SRC_HOME
  6.0.37             6037          /export/home/ssupport/6037
  7.0.13             7013          /export/home/ssupport/7013
  7.0.16             7016          /export/home/ssupport/7016
   7.1.3              713          /export/home/ssupport/713
   7.1.4              714          /export/home/ssupport/714
   7.1.6              716          /export/home/ssupport/716
   7.2.2              722          /export/home/ssupport/722
   7.2.3              723          /export/home/ssupport/722
 7.3.2.1              732          /export/home/ssupport/732
 7.3.2.3             7323          /export/home/ssupport/7323
 7.3.3.0              733          /export/home/ssupport/733
 7.3.4.0              734          /export/home/ssupport/734
   8.0.3              803          /export/home/ssupport/803
   8.0.4              804          /export/home/ssupport/804
   8.0.5              805          /export/home/ssupport/805
   8.1.5              815          /export/home/ssupport/815
$ cdb 815
Spawning sub-shell for operations
SRC_ID = 815
SRC_HOME = /export/home/ssupport815
```

Directory Structure

The source code is organized into a hierarchy of directories:

- Header files
  - rdbms/src/hdir
  - rdbms/include
- Operating system dependent code (OSD)
  - rdbms/src/server/osds
- RDBMS kernel
  - rdbms/src/server

Filename Conventions

- Each code layer is identified by file prefix; for example:
  - kt kernel transaction
  - kd kernel data
  - kc kernel cache
  - ks kernel services
- The source code files are named accordingly.

Filename Conventions

The code prefixes are used to name files in the tree. Take, for example, the file ktu.h: you can see that it must have something to do with kernel transactions (kt), and if you take a look at the the file, you find that it defines undo operations. Hence, ktu stands for kernel transaction undo.

Searching: cscope

- cscope is used exclusively on tao for browsing source code.
- cscope understands C language syntax and so can be used to find functions, callers, macro definitions, and so on.

```
$ cscope
```

- Type ? for help.

Searching: cscope

When using cscope, you must set the source root for the version that you want to view. Type cscope, then indicate the version that you want to work with. In the example below, note that you choose Oracle 8.1.5. The version is entered without the periods; that is, 8.1.5 is entered as 815.

```
$ cscope
*************************************************************************
*                                                                       *
* This is the cscope environment selection utility, please              *
* select from the following environments:                               *
*                                                                       *
*  716                                                                  *
*  722                                                                  *
*  723                                                                  *
*  7323                                                                 *
*  733                                                                  *
*  734                                                                  *
*  803                                                                  *
*  804                                                                  *
*  805                                                                  *
*  815                                                                  *
*                                                                       *
* Please make your selection, i.e.; 734                                 *
*                                                                       *
*************************************************************************
 >815
```

Common Structures

It is relatively easy to find the following structures in the source code:

- **Oracle errors (ORA-*nnn*)**
- **Internal errors (ORA-600)**
- **Initialization parameters**
- **Fixed tables and views**
- **SQL statement processing**

Common Structures

In support, you are often presented with a set of diagnostic evidence from which you are asked to provide an explanation. When the meaning is unclear, it is possible to map the diagnostic output back to the source (the code) to gain a precise understanding.

You may sometimes have insufficient information and so you have to find additional diagnostics. In this section, you will learn how to find additional diagnostics, including events, initialization parameters, and fixed tables and views.

On occasion you may need to find out how a particular Oracle facility is implemented internally. Typically features are associated with a particular SQL statement; this lesson shows you how to find them in the code.

Oracle Errors

- When the source code fails, an error is logged; for example, ORA-1575.
- Oracle errors are identified in the code by the OER(*nnn*) structure, so they are easy to find.
- You find the failing source code by using the findora command; for example:

```
$ findora 1575
```


Oracle Errors: Example from tao

```
tao-sun% cdb 805
tao-sun% findora 1575

Searching for OER(1575)
File:   /export/home/ssupport/805/rdbms/src/server/space/spcmgmt/ktm.c
Line #: 976
          if (e == OER(1595) && (e2 == KSSOERLC || e2 == OER(1575) ||
File:   /export/home/ssupport/805/rdbms/src/server/space/spcmgmt/ktm.c
Line #: 997
             (e != KSSOERLC && e != OER(1575) && !shrink_failed) ||
File:   /export/home/ssupport/805/rdbms/src/server/txn/lcltx/ktc.c
Line #: 1158
      ksesec0(OER(1575));              /* fail to get space enqueue */
File:   /export/home/ssupport/805/rdbms/src/server/txn/lcltx/ktc.c
Line #: 1193
        ksesec0(OER(1575));            /* fail to get space enqueue */
File:   /export/home/ssupport/805/rdbms/src/server/txn/lcltx/ktc.c
Line #: 1248
         ksesec0(OER(1575));           /* fail to get space enqueue */
```

Internal (600) Errors

- The Oracle code can fail with an internal error such as ORA-600 [nnn], [value1], ...
- An internal error is identified in the code by an OERI(*nnn*) or OERINM(*nnn*) structure.
- Use the find600 command with the error's first argument to find the failing code; for example:

```
$ find600 13012
```


Internal (600) Errors: Example from tao

```
tao-sun% cdb 805
tao-sun% find600 13012

Searching for OERI(13012) or OERINM("13012")
File: /export/home/ssupport/805/rdbms/src/server/sqlexec/dmldrv/updexe.c
Line #: 1870
      ksesic6(OERI(13012), ksenrg(lockiters),
```

Note: As of Oracle8, the first ORA-600 argument can be textual (OERINM).

Logging Mechanics

- The Oracle code logs internal errors by using a ksesic*n*, ASSERT*n*, or ASSERTNM (Oracle8 only) macro.
  The *n* refers to the total number of arguments passed with the error, minus 1.
- The macro arguments map to the internal error arguments.
- By using this mapping, you can discover the meaning of the error.

Example of ASSERT from ktu.c

```
/* can't find it, have to go to dictionary cache */
ASSERT1(kqrpre(KQRUSC, (ptr_t)&usn, (kqrpo **)&usc, KQRMS
       ,(kssob *)ksugcc(),&eq, mid), OERI(4000), KSESVSGN, usn);
```

Example of ksesic from ksm.c

```
if (asp > (b4)ksepec(OER(10262)))
  {
    ksdwrf("\n******** ERROR: SGA memory leak detected %ld ********\n",
           (long)asp);
    kghdmp(ksmgpga, ksmtsga, 0);                      /* dump sga heap */
    /* If heap checking enabled, then signal the space leak internal
     * error. Note that we'll still write out a space leak trace file.
     */
    if (ksepec(OER(10235)))
          ksesic2(OERI(730), ksenrg(asp), ksesrgl((text *)"space leak"));
                                                       /* signal error */
  }
```

Note: See rdbms/src/hdir/kse.h for definitions of ksesicn, ASSERT, and ASSERTNM.

Example of ASSERTNM from kse.c (8.1.5 code tree)

```
/*?? couldn't find positional match in either list-something wrong !!*/
ASSERTNM(0, OERINM("kkqsepos-2"));
```

Note: See rdbms/src/hdir/kse.h for definitions of ksesicn, ASSERT and ASSERTNM (Oracle8 only).

Variable Mapping

ksesicn is used as follows (*n* = number of arguments excluding [value1]):

```
ORA-600 [value1], [value2], [value3] ...
```

```
ksesicn(OERI(value1), value2, value3, ... );
```


Example from ksm.c

```
ORA-00600 [730], [1236], [space leak]
if (asp > (b4)ksepec(OER(10262)))
  {
    ksdwrf("\n******** ERROR: SGA memory leak detected %ld ********\n",
           (long)asp);
    kghdmp(ksmgpga, ksmtsga, 0);                    /* dump sga heap */

    /* If heap checking enabled, then signal the space leak internal
     * error.  Note that we'll still write out a space leak trace file.
     */
    if (ksepec(OER(10235)))
        ksesic2(OERI(730), ksenrg(asp), ksesrgl((text *)"space leak"));
                                                    /* signal error */
  }
 => asp = 1236
```

Note: `asp` stands for available space.

Mapping Arguments: Internal Errors

- **ASSERT*n* is used to log an internal error if <condition> is false.**
- ***n* is the number of ORA-600 arguments excluding [value1].**

```
ORA-600 [value1], [value2], [value3] ...
```

```
ASSERTn(<condition>,OERI(value1), KSE,
        value2, value3,...              )
```


Example from ktu.c

```
ORA-00600 [4000], [0]
/* can't find it, have to go to dictionary cache */
ASSERT1(kqrpre(KQRUSC, (ptr_t)&usn, (kqrpo **)&usc, KQRMS,
(kssob *)ksugcc(), &eq, mid), OERI(4000), KSESVSGN, usn);
    => usn = 0
```

The error was logged because:

```
kqrpre(KQRUSC, ...) != 0
```

Note: `usn` stands for undo segment header.

ASSERTNM

- **ASSERTNM is new in Oracle8.**
- **ASSERTNM is used like ASSERT to log internal errors, but uses text identifiers instead of numeric identifiers.**
- **Internally, ASSERTNM does not require a severity.**

ASSERTNM

The ASSERTNM interface has been added to support named internal errors in ASSERTs. It functions in the same way as the existing ASSERT macros, except that:

- It requires a named internal error (CONST char *).
- It does not require a severity.
- The corresponding assert code assumes a severity of KSESVSGN (for example, signal internal error).

Initialization Parameters

- Initialization parameters are defined in the code using the KSPPARDN, KSPPARDV, and KSPPARDYNV macros.
- Look for the *id* in the KSPPARDN definition: `#define id KSPPARDN(...)`
- *id* is the internal identifier of the parameter and can be used to find how the parameter is manipulated and used.

Example from 8.1.5

KSPPARDYNV defines a dynamic parameter. `prmwsz` is the internal identifier in this example.

```
#define prmwsz KSPPARDN(prmwsz_)
KSPPARDYNV("sort_area_size", prmwsz_, prmwsz_s, prmwsz_l, prmwsz_p,
           LCCMDINT, 0, NULLP(text), SORDFWSZ,
           KSPLS1(NULLP(text)), KSPLS1(UB4MAXVAL),
           FADDR(0),
           KSPSYSDEF|KSPSESNOR,
           0,
           KSPLS2(&kcbpbs, NULLP(word)),
           "size of in-memory sort work area")
```

Fixed Tables and Views

- Fixed (X$) tables are defined using macros defined in rdbms/src/hdir/kqf.h.
- Search for KQF in the header files of the component that you are interested in.
- Dynamic (V$) views are all defined in rdbms/src/hdir/kqfv.h.

Example from rdbms/include/kqf.h (7.3)

```
#ifndef KMLS
KQFTABL(kspi, kspi_c, "X$KSPPI", kspii, ksppno, KQFOB086, 2)
#else
KQFUTBL(kspi, kspi_c, "X$KSPPI", kspii, ksppno, KQFOB086, 2, FADDR(kspgpl))
#endif /* K_MLS */
KQFCSTP(kspi, kspip,   "KSPPINM")                       /* Name           */
KQFCINT(kspi, kspityp, "KSPPITY")                       /* Type           */
KQFCFST(kspi, "KSPPIVL", KSPMAXPLL, kspgpv, kspgpv_n)   /* Value          */
KQFCFST(kspi, "KSPPIDF", KSPBOOLEN, kspgdf, kspgdf_n)   /* Default Status */
KQFENDT(kspi)
```

Example from rdbms/include/kqfv.h (7.3)

```
KQFVIEW("V$LATCH", kqfv009_c,
"select d.kslldadr,la.latch#,d.kslldlvl,d.kslldnam,la.gets,la.misses,
        la.sleeps,la.immediate_gets,la.immediate_misses,la.waiters_woken,
        la.waits_holding_latch,la.spin_gets,la.sleep1,la.sleep2,
        la.sleep3,la.sleep4,la.sleep5,la.sleep6,la.sleep7,la.sleep8,
        la.sleep9,la.sleep10,la.sleep11
 from   x$kslld d,
       (select kslltnum latch#,
               sum(kslltwgt) gets,sum(kslltwff) misses,sum(kslltwsl) sleeps,
               sum(kslltngt) immediate_gets,sum(kslltnfa) immediate_misses,
               sum(kslltwkc) waiters_woken,
               sum(kslltwth) waits_holding_latch,
               sum(ksllthst0) spin_gets,sum(ksllthst1) sleep1,
               sum(ksllthst2) sleep2,sum(ksllthst3) sleep3,
               sum(ksllthst4) sleep4,sum(ksllthst5) sleep5,
               sum(ksllthst6) sleep6,sum(ksllthst7) sleep7,
               sum(ksllthst8) sleep8,sum(ksllthst9) sleep9,
               sum(ksllthst10) sleep10,sum(ksllthst11) sleep11
        from   x$ksllt group by kslltnum) la
 where  la.latch# = d.indx", KQFOB009, 6)
```

SQL Statement Processing

- The `opiexe.c` file contains the starting point for most SQL statements.
- Try to find the Oracle Call Type (OCT) code; with this code, you can figure out how the SQL statement is processed.

Example for CREATE TABLE

```
tao% cscope
Find this C symbol:
Find this definition:
Find functions called by this function:
Find functions calling this function:
Find assignments to:
Change this grep pattern:
Find this egrep pattern:"create table"
Find this file:
Find files #including this file:
1 /export/home/ssupport/815/rdbms/src/server/progint/opi/opiexe.c
  case OCTCTB:                                  /* create table */
  case OCTCTS:                              /* create tablespace */
```

The OCT code is OCTCTB.

Extract from opiexe.c: you see that ctcdrv() implements the create table and create cluster commands.

```
  case OCTCTB:                                  /* create table */
  case OCTCCL:                                /* create cluster */
  ctcdrv(ctx, &global_snap, selenv);
  break;
```

Case Study

Release 7.3.4 is failing with the following error:

```
ORA-00600: internal error code,
arguments: [2845], [193], [20], [525542], ...
```

What does this error mean?

Case Study

Here you have a typical support situation. The customer has experienced an internal error and he or she would like an explanation of what has gone wrong.

To begin with, all you have is a meaningless stream of numbers.

Case Study

```
ORA-00600: internal error code,
arguments: [2845], [193], [20], [525542], ...
```

```
tao% cdb734
tao% find600 2845

Searching for OERI(2845)
File:   /export/home/ssupport/732/rdbms/kernel/knl/kcf.c
Line #: 6749
   ASSERT3(fno && fno <= kcfdfk && bno > 1,
           OERI(2845), KSESVSGN,

tao% view /export/home/ssupport/732/rdbms/kernel/knl/kcf.c
```


Case Study (continued)

Log on to tao and switch to the correct version. Then use the find600 command to locate the point in the code where this error was reported. In this case it is kcf.c.

Case Study

```
tao% view /export/home/ssupport/732/rdbms/kernel/knl/kcf.c
```

```
fno = FILENUM(strtdba);
bno =  BLKNUM(strtdba);
kcfiop = &kcfios[KCFMID(fno-1, mid)];

ASSERT3(fno && fno <= kcfdfk && bno > 1, OERI(2845),
        KSESVSGN, fno, kcfdfk, bno);

kcfcko(NULLP(kcccx), fno, mid);       /* check/open file */
if (K_MNTIF(kprmry(mid) &&) !kcvcfv(fno))
             /* insure recovery component likes the file */
{
  kcfrfn(fno, NULLP(kcccx), mid);    /* record file name */
  ksesec1(KCFOERVRF, ksenrg(fno));  /* file not verified */
}
```


Case Study (continued)

Once you are inside `view`, search for 2845 (type /2845). As you can see, the arguments are [2845], [fno], [kcfdfk], [bno]. In other words:

```
fno    = 193
kcfdfk = 20
bno    = 525542
```

The condition is:

```
fno && fno <= kcfdfk && bno > 1
```

Translated, this means:

```
if    fno != 0      and
      fno <= kcfdfk and
      bno > 0
then  okay
else  log error;
```

Substituting the numbers you can see that the first condition and the third condition evaluate to true; so the error was logged because:

```
fno > kcfdfk
```

What does all this mean? Scan backwards in the file (using ^U).

Case Study

Still viewing `rdbms/kernel/knl/kcf.c`:

```
** Read Buffers from Database:
** Invoked in foreground for buffer and direct reads, never in background.
** Signals an error if read fails or file should not be read.
** Returns the number of blocks successfully read.
*/
word      kcfrbd(contig, bufp, strtdba, nblks, mid)
word      contig;                /* true IFF memory buffers continguous */
ptr_t     bufp[];                /* caller-supplied IO buffer ptr array */
reg4 kdba strtdba;               /* starting blk dba from which to read */
word      nblks;              /* number of sequential db blocks to read */
kmid      mid/*ARGUSED*/; /* internal mount id of database to read from */
{
        serc   se;                                    /* OSD error buffer */
   reg1 kfil   fno;                       /* file # corresponding to dba */
   reg2 ub4    bno;                      /* block # corresponding to dba */
   reg3 word   nread;                          /* number of blocks read */
   reg5 ub4    base;                          /* time io is started */
   reg0 kcfio *kcfiop;                    /* pointer to io stastistics */
```


Case Study (continued)

This is the procedure that is reporting the error. As you can see, there is a reasonable amount of documentation in this case. The RDBMS was in the process of reading from a database file fno, block bno at the time it failed.

To complete the picture, you need to know what kcfdfk is, so search for kcfdfk in this file (type /kcfdfk); no definitions found.

Quit the view environment, using the :q! command.

Case Study

```
 1. kcb.h: KQFTABL(kcbwat, kcbwat_f, "X$KCBFWAIT",
              kcbwf, kcfdfk, KQFOB169, 1)
 2. kcf.h: # define KCFMD1(i, j) ((i)+((j)*(kcfdfk+1)))
 3. kcf.h: # define KCFMID(i, j) ((i)+((j)*kcfdfk))
 4. kcf.h: * a double dimension array.
              kcfdfk holds the size of a single dimension.
 5. kcf.h: # define kcfdfk   KSMSGADN(kfil, kcfdfk_)
 6. kcf.h: KSMSGADV(kfil, kcfdfk_)    SGA: max # db files */
 7. kcf.h: KQFTABL(kcfio, kcfio_c, "X$KCFIO",
              kcfios, kcfdfk, KQFOB045, 1)
 8. kcl.h: KQFTABL(kclfh, kclfh_c, "X$KCLFH",
           kclhbs, kcfdfk, KQFOB229, 1)
 9. kcl.h: KQFTABL(kclfi, kclfi_c, "X$KCLFI",
           kclfis, kcfdfk, KQFOB230, 1)
10. kcv.h: must be greater or equal to the maximum number
              of datafiles (kcfdfk)
```


Case Study (continued)

If the definition is not in kcf.c, it is probably in the header file, so switch to dbms/include.

Now, search for kcfdfk using cscope; kcf.h looks hopeful. Hit items 6 and 10 give us a clue as to the meaning of kcfdk:

```
6. kcf.h:KSMSGADV(kfil,  kcfdfk_)    SGA: max # db files */
```

Just to make sure, look in kcf.h and research the definition of kcfdfk.

Case Study

Extract from `rdbms/include/kcf.h`:

```
** variables containing limits from the parameter file
*/
#define kcfcfk KSMSGADN(word, kcfcfk_)
KSMSGADV(word, kcfcfk_)                    /* SGA: max # control files */
#define kcfdfk KSMSGADN(kfil, kcfdfk_)
KSMSGADV(kfil, kcfdfk_)                     /* SGA: max # db files */
#define kcflfk KSMSGADN(word, kcflfk_)
KSMSGADV(word, kcflfk_)                    /* SGA: max # log files */
#define kcfmsw KSMSGADN(word, kcfmsw_)
KSMSGADV(word, kcfmsw_)       /* SGA: max simultaneous db file writes */
#define kcfmrf KSMSGADN(word, kcfmrf_)
KSMSGADV(word, kcfmrf_)                /* SGA: multiblock read factor */
#define kcfmmw KSMPGADN(kcfmmw_)
KSMPGADV(word, kcfmmw_) /* PGA: My maximum parallel write on db files */
```


Case Study (continued)

Here you find the answer: kcfdfk represents the maximum number of database files.

Case Study Conclusion

- Oracle was attempting to read block 525542 of file 193 when it failed.
- It failed because this database has a maximum of 20 database files.
- Serious malfunction has occurred; a bug search is the next step.

Case Study (conclusion)

You do not have a complete answer yet, but you are much closer to one. The source code can sometimes give you the whole answer. If not, it can inform a bug search, narrow down the possibilities, or suggest an avenue for further diagnostics. If you spend a little time researching the 2845, you would find out that this generally occurs as a result of an in-memory corruption of the pointer that holds the value for the maximum number of database files, a value set at database creation. This kind of heap corruption is typical of client applications or platform-specific bugs, so it provides clues on where to look next.

Using the source code can be both rewarding and time-consuming. Use the source code to resolve issues only when standard methods fail. Be sure to set customer expectations regarding time whenever a source code search is required.

Practice 2 Overview

This practice covers searching and navigating through the source code to:

- **Interpret specific errors**
- **Find the definition of a dynamic performance view**
- **Find the location of the _corrupted_rollback_segments parameter**

Practice 2 Overview

For detailed instructions on performing this practice, see Practice 2 in Appendix B.

Additional case studies are available in Appendix A.

3

Using Diagnostic Events

ORACLE

Schedule:	Timing	Topic
	60 minutes	Lecture
	15 minutes	Examples
	45 minutes	Lab Exercises
	120 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- Describe the four uses of diagnostic events
- Identify the tools used to set events, and the syntax in each case
- Find events in the source code and describe how they work

Events: Usage

Events are primarily used to:

- **Produce additional diagnostics when insufficient information is available.**
- **Work around or resolve problems by:**
  - **Changing Oracle's behavior**
  - **Enabling undocumented features**

Events: Usage

The use of the diagnostic dumps and events can be divided into four categories:

- Immediate dump
- On-error dump
- Change behavior
- Trace

Setting Events

You can set events by using:

- The event initialization parameter
- The ALTER SESSION SET EVENTS or ALTER SYSTEM SET EVENTS commands
- The sys.dbms_system.set_ev( ) procedure
- The oradebug utility

Setting Events

1. In the parameter file:
```
event = "<event name> <action>
       [:<event name> <action>]…"
```
2. For the current SQL session:
```
alter session set events
        '<event name> <action>
       [:<event name> <action>]…'
```
3. For all new sessions (8.1.4 and above):
```
alter system set events
        '<event name> <action>
       [:<event name> <action>]…'
```

3-5 ORACLE

Events are essentially a debugging aid for complex server problems. When these problems occur, it is necessary to find their cause by creating a trace file, triggered by the occurrence of the problem. To create a trace file you must set an event by:

- Issuing an alter session or alter system command from inside the tools
- Executing the sys.dbms_system.set_ev procedure
- Using the ORADEBUG utility

Examples

```
event = "604 trace name errorstack"
```

This dumps the errorstack to a trace file whenever the following error occurs:

```
ORA-00604 : error occurred at recursive SQL level x
```

Inside SQL*Plus, the event can be set by issuing the following command:

```
SQL> alter session set events '604 trace name errorstack';
```

This has the same effect as setting the event at the database level, except that it will only be in effect for the length of the SQL*Plus session.

```
SQL> alter system set events '604 trace name errorstack';
```

This will affect all new sessions; existing sessions and background processes are unaltered (see also <Note:65965.1>).

The complete syntax of the event specification in the parameter file and the alter session set events command is described in ksd.h (kernel service debug support)

Note: See also <Note:15159.1> and <Note:45217.1>

Setting Events

4. In another session from SQL:
   `sys.dbms_system.set_ev(sid, serial#, <event>, <level>, '<action>')`
5. In another session from a debug tool
   `oradebug <command>`

   For example:

```
SQL> oradebug dump processstate 10
```


The procedure dbms_system.set_ev allows you to set an event in another session.

Example

Enable SQL statement tracing by setting event 10046:

```
SQL> execute dbms_system.set_ev(8,219,10046,12,'');
```

Example of Using ORADEBUG

1. Locate the process you want to debug:

```
SQL> select spid from v$process
  2  where  username='usupport';
```

2. Make sure that the process exists:

```
ps -ef | grep 3514  (where 3514 is the spid)
```

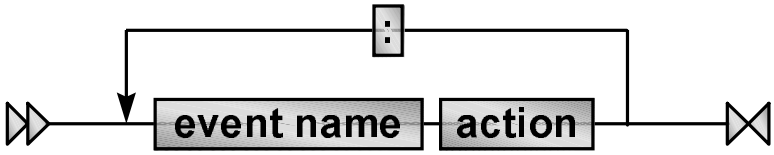
3. Use ORADEBUG to dump a process state:

```
SQL> oradebug setospid 20515
SQL> oradebug unlimit
SQL> oradebug dump processstate 10
```

Note: You can also enable tracing by using a per-process circular buffer, and you can query trace records by selecting from the x$trace fixed table. You can enable this by using the ALTER TRACING ENABLE|DISABLE "evt_type,id_low-id_high" command or by setting the initialization parameters _trace_enabled and trace_buffers_per_process. For more information, see <Note:10575.1>.

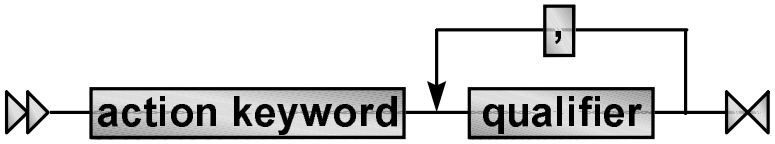
Event Specification Syntax

In init.ora and the alter session set events and the alter system set events commands:



<event name>: Symbolic name associated with the event or an event number

<action>:



Event Specification Syntax

<event name> is a symbolic name associated with the event or an event number.

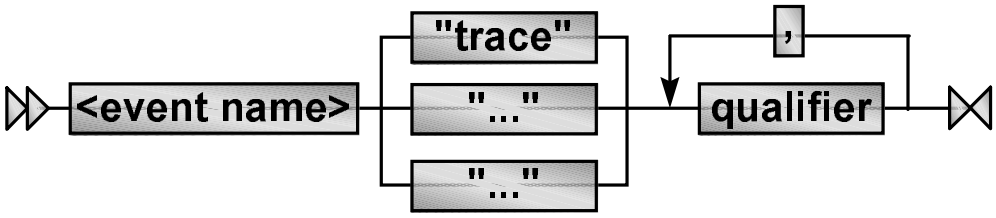
If the <event name> is a name, the parser looks it up in an event name table, if it is not "immediate". "immediate" is a special event name, indicating an immediate unconditional event. It does not wait for anyone to post it. The immediate event cannot be set in the parameter file.

If the <event name> is an event number, it is taken to be an Oracle error number (OER). Event numbers can be internal event codes, which are between 10000 and 10999. On UNIX, the internal event codes are listed in the message file `$ORACLE_HOME/rdbms/mesg/oraus.msg`. However, they are not listed in any file on Windows 95 and NT.

Note: See also <Note:9331.1> for a description of the full event syntax from ksdp.c.

Event Specification Syntax

In INIT.ORA and the alter session set events and the alter system set events command:



3-8 ORACLE

Event Specification Syntax (continued)

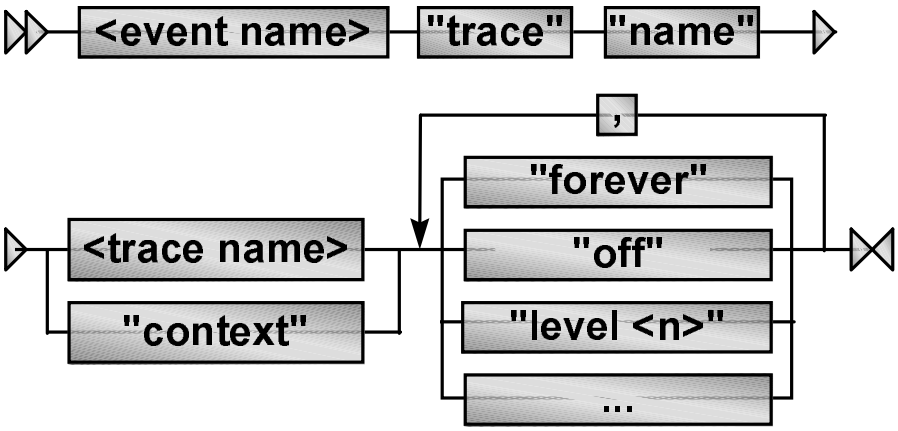
When setting events, three values for <action keyword> are supported:

- "crash" causes an Oracle crash for testing recovery.
- "debugger" invokes a system debugger if any.
- "trace" is context specific or named context-independent ones

The only value discussed in this lesson is "trace".

“trace” Event Specification Syntax

In init.ora and the alter session set events and the alter system set events commands:



3-9 ORACLE

“trace” Event Specification Syntax

The generic syntax for specific trace events is:

```
<event name> "trace" "name" <trace name> <trace qualifier> [, <trace qualifier>]...
```

The slide also shows some special values for <trace name> and <trace qualifier>.

Trace Names

<trace name> is a symbolic name associated with an internal trace id that is used to associate a trace with a (context-independent) debug dump operation.You can specify an event named “immediate.” For the case, no trace qualifiers except “level <level>” must be specified (see later in this lesson).

“context” is a special trace name.

Trace Qualifiers

“forever”: Once this trace is activated, activate it each time this event occurs.

“level <level>”: Levels are positive integers.

Typically, the higher the level setting, the more information is given. However, when performing a block dump, the level is the data block address (dba).

“off”: Disable this trace for the current event.

Setting Multiple Events

Two ways to set multiple events in init.ora:

1. **Use multiple, consecutive event lines; (simpler to comment out individual lines)**

```
event = "10015 trace name context forever"
event = "10046 trace name context forever, level 4"
```

2. **Concatenate the events with a colon (:) as the separator.**

```
event = "10015 trace name context forever: 10046
trace name context forever, level 4"
```


Setting Multiple Events

You can set multiple event lines in the init.ora parameter file, but they must be set consecutively. For example, if you want to set two events, 10015 and 10046, add the following two lines in the parameter file:

```
event = "10015 trace name context forever"
event = "10046 trace name context forever, level 4"
```

Note: If the event lines are separated by other parameters, such as

```
event = "10015 trace name context forever"
db_name = ORACLE
db_files = 20
event = "10046 trace name context forever, level 4"
```

only the second event, 10046, will be in effect.

Setting Events from SQL

dbms_system is only accessible to SYS by default; grant execute on this package to privileged users.

```
sys.dbms_system.set_ev (sid, serial#,
<event>, <level>, '<action>')
```

Example:

```
SQL> execute sys.dbms_system.set_ev -
  2  (8, 219, 10046, 12, '');
```


Setting Events from SQL

The dbms_system package is created by the dbmsutil.sql script.

- sid: The sid of the target session.
- serial#: The serial number of the target session.

The sid and serial# information can be obtained from v$session.

- <event>: The internal event codes defined in the Oracle database.
- <level>: The level of information to dump; when doing a block dump, <level> is the data block address (dba).
- '<action>': The trace name.

Example

To trace a SQL statement (event 10046 with level 12) in another session for sid 8 and serial# 219, you can issue the following command in SQL*Plus:

```
SQL> execute sys.dbms_system.set_ev -
  2  (8, 219, 10046, 12, '');
```

Setting Events from ORADEBUG

- ORADEBUG can be issued within Server Manager (<= 8.0) and within SQL*Plus (>= 8.1).
- To use ORADEBUG, you must be connected as internal or sysdba.
- "ORADEBUG help" lists all commands
- Example:

```
SQL> oradebug dump processstate 10
```


Common Mistakes Using ORADEBUG

When dumping process states, system states, and so on, the syntax should be:

```
oradebug dump <name> <level>
```

with no 'level' keyword. For example:

```
SQL> oradebug dump processstate level 1
ORA-00073: command DUMP takes between 2 and 2 argument(s)
```

The correct command is:

```
SQL> oradebug dump processstate 1
```

Do *not* use a semicolon to end your statements. In earlier versions of Server Manager, this caused ORA-73 errors. For example:

```
SQL> oradebug setospid 12345;
ORA-00073: command SETOSPID takes between 1 and 1
argument(s)
```

This is not the case with SQL*Plus in 8.1 and above.

Note: See also <Note:29786.1>.

Four Event Categories

1. Dump diagnostic information on request.
2. Dump diagnostic information when an error occurs.
3. Change database behavior.
4. Produce trace diagnostics as the database runs.

Four Event Categories

Events fall in one of the four categories, based on their usage.

Category 1: Immediate Dumps

- Immediate dump events dump diagnostic information to a trace file.
- Common immediate dumps are:
  - File header (controlf, redohdr, file_hdrs)
  - System state (systemstate)
  - Process state (processstate)

Category 1: Immediate Dumps

- The ALTER SESSION command is the most common way to set immediate dump events.

```
alter session set events
'immediate trace name <dump> level <level>'
```

- The level affects the type and amount of output produced and is specific to each dump.
- Immediate dump events cannot be invoked from the parameter file.

Category 1: Immediate Dumps

You cannot invoke immediate dump events from the parameter file. You can invoke an immediate dump by using the ORADEBUG utility or the dbms_system.set_ev( ) procedure. Here is the corresponding syntax:

ORADEBUG

```
oradebug dump <dump> <level>
```

dbms_system.set_ev( )

```
sys.dbms_system.set_ev(sid,serial#, 65535, <level>,'<dump>')
```

Note: Typically, the higher the level setting, the more information is given, such as sqlnet tracing. Refer to <Note:28988.1> for a list of common events.

Category 1: Immediate Dumps

Example:

```
SQL> alter session set events 'immediate
  2  trace name controlf level 10';
```

This produces a dump of the control file and places the output in the trace file of your process.

Immediate Dumps: Example

The control file dump can be invoked by the ORADEBUG utility or the dbms_system.set_ev( ) procedure. Here is the corresponding syntax:

ORADEBUG

```
SQL> oradebug setospid 4081
SQL> oradebug dump controlf 10
```

dbms_system.set_ev( )

```
SQL> execute sys.dbms_system.set_ev(7, 10, -
  2  65535, 10,'controlf' );
```

Note: Other common immediate dumps are: file_hdrs, redohdr, processstate, and systemstate. See <Note:28984.1> for a complete description of the control file dump.

Category 2: On-Error Dumps

- On-error dumps are similar to immediate dumps; however, they are only produced when errors occur.
- The most common one is the error stack, which produces a dump of the process call stack at the time of failure and can also dump other information.

Category 2: On-Error Dumps

On-error dumps are similar to immediate dumps; however, the dumps are only produced when a specific error occurs.

Note: Refer to <Note:37871.1> for more information on the on-error dumps.

Category 2: On-Error Dumps

- On-error dump events are most commonly activated in the parameter file.

```
event = "<error> trace name
        errorstack level <level>"
```

- The <error> can be any ORA error reported by a shadow or background process.
- The <level> is used to affect the type and amount of output produced and is specific to each dump.

Category 2: On-Error Dumps

You can also set the on-error dump event by using the alter session command or the ORADEBUG utility. Here is the corresponding syntax:

alter session

```
alter session set events '<error> trace
name errorstack level <level>'
```

ORADEBUG

After attaching to a process, execute the following command:

```
oradebug session_event <error> trace
name errorstack level <level>
```

This command sets the event on the session running in the attached process (like alter session).

```
oradebug event <error> trace name errorstack level <level>
```

This command sets the event in the currently attached process and all subsequent sessions run by that process (this is useful for MTS server processes or parallel query slaves).

Note that ORA-3113 (end-of-file on communication channel) is reported by the client and so is unsuitable for setting in an on-error event. Only server errors can be reported in this fashion.

Category 2: On-Error Dumps

Example:

```
event = "60 trace name errorstack level 1"
```

This produces an error stack in the trace file whenever an ORA-60 error is reported.

Category 2: On-Error Dumps

The example shown in the slide produces an error stack dump whenever an ORA-60 (deadlock detected while waiting for resource) error is reported.

The level for the error stack dumps are:

<level>	Description
0	Error stack only
1	Error stack and function call stack (if implemented)
2	As 1 plus the process state
3	As 2 plus the context area (all cursors and current cursor highlighted)

The corresponding alter session command for the example given above is:

```
SQL> alter session set events
  2  '60 trace name errorstack level 1';
```

The corresponding ORADEBUG command for the example given above is:

```
SQL> oradebug setospid 4018
SQL> oradebug unlimit
SQL> oradebug dump errorstack 1
```

Category 3: Change Behavior

With this event category, you can:

- **Change the Oracle server's behavior**
- **Enable hidden features**
- **Work around problems**
- **Perform specialized tuning**

Category 3: Change Behavior

- **Change behavior events are most commonly set in the parameter file.**

```
event = "<event> trace name context forever,
        level <level>"
```

- **The <event> identifies the behavior change that is required.**

Category 3: Change Behavior

Change behavior events are commonly set in the parameter file. The syntax and usage is exactly the same as for trace events, as discussed on the next page.

Example

```
event = "10269 trace name context forever, level 10"
```

Setting this event prevents SMON from coalescing free space (7.3 and above).

Category 4: Trace Events

- Trace events produce a diagnostic trace as processes are running.
- Use trace events to gather additional information so that a problem can be understood and resolved.

Category 4: Trace Events

- **Trace events are commonly set as parameters or with the *alter session* or *alter system* command.**

```
event = "<event> trace name context forever,
        level <level>"
alter session set events '<event> trace name
              context forever, level <level>';
alter system  set events '<event> trace name
              context forever, level <level>';
```

- **The <event> identifies the behavior change that is required.**
- **The <level> is used to adjust the behavior.**

Category 4: Examples

To dump the rollback segment recovery information during database startup, you can set event code 10015 (undo segment recovery) in your parameter file:

```
event = "10015 trace name context forever,level 10"
```

The 10046 trace is the equivalent of setting sql_trace = true.

The advantage of using the event is that extra details can be output to the trace file depending on the level specified with the event.

```
SQL> alter session set events
  2  '10046 trace name context forever, level 12';
```

10046 event levels are:

<level>	Description
1	Enable standard sql_trace functionality (default)
4	As level 1 plus trace bind values
8	As level 1 plus trace waits (For example, this is especially useful for spotting latch waits, but can also be used to spot full table scans and index scans.
12	As level 1 plus both trace bind values and waits

Events: Syntax Summary

Uses of diagnostic dumps and events	Event name	Trace name
Immediate dump	IMMEDIATE	<dump>
On error dump	Error#	ERRORSTACK
Change behavior	Event#	CONTEXT
Trace	Event#	CONTEXT

Location of the Events

Event# / Platform	Oracle error number	Internal event codes (10000 and 10999)
UNIX	$ORACLE_HOME/rdbms/mesg/oraus.msg	
VMS	ora_rdbms	
NT	N/A	

Common Structures
Diagnostic Events

- Diagnostic events are enabled using the event initialization parameter or the alter session or alter system commands.
- To find how diagnostic events affect code behavior, search for OER(*aaa*) with findora, where *aaa* is the event number.
- For definitions of dump types (for example, SYSTEMSTATE), search for KSDTRADV.

Example from tao

ksepec( ) is used to extract the level setting of the event. Depending on its value, the behavior is altered. In this case an additional call to ktutrc( ) is made if the event is set.

```
tao-sun% findora 10015
Searching for OER(10015)
File:  /export/home/ssupport/803/rdbms/src/server/txn/lcltx/ktu.c
Line#: 2020
       if (ksepec(OER(10015))) ktutrc("Recovering", usc);
                                                    /* trace */
File:  /export/home/ssupport/803/rdbms/src/server/txn/lcltx/ktu.c
Line#: 8785
       if (ksepec(OER(10015))) ktutrc("Recovering", usc);
                                                    /* trace */
File:  /export/home/ssupport/803/rdbms/src/server/txn/lcltx/ktur.c
Line#: 957
       if ((level = ksepec(OER(10015))) > 1)
```

How Do Events Work Internally

- Process events are initialized at process startup with the “event” initialization parameter.
- Session events are modified dynamically with an ALTER SESSION or ALTER SYSTEM command.
- Search order:
  1. Check the session events.
  2. Check the process events.

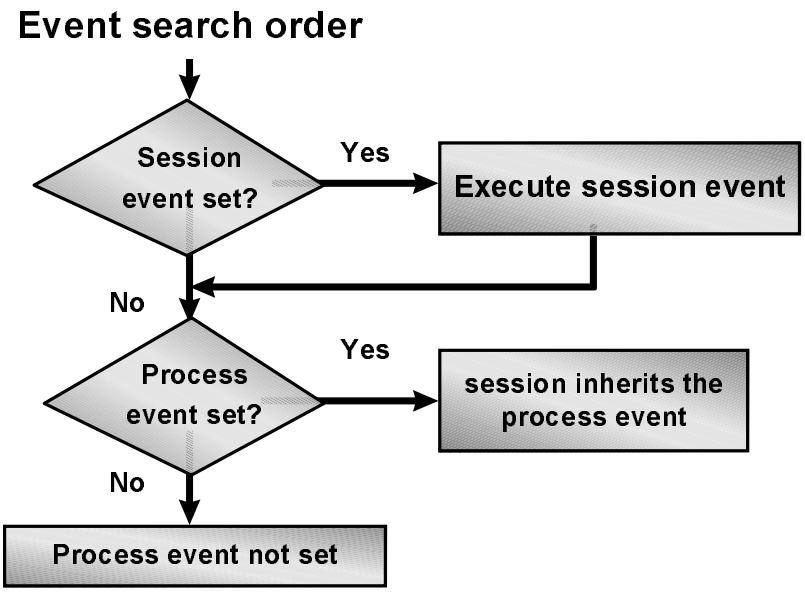
Event Groups

There are two groups of events that exist in an instance, process events and session events. Process events are initialized at process startup time with the initialization parameter “event.” Session events are modified dynamically with an alter session or alter system command.

When checking for posted events, the Oracle server first checks the session events, and then the process events.

Note: None of the debug facilities are available until after the process has actually initialized its fixed PGA part, which happens at connect time.

How Events Work Internally



How Events Work Internally

After connecting you can set a session event. The session event inherits any process events, if set.

When you disconnect, the session event disappears; however, the process event still exists.

Events and Dumps in the Source Code

- To find events in the source code, use:
  - ksepec(OER (*xxx*))
    where *xxx* is an Oracle error number
  - ksepec(OER (*yyy*))
    where *yyy* is the debug event code
- To find dumps in the source code, use KSDTRADV (KSD trace activation table entry declare variable)

```
KSDTRADV("CONTROLF", FADDR(kccdmp))
```


Finding Events in the Source Code

In the source code, search for OER(10046) or OER(600) ksepec(OER(600)) and ksepec(OER(10046)), where 600 is an Oracle error number and 10046 is a debug event code.

ksepec is defined in kse.h (Kernel Service Error manager).

Example

```
tao-sun% findora 10046
Searching for OER(10046)
File:  /export/home/ssupport/803/rdbms/src/server/progint/opi/opilof.c
Line#: 172
       if (ksepec(OER(10046))) ksdfls(); /* flush sql_trace, if enabled */
```

Finding Dumps in the Source Code

Search for KSDTRADV in the source code; it will list the trace names for the dumps.

Example

The example shown in the slide dumps control file information. KSDTRADV is defined in ksd.h (Kernel Service Debug support), “CONTROLF” is the trace name, and kccdmp is defined in kcc.c (Kernel Cache layer Control file component).

Dumps in the Source Code

1. Run cscope with the Oracle server version:

   ```
   cscope803
   ```

2. To list all the dumps, find this egrep pattern: KSDTRADV

   ```
   5 kcc.h 252
   KSDTRADV(("CONTROLF",FADDR(kccdmp))
   ```

3. Get the definition for the dump:

   ```
   Find this definition: kccdmp
   1 kcc.c <global> 4751 void kccdmp(level)
   ```


Finding Dumps in the Source Code (continued)

The following illustrates step 2:

```
Egrep pattern: KSDTRADV
  File     Line
1 kcb.h    3257 KSDTRADV("BUFFERS", FADDR(kcbdmp))
2 kcb.h    3258 KSDTRADV("SET_TSN_P1", FADDR(kcbdsts))
3 kcb.h    3259 KSDTRADV("BUFFER",  FADDR(kcbdba))
4 kcb.h    3686 KSDTRADV("RECOVERY", FADDR(kcbxre))
5 kcl2.h    919 KSDTRADV("LOCKS", kcldmp)
6 kcl2.h    920 KSDTRADV("CLASSES", kclcdp)
7 kql.h     676 KSDTRADV("LIBRARY_CACHE", FADDR(kqldmp))
8 kqr.h    1323 KSDTRADV("ROW_CACHE", FADDR(kqrdac))
9 kmmcts.h 1063 KSDTRADV("MTSSTATE", FADDR(kmmdms))

  Lines 1-9 of 70, press the space bar to display next lines
```

References

- <65965.1>: Oracle8*i* Dynamic System Wide Events
- <15159.1>: Setting Events in the Oracle Toolset
- <9331.1>: EVENT: Full Event Syntax (from ksdp.c)
- <45217.1>: EVENT: Summary Event Syntax for WWCS
- <29786.1>: SUPTOOL: ORADEBUG 7.3+ (Server Manager Debug Commands)
- <10575.1>: TRACING FOR INSOMNIAC
- <28988.1>: EVENT: LIST – List of Common Events
- <28984.1>: EVENT: CONTROLF – Obtaining & Interpreting ControlFile Dumps
- <37871.1>: EVENT: ERRORSTACK – *How to get Tips/Techniques for Debugging*, by Sanjay Gupta
- *Oracle Backup & Recovery Handbook*, by Rama Velpuri

Practice 3 Overview

This practice covers the following topics:

- **Dumping diagnostic information on request**
- **Dumping diagnostic information when an error occurs**
- **Producing trace diagnostics**
- **Looking up the source code for a special event**

Practice 3 Overview

For detailed instructions on performing this practice, see Practice 3 in Appendix B.

4

Hang, Loop, and Crash Diagnostics

ORACLE

Schedule:	Timing	Topic
	45 minutes	Lecture
	15 minutes	Examples
	60 minutes	Lab Exercises
	120 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Analyze trace files and stack traces resulting from a crash**
- **Describe state dumps and their use in analyzing hang or loop scenarios**
- **Query relevant data dictionary tables and views**

Overview

- Common failures
- Diagnostics
- State objects and state dumps
- Data dictionary tables and views

Overview

If an Oracle process or an Oracle instance terminates abnormally, before the actual exit of the process or instance, the Oracle server generates a lot of diagnostic information. In most situations, this diagnostic information, which is available as trace files at various locations, is sufficient to determine the cause of the failure. For example, when a user process dies abnormally, the RDBMS generates a trace file in user_dump_destination with a filename that looks like:

```
ora_<process_id>.trc
```

The Orace RDBMS also provides tools to generate this kind of tracing information when the process or instance is running normally. This can be done by setting events at various levels (call, session, instance). These events also generate similar tracing information.

There are also situations where the RDBMS diagnostic information does not cover all the aspects of the required diagnostics. In such situations, you must resort to other sources, such as OS, hardware traces, and so on. These, in conjunction with Oracle diagnostics, can help you determine the cause.

Common Failures

- An instance or process crash:
  - Internal errors (ora-600)
  - Operating system violation
    - Segmentation violation, bus errors (UNIX)
    - Access violation (NT)
- Hang:
  Waiting for an event that is not going to happen

Common Failures

Operating system violations are exceptions at the operating system level. There are many types of OS violations, for example:

- Segmentation violation: Improper access of a segment (data, stack, or text)
- Access violation: Improper access of a memory area (within a segment)
- Bus error: More generic error; could be either of the above

Common Failures

- **Loop:**

  A process or instance is not hanging, but is repeating some operation infinitely.

- **Slow performance:**

  Nothing is hanging, but the performance is unacceptable. OS utilities show that the processes are gaining CPU time.

Common Failures (continued)

The difference between a loop situation and a hang situation is that there is some activity; a process or instance is repeating some operation infinitely. In a hang situation you see 0% CPU usage and no activity; in a loop situation you typically see 100% CPU usage and a large amount of activity.

Diagnostic Files

- The `alert.log` file
- Trace files
- Application log files like SQL*Plus trace files
- Core dump files
- System log files

Diagnostic Files

Note that a core file is useless without the executable that generated the core file. If a usable trace file (with a call stack) is available, core files will not supply additional useful information. However, when a trace file is missing, a call stack can be generated from the core file using a debugger at the operating system level.

Note: Depending on the problem, one or more of the above files may not exist.

The `alert.log` File

Every instance generates a file called `alert.log`, which logs the following information:

- **Diagnostic data from background and foreground processes**
- **Summary information regarding errors and pointers to trace files for detailed information**
- **Information since database creation (unless purged) that might be useful in backtracking a problem**

The `alert.log` File

The `alert.log` file is written in the `background_dump_dest` directory and follows the naming convention `alert<ORACLE_SID>.log`. Here is an example `alert.log`:

```
-------------------------------------------------------------------------
Tue Jun 24 09:54:10 1999
Starting ORACLE instance (normal)
System parameters with non-default values:
  processes                = 50
  shared_pool_size         = 3500000
  user_dump_dest           = /u06/oracle/app/oracle/admin/V815/udump
PMON started
DBWR started
create database "V815"
    maxinstances 8
    maxlogfiles  32
    national character set "US7ASCII"
    datafile
     '/u06/oracle/oradata/V815/system01.dbf'   size   80M
    logfile
     '/u06/oracle/oradata/V815/redoV81501.log' size 500k,
     '/u06/oracle/oradata/V815/redoV81502.log' size 500k,
```

Trace Files

- Every server process, on encountering an exception, writes diagnostic data to a trace file.
- The trace file header contains the following information:
  - OS and version
  - Oracle version and options installed
  - Instance name
  - Process ID

Trace Files

Here is a sample trace file:

```
--------------------------------------------------------------------------
Dump file /u06/oracle/app/oracle/admin/V815/udump/ora_13758.trc
Oracle8 Enterprise Edition Release 8.0.5.0.0 - Production
With the Partitioning and Objects options
PL/SQL Release 8.0.5.0.0 - Production
ORACLE_HOME = /u06/oracle/app/oracle/product/8.0.5
System name:  HP-UX
Node name:    tchp
Release: B.10.20
Version: A
Machine: 9000/867
Instance name: V815
Redo thread mounted by this instance: 1
Oracle process number: 9
Unix process pid: 13758, image: oracleV815
Wed Sep 24 13:40:43 1999
*** SESSION ID:(8.3) 1999.09.24.13.40.43.018
Exception signal: 11, code = 0
sc_onstack: 0000000000000000, sc_mask 0000000000000000
pcsqh: 0x000040fc pcoqh: 0x00343053  pcsqt: 0x000040fc  pcoqt: 0x00343057
```

Trace Files

- One of the most important pieces of information in the trace file is the stack trace.
- Stack trace represents the order in which calls were made by the offending process before it crashed.
- Use a stack trace in conjunction with source code to understand the problem.

Trace Files (continued)

Here is an example of a stack trace:

```
----- Call Stack Trace -----

calling             call     entry              argument values in hex
location            type     point              (? means dubious value)
------------        -------- ---------          ----------------------
_ksedmp+94          CALLrel  _ksedst+0
                                                F9D8A0
_ksfdmp+e           CALLrel  _ksedmp+0          3
_kgerinv+8e         CALLreg  00000000
                                                53F18
                                                3
_kgesinv+19         CALLrel  _kgerinv+0
                                                0BC80
                                                0
                                                1015C48C
```

If it is a known problem, then you may be able to find match in GSX or WebIV for the above stack. Enter the functions in the calling location as the search criteria. The rule of thumb is to ignore the top two to three functions on the stack (these are the kse routines that are called when an exception is encountered). The prefix kse stands for kernel service error.

Application Log Files

- Application logs have information about exceptions encountered on the user side; for example:
  - Spool files from a SQL*Plus session
  - Export/import log files
- A core file (UNIX) is a process memory dump at failure; you can extract the stack trace from a core file using a debugger such as adb.

System Log Files

- System log files capture error messages and exceptions encountered at the OS level.
- These would be useful if a hardware or OS problem is suspected.

System Log Files

On Solaris, for example, look at the `/var/adm/messages` file for system error messages. Check with the product line or the operating system vendor for the location and name of the operating system message file. Some typical examples of errors at the OS level are paging failures, I/O errors, network errors, and swapping errors.

Example of a /var/adm/messages File

```
Sep 14 10:52:33 infra01 unix: NOTICE: quota_ufs: Warning: over disk limit
(pid 1 5606, uid 19113, inum 37358, fs /home)
Sep 14 10:55:42 infra01 inetd[1294]: printer/tcp: bind: Address already in use
Sep 14 11:16:08 infra01 inetd[1294]: printer/tcp: bind: Address already in use
Sep 14 11:16:46 infra01 unix: NOTICE: quota_ufs: Warning: over disk limit
(pid 1 8298, uid 34915, inum 1272507, fs /home)
Sep 14 11:46:48 infra01 inetd[1294]: printer/tcp: bind: Address already in use
Sep 14 11:53:03 infra01 unix: NOTICE: quota_ufs: over disk and time limit
(pid 2 4428, uid 13952, inum 1542289, fs /home)
Sep 14 11:56:58 infra01 inetd[1294]: printer/tcp: bind: Address already in use
Sep 14 12:10:25 infra01 unix: NOTICE: quota_ufs: over disk and time limit
(pid 2 4401, uid 13952, inum 1542526, fs /home)
```

Hang Situations

- The process is waiting for something that would never happen.
- To find the event, the most useful tools are:
  - Multiple state dumps
  - v$session_wait, v$lock, v$latch, v$latchholder
- The views or the state dumps reveal the information leading to the hang.

Hang Situations

A hang situation is a CPU polling situation. There is almost no CPU usage. You should look into process state dump, system state dump, and error stacks. Look for WAIT events that do not resolve themselves. Verify that the database is hung, not other applications. V$SESSION_WAIT is a good source of information at the database level in this situation. Useful columns to select from this view are:

- sid: Session id (v$session)
- seq#: The sequence number of the wait for this session
- event: The name of the event that the session is waiting for or has just finished waiting for
- wait_time: The time waited for this event
- seconds_in_wait: The approximate wall clock time in seconds at the start of the wait
- state: The following values are possible:

State	Value	Meaning
WAITING	0	The session is currently waiting for this event.
WAITED UNKNOWN TIME	-2	Timing is not enabled.
WAITED SHORT TIME	-1	The session woke up the same clock tick as it went to sleep (<10 ms).
WAITED KNOWN TIME	> 0	Session waited wait_time.

Looping

- The process is looping on something that is not going to happen, such as acquiring a resource that is held by another state object.

  These occur in system states as *wait events.*
- To find the event, use tools such as:
  - Multiple state dumps, error stacks, bugs
  - v$session_wait, v$lock, v$latch, v$latchholder
- The views or the dumps should reveal the information leading to the looping behavior.

Looping

To get a system state dump, do the following:

```
SQL> alter session set events
  2  'immediate trace name systemstate level 10';
```

Alternatively, attach to the process using ORADEBUG as follows:

```
SQL> oradebug setospid <PID>
SQL> oradebug dump systemstate 10
```

You can obtain additional information from views like v$session_wait.

Diagnosing Looping Situations

To identify a looping problem:

- **Take multiple system state dumps at frequent intervals.**
- **Get a stack trace by attaching to the process through ORADEBUG.**
- **Query relevant data dictionary views at frequent intervals and look for changes in the process state.**

Diagnosing Looping Situations

The output below resulted from processing a system state dump with the ass.awk tool (See WebIV for usage and availability). This tool is designed to extract important information from these dumps. Note the repeated attempts to acquire the row cache locks 333b5f30 and 336e6540. The resource, holder, and state section indicates a classic deadlock scenario.

```
Starting Systemstate 1
~~~~~~~~~~~~~~~~~~~~~
1:
2:  waiting for 'pmon timer'
3:  waiting for 'rdbms ipc message'
4:  ...
29: waiting for 'row cache lock'       [Rcache object=333b5f30,]
30: waiting for 'row cache lock'       [Rcache object=336e6540,]
31: waiting for 'row cache lock'       [Rcache object=333b5f30,]
32: waiting for 'row cache lock'       [Rcache object=336e6540,]
33: waiting for 'row cache lock'       [Rcache object=333b5f30,]
34: last wait for 'db file sequential read' (2d,29a1a,1)
35: waiting for 'row cache lock'       [Rcache object=333b5f30,]
     Resource                    Holder   State
     Rcache object=333b5f30,     15:      15: is waiting for 12:
     Rcache object=336e6540,     12:      12: is waiting for 15:
```

Slow Performance

- A process might just be slow.
- If the process is gaining CPU time, it is not hung; could either be looping or just too slow.

Slow Performance

- If a process is not hung and not looping it is probably just too slow; this needs tuning.
- You can tune at different levels, including the hardware, database, and application level.
- Analyzing BSTAT/ESTAT and TKPROF output and understanding the application and the database layout are a good starting point.

Slow Performance

On UNIX you can do a `'ps -ef | grep <PID>'` and look at the time field value:

- If the value does not increase, then the process is not gaining any CPU time.
- If the value is increasing, then the process is not hung.

This is a typical performance issue: the components of this process (application, network, database) need to be tuned. You should look for contention, locking issues, and disk I/O issues.

A good diagnostic is the BSTAT/ESTAT report. Diagnosing and tuning performance problems are beyond the scope of this lesson.

State Objects

- State objects are structures in the SGA associated with various database entities, such as:
  - Background and foreground processes
  - Sessions
  - Latches and enqueues
- There are several types of state objects, including process, session, and transaction state objects.
- Each state object is designed for a specific purpose.
- They are used by PMON to clean up resources in case of failures.

State Objects

The information in the system state is vast and often confusing. Refer to appendix A for more information on system states. Take a few system state dumps and try to identify some of the most useful pieces of information. System state dumps become clearer as you read a few of them.

Process State Objects

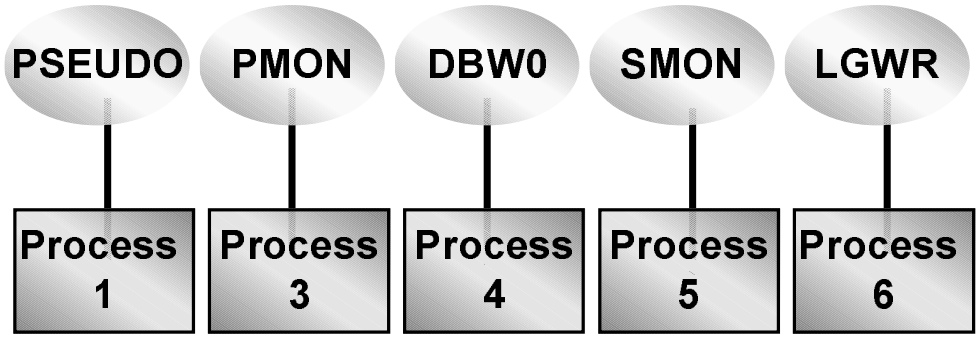
- Each Oracle background and foreground process has a process state object.
- There is an additional state object for the PSEUDO process that is used in the MTS environment.

Process State Objects

There is a one-to-one correspondence between Oracle processes and state objects. The only exception is the PSEUDO process, which is used to hold sessions detached by a user during session migration. A detached user session is owned by the PSEUDO process until some other process switches to it, in which case it is moved to the state object of the new process.

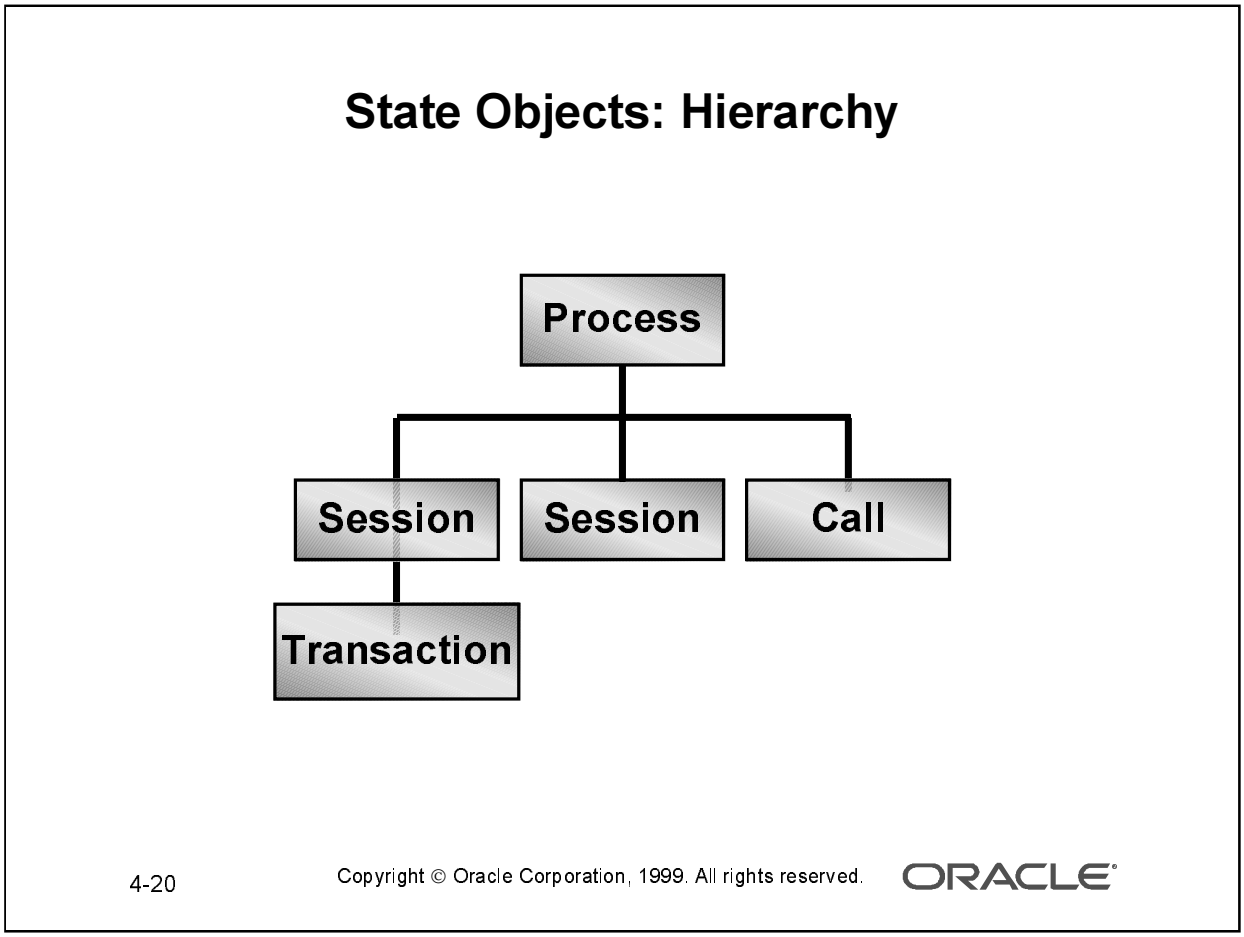
Process State Objects

Processes:



Process State Objects (continued)

The PSEUDO process is used in the MTS environment to accommodate session migration. There may be platform-specific differences in the numbering of the processes.



State Objects: Hierarchy

The hierarchy shown in the slide illustrates a typical process state. Note that the process is running a session. This session has an open transaction.

Note: Typically, a process has only one session object. Processes with multiple sessions are usually found in XA and MTS environments or OCI applications.

State Dumps

- You can dump state objects to a trace file to diagnose a problem.
- There are two types of dumps:
  - Process state dumps
  - System state dumps
- A process state dump is a dump of all state objects for a process.
- A system state dump is a dump of all state objects for all the processes on the system.

State Dumps

If you have isolated the problem to a particular process, then a process state dump may be sufficient to diagnose the problem. Note, however, that the problem may be somewhere else, in which case a complete system state dump (usually a few of them) is required to isolate the offending process.

Process State Dumps

Dump a process state:

```
SQL> alter session set events 'immediate
  2  trace name processstate level 10' ;
```

Dump a process state on error (from init.ora):

```
<error#> "trace name processstate level 10"
```

Dump a process state of a currently running process:

```
SQL> oradebug setospid <process id>
SQL> oradebug dump processsate 10
```

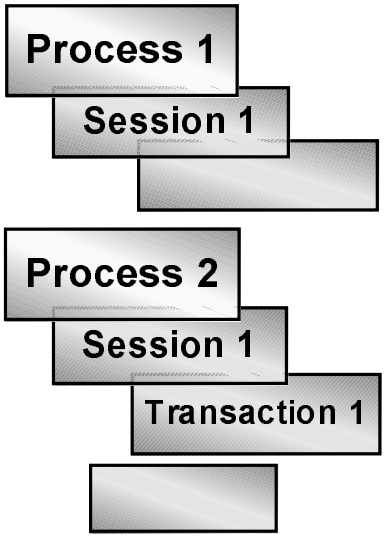

Example of a Process State Dump

```
PROCESS STATE
-------------
SO: 8004790c, type: 3, owner: 8003cbb4, flag: INIT/-/-/0x00
    (session) trans: 0, creator: 8003cbb4, flag: (41) USR/- -/-/-/-/-/-
              DID: 0001-0008-00000005, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    O/S info: user: dsi, term: pts/1, ospid: 12129, machine: ccd-orsun3
              program: svrmgrl@ccd-orsun3 (TNS V1-V3)
    last wait for 'SQL*Net message from client' seq=234 wait_time=-2
                driver id=62657100, #bytes=1, =0
-------------------------------------------------------------
```

Component	Definition
SO	State object address
type	Index entry for object type, printed on next line (process, session, and so on)
owner	State object address that owns this state object
flag	INIT: object is initialized; FLST: on freelist; CLN: object freed by PMON
DID	Resource ID (Oracle8 only)

System State Dumps

A system state is a process state dump of all the processes for the instance.



ORACLE

System State Dumps

- **Dump a system state immediately:**

```
SQL> alter session set events
  2  'immediate trace name systemstate
  3  level 10' ;
```

- **Dump a system state on error (from init.ora):**

```
'<error#> trace name systemstate level 10'
```

Data Dictionary Views

The following data dictionary views are useful when diagnosing a hang or loop situation:

- **v$session_wait, v$session_event**
- **v$latch, v$latchholder, v$latchname**
- **v$sysstat, v$lock**
- **$process, v$session, v$transaction**
- **x$kcbfwait (buffer waits), x$ksqst (enqueues)**

Data Dictionary Views

An example from v$session_wait:

```
SQL> select sid, seq#, event wait_time, seconds_in_wait, state
  2  from   v$session_wait;

SID  SEQ#  EVENT                       WAIT_TIME  STATE
---  ----  -------------------------  ---------  -------
  1  3335  pmon timer                 381684631  WAITING
  2  3352  rdbms ipc message              10031  WAITING
  3  3391  rdbms ipc message               9729  WAITING
  6    15  rdbms ipc message               9975  WAITING
  4  6666  rdbms ipc message                  0  WAITING
  5    75  smon timer                     2532  WAITING
  7    65  SQL*Net message from client        0  TIME
```

References

- *Oracle Diagnostic Guide Presentation*, by Richard Exley
- *State Dump Analysis*, by Russell Green and Richard Exley
- *Understanding System Dumps*, by Deana Holmer
- UK notes 33372.1, 33368.1
- *Session Wait Information in Oracle7*, by Anjo Kolk

Practice 4 Overview

This practice covers the following topics:

- **Identify and resolve database hanging situations.**
- **Identify and resolve database looping situations.**

Practice 4 Overview

For detailed instructions on performing this practice, see Practice 4 in Appendix B.

5

Heap Corruption Diagnostics

ORACLE

Schedule:	Timing	Topic
	45 minutes	Lecture
	15 minutes	Examples
	60 minutes	Lab Exercises
	120 minutes	Total

Objectives

After completing of this lesson, you should be able to do the following:

- **Explain heap corruptions**
- **Identify when a heap corruption has occurred**
- **Use initialization parameters and diagnostic events to help resolve heap corruptions**

Note: Heap corruptions can be seen as a special case of a crash.

Oracle Server Memory Management

Oracle server processes use the following memory structures, among others:

- **PGA: Allocated and used by one process**
- **UGA: Allocated when a session starts and used by that session**
- **SGA: Allocated at startup and accessible by all processes**

Oracle Server Memory Management

Oracle server memory is managed by a generic heap manager that provides such services to its clients as allocation, freeing, and locating memory. The following basic classes of memory are grouped by the required persistence of that memory:

- PGA memory is only accessed by the Oracle process that allocates it, and can be released when the Oracle process finishes. PGA memory dynamically grows and shrinks during the life of the Oracle process.
- UGA memory is associated with a particular session and is only accessed by the Oracle process that is currently running the session. If the session can migrate from one process to another (for example, in a multi-threaded server environment), then the UGA must be globally accessible. UGA memory requirements grow and shrink dynamically during the life of the session.
- SGA memory is accessed by all Oracle processes.

UNIX Implementation

An Oracle instance is implemented as a collection of separate processes, with each Oracle process corresponding to a UNIX process.

SGA memory is implemented as a shared memory area, which is allocated at instance startup and remains fixed in size. This is typically known as the SGA even though it may also contain UGA memory.

Oracle Server Memory Management

The heap implementation is platform-specific; for example:

- **On UNIX, UGA and PGA are allocated from the process heap; SGA is shared memory and is allocated at instance startup.**
- **On NT, SGA, PGA, and UGA memory is allocated from the process heap**

Oracle Server Memory Management (continued)

UNIX Implementation (continued)

UGA memory is allocated out of the SGA if a session can migrate; otherwise it is implemented as a UNIX process heap.

PGA memory is implemented as a UNIX process heap. One side effect of this is that UNIX PGA can *only* be accessed by the owning process.

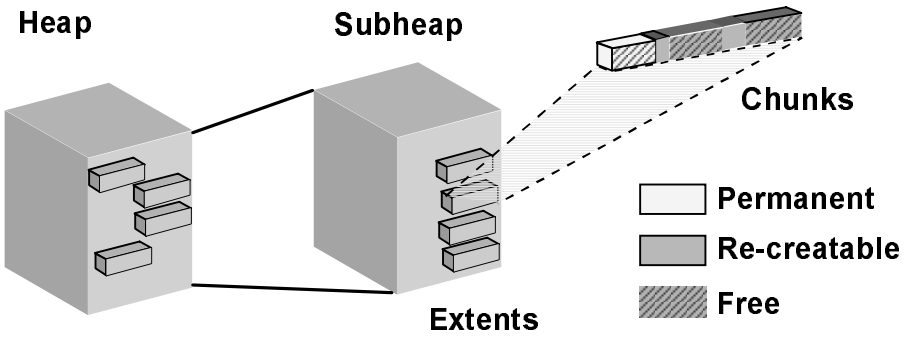
NT Implementation

An Oracle instance is implemented as a single NT process, with each Oracle process implemented as a single thread.

SGA, PGA, and UGA memory is allocated from the process heap and so is accessible by all threads.

Note: See kgh.h for details on how kgh works.

Oracle Server Memory Management



- A heap is composed of one or more contiguous memory areas called extents, and memory is allocated from extents in chunks.
- A chunk from one heap may contain another heap; this is known as a subheap.

Oracle Server Memory Management (continued)

Oracle server memory is managed by a generic heap manager. When a user asks for a chunk of space, that chunk comes from a particular heap. Within a heap, the user can ask for space that is permanently allocated in the heap or for space that can be freed and reused.

When a chunk of space that may be freed is allocated from a heap, it is possible to specify that the contents of the chunk are RECREATABLE. If this option is specified, then the chunk of space can be explicitly UNPINNED when not in use.

When a user requests space from a heap and no more space is available, the heap manager can use a callback routine to request that the owner of an unpinned, re-creatable chunk of space free the chunk. Unpinned chunks of space are kept on an LRU list so that the heap manager can determine which one to try to free up first.

A heap is composed of a set of contiguous chunks of space called extents. When the user asks for a chunk of space from a heap, the heap manager looks in the set of extents contained in the heap for an unused piece of space of the requested size. If one cannot be found, then the heap manager uses a callback to request a new extent, and adds it to the heap.

If the extents of a heap are allocated from another heap, then the heap is said to be a subheap of the other heap, the parent heap.

Note: See kgh.h for details on how kgh works.

The Potential for Corruption

- Memory corruption, known as heap corruption in the RDBMS, occurs when the wrong data is written to memory.
- It is most commonly caused by a rogue process accidentally updating the wrong memory location.
- Heap corruption can happen in private or shared memory.
- Shared memory corruption can and usually does impact more than one process.

The Potential for Corruption

A heap is the section of memory in which addresses are stored. These addresses can be interpreted as pointers to structures or other addresses. These addresses and pointers constantly change as needed during run time. A heap corruption occurs when the pointers or addresses are overwritten when they are not yet ready to be freed. When the Oracle server tries to read that part of the heap and encounters incorrect information, errors will be signalled.

Heap Corruption Symptoms

- The RDBMS, and software in general, normally works predictably and delivers repeatable results.
- Heap corruption, especially in shared memory, destroys this predictability by allowing separate code areas or processes to interact in an uncontrolled way.
- This unpredictability is a key characteristic of heap corruption.

Heap Corruption Symptoms

- Heap corruption typically causes random and inconsistent crashes, including:
  - ORA-600 errors
  - Core dumps (UNIX)
  - GPFs (NT)
- Restarting the instance offers temporary relief; the symptoms return when the “corrupter” reintroduces the corruption.

Symptoms: ORA-600

- **ORA-600 errors resulting from the failure of a consistency check are typical in heap corruption situations.**
- **Any random heap corruption would be likely to cause inconsistency and lead to this type of failure.**

Example

kcfdfk is an SGA memory variable, that stores the maximum number of data files allowed in the database.

```
# define kcfdfk   KSMSGADN(kfil, kcfdfk_)
KSMSGADV(kfil,    kcfdfk_)      /* SGA: max # db files */
```

If this memory location was corrupted–for example, changed to 0–an attempt to read a block from any file would fail because the file number was greater than the kcfdfk value.

```
ORA-00600: internal error code, arguments: [2845], [1], [0], [1095]
```

Here is the code:

```
ASSERT3(fno && fno <= kcfdfk && bno > 1, OERI(2845),KSESVSGN, fno, kcfdfk, bno);
```

Note that ORA-600 errors can also be the result of many problems, such as code bugs or disk corruptions.

Every heap has a header, and when the RDBMS accesses a heap, a sanity check is performed on this header. If the header is corrupted, the RDBMS generates a dump.

Symptoms: Core Dumps/GPFs

- UNIX: Core dump ORA-7445
  - SIGSEGV: signal 11
  - SIGBUS: signal 10
- NT: GPF (ACCVIO)
- These errors indicate that a process has attempted to address outside its memory space.
- The most likely cause is the corruption of a memory pointer.

Example

kcbbbh is an SGA memory variable that contains a pointer to the location of the buffer headers in the buffer cache.

```
#define kcbbbh   KSMSGADN(kcbbh**, kcbbbh_)
KSMSGADV(kcbbh**, kcbbbh_)      /* buffer header base */
```

If this variable were to be corrupted, the RDBMS, unaware of the corruption, would use the corrupted value and most likely attempt to address outside its available memory.

```
ORA-07445: exception encountered: core dump [] [SIGBUS]
[Invalid address alignment] [11111595] [] []
```

Note: A code bug that resulted in a miscalculated memory address could also result in similar behavior.

Heap Corruption Diagnostics

It can be difficult to find the cause of heap corruptions because:

- **Heap corruptions can persist for some time before they are discovered**
- **In shared memory, the discoverer is often different from the corruptor**

Heap Corruption Diagnostics

Heap corruptions are often difficult to resolve because the corruption can persist for some time before it is discovered, hence obscuring the cause.

In shared memory, the discoverer is often different from the corruptor, making the cause even more difficult to find.

Heap Corruption Diagnostics

To identify the corrupter, you can use three methods to force failures at the point of corruption:

- **The _db_block_cache_protect parameter**
- **Event 10235**
- **Event 10049**

_db_block_cache_protect

_db_block_cache_protect = true

protects the buffer cache from accidental writes:

- **Each buffer is made read-only on startup.**
- **Each buffer must be explicitly set to read-write before an update and then returned to read-only afterwards.**

The _db_block_cache_protect Initialization Parameter

The availability of this feature depends on the OS being able to offer memory protection and the Oracle kernel implementing it in the OSD routines. _db_block_cache_protect is *not* implemented on most platforms; that is, setting it to true will have no effect. On most UNIX ports, the implementation depends on the availability of the mprotect( ) system call.

For example, this functionality is implemented on OpenVMS and Digital UNIX, but not supported on Windows NT.

The default value is false.

Note: This feature can incur additional memory and CPU overhead, so use it with caution. Refer also to <Note:18144.1>.

Diagnostics: Event 10235

- "Check memory manager internal structures"
- Enables additional checking and processing in the heap manager
- Enables memory protection on some platforms (higher levels)
- Detects heap corruption more quickly and so closer to when the corruption occurred

Event 10235

This event ("check memory manager internal structures") can be used to try to catch heap corruption problems closer to when they occur.

Syntax

```
event = "10235 trace name context forever, level <level>"
```

<level>	Description
1	Fast check on heap free (kghfrh)
2	Do 1 and fill memory with junk on alloc/free
3	Do 2 and ensure the chunk belongs to given heap on free
4	Do 3 and make permanent chunks freeable so they can also be checked

Higher levels are bitwise triggers:

8xN	Do 4 and check the given heap on every operation
16xN	Also check top PGA (recursively) and top SGA (non-recursively) on each operation

Note: This event can incur additional CPU and contention overhead, so use it with caution. Refer to <Note 21208.1> for details.

Diagnostics: Event 10049

- "Protect library cache memory heaps"
- You can use this event to enable library cache object protection.

Event 10049

The event ("protect library cache memory heaps") enables library cache object protection.

Syntax

```
event = "10049 trace name context forever, level <level>"
```

<level>	Description
10<level<10000	Heaps in unpinned objects are given no access
10000	The library cache manager never keeps read-only unpinned/unlocked/unreferenced library objects in the cache
10001	The library cache manager never keeps non-read-only unpinned/unlocked/unreferenced library objects in the cache
10099	The library cache manager never keeps unpinned/unlocked/unreferenced library objects in the cache
10100	The library cache manager print a warning in the trace file if an object load fails due to insufficient shared memory

…

Note: This event can incur additional memory and CPU overhead, so use it with caution. Refer to <Note 21157.1> for details.

Memory Dumps

- Gather more evidence by dumping memory.
- Examples:
  - heapdump: Dump PGA, SGA, UGA
  - heapdump_addr: Dump a subheap
  - row_cache: Dump the dictionary cache
  - buffers: Dump the buffer cache
  - library_cache: Dump the library cache

Heap Dumps

There are three ways to dump heaps:

- Using the alter session command
- Using the ORADEBUG utility
- Using the event initialization parameter

Memory Dumps

A heap dump consists of three parts:

- **Dump of the heap descriptor**
- **Dump of the chunks within each extent on the heap's extent list**
- **Dump of free lists, lru lists, permanent chunk lists, and marked chunk lists**

Heap Dump Example

First Part

The first part of a heap dump is a dump of the heap descriptor:

```
HEAP DUMP heap name="session heap"  desc=0x1229190
extent sz=0x108c alt=32767 het=32767 rec=0 flg=3 opc=3
parent=faec44 owner=80024858 nex=0 xsz=0x1090
```

Second Part

The chunks for each extent are dumped with address, size, type, allocation comment, and extra information that corresponds to the chunk type:

```
EXTENT 0
 Chunk  1254500 sz=   1752  free      "               "
EXTENT 1
 Chunk  125506c sz=    604  free      "               "
...
EXTENT 4
 Chunk  125336c sz=   3348  perm      "perm           "  alo=3348
 Chunk  1254080 sz=     68  free      "               "
 Chunk  12540c4 sz=     20  freeable  "frame segment  "
 Chunk  12540d8 sz=     36  freeable  "frame          "
...
```

Second Part (continued)

If there is a corrupted chunk header in the heap dump, kgh writes a "bad magic number" message in the chunk's record and stops scanning the event:

```
   Chunk  1254294 sz=      104    freeable  "kgiob          "
   Chunk  12542fc sz=      104    freeable  "kgiob          "
   Chunk  1254364 sz= 57173760    ERROR, BAD MAGIC NUMBER (6b676800)
```

Third Part

The third part of the heap dump is the dump of free lists, lru lists, permanent chunk lists and marked chunk lists. The chunks on these lists should already have been dumped in the extent list:

```
FREE LISTS:
  Bucket 0 size=76
    Chunk  1254080 sz=       68    free      "               "
  Bucket 1 size=140
  Bucket 2 size=268
    Chunk  1254d98 sz=      292    free      "               "
Total free space   =     2716
  UNPINNED RECREATABLE CHUNKS (lru first):
  PERMANENT CHUNKS:
    Chunk  12552dc sz=     3512    perm      "perm           "  alo=300
    Chunk  125336c sz=     3348    perm      "perm           "  alo=3348
...
Permanent space    =    16824
```

Note: Refer to <Note:61866.1> for more information.

Memory Dumps

- heapdump: Dump PGA, UGA, and SGA
- Syntax:

  ```
  alter session set events 'immediate
  trace name heapdump level <level>'
  ```

  Dump on a specific error:

  ```
  alter session set events '<error#>
  trace name heapdump level <level>'
  ```

- "With contents" dumps a hex dump of each memory chunk on the heap.

Heap Dump Level Settings

Note the new heap dump level settings in Oracle8:

Heap description	<level> Hex | Dec	<level> with contents Hex | Dec
Top PGA	0x01 1	0x401 1025
Top SGA	0x02 2	0x802 2050
Top UGA	0x04 4	0x1004 5000
Current call	0x08 8	0x2008 8200
User call	0x10 16	0x4010 16400
Large pool	0x20 32	0x8020 32800

Example

```
SQL> alter session set events 'immediate trace name
  2  heapdump level 1';
```

Note: Refer to <Note:39686.1> for more information.

Memory Dumps

- **heapdump_addr: Dump a subheap**
- **Syntax:**

```
alter session set events 'immediate
trace name heapdump_addr <addr>'
```


Subheap Dumps

The heapdump_addr event was introduced in 7.3 when the heap dump event level numbering changed to allow subheaps to be dumped.

Items in the heapdump can themselves be heaps. These usually have a label similar to “heap” or “hp”.

Example

UGA dump:

```
Chunk 200ddbcc sz=560  freeable  "bind var heap  " ds=200c7c68
```

Dump the subheap as follows:

1. Convert the ds entry (200c7c68) to decimal.
2. Issue the following command:

```
SQL> alter session set events 'immediate trace name
 2   heapdump_addr 537689192';
```

Note: Refer to <Note:47632.1> for more information.

Memory Dumps

- row_cache: Dump the dictionary cache
- Syntax:

```
alter session set events 'immediate
trace name row_cache level <level>'
```


Row Cache Dumps

Row cache dumps write the entire row cache to the user trace file.

Note: Refer to <Note:47398.1> for more information.

Memory Dumps

- buffers: Dump the buffer cache
- Syntax from a SQL session:

```
alter session set events 'immediate
trace name buffers level <level>'
```

Dump buffers on error:

```
event = '<error#>
trace name buffers level <level>'
```


Buffers Dumps

You can use the buffers dump to dump the buffer cache with various amounts and types of information.

Buffers description	<level> Headers	<level> Brief Block	<level> Full Block
Block	1	2	3
and latch lru	4	5	6
and users/waiters	8	9	10

Note: Refer to <Note: 43143.1> for more information.

Memory Dumps

- library_cache: Dump the library cache
- Syntax:

```
alter session set events 'immediate
trace name library_cache level <addr>'
```


Library Cache Dumps

Specify an address or use level 10 to get a full library cache dump.

Note: Refer to <Note 35053.1> for more information.

Resolution Techniques

- The key to resolution is to find the corrupter.
- This is often the process, job, query, or application that ran immediately before the corruption was reported.
- Spotting the link is the key; diagnostics can be used where necessary.

Resolution Techniques

The key to resolution is to find the corrupter. When you have identified the corrupter, a test case allows product development to identify the cause and create a fix. As an example, try finding out if the corruption always happens at a certain time; for example, when a batch job is being run.

References

- 18144.1: Parameter: db_block_cache_protect (hidden)
- 21208.1: Event 10235
  "Check memory manager internal structures"
- 21157.1: Event 10049
  "Protect library cache memory heaps"
- 61866.1: Internal: Debugging KGH
- 39686.1: Event heapdump: How to get
- 47632.1: Event heapdump_addr
  Dumping subheaps by address
- 47398.1: Event row_cache: How to get
- 43143.1: Event buffers
  How to dump the entire buffer cache
- 35053.1: Event library_cache: How to get

Practice 5 Overview

This practice covers locating heap corruptions.

Practice 5 Overview

For detailed instructions on performing this practice, see Practice 5 in Appendix B.

6

Block Dump Analysis

ORACLE

Schedule:	Timing	Topic
	60 minutes	Lecture
	15 minutes	Examples
	45 minutes	Lab Exercises
	120 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Dump Oracle blocks at the OS level**
- **Interpret formatted dumps**
- **Compare OS and formatted block dumps**
- **Identify key data structures in Oracle7 and Oracle8 blocks**

RDBMS Data Blocks

- The RDBMS stores data in data blocks.
- One data block corresponds to a specific number of bytes of physical database space on disk.
- Space is allocated in a multiple of blocks.

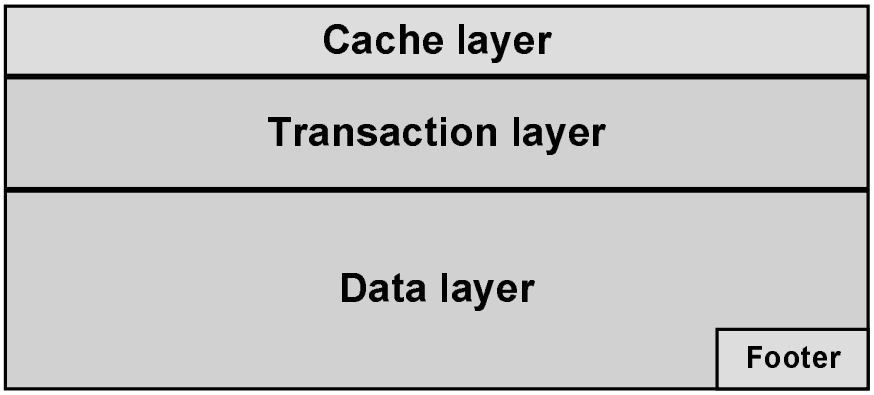
RDBMS Data Blocks

Once the database is created with a certain block size, you cannot alter it unless you re-create the database. All Oracle server blocks have the same block size for that specific database. Typical block sizes are 2 KB, 4 KB, 8 KB, and 16KB; the maximum size is OS-specific. There are minor differences between Oracle7 and Oracle8 data blocks. These differences are noted in this lesson.

Note: There is no difference and therefore no distinction made in this lesson between Oracle8 and Oracle8*i* blocks.

Data Block Structure

An Oracle server data block has three parts:



Data Block Components

The three layers of an Oracle server data block are represented by C structures, which are mapped on to the blocks in the SGA kcbh (kernel cache block header).

The Cache Layer contains information about the block format (Oracle7 or Oracle8), type (data, index, header, and so on), and sequencing data.

The transaction layer stores information about transactions that have an interest in the block.

This structure is at the beginning of every data block in the database. It even appears in sort blocks which are not changed by redo. It also appears at the beginning of data file header blocks and controlfile header blocks. The cache header provides checking for corrupt data. It is used to ensure that the correct block has been read and that the block is not fractured or otherwise damaged. A fractured block is one that was partially written to disk, leaving a previous version on part of the block.

ROWIDs and Block Dumps

- ROWIDs are pointers to a particular row in a block.
- The ROWID components are version dependent:
  - Oracle7: File number, block number, and the row number for a particular block.
  - Oracle8: Additional data object number; the reason is that the file number is not unique to the database but is relative to the tablespace.
- The Oracle7 ROWID format is still used in Oracle8, and referred to as restricted ROWID format.

Why ROWIDs?

The reason that ROWID formats are introduced with block dumps is so that you can dump blocks when the only information you have is a set of ROWIDs and not the actual file number and block number.

ROWIDs and Block Dumps

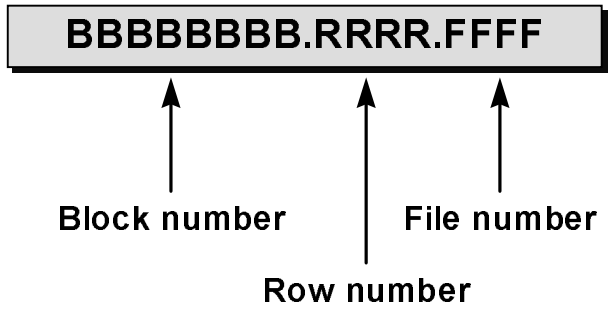
When to use the ROWID access method:

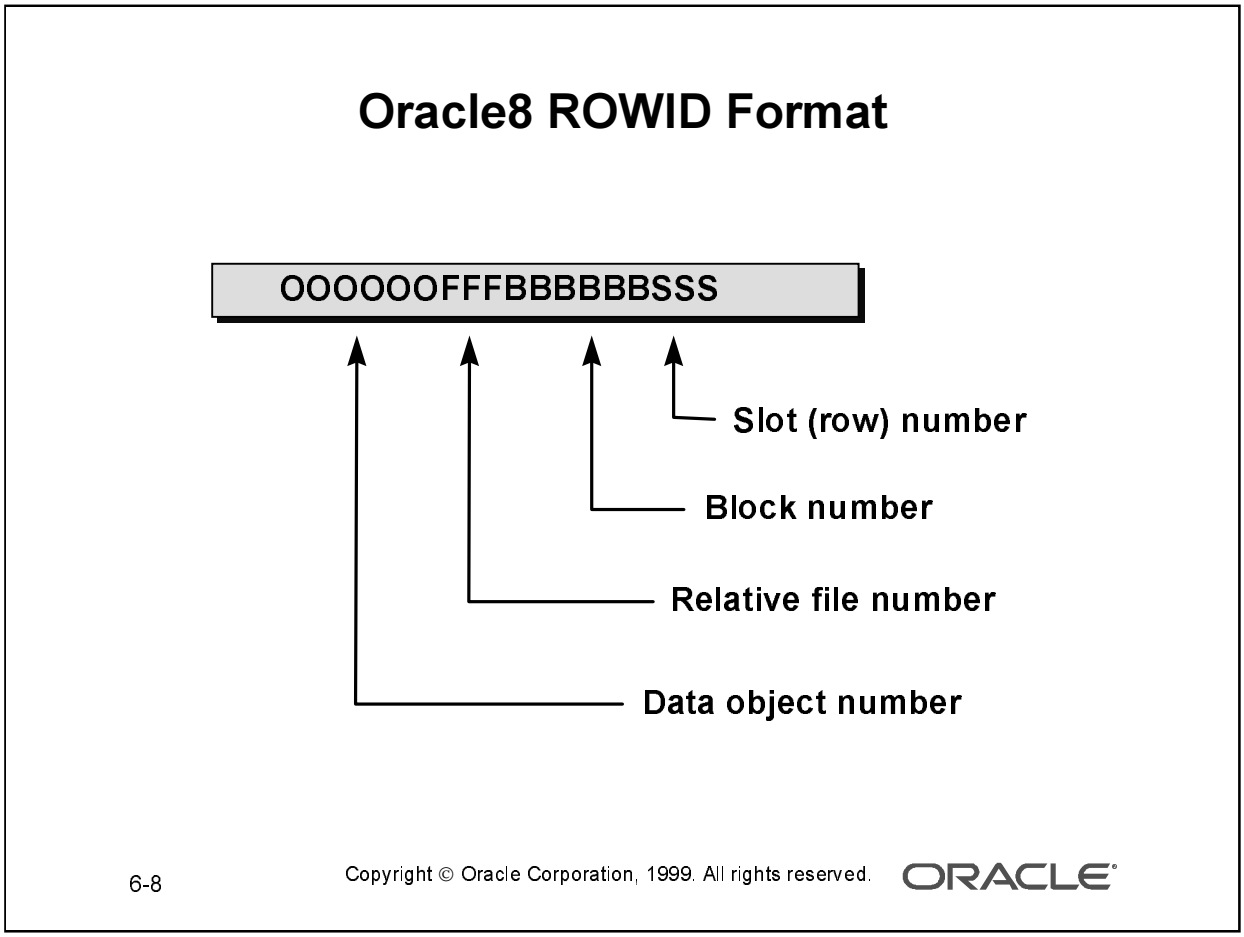
- **When data is indexed, the index block contains the ROWID for each row for faster access.**
- **This ROWID can also be used to extract data from the block when regular access methods fail.**

ROWIDs and Block Dumps

You can use the ROWID access method when the actual data is corrupted and regular data access methods like select and export fail.

Oracle8 Restricted ROWID Format





Oracle8 ROWID Format

Oracle8 and Oracle8*i* use the extended ROWID data type to store the address of every row in the database. The extended ROWID efficiently identifies rows in partitioned tables and indexes, as well as nonpartitioned tables and indexes. It supports tablespace-relative data block addresses. A restricted ROWID datatype is also available for backward compatibility with existing applications.

Extended ROWIDs use a base-64 (A–Z, a–z, 0–9, +, /) encoding of the physical address for each row selected.

Example

```
SQL> select rowid,ename
  2  from   emp
  3  where  ename = 'KASSIS';

  ROWID                ENAME
  ------------------   ----------
  AAAAwQAAFAAAAANAAI   KASSIS
```

Oracle8 ROWID Format (continued)

You can use the DBMS_ROWID package to get all the information that you need about ROWIDs. You can find out the data block number, the object number, and other components of the ROWID without having to write code to interpret the base-64 character external ROWID format.

DBMS_ROWID Example

The ROWID_BLOCK_NUMBER function returns the database block number for the input ROWID.

```
SQL> insert into T2
  2  (select dbms_rowid.rowid_block_number(ROWID)
  3   from   some_table
  4   where  key_value = 42);
```

There is also a rowid utility, that converts an Oracle8 ROWID to its object number, relative file number, block number, and row number. This utility was developed by support on Solaris and is available on most Test Center Solaris hosts.

Example of the rowid utility:

```
$ rowid AAAAwQAAFAAAAANAAA

Object number :3088  0xc10
Relative File number :5  0x5
Block number :13  0xd
Row number :0  0x0
```

Oracle Block Dumps

You can use two methods to dump Oracle blocks:

- Raw dumps performed with operating system utilities
- Formatted dumps performed through the RDBMS

OS Dump for Oracle7 and Oracle8

- On UNIX platforms, the dd utility is used to dump blocks.
- Do not place spaces around the equal sign (=) when using dd.

```
dd bs=db_block_size if=dbfile.dbf \
skip=(block-1) \
count=3 | od -xv > file.out
```


UNIX dd Utility

bs: Block size (For our purposes, this is the Oracle block size; 2 KB , 4 KB, 1024, 4096, and so on)

if: Input file

skip: Number of blocks to skip

count: Number of blocks to dump

od: UNIX octal dump utility

-x: Option to the od command to convert output to hex

-v Show all input data (verbose). Without the -v option, all groups of output lines that would be identical to the preceding line are replaced with a line containing only an asterisk (*).

Example

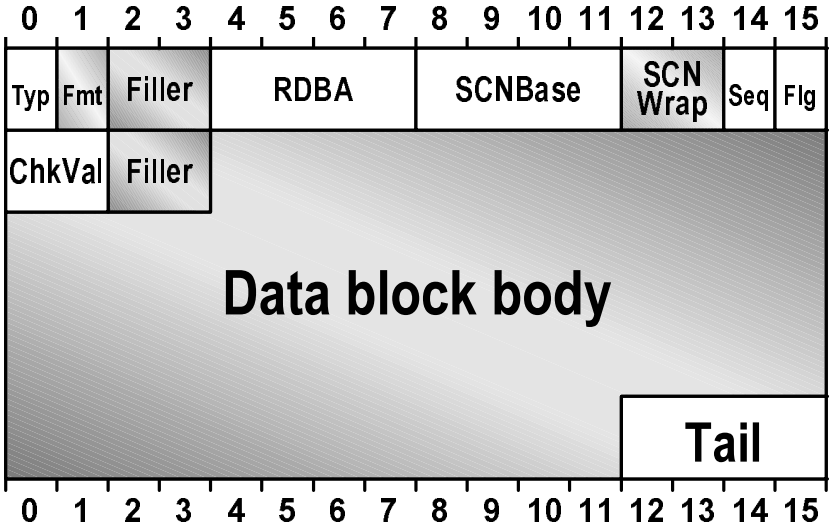
You have looked in your alert.log file and discovered that block 100 from file #1 has been reported as corrupted. Your TEAMA database was created using a 4 KB block size. How would you dump this block from the operating system?

```
$ dd if=systemTEAMA01.dbf bs=4k skip=99 count=3 | od -xv > file.out
```

The example above would dump blocks 99, 100, and 101 into a file called file.out. You will often find it useful to dump the blocks directly preceding and following your target block, for reference purposes.

Note: Remember that dd reads from disk. If the database is up, force a checkpoint to flush dirty buffers to disk.

Oracle8 Data Block Layout



Oracle8 Data Block Layout

Typ: Block type (See one of the following pages for accepted values.)

Fmt: Block format; in Oracle8 this is 0x02, converted on the fly

Filler: Not currently used

RDBA: Relative database address of the block

SCNBase: SCN base

SCNWrap: SCN wrap

Seq: Sequence number; incremented for every change made to the block at the same SCN

Flg: Flag (defined in kcbh.h)

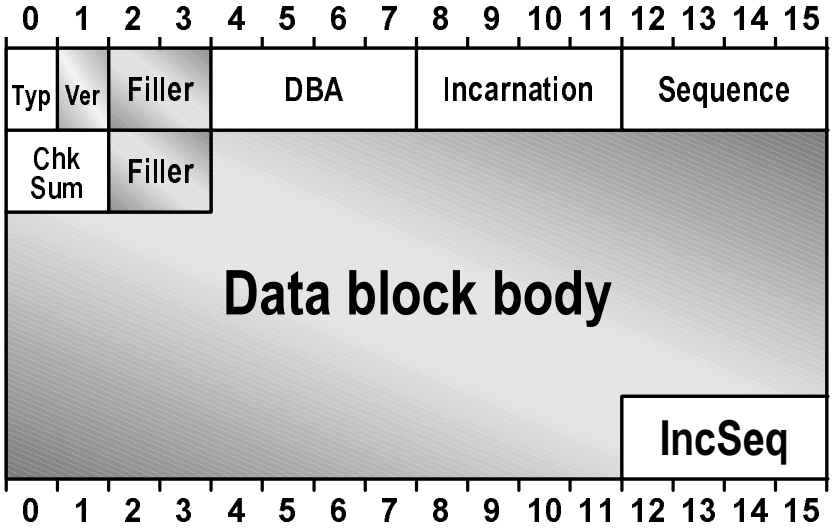
```
#define KCBHFNEW 0x01 /* new block - zeroed data area           */
#define KCBHFDLC 0x02 /* Delayed Logging Change advanced SCN/seq */
#define KCBHFCKV 0x04 /* CheCK Value saved - block xor's to zero */
#define KCBHFTMP 0x08 /* Temporary block                        */
```

ChkVal: Optional check value for the block

Tail: Consistency information to verify that the beginning and the end of the block are of the same version (lower order 2 bytes of SCNBase, plus block type, plus SCN Seq number)

Note: If there are 254 changes at the same SCN, then it is necessary to force a new SCN allocation on the next change of the block.

Oracle7 Data Block Layout



Oracle 7 Data Block Layout

The bytes in the header are defined as follows:

Typ:	Block type (See the following page for accepted values)
Ver:	Block version; in Oracle7 this is 0x01
Filler:	Not used at present
DBA:	Database address of the block
Inc:	Incarnation of the block; incremented each time the block is "newed"
Seq:	Sequence number of the block (A change version for the current incarnation)
ChkSum:	A simple checksum on the contents of the block (It uses the routine smschk( ) and is only present in data blocks if DB_BLOCK_CHECKSUM is set. However, it is always present in file header blocks.)

Oracle8 Versus Oracle7 Data Blocks

Oracle8 has introduced a few changes to the data block header format. The incarnation and sequence numbers have been replaced with an SCN number and sequence number. IncSeq (lower order two bytes of incarnation plus lower two order bytes of the sequence number) has been replaced by tail, which consists of the lower order two bytes of SCNBase plus block type plus SCN seq number. The DBA has been replaced by the relative DBA, RDBA. The sequence number has been split to SCNWrap, Seq, and Flg.

Type: Block Type (Defined in k.h)

Id	Type
1	Undo segment header
2	Undo data block
3	Save undo header
4	Save undo data block
5	Data segment header (temp, index, data, and so on)
6	KTB-managed data block (with itl)
7	Temp table data block (no itl)
8	Sort key
9	Sort run
10	Segment free list block
11	Data file header

Note: Block types valid for Oracle7 and Oracle8 blocks.

Oracle8 Binary (OS) Dump

```
<Offset>  <-------------- Data ----------------->
 0004000  0602 0000 0140 000d 0004 10e7 0000 0102
 0004020  0000 0000 0100 0000 0000 0c10 0004 10e6
 ...
 0007760  0c11 0101 0102 c209 ff02 c115 10e7 0601
```

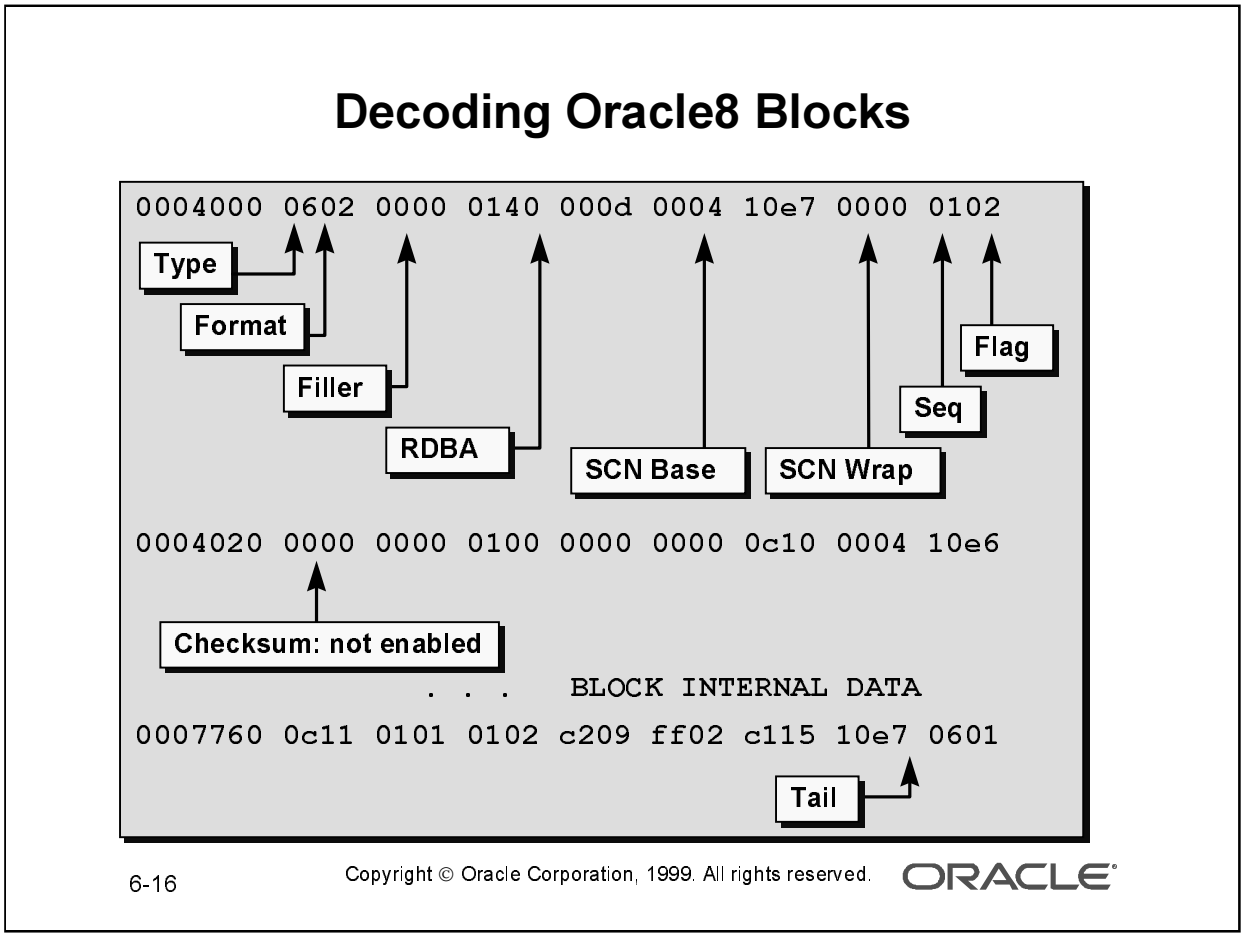

Oracle8 OS Dump

This is a dump of an Oracle8 block from a Solaris host. The following command was used to produce the dump:

```
$ dd bs=2k if=users01.dbf skip=12 count=3|od -x > /tmp/file.out
```

It is important to remember that although the data output is in hex, the offset is still calculated in octal. The offset column is added by the UNIX od command and is not part of the dump.

Note: An Oracle7 block would look the same but the header and footer would evaluate slightly differently. Refer to the previous slides.



Decoding Oracle8 Blocks

The tail of an Oracle8 block is comprised of the lower order two bytes of SCNBase plus block type plus SCN seq number. As you can see in the example the tail is:

10e7 (last two bytes of SCN Base) + 06 (type) + 01(seq) or 10e7 0601

Example from kcbh.h:

```
struct kcbh
{ub1   type_kcbh;     /* Block type* /
 ub1   frmt_kcbh;     /* #define KCBH_FRMT8 2 */
 ub1   spare1_kcbh;
 ub1   spare2_kcbh;
 krdba rdba_kcbh;     /* relative DBA /
 ub4   bas_kcbh;      /* base of SCN */
 ub2   wrp_kcbh;      /* wrap of SCN */
 ub1   seq_kcbh;      /* sequence # of changes at same scn */
 ub1   flg_kcbh;
 ub2   chkval_kcbh;
};
```

Oracle Formatted Block Dumps

- You can dump the block from the RDBMS.
- A trace file containing the dump is generated in the user dump destination.

Oracle8:

```
SQL> alter system dump datafile 5 block 13;
```

Oracle7:

```
SQL> alter session set events 'immediate
  2  trace name blockdump level 67110390';
```


Oracle Formatted Block Dumps

The next several slides show an Oracle8 formatted block dump with appropriate legend in the notes.

Note: In the example for Oracle7 in the slide, the level is actually the block number to be dumped; that is, the decimal equivalent of the hexadecimal DBA.

Oracle8 Formatted Block Dump

```
Mon Aug 25 16:42:04 1997
*** SESSION ID:(6.38) 1997.08.25.16.42.04.000
Start dump data blocks tsn: 4 file#: 5
minblk 13 maxblk 13
buffer tsn: 4 rdba: 0x0140000d (5/13)
scn:0x0000.000410e7 seq:0x01 flg:0x02 tail:0x10e70601
frmt:0x02 chkval:0x0000 type:0x06=trans data

Block header dump: rdba: 0x0140000d
Object id on Block? Y
```


Oracle8 Formatted Block Dump

There is a block header for any block dump. It determines the format of the rest of the dump. At the head of any Oracle block dump you should see a block header.

Rdba: Relative DBA of the block

Scn: SCN number

Seq: Sequence number:

```
SEQ -> 0              /* non-logged changes - do not advance seq# */
SEQ -> (UB1MAXVAL-1)  /* maximum possible sequence number         */
SEQ -> (UB1MAXVAL)    /* seq# to indicate a block is corrupt,
                         equal to FF                              */
```

Flg: Flag (defined in kcbh.h)

Tail: Consistency data used to verify that the beginning and the end of the block are of the same version (Consists of lower order two bytes of SCNBase plus block Type plus SCN Seq number.)

Frmt: Block format; in Oracle8 this is 2

Chkval: Optional check value for the block if DB_BLOCK_CHECKSUM = TRUE

Type: Block type (defined in kcb.h)

Oracle8 Formatted Block Dump

```
Block header dump: rdba: 0x0140000d
Object id on Block? Y
seg/obj: 0xc10  csc: 0x00.410e6  itc: 1  flg: O
typ: 1 - DATA

Seg/Obj ID in dictionary.
fsl: 0  fnx: 0x0 ver: 0x01

Itl   Xid          Uba          Flag  Lck  Scn/Fsc
0x01  0x0002.003   0x00800604   --U-   14  fsc 0x0000
      .000002fa    .025d.06                .000410e7
```


Oracle8 Formatted Block Dump (continued)

Rdba: Relative data block address

Seg/Obj: Seg/Obj ID

Csc: SCN at last block cleanout

Itc: Number of itl slots

Flg: 0 = on the freelist

Typ: 1 = DATA; 2 = INDEX

Fsl: ITL TX freelist slot

Fnx: DBA of NEXT block on freelist

Itl: Interested transaction list index (ITLs determined by INITRANS and MAXTRANS)

Xid: Transaction ID (UndoSeg.Slot.Wrap)

Uba: Undo address (UndoDBA.SeqNo.RecordNo)

Flg: C = Committed; U = Commit Upper Bound; T = Active at CSC;
B = Rollback of this UBA gives before image of the ITL.

Lck: Number of rows affected by this transaction

Scn/Fsc: Scn = SCN of commited TX; Fsc = Free space credit (bytes)

Oracle8 Formatted Block Dump

```
data_block_dump
===============
tsiz: 0x7b8
hsiz: 0x2e
pbl: 0x01182e04
bdba: 0x0140000d
flag=--------
ntab=1
nrow=1
frre=-1
fsbo=0x2e
fseo=0x581
avsp=0x553
tosp=0x553
0xe:pti[0]      nrow=14 offs=0
```

6-20

ORACLE

Oracle8 Formatted Block Dump (continued)

Tsiz: Total data area size

Hsiz: Data header size

Pbl: Pointer to buffer holding the block

Bdba: Block relative data block address (RDBA)

Flag: N = pctfree hit(clusters); F = do not put on free list; K = flushable cluster keys

Ntab: Number of tables (>1 in clusters)

Nrow: Number of ROWS

Frre: First free row index entry; -1 = you have to add one

Fsbo: Free space begin offset

Fseo: Free space end offset

Avsp: Available space in the block

Tosp: Total available space when all TXs commit

Nrow: Number of rows for first table

Oracle8 Formatted Block Dump: Data Layer

```
tab 0, row 8, @0x646
tl: 38 fb: --H-FL-- lb: 0x1 cc: 8
col 0: [ 3]  c2 4f 28
col 1: [ 6]  4b 41 53 53 49 53
col 2: [ 9]  50 52 45 53 49 44 45 4e 54
col 3: *NULL*
col 4: [ 7]  77 b5 0b 11 01 01 01
col 5: [ 2]  c2 33
col 6: *NULL*
col 7: [ 2]  c1 0b
```


Oracle Formatted Block Dump (continued)

Tab: Table 0, row 8, offset

Cc: Number of columns in this ROW piece

Lb: Lock byte: ITL entry that has this row locked

Fb: Flag byte
H = head of row piece; K = Cluster key; C = Cluster table member; D = Deleted row;
F = First data piece; L = last data piece; P = First column continues from previous piece;
N = Last column continues in next piece

Tl: Row size (number of bytes plus data)

Col: Column data

Translation

The following is an ascii-to-hex map for reference to help translate some of the information in the formatted block dump:

```
  20    , 2c    8 38    D 44    P 50    \ 5c    h 68    t 74
! 21    - 2d    9 39    E 45    Q 51    ] 5d    i 69    u 75
" 22    . 2e    : 3a    F 46    R 52    ^ 5e    j 6a    v 76
# 23    / 2f    ; 3b    G 47    S 53    _ 5f    k 6b    w 77
$ 24    0 30    < 3c    H 48    T 54    ` 60    l 6c    x 78
% 25    1 31    = 3d    I 49    U 55    a 61    m 6d    y 79
& 26    2 32    > 3e    J 4a    V 56    b 62    n 6e    z 7a
' 27    3 33    ? 3f    K 4b    W 57    c 63    o 6f    { 7b
( 28    4 34    @ 40    L 4c    X 58    d 64    p 70    | 7c
) 29    5 35    A 41    M 4d    Y 59    e 65    q 71    } 7d
* 2a    6 36    B 42    N 4e    Z 5a    f 66    r 72    ~ 7e
+ 2b    7 37    C 43    O 4f    [ 5b    g 67    s 73    ^? 7f
```

You can also obtain this ascii-to-hex conversion map on any UNIX machine by typing “man ascii” at a shell prompt.

Translation Example

```
block_row_dump:
tab 0, row 8, @0x792
...
col 1: [ 6] 4b  41  53  53  49  53       (in Hex)
            K   A   S   S   I   S   <==  (in Ascii)
```

Practice 6 Overview

This practice covers the following topics:

- **Performing a UNIX dump**
- **Performing an Oracle formatted dump**

Practice 6 Overview

For detailed instructions on performing this practice, see Practice 6 in Appendix B.

7

Block Corruption Diagnostics and Recovery

ORACLE

Schedule:	Timing	Topic
	45 minutes	Lecture
	15 minutes	Examples
	60 minutes	Lab Exercises
	120 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- Identify and explain block corruptions
- Collect and interpret relevant diagnostic information
- Apply appropriate recovery actions

What Is Block Corruption?

Whenever the RDBMS reads or writes a block to or from disk, a consistency check is performed:

- **Block version**
- **The DBA value in cache is compared to the DBA value in the block buffer**
- **Block-checksum, if enabled**

Which Checks Are Performed?

- In Oracle7:
  - The incarnation and sequence number in the header
  - The IncSeq in the footer
  - The block type
- In Oracle8:
  - The SCNBase and seq in the header
  - The SCNBase and seq in the footer
  - The block type

Which Checks Are Performed?

The block type is stored in the first word of the data block.

Examples

These are typical examples of ORA-600 errors due to block corruption:

- ORA-600 (4519): Cache layer block type is incorrect
- ORA-600 (4136): Check rollback segment block
- ORA-600 (4154): Check rollback segment block

Note: In Oracle7, the block header with free list is checked and fails with ORA-00 (4393). This is no longer used in Oracle8.

Block Corruption Types

- **Media corrupt:**
  - **Oracle7: INC = 0**
  - **Oracle8: SCN = 0**
- **Soft corrupt:**
  - **Oracle7: Seq = 0**
  - **Oracle8: SCN = *XXX***
    Seq = UB1MAXVAL

    where *XXX* means that only a change at a higher SCN, which reconstructs the entire block, can be applied to the block

Block Corruption Types

There are two types of block corruption:

- Media corrupt blocks: The information in the block does not make any sense after the read from disk.
- Software corrupt blocks: The RDBMS marked the block corrupt after detecting a corruption.

In Oracle8, the seq value in a block has a maximum value of UB1MAXVAL-1. Therefore, the value UB1MAXVAL can be used to mark a block as “soft corrupt”.

Symptoms: ORA-1578

ORA-01578: "ORACLE data block corrupted (file # %s, block # %s)"

- **This error is generated when a corrupted data block is found.**
- **The error always returns the absolute file number and block number.**
- **Check the `alert.log` file.**

Symptoms: ORA-1578

Usually, the ORA-1578 error is the result of a hardware problem. If the ORA-1578 error always comes back with the same arguments, it is most likely a physically corrupted block.

If the arguments change each time, there may be a hardware problem, and the customer should have the memory and page space checked, and the I/O subsystem checked for bad controllers.

Once you have the decimal block number, you can use the UNIX dd command to try to dump the block from disk. If you get a read error, you know that this is a hardware failure. Also, you can try to dump the block a few times and see if you always get the same data back. This could be a check for the I/O subsystem or bad controllers.

Note: See <Note:61685.1> for additional information on ORA-1578 in Oracle8 and <Note:28814.1> for additional information on ORA-1578 in Oracle7.

Symptoms: ORA-600

- ORA-600 [3374], [a], [b], [c], [d]
  - Block check failure for write
  - Corrupted in memory
  - Checks dba, type, version, and inc#
- ORA-[600] [kcbzpb_1], [d], [kind], [chk], [ ], [ ]
  - Replaces ORA-600 [3374] for Oracle8

Symptoms: ORA-600

ORA-600[kcbzpb_1],[d],[kind],[chk] is signaled when a block is corrupted in memory. The only way it should be bad is if a stray store into memory destroyed the header or tail.

- kcbzpb_1 is a named internal error, referencing a source code module name
- d = blocknumber
- kind = kind of corruption detected
- chk = checksum flag

The ORA-600[3339] error comes with ORA-1578 and means either disk corruption or in-memory corruption after read.

How to Handle Corruptions

- Check the alert file and system log file.
- Use available diagnostic tools to find out the type of corruption.
- Perform a block dump.
- Use patch utilities to see what is wrong with the block.
- Determine if the error persists; run checks multiple times.
- Recover data from the corrupted object if necessary.

How to Handle Corruptions

Always try to find out if the error is permanent. Run the analyze command multiple times or, if possible, perform a shutdown and a startup and try again to perform the operation that failed earlier.

Find out if there are more corruptions. If you encounter one, there may be other corrupted blocks, as well. Use tools like DBVERIFY for this.

Before you try to salvage the data, perform a block dump as evidence to identify the actual cause of the corruption.

- Make a hex dump of the bad block, using UNIX dd and od -x.
- Consider performing a redo log dump to check all the changes that were made to the block so that you can discover when the corruption occurred.

Note: Remember that when you have a block corruption, performing media recovery is the recommended process after the hardware is verified.

Patch utilities are discussed later in this lesson.

How to Handle Corruptions

- Resolve any hardware issues:
  - Memory boards
  - Disk controllers
  - Disks
- Recover or restore data from the corrupt object if necessary.

How to Handle Corruptions (continued)

There is no point in continuing to work if there are hardware failures. When you encounter hardware problems, the vendor should be contacted and the machine should be checked and fixed before continuing. A full hardware diagnostics should be run.

Many types of hardware failures are possible:

- Bad I/O hardware or firmware
- Operating system I/O or caching problem
- Memory or paging problems
- Disk repair utilities

DBVERIFY Utility

- Data file verification utility that comes with Oracle release 7.3
- Only works on data files; redo log files cannot be checked
- Checks block consistency
- Opens data files read-only; can be done while the database is up and running

DBVERIFY: Example

```
$ dbv file=data01_01.dbf blocksize=4096
DBVERIFY: Release 8.1.5.0.0 - Production on Thu Nov 11 17:17:21 1999
(c) Copyright 1999 Oracle Corporation. All rights reserved.
DBVERIFY - Verification starting : FILE = data01_01.dbf
DBVERIFY - Verification complete
Total Pages Examined         : 7680
Total Pages Processed (Data) : 3073
Total Pages Failing   (Data) : 0
Total Pages Processed (Index): 1763
Total Pages Failing   (Index): 0
Total Pages Processed (Other): 33
Total Pages Empty            : 2811
Total Pages Marked Corrupt   : 0
Total Pages Influx           : 0
```

Note: Refer to <Note:35512.1> for details on DBVERIFY. Using DBVERIFY while the database is running is not recommended, because blocks in the cache could reinitiate corruptions.

DBVERIFY Utility

- The page number is the block number within the data file.
- DBVERIFY uses the standard kcb routines to check the block integrity.
- If head and tail do not match, DBVERIFY reads the block again. If they match, an influx block is reported; otherwise a corruption.

DBVERIFY Utility

Influx blocks are split blocks. If DBVERIFY does not report corruption it means that when it read the block the first time, DBW0 was writing a new version and it caught part of the old and part of the new version of this block. Thus, head and tail do not match.

Note: DBVERIFY only checks for logical corruption; that is, it uses kcb.h routines to check header/footer information only. Therefore, it is possible for corruption to occur above the high-water mark.

The ANALYZE Command

- Performs a logical block check
- Does not mark blocks as soft corrupt; only reports them
- Validates index and table entries

The ANALYZE Command

```
SQL> analyze table xxx validate structure cascade;
SQL> analyze index xxx validate structure;
SQL> analyze cluster xxx validate structure;
```

The cascade option validates an object, including all related objects.

The advantage of this option is that you can run it in a SQL*Plus session on a specific object, to do an integrity check and to determine if an error is persistent, by running the same analyze command a number of times.

```
SQL> analyze table xxx partition (p1)
  2  validate structure into invalid_rows;
```

For a partitioned table, the Oracle server also verifies that the row belongs to the correct partition. If the row does not collate correctly, the rowid is inserted into the invalid_rows table.

Note: A simple select statement (select * from <table>) does a full table scan, which means that it reads all the data blocks up to the high-water mark of the table. You could use this to perform a quick check for corruptions in your current table data.

Diagnostic Events

- **Event 10231:**
  "Skip corrupted blocks during a full table scan"
- **Event 10233:**
  "Skip corrupted data or index blocks on index range scans"

Diagnostic Events

Event specification syntax in the parameter file:

```
event = "10231 trace name context forever, level 10"
event = "10233 trace name context forever, level 10"
```

Note: An index range scan is used when you query the database based on a range of values, or if your query uses a nonunique index.

Diagnostic Events

- **Event 10232:**
  "Dump corrupted blocks in a trace file"
- **Event 10210:**
  "Force logical block checking"

  Whenever a table data block is modified, an integrity check is done; bad blocks are marked as soft corrupt.
- **Event 10211: As event 10210 for index blocks**
- **Event 10212: As event 10210 for cluster blocks**

Diagnostic Events (continued)

For events 10210, 10211, and 10212, the kdbchk (block integrity check) returns FALSE on bad blocks.

It checks for the following:

- Free slot list structure
- Row positions for containment within block
- Row positions for overlap
- Lock counts for transactions against the itl

These three events allow potential corruptions to be spotted as soon as they appear and an error to be signaled; however, performance is affected because the blocks are copied when they are checked. Note that all blocks that fail the verification check are immediately marked "soft corrupt", which means that the data in the block is lost.

To enable these events, add the following to the init.ora parameter file:

```
event = "xxx trace name context forever, level 10"
```

where *xxx* is 10210, 10211, or 10212.

Note: These events are expensive; make sure to turn them off when not needed. Event 10231 and event 10233 only skip soft corrupt blocks.

Initialization Parameter db_block_checking

- Introduced in Oracle8*i*
- Can be set by using the ALTER SESSION or ALTER SYSTEM DEFERRED command
- Replaces events 10210, 10211, and 10212
- Default value is false

Initialization Parameter: db_block_checking

The db_block_checking initialization parameter invokes the same kdb checks as events 10210, 10211, and 10212.

The default setting is false and is provided for compatibility with earlier releases where block checking is disabled as a default.

Because the parameter is dynamic, it provides more flexibility than events 10210, 10211, and 10212, which it will replace in future releases. Note that the setting of db_block_checking overrides any setting of events 10210, 10211, and 10212.

Note: Oracle release 7.2 and later includes a special provision, the db_block_checksum initialization parameter, to add checksums to Oracle database blocks. This helps detect media corruptions when a block is next read by the Oracle server.

If db_block_checksum is set to true, DBWn and the direct loader calculate a checksum and store it in the cache header of every data block when writing it to disk. When the block is subsequently read, the checksum is recomputed and the stored value is checked with this computed value (see also <Note:32969.1>).

The Export Utility

- A full export can be used to check the logical consistency of a database.
- Export performs a full table scan on all the tables to retrieve the data.
- Export does not detect all corruptions:
  - Does not detect disk corruptions above the high-water mark
  - Does not detect corruptions in indexes, or in free or temporary extents

The Export Utility

You do not have to create a dump file when you perform an export, if all you want is to check the consistency of the data. On UNIX systems, use /dev/null as the file name; for OpenVMS use NULL:, and for Windows NT use NUL. When you do this, the export utility reads all the data but writes it to the null device, which is basically not writing at all. This is a good method for checking the logical consistency of the database and detecting corruption on a regular basis, both for physical corruption (in used blocks) and for logical corruption (data-dictionary issues).

Example

```
$ exp system/manager file=/dev/null full=y
Export: Release 8.1.5.0.0 - Production on Thu Nov 11 18:35:18 1999
(c) Copyright 1999 Oracle Corporation. All rights reserved.
Connected to: Oracle8i Enterprise Edition Release 8.1.5.0.0 -
Production  With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
Volume size (<ret> for no restriction) >
Export done in WE8ISO8859P1 character set
and WE8ISO8859P1 NCHAR character set
About to export the entire database ...
. exporting tablespace definitions
```

The Export Utility

Export only reads:

- **User data below the high-water mark**
- **Parts of the data dictionary, while looking up information concerning the objects being exported**

The Export Utility (continued)

Note: Export does not read all of the data dictionary, so there may be undetected corruptions in the SYSTEM tablespace.

Media Recovery

The database must be in archivelog mode.
Perform the following steps:

- **Resolve any hardware issues; for example, replace the disk.**
- **Find out which data file contains the corrupted blocks.**
- **Restore a backup of the data file.**
- **Recover the data file.**

Media Recovery

Find out which absolute data file number contains the corrupted block and then query v$datafile or dba_data_files to retrieve the file name.

Make sure that any hardware problems are resolved before you restore the backup of the data file. If the disk is bad and you have enough room on another disk, it is always possible to put the backup on a good disk, rename the data file to the new location, and recover.

Also, after the recovery you may want to check the recovered data file for corruptions.

If you do not have a hot backup, you must go back to restoring a full cold backup.

Which Object Is Corrupted?

- Index: Just drop and recreate the index.
- Rollback segment: See one of the later lessons.
- Table: The data in the corrupted block is lost.
  - Drop the table and re-create it, and import data from an export dump.
  - Set event 10231 to skip corrupt blocks on export, drop the table, and re-create it from the export dump.
  - Use SQL or PL/SQL to pull data out of the table into a newly created table.

Which Object Is Corrupted?

If you do not plan to restore data files and recover, use the following statement to determine which object has corrupted blocks.

The absolute file number (for example, 5) and block number (for example, 2) can be found in the error message; for example:

```
ORA-01578: ORACLE data block corrupted (file #5, block # 2)
```

In Oracle7, issue:

```
SQL> select segment_name, segment_type
  2  from   dba_extents
  3  where  file_id = 5
  4  and    2 between block_id and block_id + blocks - 1;
```

In Oracle8, issue:

```
SQL> select segment_name, segment_type, relative_fno
  2  from   dba_extents
  3  where  file_id = 5
  4  and    2 between block_id and block_id + blocks - 1;

SEGMENT_NAME    SEGMENT_TYPE     RELATIVE_FNO
--------------  ---------------  ------------
EXAMPLE         TABLE PARTITION             5
```

Using SQL or PL/SQL

- If the corrupted table has a unique index, you can use the index.
- Do a range scan with the ROWID hint to select around the corrupted block.
- You must use PL/SQL to retrieve data from tables containing a LONG column.

Using SQL

If you use SQL, you can create a new table using:

```
SQL> create table temp as
  2  select * from tab_name
  3  where  1 = 2;
```

Create the required ROWIDs with the following dbms_rowid.rowid_create function:

```
function rowid_create
 (rowid_type   IN number, object_number IN number
 ,relative_fno IN number, block_number  IN number
 ,row_number   IN number) return rowid;
 pragma RESTRICT_REFERENCES(rowid_create,WNDS,RNDS,WNPS,RNPS)
```

Then insert the required rows into that table around the corruption:

```
SQL> insert into temp
  2  select * from tab_name
  3  where  rowid < DBMS_ROWID.ROWID_CREATE(1,2168,5,3,0);
 SQL> insert into temp
  2  select * from tab_name
  3  where  rowid > DBMS_ROWID.ROWID_CREATE(1,2168,5,1,0);
```

Because the corrupt block is block 2, the last possible ROWID before the corrupt block contains block 1, and the first ROWID after the corrupt block contains block 3. Use row number 0, because you do not know how many row numbers are in the blocks.

Using SQL (continued)

To receive the object number, query dba_objects for non partitioned tables and tabpart$ for partitioned tables:

```
SQL> select object_id from dba_objects
  2  where  object_name = 'TAB_NAME';
SQL> select obj# from tabpart$
  2  where  file# = 5
  3  and    block# = 2;
```

Note: Refer to <Note 61685.1> for more details.

Chained Rows

If you have chained rows that have row pieces in the corrupt block, those rows are also lost. To find out which rows you must skip additionally, select ROWIDs from the table with a WHERE clause that skips the rows in the corrupt blocks. If an error occurs, look at the last returned ROWID, and you know that the next rowid has a problem. Then also exclude this one and relaunch the altered query.

LONG Columns

If the table contains a LONG column, you have to write a small PL/SQL block, which handles the LONG column. This has restrictions on the maximum size of the data, for example, 32 KB (port-specific).

If the LONG data is too large, writing a Pro*C or OCI program might be an alternative.

SQL*Plus can copy LONG data by specifying the longchunksize parameter and using the SQL*Plus COPY command. However, there is a limitation on the size of the LONG, and this limitation is OS dependent. Check the user's guide.

The dbms_repair Package

- Introduced in Oracle8*i*
- Works with corruption in the transaction and data layer
- Only marks the block “software corrupt” in the initial release

The dbms_repair Package

The dbms_repair package can indicate but not repair block corruption, and can help to extract meaningful data even if the block has not yet been marked as corrupt.

- After the block is marked as corrupt, the entire block must be skipped.
- For more details on this package, refer to chapter 18 in the *Oracle8i Administrator's Guide*, *the Oracle8i Supplied Packages Reference*, and to <Note:68013.1>.

The following section covers setting up the dbms_repair tables and checking the objects for corruptions.

Example

First create the repair tables and views with the following procedure:

```
SQL> execute dbms_repair.admin_tables
  2  (table_name => 'REPAIR_TABLE',-
  3   table_type => dbms_repair.repair_table,-
  4   action => dbms_repair.create_action);
```

Use the check_object procedure to check the specified object and populate the repair table:

```
SQL> dbms_repair.check_object (schema_name => 'SYSTEM',-
  2  object_name => 'T1',-
  3  repair_table_name => 'REPAIR_TABLE',-
  4  corrupt_count => :count);
```

dbms_repair Package

- admin_tables: Create the repair tables.
- check_object: Populate the repair tables with information about corruptions and repair directives.
- fix_corrupt_blocks: Fix the corrupt blocks in specified objects based on information in the repair table (ORA-1578).
- dump_orphan_keys: Report on index entries that point to rows in corrupt data blocks (optional).
- skip_corrupt_blocks: Skip corrupt blocks during index and table scans.

Example (continued)

```
SQL> print count
count: 1

SQL> select object_name, block_id
  2  ,      corrupt_type, marked_corrupt
  3  ,      corrupt_description, repair_description
  4  from   repair_table;

OBJECT_NAME  BLOCK_ID CORRUPT_TYPE MARKED_COR
---------- --------- ------------ ----------
CORRUPT_DESCRIPTION
--------------------------------------------
REPAIR_DESCRIPTION
--------------------------------------------
T1                   3            1 FALSE
kdbchk: row locked by non-existent transaction
        table=0   slot=0
        lockid=32   ktbbhitc=1
mark block software corrupt
```

Now query the corrupt object and extract as much information as possible. Dump the corrupted block and use tools such as BBED or ORAPATCH (see following section) to mine data from the hex dumps.

Patching with ORAPATCH

- ORAPATCH acts like a pure disk or file editor.
- "ORAPATCH help" lists all commands.
- You can view and modify blocks in hexadecimal for all files and block types supported by RDBMS.
- It is possible to patch everything, but it is very time-consuming to rebuild an entire block.

Patching with ORAPATCH

ORAPATCH is an internal support tool and is not provided to customers.

Note: Refer to <Note:28864.1> for a quick reference or see <Note:67996.1> for an example.

Before trying to patch a block, make sure that there is an agreement with the customer as to what is possible and what is not. Think about the time and effort involved, and be sure of what the customer really wants. Usually the data in the block is lost, and it is very difficult to rebuild an entire block.

Note: ORAPATCH works on 512 bytes blocks. If your alert.log file reported that block 10 was corrupted, then multiply 10 by 8, if the Oracle block size is 4 KB (4098 divided by 512 = 8) and then add 1 for the file header block. As a result, specify block 81.

Example

```
$ orapatch open systemTEAMA01.dbf write
patch> set hex
patch> display 81
```

Patching with BBED

- BBED stands for block browser/editor.
- It is a tool to browse and edit disk data structures while the database is open.
- It supports physical (byte-level) and symbolic (structure and field-level) interfaces for displaying and editing.

```
C:\dsi>bbed.exe listfile=files.bbed mode=edit
Password: *********
BBED: Release 2.0.0.0.0 - Limited Production on Mo Nov 15
12:08:36 1999
(c) Copyright 1999 Oracle Corporation.  All rights reserved
********** !!! For Oracle Internal Use only !!! ***********
BBED>
```


Patching with BBED

BBED is shipped with Oracle8 releases and with some Oracle 7.3 releases. It includes all the features of ORAPATCH, and also displays separate data structures within blocks. This can be very helpful in determining what exactly is wrong in the block. Another advantage that BBED has over ORAPATCH is the fact that BBED saves the "before image", so you can always undo changes.

On UNIX, you need to relink BBED with the following command:

```
$ make -f ins_rdbms.mk $ORACLE_HOME/rdbms/lib/bbed
```

Note: The link command does not work on all platforms for release 8.0. You must edit the make file to create an executable.

On Windows NT, a BBED.EXE is shipped as an executable and is protected with the password "blockedit".

Using BBED: Example

```
BBED> set file 1
 FILE#           1
BBED> map
 File: D:\ORACLE\ORADATA\GB500\SYSTEM01.DBF (1)
 Block: 5                                Dba:0x00400005
------------------------------------------------------
 Undo Data
 struct kcbh, 20 bytes                      @0
 struct ktubh, 32 bytes                     @20
 ub1 freespace[296]                         @52
 ub1 undodata[1696]                         @348
 ub4 tailchk                                @2044
BBED>
```


Patching with BBED (continued)

Invoke BBED in command line mode with the following parameters:

DATAFILE:	File to browse or edit
BLOCKSIZE:	Block size <in bytes>
MODE:	Browse or edit
REVERT:	To revert changes made in a previous session (y[es]/n[o])
SILENT:	y[es]/n[o]
SPOOL:	y[es]/n[o]
LISTFILE:	List of files; the format is: <absolute filenumber> <name> [sizes in bytes]
CMDFILE:	Command file name
BIFILE:	Before-Image file; default is bifile.bbd
LOGFILE:	User log file; default is log.bbd
PARFILE:	Parameter file

After launching BBED, issue specific implemented commands in interactive mode. Type in "help <command>" or "help all" to receive a list of the BBED commands. Refer to <Note:62015.1> for a detailed description and an example of how to use BBED.

Patching with BBED or ORAPATCH

- **ORAPATCH and BBED are tools that can potentially corrupt the database.**
- **Never use a customer system as a test system.**
- **Make sure to address customer expectations *before* you make an attempt.**

Patching with BBED or ORAPATCH

Remember at all times that the customer owns his or her database. When you want to help with recovery or patching, you must first consult with the customer to see what can be done, explain what you want to do, explain the risks and the time involved, and also the time it would take when the recovery or patching fails. Then ask the customer if he or she agrees to the proposed plan, and go ahead.

This cannot be stressed enough; it may seem obvious, but is often neglected.

Note: Patching utilities like ORAPATCH and BBED are useful when logical corruption has occurred; that is, when the header and footer do not agree. They are not useful when physical corruption has occurred and data has been corrupted.

References

- 61685.1: Handling block corruptions (Oracle8)
- 28814.1: Handling block corruptions (Oracle7)
- 35512.1: DBVERIFY: Database file verification utility
- 68013.1: DBMS_REPAIR Example
- 28864.1: ORAPATCH Quick reference
- 67996.1: DBMS_REPAIR Comprehensive example
- 62015.1: BBED: 7.3.2+ Database block editor

Practice 7 Overview

This practice covers the following topics:

- **Introducing block corruptions**
- **Determining block corruptions**

Practice 7 Overview

For detailed instructions on performing this practice, see Practice 7 in Appendix B.

8

Rollback Segment Corruption Recovery

ORACLE

Schedule:	Timing	Topic
	60 minutes	Lecture
	15 minutes	Examples
	60 minutes	Lab Exercises
	135 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Identify and explain rollback segment corruption**
- **Collect and interpret relevant diagnostic information**
- **Formulate and apply appropriate recovery procedures**

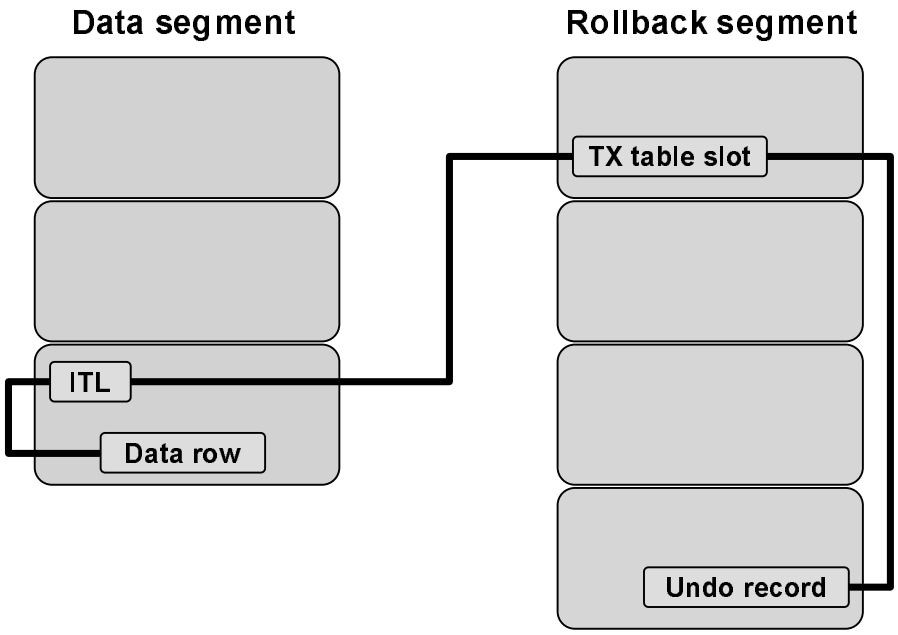
Objectives

Rollback segments are central to the RDBMS, being used for both rollback (transaction) recovery and for building consistent read database snapshots. When they become corrupt, the database availability is affected, and if the correct preparations are not made beforehand, it may be impossible to restore data consistency.

When customers call support, it is typically because they have failed to recover the database after a corruption. They may be under pressure to get the database working again and to prove that they are in control. It is critically important that, despite these distractions, you take the time to thoroughly understand the problem and, using a knowledge of how RDBMS works and the tools available, propose and justify a strategy that will return the database to a consistent state. When total consistency cannot be achieved, you should be able to explain the areas of risk and suggest effective remedial action.

This lesson covers the internals of transaction management, database startup, and rollback segment acquisition, and how rollback segment corruption can interfere with them. This will help you understand the symptoms of rollback segment corruption, additional diagnostics, and how the RDBMS behavior can be changed by using certain undocumented parameters.

Transaction Internals: Overview



Transaction Internals

A transaction begins when a slot is allocated in the transaction table at the head of a rollback segment. This is the physical manifestation of a transaction. The transaction identifier (txid) is a pointer to this location. Transaction identifiers have the following structure:

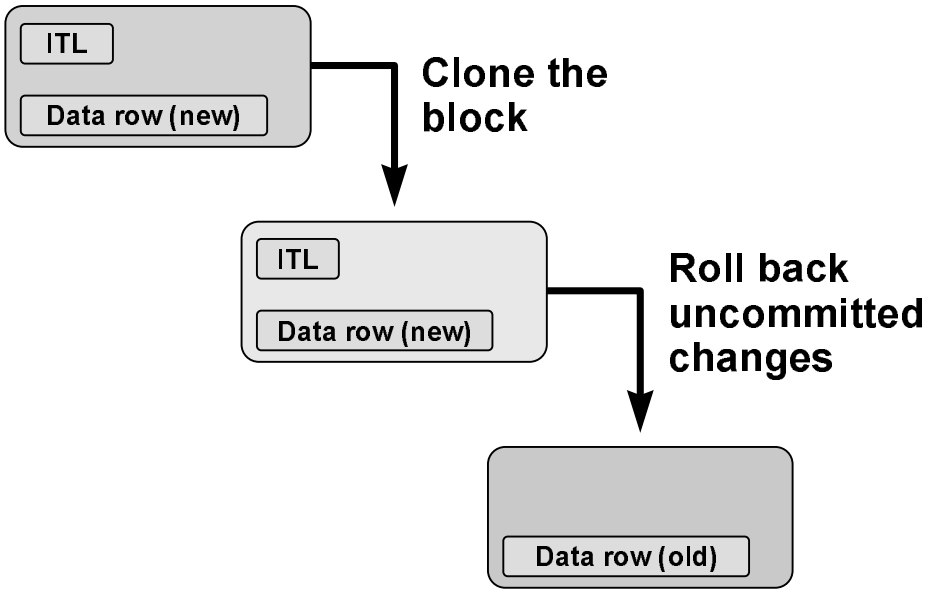
`txid = usn.slot.wrap` (undo_segment_number.transaction slot id.SCN wrap)

Before a transaction inserts, updates, or deletes a table row, an ITL is allocated in the block containing the row. The ITL is used to mark the row as locked until the transaction is either committed or rolled back. The ITL contains the transaction identifier.

When the change is applied to the block, undo information is also generated and is stored in the rollback segment. The transaction table slot contains a pointer to the undo.

Note: When querying v$lock, *usn.slot* shows up as id1 and *wrap* shows up as id2.

Transaction Internals: Read Consistency



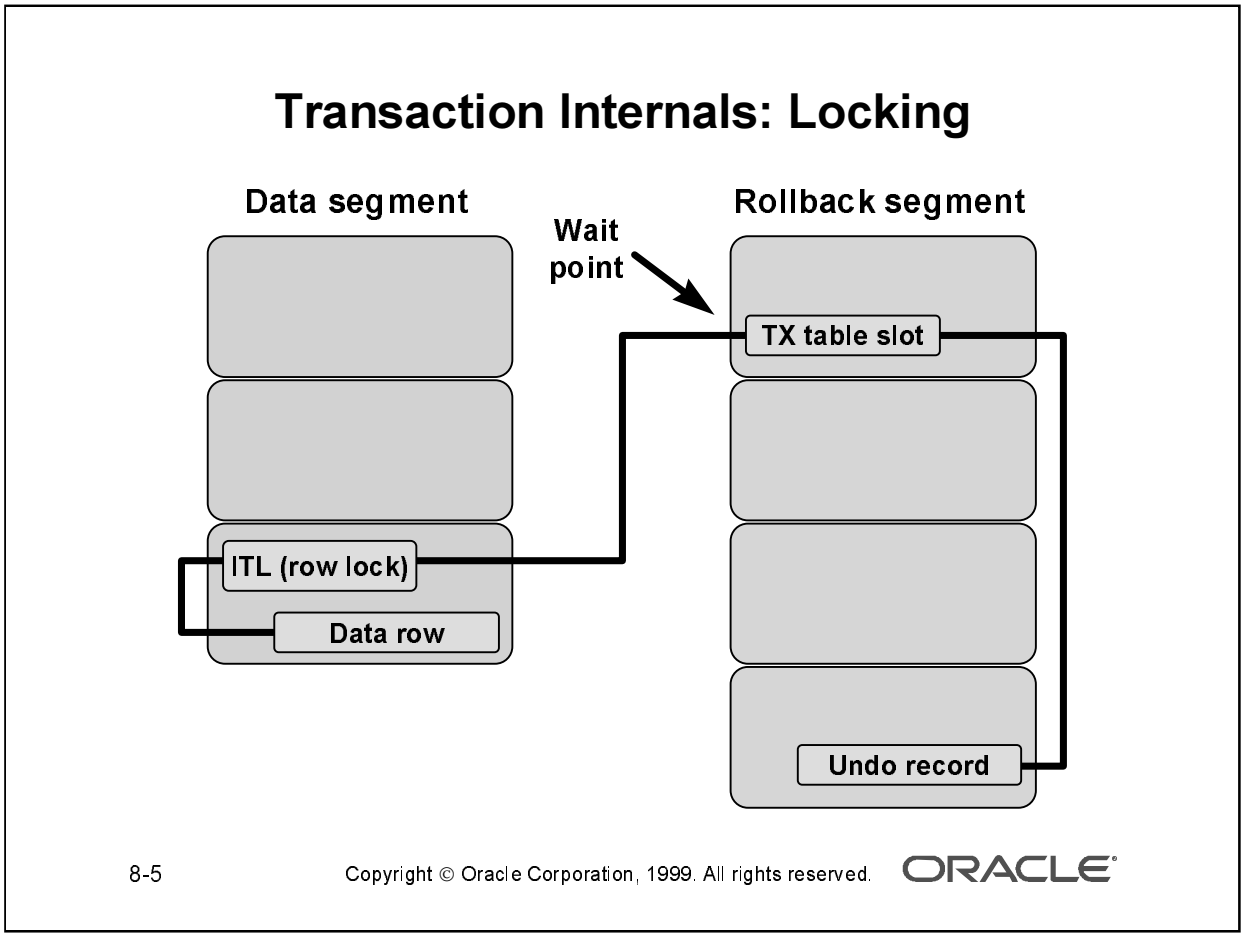
Read Consistency

Suppose a different session wants to read the same block, and the first transaction has not yet been committed.

The session, when reading the block (possibly still in cache), finds the open ITL. It checks the transaction status (by reading the rollback segment header) and finds that it is still active. This means that the session must roll back the changes made by the active transaction to create a view (snapshot) of the block before the first change was made. This is called making a consistent read (CR) copy of the block, and is achieved by first cloning the block and then rolling back the latest changes by applying undo records from the rollback segment.

Note that CR requires access to the data block, rollback segment header, and undo records. If any of these are corrupted, CR is affected.

Note: This is a simplified view of the CR mechanism. See the DSI302 course for a more detailed description.

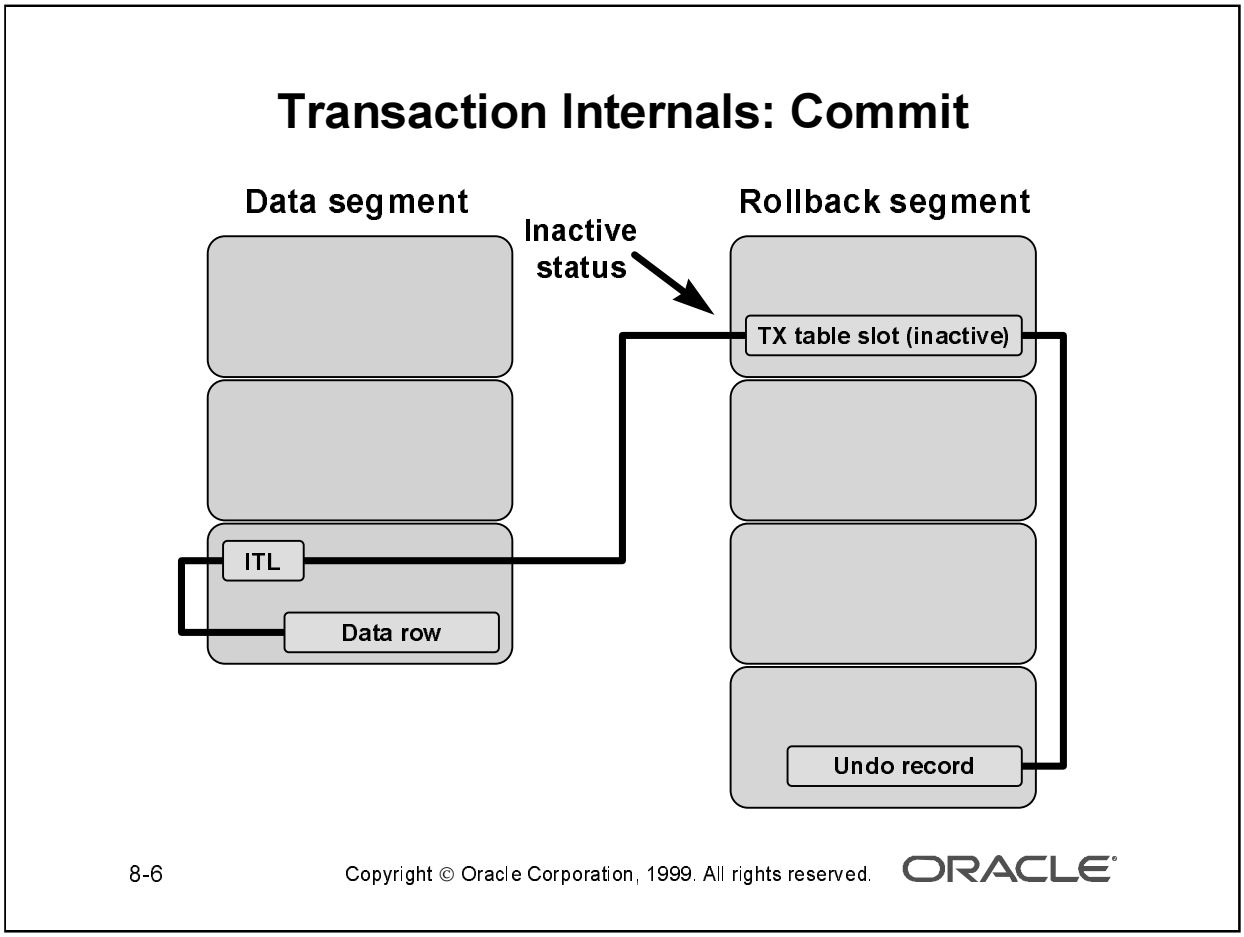


Locking

Continuing the same example, suppose another session wants to change (update or delete) the row that was updated by the first transaction, and the first transaction has not yet been committed.

The new session, when reading the block (possibly still in cache) will find the open ITL. It checks the transaction status (by reading the rollback segment header) and finds that it is still active. This tells the session that progress cannot be made at this stage, so it waits for the transaction to complete (commit or roll back).

Note that it is necessary to read the rollback segment header to establish the status of the ITL. If the rollback segment header is corrupted, then locking is affected.



Commit and Delayed Block Cleanout

Continuing the same example, suppose the first transaction is now committed. The event is immediately recorded by marking the transaction table slot as inactive; however, the data block itself may not be updated until later. This means that the ITL in the block may remain open for some time after the commit.

When the RDBMS next reads the block, the transaction table is checked, the commit is confirmed, and the ITL is closed. This is known as *delayed block cleanout*. If the rollback segment has since been deleted, the SCN recorded in undo$ is used to confirm the commit.

Note that the RDBMS must read the rollback segment header to perform delayed block cleanout. If the rollback segment is corrupted, then delayed block cleanout is affected.

In release 7.3, a new feature called delayed *logging* block cleanout (DLBC) has changed the RDBMS behavior. This feature is enabled by setting delayed_logging_block_cleanouts to true, as the default setting. With DLBC, ITL cleanout is partially completed at commit time for blocks that remain in the cache. References to the rollback segment header are no longer required to clean out those ITLs, so an I/O is saved. The ITL cleanout is completed when the ITL is reused or when a row covered by the ITL is next locked.

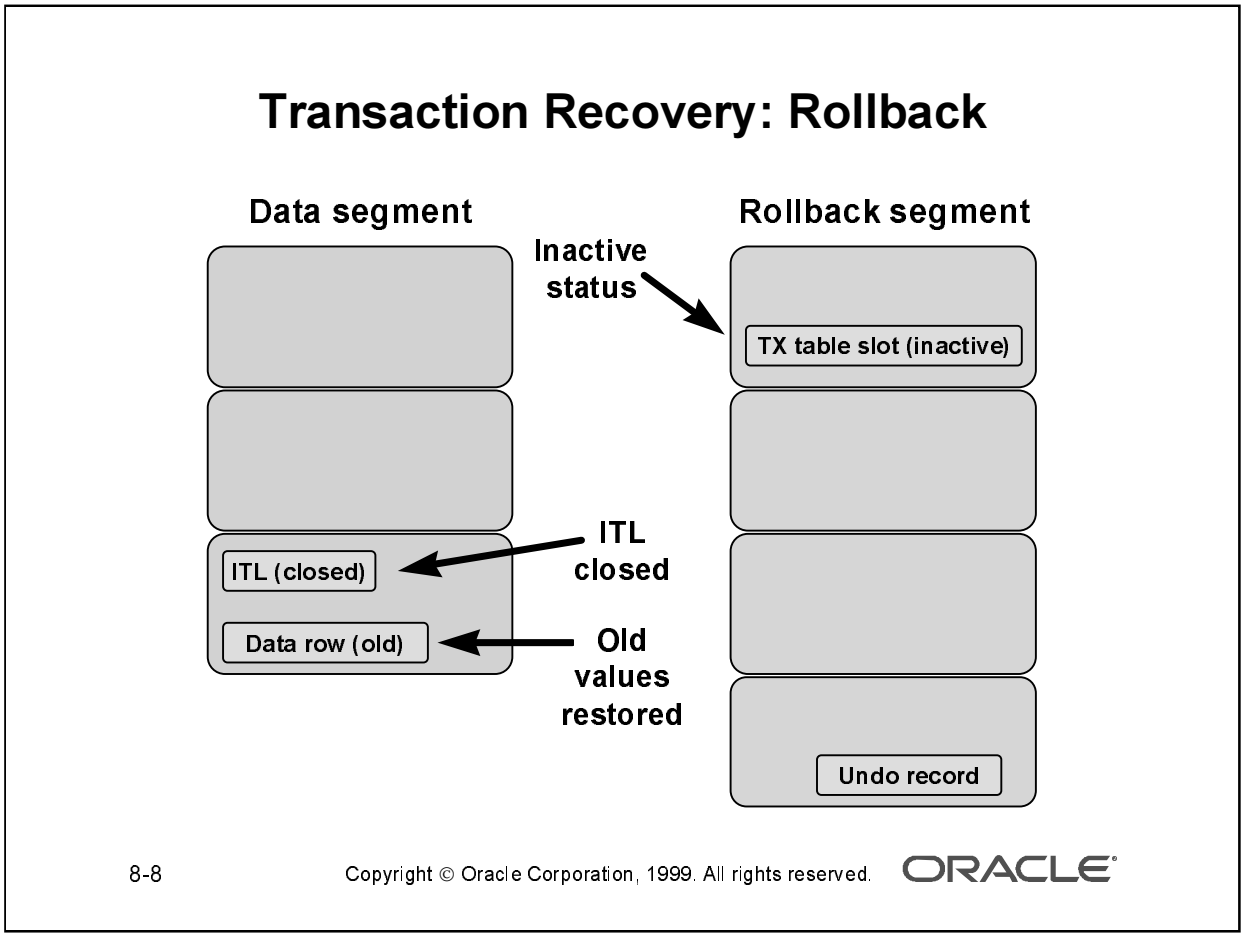
In release 8.1.3, the delayed_logging_block_cleanouts parameter has been removed and its behavior is incorporated into the RDBMS.

Note: The above description is a simplification of the exact DLBC mechanism.

Transaction Recovery

Transaction recovery (the process of rolling back transactions) is performed:

- By the shadow process when a rollback statement is issued
- By PMON when a session (process) crashes with a transaction in progress
- By SMON or a shadow process on opening a database that crashed with active transactions

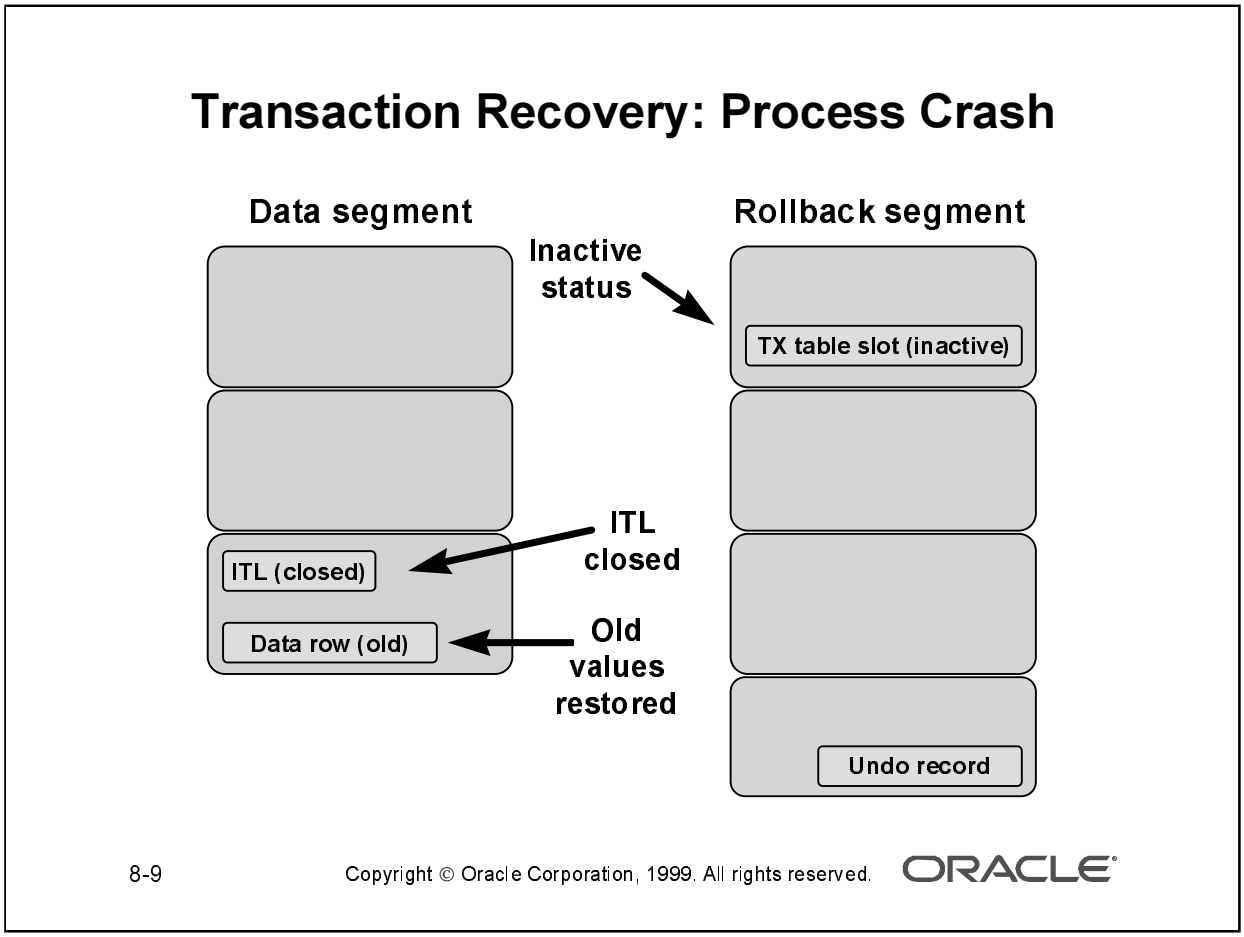


Transaction Recovery On Rollback

When a rollback statement is issued, the RDBMS scans the undo records for the current transaction in reverse order (latest first) and applies them to the database. When the statement returns, all the block changes have been undone and the ITL has been cleared. There is no delay on rollback.

The rollback operation requires access to the rollback segment header, undo records, and data blocks. If any of these are corrupted, then rollback will be affected.

Note that the process of performing rollback generates new redo, which is written to the redo logs.



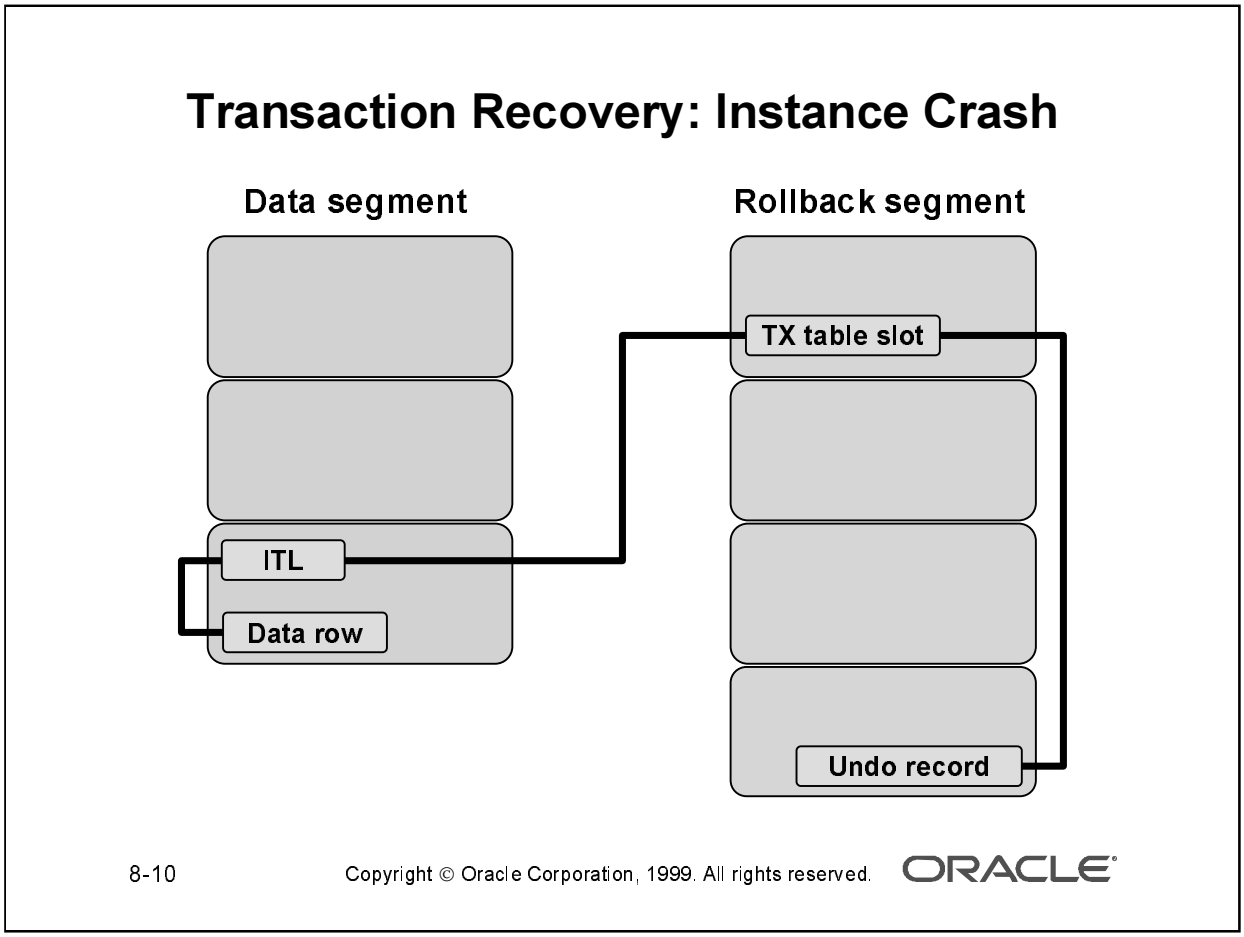
Transaction Recovery After a Process Crash

If the RDBMS shadow process crashes with an active transaction, PMON detects the failure and immediately rolls back the transaction. You can monitor this by setting the following events in the init.ora file or on the PMON process:

- 10012 trace name context forever, level 1 (abort transaction)
- 10246 trace name context forever, level 1 (trace PMON actions to trace file + IO slave trace)

Example

```
Dump file /u01/app/oracle/admin/V805/bdump/v805_pmon_223259.trc
...
Fri Dec 12 11:18:00 1997
*** SESSION ID:(1.1) 1997.12.12.11.18.00.000
marked process 8002a17c pid=12 serial=4 dead
Fri Dec 12 11:18:00 1997
deleting process 8002a17c pid=12 seq=4
Synching redo buffers
ABORT TRANSACTION - xid: 0x0004.005.0000019d
deletion of process 8002a17c pid=12 seq=4 successful
```



Transaction Recovery After an Instance Failure

If the RDBMS instance crashes before the transaction is committed, there is no time to roll back the transaction. The next time the database is opened, crash recovery rolls the database forward, returning it to its precrash state (as above). It is then the responsibility of transaction (rollback) recovery to remove any incomplete transactions.

Transaction Recovery: Database Open

- **Active transactions in the SYSTEM rollback segment are immediately rolled back.**
- **Active transactions in other rollback segments are marked as "dead."**
- **At a later time, SMON scans the segments again and performs a rollback on dead transactions.**

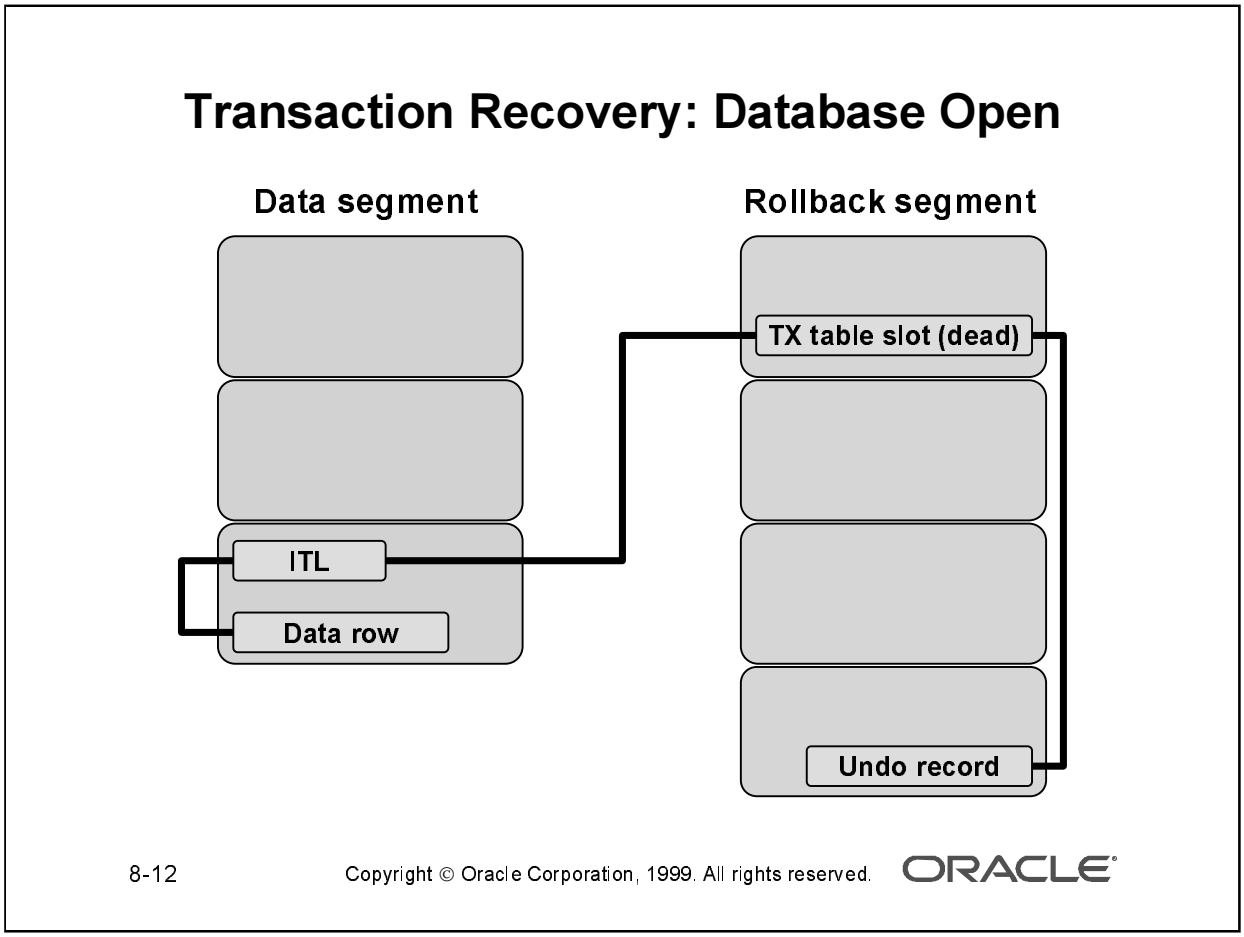
Transaction Recovery at Database Open

The RDBMS starts up as fast as possible after an untidy shutdown. This is especially useful when downtime must be kept to an absolute minimum. Another benefit is that if a new transaction wants to update a row locked by a dead transaction, the new transaction rolls back the transaction itself and does not have to wait for SMON.

In release 7.2 and earlier, all active transactions are rolled back before the database is opened; a corrupt rollback segment typically prevents the database from opening.

A side effect of the release 7.3 changes is that databases now open, in most cases, even if a rollback segment is corrupt, with errors logged to alert and trace files (SMON). This makes it much easier to diagnose the failure, because you have access to the database. However, it would be easy for customers to run for some time without realizing that they have a problem. Note that even in Oracle8, if the server is unable to read the rollback segment header (because the data file is offline or corrupted), then you cannot open the database.

All rollback segments (found in undo$), not just those specified with the rollback_segments parameter, are checked for active transactions when the database is opened.



Transaction Recovery at Database Open (continued)

In Oracle8, you can list dead transactions by issuing the following query:

```
SQL> select * from x$ktuxe where ktuxecfl='DEAD';
```

You can also find dead transactions by dumping the rollback segment header and checking the cflags column of the transaction table dump. To dump the rollback segment header, use the following command:

```
SQL> alter system dump undo header R01;
```

where *R01* is the rollback segment name. A dead transaction is identified by having cflags = '0x10'.

Example

```
Undo Segment: R01 (2)
TRN TBL
index state cflags wrap#  uel    scn             dba         parent-xid
----- ----- ------ ------ ------ --------------- ----------- -------------------
0x00     9   0x00   0x0452 0x0003 0x0465.001a3382 0x008015f8 0x0000.000.00000000
0x11     9   0x10   0x044e 0x0006 0x0465.001a81df 0x00000000 0x0000.000.00000000
```

Rollback Segment Acquisition

- After transaction recovery, rollback segments are acquired by the instance.
  - Segments specified by the rollback_segments parameter are acquired first.
  - Public rollback segments may also be acquired when necessary.
- Acquisition failure typically causes ORA-1545: "rollback segment %s specified not available."

Rollback Segment Acquisition

The number of rollback segments acquired by an instance is governed by the transactions and transactions_per_rollback_segment initialization parameters.

The requirement is:

1 (for system) + transactions / transactions_per_rollback_segment

This number is rounded up to the next whole number.

A segment with a status of "Needs Recovery" cannot be acquired when the database is opened, so if it is specified in rollback_segments, the instance will not open (ORA-1545).

Rollback segments containing dead transactions are available to be acquired for new transactions.

Diagnostic Events

- **Event 10013: Monitor transaction recovery during startup. To set in init.ora:**

```
10013 trace name context forever, level 10
```

- **Event 10015: Dump rollback segment headers (excluding those not listed in undo$) before and after transaction recovery. To set in init.ora:**

```
10015 trace name context forever, level 10
```


Diagnostic Events

Sample output from diagnostic event 10013:

```
Recovering transaction (4, 5)
Fully recovered transaction (4, 5) by (1) rec
```

Sample output from diagnostic event 10015:

```
UNDO SEG (BEFORE RECOVERY): usn = 4  Extent Control Header
-----------------------------------------------------------
Extent Control:: inc#: 33808  tsn: 1    object#: 0
                 #extents: 3            end of table: 1012
                 HWM ext#: 1  size: 65  offset: 47
                 Unlocked
UNDO SEG  (AFTER RECOVERY): usn = 4  Extent Control Header
-----------------------------------------------------------
Extent Control:: inc#: 33808  tsn: 1    object#: 0
                 #extents: 3            end of table: 1012
                 HWM ext#: 1  size: 65  offset: 47
                 Unlocked
```

Undocumented Parameters: Overview

```
_offline_rollback_segments
_corrupted_rollback_segments
```

- Undocumented and unsupported initialization parameters, used as a last resort
- Specify a comma-separated list of rollback segment names; for example:

```
_offline_rollback_segments = (r01, r02, r03)
```

- Neither the SYSTEM rollback segment nor segments already listed in rollback_segments can be specified

Undocumented Parameters

Internally, when the database is opened, a list is constructed of all the _offline (defined in ktugdors) and all the _corrupted (defined in ktugdcrs) undo segment numbers (USNs).

From ktucts.h

```
/* ktugt - Kernel Transaction Undo Global Type */
struct ktugt
{
#ifdef SYS_DFS
  ksils          ktugdiup;        /* instance existence lock */
  ksils          ktugdisc;        /* instance lock to init SCN */
#endif
   kusn          *ktugdors;       /* list of offline rollback
   segment */
   kusn          *ktugdcrs;       /* list corrupted rollback
   segments */
   ...
```

A function called ktuuon( ) checks these lists and returns:

- KTUINVAL (kernel transacion undo invalid) if the segment is _corrupted
- KTUAVAIL (kernel transaction available) otherwise

Undocumented Parameters (continued)

From ktu.h

```
/*
 * ktuuon - Kernel transaction Undo Undo Online?
 * returns KTSINVAL - if undo is indicated to be corrupted in
   init.ora
 * returns KTSOFFL  - if undo is indicated to be offline in
   init.ora
 * else returns KTSAVAIL
 */
```

Example init.ora File (fragment)

```
rollback_segments             = (r01, r02, r03)
_corrupted _rollback_segments = (r04)
```

Note

Unsupported means:

- The Oracle server is not designed to work with these parameters
- The parameters have not been tested, so you may be the first to try them in this combination
- The consistency of the database will always be in-doubt afterwards
- There is a high risk of unexpected and undesirable side effects

Undocumented Parameters: Overview

Using the _offline_rollback_segments or _corrupted_rollback_segments parameters changes the behavior of the RDBMS when:

- Opening the database
- Performing consistent read and delayed block cleanout
- Dropping a rollback segment

Undocumented Parameters: On Database Open

When opening a database, any rollback segments listed in _offline or _corrupted parameters:

- **Are not scanned, and any active transactions are neither marked as dead nor rolled back**
- **Appear offline in dba_rollback_segs (undo$)**
- **Cannot be acquired by the instance for new transactions**

Undocumented Parameters (continued)

When using these undocumented parameters, the transaction table is not read when the database is opened, so transactions are not marked as dead or rolled back. This is where the situation becomes dangerous, because if there are active transactions in the segment, they become highly abnormal.

They are not dead, and yet they belong to a session that was active before the instance started.

Unexpected and undesirable effects will occur; remember: *The database is in an unsupported state*.

Undocumented Parameters: On Database Open

- When opening a database, _corrupted additionally causes all distributed transactions for a named rollback segment to be marked as “forced commit”.
- This is done by updating pending_trans$ and dba_2pc_pending (two phase commits).

Undocumented Parameters (continued)

In the situation described in the slide, the rollback segment itself is not updated; the Oracle server just performs an update on pending_trans$ and dba_2pc_pending.

Undocumented Parameters: CR and Cleanout

If an open ITL is found to be associated with an _offline segment, the segment is read to find the transaction status.

- **If committed, the block is cleaned out.**
- **If active and you want to read the block, a CR copy is constructed using undo from the segment.**
- **If active and you want to lock the row, *undesirable behavior* may result.**

CR and Cleanout

Note that although the rollback segment is _offline, the Oracle server actually reads the segment to find transaction statuses and to gather undo records to perform rollback. If a corrupt block is found, the Oracle server will still fail.

When you update a block covered by an active dead transaction, the shadow process loops indefinitely, consuming CPU. Presumably, this is because the old transaction appears active, so the Oracle server attempts to wait for the TX enqueue. You get this enqueue immediately, because the old transaction is no longer there. The Oracle server then checks the block again and finds it is still active, so the process is repeated again and again.

Undocumented Parameters: CR and Cleanout

- If an open ITL is found to be associated with a _corrupted segment, the segment is not read to find the transaction status.
- It is as if the rollback segment had been dropped; the transaction is assumed to be committed and delayed block cleanout is performed.
- If the transaction was not committed, logical corruption will occur.

CR and Cleanout (continued)

Most important, the Oracle server does not read the segment in this case. It is as if the segment has been dropped. This is the most important difference between _offline and _corrupted.

Note that if some ITLs are cleared and the rollback segment is then reintroduced to the database (by removing the _corrupted parameter), you may try to roll back a block that you just committed.

Tests show that this typically causes data or block corruption. To prevent this from happening, always drop a rollback segment if you have ever marked it as _corrupted.

Undocumented Parameters: Drop Rollback Segment

- Typically, a rollback segment is prevented from being dropped if it contains active transactions.
- When dropping an _offline or _corrupted rollback segment, this check is not performed.
- As a result, the segment may be dropped even if it contains active transactions.

Dropping Rollback Segments

Normally you cannot drop a rollback segment if it contains active transactions. You can circumvent this by using the parameters discussed in this lesson.

If you drop an _offline or _corrupted rollback segment that contains active transactions, you risk logical corruption, possibly in the data dictionary.

Always make sure to change your database back into a supported state by solving the problems, removing the special settings in your parameter file, shutting the instance down, and performing a normal startup.

Although the database may seem to run smoothly, certain corruption problems can come back even after a long time, potentially causing a lot more problems than they did in the first place.

Recovery From Corruption

- In all cases, media recovery is the preferred recovery approach, because it is supported and guarantees consistency.
- When media recovery is impossible, alternative techniques are suggested and the risks explained.

Recovery From Corruption

In this lesson, the term object corruption means that a block has become corrupt in a user object (table, index, cluster) that is part of an active transaction. The block is typically found to be corrupt when the RDBMS attempts to roll back the transaction.

Recovery Guidelines

- **Rollback segment corruption can interfere with CR, row locking, block cleanout, and rollback.**
- **If a rollback segment header is corrupt, database startup is prevented.**
- **Undocumented parameters may enable recovery; however, they should only be used as a last resort, and only when the many risks are understood and explained to the customer.**

Recovery Guidelines

Remember at all times that the customer owns his or her database. When you want to help with recovery or patching, you must first consult with the customer to see what can be done, explain what you want to do, explain the risks and the time involved, and also the time it would take if the recovery fails. Then ask the customer if he or she agrees to the proposed plan, and go ahead. This cannot be stressed enough; it may seem obvious, but it is often neglected.

These undocumented parameters are very dangerous: tell the customer what they can do to his or her database *before* using them. When you are not exactly sure what they do, ask for assistance.

References

Oracle8 Rollback Segment Corruption Recovery
by Richard Exley and Ashish Prabhu

Practice 8 Overview

This practice covers the following topics:

- **Identifying block corruption**
- **Recovering from corruption**

Practice 8 Overview

For detailed instructions on performing this practice, see Practice 8 in Appendix B. Additional case studies are available in Appendix A.

9

Data Salvage Using the Data Unloader Utility (DUL)

ORACLE

Schedule:	Timing	Topic
	60 minutes	Lecture
	15 minutes	Examples
	45 minutes	Lab Exercises
	120 minutes	Total

Objectives

After completing this lesson, you should be able to do the following:

- **Describe the features of DUL**
- **Explain the limitations of DUL**
- **Configure DUL**
- **Use DUL**

Data Unloader: Usage

- Developed by Bernard Van Duijnen
- Stand-alone utility
- Unloads data from data files of a crashed database
- Creates SQL*Loader control files and data files

Data Unloader: Usage

A useful utility has been developed in the Netherlands, called the Data Unloader (DUL). DUL is a standalone C program that directly retrieves rows from tables in data files; the RDBMS is not necessary.

DUL is intended to retrieve data from the database that cannot otherwise be retrieved. The database may be corrupted, but an individual data block to be used by DUL must be 100% correct.

DUL unloads data from files containing table and cluster data. It creates no scripts for triggers, procedures, tables, and views; it can only read the definitions from the data dictionary tables.

DUL can create loader files or export files.

DUL Features

- Implemented on Oracle versions 6, 7, and 8
- Available on many platforms
- Not an alternative to standard recovery methods
- Not a supported utility
- Used as a last resort only

DUL Features

DUL version 3 is implemented and tested for Oracle7. The most recent version is DUL version 8; it is implemented for Oracle version 8.0.4.

DUL is available on the following platforms: Sequent/ptx, Vax Vms, Alpha Vms, MVS, HP9000/8xx, IBM AIX, SCO Unix, Linux, Alpha OSF/1, Intel Windows NT, and Windows95.

The executables can be obtained from the Dutch intranet site `http://www.nl.oracle.com/support/dul/`. For questions and discussions, there is a mailing list helpdul.nl.

DUL is not a supported utility; that is, there is no guarantee for bug fixes.

DUL should be used as a last resort, after all other recovery alternatives have been exhausted. This includes the use of destructive events. However, to use these destructive events, the database must be in reasonably good shape to be opened and to remain open while an export is being performed. There are examples of databases that were restored from two disparate backups taken two weeks apart. In this situation it is often very difficult to successfully export, even if the database can be opened.

DUL Features

- Reads data dictionary if files available
- Can work without dictionary files
- Can unload individual tables and schemas, or the complete database
- Supports cross-platform unloading

DUL Features (continued)

If the system tablespace files are available, DUL reads the data dictionary information and unloads the tables user$, obj$, tab$, and col$. Old system tablespace backup files from the same database are better than nothing, and can also be used.

If the system data files are not available, DUL scans the data files for segments and extents, and scans the found segments for rows. Then you unload the found segments with the UNLOAD TABLE command. To do this, you must have a good knowledge of the database structure, such as column and table storage information.

Databases can be unloaded on platforms other than the DUL host platform. Simply copy the data files, and modify the configuration files appropriately.

DUL Support

The following standard constructs are supported:

- **Chained or migrated rows**
- **Hash or index clusters**
- **NULLs, trailing NULL columns**
- **LONGs, RAWs, DATEs, NUMBERs**
- **ROWIDs (Oracle7 and Oracle8 format)**
- **Multiple free list groups**
- **Unlimited extents**
- **Partitioned tables**
- **Index-organized tables**

DUL Support

DUL supports the standard data types for Oracle8, and some Oracle8 features such as index-organized tables and partitioned tables. LOB support is under development.

DUL Restrictions

- The database can be corrupt, but individual blocks must be correct
- DUL unloads LONG RAWs but LONG RAW data cannot be loaded by using SQL*Loader
- No support for MLSLABELs
- No support for multibyte character sets
- No support for lobs, varrays, objects, and nested tables
- DUL performs dirty reads

DUL Restrictions

The database may be corrupted, but an individual data block that is used must be correct. During all unloading, blocks are checked to verify that they are not corrupted and belong to the correct segment. If a corrupted block is detected, an error message is generated to the loader file.

There is no suitable format in SQL*Loader to preserve all LONG RAWs. Use the export mode instead, or write a Pro*C program to load the data.

DUL is essentially a single byte application. The command parser does not understand multi-byte characters.

Multi-level security labels (MLSLABELs, Trusted Oracle only) are not supported.

Lobs, varrays, objects, and nested tables are not yet supported.

DUL performs dirty reads from the data files.

Data will likely be logically inconsistent, and therefore will have to be revalidated.

Configuring DUL

Two configuration files are necessary:

- **init.dul**
- **control.dul**

DUL Configuration Files

There are two configuration files for DUL:

- init.dul contains all configuration parameters, such as size of caches, details of header layout, Oracle block size, and output file format.
- In the control file, control.dul, the data file names and the Oracle file numbers must be specified.

Note: The following slides explain the usage of DUL version 8.

init.dul Parameters

The init.dul file contains the following database-specific parameters:

- **Database configuration parameters**
- **Dictionary cache information:**
  - **dc_columns**
  - **dc_tables**
  - **dc_objects**
  - **dc_users**

init.dul: Database-Specific Parameters

The init.dul file contains database configuration parameters such as db_block_size or compatible and dictionary cache information.

The DUL cache must be large enough to hold all entries from the dictionary tables col$, tab$, obj$, and user$. If the DUL cache is too small, DUL does not start and reports the following error:

```
DUL: Error: parameter dc_xxxxxx too low
```

The dc_xxx parameter must be increased and DUL restarted.

Note: Oracle version 6 had more than 20 dc_ initialization parameters. These parameters were replaced in Oracle7 by the shared_pool_size parameter.

init.dul Parameters

OS-specific parameters:

- **osd_big_endian_flag**
- **osd_dba_file_bits**
- **osd_c_struct_alignment**
- **osd_file_leader_size**
- **osd_word_size**

init.dul: OS-Specific Parameters

- osd_big_endian_flag (boolean) determines whether or not the platform is byte-swapped. DUL8 sets the default according to the machine it is running on.
- HP, Sun, and mainframes generally are big endian (true);
  Intel and DEC are little endian (false).
- osd_dba_file_bits is the number of bits in a dba used for the low order part of the file number. For example, set osd_dba_file_bits on Sun Sparc Solaris to 10 (Oracle8).
- osd_c_struct_alignment is the C structure member alignment (0,16, or 32), and must be set to 32 for most ports.
- osd_file_leader_size contains the number of bytes/blocks added before the real Oracle file header block.
- osd_word_size is the size of a machine word; always 32, except for MS-DOS(16).

Note: A complete list of OS-specific parameters and the values for the required platforms can be obtained from the Web site `http:/www.nl.oracle.com/support/dul`.

init.dul Parameters

Other common parameters:

- **control_file**
- **ldr_enclose_char**
- **ldr_phys_rec_size**
- **buffer**
- **compatible**
- **export_mode**

init.dul: Other Parameters

- control_file is the name of the DUL control file (default: control.dul).
- ldr_enclose_char contains the character to enclose fields in SQL*Loader mode.
- ldr_phys_rec_size is the physical record size for the generated loader data file:
  - 0: No fixed records, each record is terminated with a newline.
  - >2: Fixed record size.
- buffer indicates the row output buffer size (used in export mode only).
- compatible is the database version; valid values are 6, 7, or 8.
  This parameter must be specified.
- export_mode (boolean) determines whether the output mode is the export format or SQL*Loader format.

init.dul Example

```
# sample init.dul configuration parameters
# Database specific parameters
dc_columns = 2000000
dc_tables = 10000
dc_objects = 100000
dc_users = 400
dc_segments = 100000
db_block_size = 2048
compatible = 8.0
# O/S specific parameters
osd_big_endian_flag = true
osd_dba_file_bits = 10
osd_c_struct_alignment = 32
osd_file_leader_size = 1
osd_word_size = 32
# Other parameters
export_mode = true
control_file= /u05/home/dsi/bin/control.dul
```


init.dul Example

See also <Note:72554.1> and the Web site
`http://www.nl.oracle.com/support/dul` for more details.

control.dul

- The control file contains the mapping of file numbers to file names (see v$datafile).
- The format for Oracle8 files is:

```
tablespace_no relative_file_number data_file_name
[optional extra leader offset]
[startblock block_no] [endblock block_no]
```

- Each entry is located on a separate line and can contain a part of a datafile.

The DUL Control File

A DUL control file (default name: control.dul) is used to translate the file numbers to file names. The order is important. The optional extra leader offset is an extra byte offset that will be added to all lseek( ) operations for that data file. This makes it possible to skip over the extra block for AIX on raw devices or unload from fragments of a data file.

Note: Refer to the *DUL User's and Configuration Guide* for the the syntax of Oracle7 or Oracle6 file format.

control.dul Example

```
# sample Oracle8 control file
0 1 /u05/home/dsi/dsi301/TEAMB/systemTEAMB01.dbf
1 2 /u05/home/dsi/dsi301/TEAMB/rbsTEAMB01.dbf
2 3 /u05/home/dsi/dsi301/TEAMB/tempTEAMB.dbf
3 4 /u05/home/dsi/dsi301/TEAMB/data01TEAMB.dbf
4 5 /u05/home/dsi/dsi301/TEAMB/index01TEAMB.dbf
```


control.dul: Example

Issue the following query and edit the spool file to receive a valid control file:

```
SQL> select ts#, rfile#, name
  2  from   v$datafile;
```

Using DUL

1. Create an appropriate init.dul and control.dul file.
2. Unload the object information.
3. Invoke DUL.
4. Rebuild the database.

Using DUL

The subsequent section discusses how to load data in the case of available or unavailable system tablespace files.

Unload Data:
With System Tablespace Files

1. Create init.dul and control.dul files.
2. Unload the dictionary tables with the appropriate ddl script; for example, on UNIX:

```
$ dul dictv8.ddl
```

3. Restart DUL and unload the objects; for example:

```
DUL> unload table <username.table>;
DUL> unload user <username>;
DUL> unload database;
```


How to Unload Data With Available System Tablespace Files

1. Create the init.dul and the control files.
2. Unload the dictionary tables with the appropriate DDL script.
3. Invoke the DUL in interactive mode.

Commonly Used DUL Commands

The most commonly used DUL commands are:

```
DUL> unload database;
```

This unloads the entire database.

```
DUL> unload user scott;
```

This unloads the schema SCOTT.

```
DUL> unload table scott.emp;
```

This unloads the specified table.

Unload Data: Without System Tablespace Files

1. Create init.dul and control.dul files.
2. Start DUL and scan the database for segment headers and extents:

```
DUL> scan database;
```

3. Restart DUL and scan the found tables for column statistics:

```
DUL> scan tables;
```

4. Identify the scanned tables.
5. Unload the identified tables.

How to Unload Data Without System Tablespace Files

1. Create the init.dul and the control files.
2. Scan all blocks of all data files with the following command:

```
DUL> scan database;
```

The two following two files are generated:

- seg.dat contains information regarding found segment headers.
- ext.dat contains information regarding contiguous table/cluster data blocks.

3. Indicate to DUL that it should use its own generated extent map rather than the segment header information, and scan all tables in all data segments using the commands:

```
DUL> alter session set use_scanned_extent_map=true;
DUL> scan tables;
```

You can also use:

```
DUL> scan extents;
```

4. Identify scanned tables. The scanned information should look familiar. Use this information and your knowledge of the table to compose the unload syntax of the lost tables.
5. Unload the identified tables; for example, with the following command:

```
DUL> unload table dept (deptno number,dname
DUL 2> varchar2(13),loc varchar2(13))storage(objno 1078);
```

Rebuilding the Database

- **Load the database by using Import.**
- **Load the database by using SQL*Loader.**

Rebuilding the Database

To rebuild the database, load the unloaded information with the Import utility or the SQL*Loader utility.

Load the Database by Using Import

If you used the export mode to unload the data, DUL generated for each table a separate import file with minimal information in it. That is, grants, storage clauses, or triggers are not included. The output file name generated by DUL is ownername_tablename.dmp; for example, SCOTT_EMP.dmp.

Load the Database by Using SQL*Loader

For SQL*Loader output formats, the columns are separated by spaces and enclosed in double quotes. The output file names generated by DUL are ownername_tablename.dat for the data file and ownername_tablename.ctl for the control file.

References

- DUL Web page:
  - `http://www-sup.nl.oracle.com/support/dul`
  - DUL User's and Configuration Guide
- Note 72554.1:
  Using DUL to Recover from Database Corruption

Practice 9 Overview

This practice covers unloading user data with available system tablespace files.

Practice 9 Overview

For detailed instructions on performing this practice, see Practice 9 in Appendix B. Additional case studies are available in Appendix A.

A
Supplemental Information

Case Studies 2: Source Code, The Ultimate Oracle

ORA-600 Examples:
A. ORA-600 [4137] (Oracle Version 7.3.4)

```
tao-sun% cdb 734
tao-sun% find600 4137
Searching for OERI(4137)
File:   /export/home/ssupport/734/rdbms/src/server/txn/txmgmt/ktur.c
Line #: 1733
    ksesic0(OERI(4137));
```

View the file.

```
1. /* NAME
2.      ktur.c - Kernel Transaction Undo Recovery
3.    FUNCTION
4.      Undo routines dealing with recovery
```

/4137

```
1. if (!KXIDEQ(xid, &ubh->ktubhxid))            /* make sure the txid matches */
2.    {
3.      ksdwrf("XID passed in =");
4.      KXIDDMP(xid);
5.      ksdwrf("\nXID from Undo block = ");
6.      KXIDDMP(&ubh->ktubhxid);
7.      ksdwrf("\n");
8.      ksesic0(OERI(4137));
9.    }
```

Scan upwards.

```
1. /*
2.  * ktubko - Kernel transaction Undo BacK Out
3.  *
4.  *          Get the undo block (current and share) pointed to by the uba
5.  *          and retrieve the next undo record to rollback.
6.  *          Calls ktundo to actually backs out the change.
7.  *          Also returns the uba of the next undo record to apply
8.  *          WARNING: rpi call made, caller should not have any pinned blks
9.  *
10. */
11. STATICF void        ktubko(xid, uba, flag, udes, ubdes, nuba)
12.      kxid  *xid;                                          /* pointer to xid */
13.      kuba  *uba;                                 /* uba of undo record to apply */
14.      word   flag;                                /* action flag: KTU_XR, KTU_XA */
15.      ktusd  *udes;                      /* undo segment descriptor (not pinned) */
16.      kcbds  *ubdes;                       /* undo block descriptor (not pinned) */
17.      kuba   *nuba;/* uba of the next undo record to apply after this (return) */
18. {
19.    ptr_t   bptr;                                        /* undo block pointer */
20.    kturx   *ctx;                          /* context for retrieving undo record */
21.    ktubh  *ubh;                                          /* undo block header */
22.    word    level;
```

The RDBMS is rolling back a transaction with id *xid*. It does this by scanning the rollback segment (undo blocks) and undoing the changes made by the transaction, one record at a time. As a check, it compares the *xid* with the transaction id in the undo block header *ubh.ktubhxid*. If the check fails, it logs *xid*, *ubh.ktubhxid* and ORA-600 [4137].

B. ORA-600 [17069] [3523916156] (Oracle Version 8.1.5)

```
tao-sun% cdb 815
tao-sun% find600 17069
$ find600 17069
Searching for OERI(17069) or OERINM("17069")
File:   /export/home/ssupport/815/rdbms/src/generic/dict/libcache/kgl.c
Line #: 2394
    if (i >= 50) kgesic1(gp, kgefac(gp), OERI(17069), kgenpa(hd));
```

View kgl.c with cscope.

```
1.  NAME
2.      kgl - Kernel Generic Library cache manager
3.    FUNCTION
4.      Manages the library cache.
```

/17069

```
1.     /* blow off if we have been retrying for 50 times */
2.     if (i >= 50) kgesic1(gp, kgefac(gp), OERI(17069), kgenpa(hd));
```

Scan upwards.

```
1.  /* kglgob - KGL GeT an Object locked and pinned */
2.  kglhd  *kglgob(gp, ds, lock_mode, pin_mode,
3.                 lock_persist, pin_persist, lk_, pn_)
4.  reg2
5.  kgsmp  *gp;                                          /* client's generic PGA */
6.  kglds  *ds;              /* descriptor of the object to be locked and pinned */
7.  eword   lock_mode;                                       /* mode of the lock */
8.  eword   pin_mode;                                        /* mode of the pin */
9.  eword   lock_persist;                               /* persistence of the lock */
10. eword   pin_persist;                                /* persistence of the pin */
11. kgllk **lk_;                               /* library object lock to be gotten */
12. kglpn **pn_;                                /* library object pin to be gotten */
13. {
14.   reg3 kglhd        *hd;                /* handle of the object to be locked */
15.   reg0 kglob        *ob;                              /* object to be locked */
16.   reg1 CONST kglsf  *sf = &kgsfpt(gp)->kgsfkgl; /* client's service function */
17.   ub4                flags;                                    /* saved flags */
18.   kgllk             *lk;                             /* library object lock */
19.   kglpn             *pn;                              /* library object pin */
20.   eword              i;                                        /* counter */
```

It looks like the RDBMS has been looping. What does the loop do?

```
1.  /*
2.     * Lock and pin the desired object in the desired modes. If the object is
3.     * wanted to be locally represented but is not (e.g., no obj$ row
4.     * in the database), is wanted to be valid but is invalid, or is wanted to
5.     * be authorized but is not, create, recompile, or reauthorize
6.     * the desired object by calling the callback service
7.     * function for creating, reauthorizing, or recompiling an object.
8.     * Repeat the above process until the desired object is locked and
9.     * pinned in the desired modes and has the wanted status.
10.    */
11.   for (i = 0;; ++i)
12.   {
13.     noreg eword   operation;                  /* validation operation code */
14.     eword   e;                                          /* error code */
15.
16.     /* get and lock the object in the library cache */
17.     if (!(hd = kglget(gp, ds, lock_mode, lock_persist, &lk)))
18.     {
```

```
19.       /*
20.        * Failed to get the handle. This can only happen if this is a
21.        * get by dependency and the dependency is invalid. In this case,
22.        * just return null.
23.        */
24.       KGEASSERT1(gp, ds->kgldsbyw == KGLDSBYD, kgefac(gp),
25.                  OERI(17071), KGESVSGN, ds->kgldsbyw);
26.       break;
27.     }
28.
29.     /* pin the object in the library cache */
30.     KGEONERROR(gp, e)
31.     {
32.       kglrls(gp, &lk);
33.       KGERESIGERROR(gp);
34.     }
35.     KGEENDERROR(gp)
36.       /* pin the body library object */
37.       ob = kglpin(gp, ds, lk, hd, pin_mode, pin_persist, &pn);
38.     KGEPOPERROR(gp)
39.     if (!ob)
40.     {
41.       /*
42.        * Failed to pin. If it is because the lock is broken and this is
43.        * the first lock, release the lock and retry.
44.        */
45.       if (ds->kgldssta == KGLDSBRO && lk->kgllkcnt == 1)
46.       {
47.         kglrls(gp, &lk);
48.         continue;
49.       }
50.
51.       /* otherwise, just return null */
52.       kglrls(gp, &lk);
53.       hd = (kglhd *)0;
54.       break;
55.     }
56.
57.     /* blow off if we have been retrying for 50 times */
58.     if (i >= 50) kgesic1(gp, kgefac(gp), OERI(17069), kgenpa(hd));
```

The RDBMS has been trying to lock and pin an object in the library cache. After 50 attempts, it quits and generates an ORA-600 [17069] [hd] - hd is the object handle.

Case Studies 4: Hang, Loop and Crash Diagnostics

A. After execution of the 'alter session set events immediate trace name systemstate level 10' command, the following system state dump will be created in the user_dump_dest directory:

```
1.  ********************************************************************************
2.  Header section : Generic System and Database Information
3.  ********************************************************************************

4.  Dump file /home/users/nvengurl/admin/udump/ora_4801.trc
5.  Oracle8 Enterprise Edition Release 8.0.5.0.0 - Production
6.  With the Partitioning and Objects options
7.  PL/SQL Release 8.0.5.0.0 - Production
8.  ORACLE_HOME = /u03/app/oracle/product/8.0.5
9.  System name:       OSF1
10. Node name: charlie1.us.oracle.com
11. Release:   V4.0
12. Version:   464
13. Machine:   alpha
14. Instance name: nitin
15. Redo thread mounted by this instance: 1
16. Oracle process number: 16
17. Unix process pid: 4801, image: oraclenitin
18.
19. Fri Mar  6 16:18:52 1999
20. *** SESSION ID:(16.2) 1999.03.06.16.18.52.970
21. ==============================================
22. SYSTEM STATE
23. ------------
24.
25. ********************************************************************************
26. Information same as from v$sysstat.
27. ********************************************************************************
28.
29. System global information:
30.      processes: base 40002adf0, size 50, cleanup 40002b5b0
31.      allocation: free sessions 400041010, free calls 40005d468
32.      control alloc errors: 0 (process), 0 (session), 0 (call)
33.      system statistics:
34.              logons cumulative         11           0
35.                 logons current          0           0
36.           opened cursors cumulative   291           0
37.           opened cursors current        0           0
38.                   user commits          1           0
39. ....
40. ....
41.
42. ********************************************************************************
43.
44.
45. ********************************************************************************
46. PROCESS 1:     -----> PSEUDO process
47.   ----------------------------------------
48.   SO: 40002b1d0, type: 1, owner: 0, flag: INIT/-/-/0x00
49.   (process) Oracle pid=1, calls cur/top: 0/0, flag: (10) PSEUDO
50.             int error: 0, call error: 0, sess error: 0, txn error 0
51.     OSD pid info: Unix process pid: 0, image:    ----> note pid = 0
52. ********************************************************************************
53. PROCESS 2:       -----> PMON
54.   ----------------------------------------
55.   SO: 40002b5b0, type: 1, owner: 0, flag: INIT/-/-/0x00
56.
57. ********************************************************************************
58.
59. PROCESS 3:       -----> DBW0
60.   ----------------------------------------
61.   SO: 40002b990, type: 1, owner: 0, flag: INIT/-/-/0x00
62.   (process) Oracle pid=3, calls cur/top: 40005c500/40005c500, flag: (2) SYSTEM
63.             int error: 0, call error: 0, sess error: 0, txn error 0
64.   (post info) last post received: 0 0 23
65.               last post received-location: ksfvrdp
```

```
              last process to post me: 40002d890 1 2      -----> note
              last post sent: 0 0 22
              last post sent-location: ksfvsubmit
              last process posted by me: 40002d890 1 2     -----> note
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ?, ospid: 31488
    OSD pid info: Unix process pid: 31488, image: ora_dbwr_nitin
    ----------------------------------------
    SO: 400037b78, type: 3, owner: 40002b990, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (51) USR/- BSY/-/-/-/-/-
              DID: 0001-0003-00000002, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    waiting for 'io done' seq=15657 wait_time=0      ------> note
                msg ptr=0, =0, =0
    ----------------------------------------

**********************************************************************

PROCESS 4:   ------> LGWR
  ----------------------------------------
  SO: 40002bd70, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=4, calls cur/top: 40005c588/40005c588, flag: (2) SYSTEM
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 0 0 15
              last post received-location: ksasnd
              last process to post me: 40002e430 1 0
              last post sent: 43 0 4
              last post sent-location: kslpsr
              last process posted by me: 40002e430 1 0
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ?, ospid: 32284
    OSD pid info: Unix process pid: 32284, image: ora_lgwr_nitin
    ----------------------------------------
    SO: 40007dbe0, type: 4, owner: 40002bd70, flag: INIT/-/-/0x00
    (enqueue) RT-00000001-00000000       DID: 0000-0004-00000002
    lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
    res:400092558, mode: X, prv: 400092568, sess: 400038560, proc: 40002bd70
    ----------------------------------------
    SO: 400038560, type: 3, owner: 40002bd70, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (51) USR/- BSY/-/-/-/-/-
              DID: 0001-0004-00000003, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    waiting for 'rdbms ipc message' seq=5638 wait_time=0     ----> note
                timeout=12c, =0, =0
    ----------------------------------------
    SO: 40005c588, type: 2, owner: 40002bd70, flag: INIT/-/-/0x00
    (call) sess: cur 400038560, rec 0, usr 400038560; depth: 0

**********************************************************************

PROCESS 5:   -----> CKPT
  ----------------------------------------
  SO: 40002c150, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=5, calls cur/top: 40005c610/40005c610, flag: (2) SYSTEM
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 0 0 0
              last post received-location: No post
              last process to post me: none
              last post sent: 0 0 12
              last post sent-location: ksbrdp
              last process posted by me: 400036bd0 1 0
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ?, ospid: 32646
    OSD pid info: Unix process pid: 32646, image: ora_ckpt_nitin
    ----------------------------------------
    SO: 400038f48, type: 3, owner: 40002c150, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (51) USR/- BSY/-/-/-/-/-
              DID: 0000-0000-00000000, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    waiting for 'rdbms ipc message' seq=5441 wait_time=0
                timeout=12c, =0, =0
    ----------------------------------------
```

```
    SO: 40005c610, type: 2, owner: 40002c150, flag: INIT/-/-/0x00
    (call) sess: cur 400038f48, rec 0, usr 400038f48; depth: 0

*******************************************************************************

PROCESS 6: -----> SMON
  ----------------------------------------
  SO: 40002c530, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=6, calls cur/top: 40005c698/40005c698, flag: (a) SYSTEM
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 0 0 51
              last post received-location: ktmchg
              last process to post me: 40002ccf0 2 0
              last post sent: 79 0 4
              last post sent-location: kslpsr
              last process posted by me: 40002ccf0 2 0
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ?, ospid: 32494
    OSD pid info: Unix process pid: 32494, image: ora_smon_nitin
    ----------------------------------------
    SO: 400039930, type: 3, owner: 40002c530, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (51) USR/- BSY/-/-/-/-/-
              DID: 0001-0006-00000001, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    waiting for 'smon timer' seq=6639 wait_time=0   ------> note
                sleep time=12c, failed=0, =0
    ----------------------------------------
    SO: 40005c698, type: 2, owner: 40002c530, flag: INIT/-/-/0x00
    (call) sess: cur 400039930, rec 0, usr 400039930; depth: 0

*******************************************************************************

 PROCESS 7: -----> RECO
  ----------------------------------------
  SO: 40002c910, type: 1, owner: 0, flag: INIT/-/-/0x00

*******************************************************************************
```

(All the background processes are started at this point, so the subsequent processes are user processes.)

```
 PROCESS 8: -----> user process
  ----------------------------------------
  SO: 40002ccf0, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=8, calls cur/top: 40005c3f0/40005c3f0, flag: (0) -
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 43 0 4
              last post received-location: kslpsr
              last process to post me: 40002bd70 1 2
              last post sent: 0 0 15
              last post sent-location: ksasnd
              last process posted by me: 40002bd70 1 2
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ttypf, ospid: 12216
    OSD pid info: Unix process pid: 12216, image: oraclenitin
    ----------------------------------------
    SO: 40003ad00, type: 3, owner: 40002ccf0, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (141) USR/- BSY/-/-/-/-/-
              DID: 0001-0008-00000001, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 5/SYSTEM
    O/S info: user: nvengurl, term: ttypf, ospid: 2447, machine: charlie1.us.oracle.com
              program: <sqlplus>@charlie1.us.oracle.com (TNS V1-V3)
    application name: SQL*Plus, hash value=3669949024
    waiting for 'rdbms ipc reply' seq=32168 wait_time=0
                from_process=3, timeout=12ab6b8, =0
      ----------------------------------------
      SO: 400267a40, type: 23, owner: 40003ad00, flag: INIT/-/-/0x00
      LIBRARY OBJECT LOCK: lock=400267a40 handle=4005fc1d8 mode=N
      call pin=400268a78 session pin=0
      user=40003ad00 session=40003ad00 count=1 flags=[00] savepoint=6586
      LIBRARY OBJECT HANDLE: handle=4005fc1d8
      name=SYS.DBMS_APPLICATION_INFO
      hash=fcfd1282 timestamp=09-17-1997 16:07:16
      namespace=BODY/TYBD flags=TIM/SML/[02000000]
      kkkk-dddd-llll=0000-0011-0011 lock=N pin=0 latch=0
      lwt=4005fc208[4005fc208,4005fc208] ltm=4005fc218[4005fc218,4005fc218]
```

```
      pwt=4005fc238[4005fc238,4005fc238] ptm=4005fc2d0[4005fc2d0,4005fc2d0]
      lwt=40060ea80[40060ea80,40060ea80]
ltm=40060ea90[40060ea90,40060ea90]
      pwt=40060eab0[40060eab0,40060eab0]
ptm=40060eb48[40060eb48,40060eb48]
      ref=40060ea60[400612498,40061e558]
        LIBRARY OBJECT: object=40060e6b8
        type=PCKG flags=EXS/LOC[0005] pflags=NST [01] status=VALD load=0
        DEPENDENCIES: count=1 size=16
        DATA BLOCKS:
        data#     heap  pointer status pins change
        ----- -------- -------- ------ ---- ------
            0 40060e988 40060e850 I/-/A     0 NONE
            2 40060e7c8 40060db38 I/-/A     0 NONE
            4 40060e420 40066ceb0 I/-/A     0 NONE
            9 40060e378 400609a40 I/-/A     0 NONE
      ----------------------------------------
      SO: 4006d3908, type: 22, owner: 40003ad00, flag: INIT/-/-/0x00
      user lock: lock=4006d3908 mode=S
      user resource: user=400269ce8 uid=5 mode=S
      ----------------------------------------
      SO: 4006ad940, type: 22, owner: 40003ad00, flag: INIT/-/-/0x00
      user lock: lock=4006ad940 mode=S
      user resource: user=400269ce8 uid=5 mode=S
      ----------------------------------------
      SO: 400267b10, type: 23, owner: 40003ad00, flag: INIT/-/-/0x00
      LIBRARY OBJECT LOCK: lock=400267b10 handle=400702b58 mode=N
      call pin=400268b68 session pin=0
      user=40003ad00 session=40003ad00 count=1 flags=[00] savepoint=1
      LIBRARY OBJECT HANDLE: handle=400702b58
      name=SYS.IDGEN1$
      hash=617ab9e8 timestamp=09-17-1997 15:56:24
      namespace=TABL/PRCD/TYPE flags=TIM/SML/[02000000]
      kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
      lwt=400702b88[400702b88,400702b88]
ltm=400702b98[400702b98,400702b98]
      pwt=400702bb8[400702bb8,400702bb8]
ptm=400702c50[400702c50,400702c50]
      ref=400702b68[400702b68,400702b68]
        LIBRARY OBJECT: object=400702820
        type=SQNC flags=EXS/LOC[0005] pflags= [00] status=VALD load=0
        DATA BLOCKS:
        data#     heap  pointer status pins change
        ----- -------- -------- ------ ---- ------
            0 400702a90 400702910 I/-/A     0 NONE
    ----------------------------------------
    SO: 40005c3f0, type: 2, owner: 40002ccf0, flag: INIT/-/-/0x00
    (call) sess: cur 40003ad00, rec 40003b6e8, usr 40003ad00; depth: 0
      ----------------------------------------
      SO: 40007db78, type: 4, owner: 40005c3f0, flag: INIT/-/-/0x00
      (enqueue) CI-00000012-00000005     DID: 0001-0008-00000001
      lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      res:4000924f0, mode: X, prv: 400092500, sess: 40003ad00, proc: 40002ccf0


      ------> holding CI lock enqueue in 'X' mode. CI locks are held
        for temp segments cleanout.
      ----------------------------------------
      SO: 40024b2f8, type: 21, owner: 40005c3f0, flag: INIT/-/-/0x00
      row cache enqueue: count=1 session=40003ad00 object=400714088, mode=S
      row cache parent object: address=400714088 type=2(dc_segments)
      transaction=0 mode=S flags=0002
      status=VALID/-/-/-/-/-/-/-
      data=
      00000003 00000004 00000110 00000005 00000005 00000001 000000f9 00000005
      00000032 00000003 00000005 00000001 00000000 00000000
      ----------------------------------------
      SO: 40007db10, type: 4, owner: 40005c3f0, flag: INIT/-/-/0x00
      (enqueue) TS-00000003-01000110     DID: 0001-0008-00000001
      lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
      res:400092488, mode: X, prv: 400092498, sess: 40003ad00, proc: 40002ccf0
      ----------------------------------------
        hash=cb1510c7 timestamp=02-11-1999 17:29:40
        namespace=CRSR flags=RON/TIM/PN0/LRG/[10010001]
        kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
        lwt=400708538[400708538,400708538]
ltm=400708548[400708548,400708548]
        pwt=400708568[400708568,400708568]
```

```
ptm=400708600[400708600,400708600]
        ref=400708518[400708518,400708518]
          LIBRARY OBJECT: object=4007044c8
          type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
          CHILDREN: size=16
          child#    table reference   handle
          ------ -------- --------- --------
               0 4007043e0 400704720 400704030
          DATA BLOCKS:
          data#     heap  pointer status pins change
          ----- -------- -------- ------ ---- ------
              0 400708440 4007045b8 I/P/A     0 NONE
********************************************************************************
PROCESS 9:
  ----------------------------------------
  SO: 40002d0d0, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=9, calls cur/top: 0/40005cb60, flag: (0) -
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 43 0 4
              last post received-location: kslpsr
              last process to post me: 40002bd70 1 2
              last post sent: 0 0 15
              last post sent-location: ksasnd
              last process posted by me: 40002bd70 1 2
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ttyp5, ospid: 3709
    OSD pid info: Unix process pid: 3709, image: oraclenitin
    ----------------------------------------
    SO: 40005cb60, type: 2, owner: 40002d0d0, flag: INIT/-/-/0x00
    (call) sess: cur 0, rec 0, usr 0; depth: 0
PROCESS 10:
  ----------------------------------------
  SO: 40002d4b0, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=10, calls cur/top: 0/40005ccf8, flag: (0) -
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 43 0 4
              last post received-location: kslpsr
              last process to post me: 40002bd70 1 2
              last post sent: 0 0 15
              last post sent-location: ksasnd
              last process posted by me: 40002bd70 1 2
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ttyp5, ospid: 3698
    OSD pid info: Unix process pid: 3698, image: oraclenitin
    ----------------------------------------
    SO: 40005ccf8, type: 2, owner: 40002d4b0, flag: INIT/-/-/0x00
    (call) sess: cur 0, rec 0, usr 0; depth: 0

********************************************************************************

PROCESS 11:
  ----------------------------------------
  SO: 40002d890, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=11, calls cur/top: 40005cad8/40005cad8, flag: (2) SYSTEM
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 0 0 26
              last post received-location: ksfvfcl
              last process to post me: 40002d0d0 3 0
              last post sent: 0 0 24
              last post sent-location: ksfvrdp: imm op done
              last process posted by me: 40002d0d0 3 0
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ?, ospid: 32644
    OSD pid info: Unix process pid: 32644, image: ora_i103_nitin

```

---> This is the database writer i/o slave process, 1 is adapter (disk in this case) and 3 is the slave number. Something strange here as well. This process should have started after database open before any user process but for some reason it started late.

```

    ----------------------------------------
    SO: 40003cab8, type: 3, owner: 40002d890, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (51) USR/- BSY/-/-/-/-/-
              DID: 0000-0000-00000000, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    waiting for 'io done' seq=15574 wait_time=0    ----> waiting for i/o done
                  msg ptr=4006e0460, =0, =0
```

```
      ----------------------------------------
      SO: 4006e03c8, type: 8, owner: 40003cab8, flag: -/-/-/0x00
IO Slave State
--------------
slave num: 3, KSFV context: 40078f860, idle since: 1360a01f
Flags: (118) IDLE
    ----------------------------------------
    SO: 40005cad8, type: 2, owner: 40002d890, flag: INIT/-/-/0x00
    (call) sess: cur 40003cab8, rec 0, usr 40003cab8; depth: 0

********************************************************************************

PROCESS 13:
  ----------------------------------------
  SO: 40002e050, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=13, calls cur/top: 40005cfa0/40005cfa0, flag: (0) -
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 43 0 4
              last post received-location: kslpsr
              last post sent: 0 0 15
              last post sent-location: ksasnd
              last process posted by me: 40002bd70 1 2
    (latch info) wait_event=0 bits=0
    O/S info: user: nvengurl, term: ttyp1, ospid: 4436
    OSD pid info: Unix process pid: 4436, image: oraclenitin
    ----------------------------------------
    SO: 40003d4a0, type: 3, owner: 40002e050, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (100141) USR/- BSY/-/-/-/-/-
              DID: 0001-000D-00000001, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 0, prv: 0, user: 0/SYS
    O/S info: user: nvengurl, term: ttyp1, ospid: 5284, machine: charlie1.us.oracle.com
              program: <sqlplus>@charlie1.us.oracle.com (TNS V1-V3)
    application name: 01@ undo_byt.sql, hash value=679922013
    waiting for 'enqueue' seq=943 wait_time=0
                name|mode=43490006, id1=12, id2=5

------> note - this process is waiting for an enqueue
      ----------------------------------------
      SO: 40026c840, type: 24, owner: 40003d4a0, flag: INIT/-/-/0x00
      LIBRARY OBJECT PIN: pin=40026c840 handle=0 lock=40026dc18
      user=40003d4a0 session=40003d4a0 count=0 mask=0000 savepoint=170 flags=[00]
        LIBRARY OBJECT: object=40060e6b8
        type=PCKG flags=EXS/LOC[0005] pflags=NST [01] status=VALD load=0
        DEPENDENCIES: count=1 size=16
        DATA BLOCKS:
        data#     heap  pointer status pins change
        ----- -------- -------- ------ ---- ------
            0 40060e988 40060e850 I/-/A     0 NONE
            2 40060e7c8 40060db38 I/-/A     0 NONE
            4 40060e420 40066ceb0 I/-/A     0 NONE
            9 40060e378 400609a40 I/-/A     0 NONE
      ----------------------------------------
      SO: 4004a7308, type: 22, owner: 40003d4a0, flag: INIT/-/-/0x00
      user lock: lock=4004a7308 mode=S
      user resource: user=40026de88 uid=0 mode=S
      ----------------------------------------
      SO: 4004bba80, type: 22, owner: 40003d4a0, flag: INIT/-/-/0x00
      user lock: lock=4004bba80 mode=S
      user resource: user=40026de88 uid=0 mode=S
      ----------------------------------------
      SO: 40026d390, type: 23, owner: 40003d4a0, flag: INIT/-/-/0x00
      LIBRARY OBJECT LOCK: lock=40026d390 handle=400702b58 mode=N
      call pin=40026bfd0 session pin=0
      user=40003d4a0 session=40003d4a0 count=1 flags=[00] savepoint=1
      LIBRARY OBJECT HANDLE: handle=400702b58
      name=SYS.IDGEN1$
      hash=617ab9e8 timestamp=09-17-1997 15:56:24
      namespace=TABL/PRCD/TYPE flags=TIM/SML/[02000000]
      kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
      lwt=400702b88[400702b88,400702b88]
ltm=400702b98[400702b98,400702b98]
      pwt=400702bb8[400702bb8,400702bb8]
ptm=400702c50[400702c50,400702c50]
      ref=400702b68[400702b68,400702b68]
        LIBRARY OBJECT: object=400702820
        type=SQNC flags=EXS/LOC[0005] pflags= [00] status=VALD load=0
```

```
448.        DATA BLOCKS:
449.        data#     heap  pointer status pins change
450.        ----- -------- -------- ------ ---- ------
451.            0 400702a90 400702910 I/-/A     0 NONE
452.    ----------------------------------------
453.    SO: 40005cfa0, type: 2, owner: 40002e050, flag: INIT/-/-/0x00
454.    (call) sess: cur 40003d4a0, rec 40003de88, usr 40003d4a0; depth: 0
455.      ----------------------------------------
456.      SO: 40007df88, type: 4, owner: 40005cfa0, flag: INIT/-/-/0x00
457.      (enqueue) CI-00000012-00000005     DID: 0001-000D-00000001
458.      lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
459.      res:4000924f0, req: X, prv: 400092510, sess: 40003d4a0, proc: 40002e050
460.
461.  -----> requesting a CI lock in exclusive mode which is held by
462.         process 8 in 'X' mode, so this process is going to wait
463.
464.      ----------------------------------------
465.      SO: 40024b2a0, type: 21, owner: 40005cfa0, flag: INIT/-/-/0x00
466.      row cache enqueue: count=1 session=40003d4a0 object=4003d1540, mode=S
467.      row cache parent object: address=4003d1540 type=2(dc_segments)
468.      transaction=0 mode=S flags=0002
469.      status=VALID/-/-/-/-/-/-/-
470.      data=
471.      00000000 00000001 00001a15 00000000 00000003 00000001 000000f9 00000003
472.      00000032 00000003 00000003 00000001 00000000 00000000
473.      ----------------------------------------
474.      SO: 40007dcb0, type: 4, owner: 40005cfa0, flag: INIT/-/-/0x00
475.      (enqueue) TS-00000000-00401A15     DID: 0001-000D-00000001
476.      lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
477.      res:400092c40, mode: X, prv: 400092c50, sess: 40003d4a0, proc: 40002e050
478.      ----------------------------------------
479.      SO: 40026abf8, type: 13, owner: 40005cfa0, flag: INIT/-/-/0x00
480.      TEMPORARY TABLE LOCK: tsn=0000 dba=00000000 flg=00
481.      (enqueue) TS-00000000-00401A15     DID: 0001-000D-00000001
482.      lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
483.        call pin=0 session pin=40026be68
484.        user=40003d4a0 session=40003de88 count=1 flags=[00] savepoint=0
485.        LIBRARY OBJECT HANDLE: handle=40057aed8
486.        namespace=CRSR flags=RON/PN0/[10010000]
487.        kkkk-dddd-llll=0000-0041-0041 lock=N pin=0 latch=0
488.        lwt=40057af08[40057af08,40057af08] ltm=40057af18[40057af18,40057af18]
489.        pwt=40057af38[40057af38,40057af38] ptm=40057afd0[40057afd0,40057afd0]
490.        ref=40057aee8[40057b030,40057b030]
491.          LIBRARY OBJECT: object=40057ab40
492.          type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
493.          DEPENDENCIES: count=1 size=16
494.          AUTHORIZATIONS: count=1 size=16 minimum entrysize=16
495.          ACCESSES: count=1 size=16
496.          TRANSLATIONS: count=1 size=16
497.          DATA BLOCKS:
498.          data#     heap  pointer status pins change
499.          ----- -------- -------- ------ ---- ------
500.              0 40057ae10 40057acd8 I/P/A     0 NONE
501.              6 40057ac50 40057a640 I/-/A     0 NONE
502.        ----------------------------------------
503.        SO: 400266da8, type: 23, owner: 40003de88, flag: INIT/-/-/0x00
504.        LIBRARY OBJECT LOCK: lock=400266da8 handle=40057f620 mode=N
505.        call pin=400267bf0 session pin=0
506.        user=40003d4a0 session=40003de88 count=1 flags=[00] savepoint=0
507.        LIBRARY OBJECT HANDLE: handle=40057f620
508.        name=delete from col$ where obj#=:1
509.        hash=864ff862 timestamp=02-12-1999 16:51:11
510.        namespace=CRSR flags=RON/TIM/PN0/SML/[12010000]
511.        kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
512.        lwt=40057f650[40057f650,40057f650] ltm=40057f660[40057f660,40057f660]
513.        pwt=40057f680[40057f680,40057f680] ptm=40057f718[40057f718,40057f718]
514.        ref=40057f630[40057f630,40057f630]
515.          LIBRARY OBJECT: object=40057f2e8
516.          type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
517.          CHILDREN: size=16
518.          child#    table reference   handle
519.          ------ -------- --------- --------
520.               0 40057b288 40057b030 40057aed8
521.          DATA BLOCKS:
522.          data#     heap  pointer status pins change
523.          ----- -------- -------- ------ ---- ------
524.              0 40057f558 40057f3d8 I/P/A     0 NONE
525.
```

```
526.*******************************************************************************
527.
528.PROCESS 14:
529.  ----------------------------------------
530.  SO: 40002e430, type: 1, owner: 0, flag: INIT/-/-/0x00
531.  (process) Oracle pid=14, calls cur/top: 40005d138/40005d138, flag: (0) -
532.            int error: 0, call error: 0, sess error: 0, txn error 0
533.  (post info) last post received: 43 0 4
534.              last post received-location: kslpsr
535.              last process to post me: 40002bd70 1 2
536.              last post sent: 0 0 15
537.              last post sent-location: ksasnd
538.              last process posted by me: 40002bd70 1 2
539.    (latch info) wait_event=0 bits=0
540.    O/S info: user: nvengurl, term: ttyp1, ospid: 5464
541.    OSD pid info: Unix process pid: 5464, image: oraclenitin
542.    ----------------------------------------
543.    SO: 40003e870, type: 3, owner: 40002e430, flag: INIT/-/-/0x00
544.    (session) trans: 0, flag: (100141) USR/- BSY/-/-/-/-/-
545.              DID: 0001-000E-00000001, short-term DID: 0000-0000-00000000
546.              txn branch: 0
547.              oct: 0, prv: 0, user: 0/SYS
548.    O/S info: user: nvengurl, term: ttyp1, ospid: 5376, machine: charlie1.us.oracle.com
549.              program: <sqlplus>@charlie1.us.oracle.com (TNS V1-V3)
550.    application name: SQL*Plus, hash value=3669949024
551.    waiting for 'enqueue' seq=412 wait_time=0
552.                name|mode=43490006, id1=12, id2=5
553.
554. -----> note: waiting for enqueue
555.       ----------------------------------------
556.       SO: 40026cc00, type: 24, owner: 40003e870, flag: INIT/-/-/0x00
557.       LIBRARY OBJECT PIN: pin=40026cc00 handle=0 lock=400266ba0
558.       user=40003e870 session=40003e870 count=0 mask=0000 savepoint=1100 flags=[00]
559.       ----------------------------------------
560.       SO: 400266ba0, type: 23, owner: 40003e870, flag: INIT/-/-/0x00
561.       LIBRARY OBJECT LOCK: lock=400266ba0 handle=4003e5790 mode=N
562.       call pin=0 session pin=40026cc00
563.       user=40003e870 session=40003e870 count=1 flags=[00] savepoint=165
564.       LIBRARY OBJECT HANDLE: handle=4003e5790
565.       namespace=CRSR flags=RON/PN0/[10010000]
566.       kkkk-dddd-llll=0000-0041-0041 lock=N pin=0 latch=0
567.       lwt=4003e57c0[4003e57c0,4003e57c0]
568.ltm=4003e57d0[4003e57d0,4003e57d0]
569.       pwt=4003e57f0[4003e57f0,4003e57f0]
570.ptm=4003e5888[4003e5888,4003e5888]
571.       ref=4003e57a0[4003e6e98,4003e6e98]
572.         LIBRARY OBJECT: object=4003e5508
573.         type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
574.         DEPENDENCIES: count=1 size=16
575.         AUTHORIZATIONS: count=1 size=16 minimum entrysize=16
576.         ACCESSES: count=1 size=16
577.         TRANSLATIONS: count=1 size=16
578.         DATA BLOCKS:
579.         data#     heap  pointer status pins change
580.         ----- -------- -------- ------ ---- ------
581.             0 4003e6b48 4003e5088 I/P/A     0 NONE
582.             6 4003e5618 4003e4848 I/-/A     0 NONE
583.       ----------------------------------------
584.       SO: 40026d598, type: 23, owner: 40003e870, flag: INIT/-/-/0x00
585.       LIBRARY OBJECT LOCK: lock=40026d598 handle=4003f0cc8 mode=N
586.       call pin=40026c228 session pin=0
587.       user=40003e870 session=40003e870 count=1 flags=[00] savepoint=165
588.       LIBRARY OBJECT HANDLE: handle=4003f0cc8
589.       name=SELECT RAWTOHEX(KSMMMVAL)   FROM X$KSMMEM  WHERE INDX = :b1
590.       hash=83b422bd timestamp=03-06-1999 13:42:20
591.       namespace=CRSR flags=RON/TIM/PN0/SML/[12010000]
592.       kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
593.       lwt=4003f0cf8[4003f0cf8,4003f0cf8] ltm=4003f0d08[4003f0d08,4003f0d08]
594.       pwt=4003f0d28[4003f0d28,4003f0d28] ptm=4003f0dc0[4003f0dc0,4003f0dc0]
595.       ref=4003f0cd8[4003f8bd8,4003f8bd8]
596.         LIBRARY OBJECT: object=4003e6c40
597.         type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
598.         CHILDREN: size=16
599.         child#    table reference   handle
600.         ------ -------- --------- --------
601.              0 4003e5b40 4003e6e98 4003e5790
602.         DATA BLOCKS:
603.         data#     heap  pointer status pins change
```

```
604.        ----- -------- -------- ------ ---- ------
605.            0 4003e6f10 4003e6d30 I/P/A     0 NONE
606.       ----------------------------------------
607.       SO: 40026c408, type: 24, owner: 40003e870, flag: INIT/-/-/0x00
608.       LIBRARY OBJECT PIN: pin=40026c408 handle=0 lock=40026d870
609.       user=40003e870 session=40003e870 count=0 mask=0000 savepoint=1092 flags=[00]
610.       ----------------------------------------
611.       SO: 40026d870, type: 23, owner: 40003e870, flag: INIT/-/-/0x00
612.       LIBRARY OBJECT LOCK: lock=40026d870 handle=4003e7f50 mode=N
613.       call pin=0 session pin=40026c408
614.       user=40003e870 session=40003e870 count=1 flags=[00] savepoint=159
615.       LIBRARY OBJECT HANDLE: handle=4003e7f50
616.       namespace=CRSR flags=RON/PN0/[10010000]
617.       kkkk-dddd-llll=0000-0041-0041 lock=N pin=0 latch=0
618.       name=SYS.DBMS_OUTPUT
619.       hash=5e736e89 timestamp=09-17-1997 16:06:59
620.       namespace=TABL/PRCD/TYPE flags=TIM/SML/[02000000]
621.       kkkk-dddd-llll=0000-0215-0215 lock=N pin=0 latch=0
622.       lwt=40060ea80[40060ea80,40060ea80] ltm=40060ea90[40060ea90,40060ea90]
623.       pwt=40060eab0[40060eab0,40060eab0] ptm=40060eb48[40060eb48,40060eb48]
624.       ref=40060ea60[400612498,40061e558]
625.         LIBRARY OBJECT: object=40060e6b8
626.         type=PCKG flags=EXS/LOC[0005] pflags=NST [01] status=VALD load=0
627.         DEPENDENCIES: count=1 size=16
628.         DATA BLOCKS:
629.         data#     heap  pointer status pins change
630.         ----- -------- -------- ------ ---- ------
631.             0 40060e988 40060e850 I/-/A     0 NONE
632.             2 40060e7c8 40060db38 I/-/A     0 NONE
633.             4 40060e420 40066ceb0 I/-/A     0 NONE
634.             9 40060e378 400609a40 I/-/A     0 NONE
635.       ----------------------------------------
636.       SO: 40037bfd8, type: 22, owner: 40003e870, flag: INIT/-/-/0x00
637.       user lock: lock=40037bfd8 mode=S
638.       user resource: user=40026de88 uid=0 mode=S
639.       ----------------------------------------
640.       SO: 400361f60, type: 22, owner: 40003e870, flag: INIT/-/-/0x00
641.       user lock: lock=400361f60 mode=S
642.       user resource: user=40026de88 uid=0 mode=S
643.       ----------------------------------------
644.       SO: 40026de20, type: 23, owner: 40003e870, flag: INIT/-/-/0x00
645.       LIBRARY OBJECT LOCK: lock=40026de20 handle=400702b58 mode=N
646.       call pin=40026ca98 session pin=0
647.       user=40003e870 session=40003e870 count=1 flags=[00] savepoint=1
648.       LIBRARY OBJECT HANDLE: handle=400702b58
649.       name=SYS.IDGEN1$
650.       hash=617ab9e8 timestamp=09-17-1997 15:56:24
651.       namespace=TABL/PRCD/TYPE flags=TIM/SML/[02000000]
652.       kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
653.       lwt=400702b88[400702b88,400702b88]
654.ltm=400702b98[400702b98,400702b98]
655.       pwt=400702bb8[400702bb8,400702bb8]
656.ptm=400702c50[400702c50,400702c50]
657.       ref=400702b68[400702b68,400702b68]
658.         LIBRARY OBJECT: object=400702820
659.         type=SQNC flags=EXS/LOC[0005] pflags= [00] status=VALD load=0
660.         DATA BLOCKS:
661.         data#     heap  pointer status pins change
662.         ----- -------- -------- ------ ---- ------
663.             0 400702a90 400702910 I/-/A     0 NONE
664.    ----------------------------------------
665.    SO: 40005d138, type: 2, owner: 40002e430, flag: INIT/-/-/0x00
666.    (call) sess: cur 40003e870, rec 40003f258, usr 40003e870; depth: 0
667.       ----------------------------------------
668.       SO: 40007e058, type: 4, owner: 40005d138, flag: INIT/-/-/0x00
669.       (enqueue) CI-00000012-00000005     DID: 0001-000E-00000001
670.       lv: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
671.       res:4000924f0, req: X, prv: 40007dfb8, sess: 40003e870, proc: 40002e430
672.       ----------------------------------------
673.       SO: 40024b350, type: 21, owner: 40005d138, flag: INIT/-/-/0x00
674.       row cache enqueue: count=1 session=40003e870 object=40039ddf0, mode=S
675.       row cache parent object: address=40039ddf0 type=2(dc_segments)
676.       transaction=0 mode=S flags=0002
677.       status=VALID/-/-/-/-/-/-/-
678.       data=
679.       00000004 00000007 00000a0e 00000013 00000005 00000001 000000f9 00000005
680.       00000032 00000003 00000005 00000001 00000000 00000000
681.       ----------------------------------------
682.
```

```
        SO: 40026edf8, type: 24, owner: 40003f258, flag: INIT/-/-/0x00
        LIBRARY OBJECT PIN: pin=40026edf8 handle=0 lock=40026d188
        user=40003e870 session=40003f258 count=0 mask=0000 savepoint=0 flags=[00]
        ----------------------------------------
        SO: 40026d188, type: 23, owner: 40003f258, flag: INIT/-/-/0x00
        LIBRARY OBJECT LOCK: lock=40026d188 handle=40057aed8 mode=N
        call pin=0 session pin=40026edf8
        user=40003e870 session=40003f258 count=1 flags=[00] savepoint=0
        LIBRARY OBJECT HANDLE: handle=40057aed8
        namespace=CRSR flags=RON/PN0/[10010000]
        kkkk-dddd-llll=0000-0041-0041 lock=N pin=0 latch=0
        lwt=40057af08[40057af08,40057af08]
ltm=40057af18[40057af18,40057af18]
        pwt=40057af38[40057af38,40057af38]
ptm=40057afd0[40057afd0,40057afd0]
        ref=40057aee8[40057b030,40057b030]
          LIBRARY OBJECT: object=40057ab40
          type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
          DEPENDENCIES: count=1 size=16
          AUTHORIZATIONS: count=1 size=16 minimum entrysize=16
          ACCESSES: count=1 size=16
          TRANSLATIONS: count=1 size=16
          DATA BLOCKS:
          data#     heap  pointer status pins change
          ----- -------- -------- ------ ---- ------
              0 40057ae10 40057acd8 I/P/A     0 NONE
              6 40057ac50 40057a640 I/-/A     0 NONE
        ----------------------------------------
        SO: 40026d050, type: 23, owner: 40003f258, flag: INIT/-/-/0x00
        LIBRARY OBJECT LOCK: lock=40026d050 handle=40057f620 mode=N
        call pin=40026ec90 session pin=0
        user=40003e870 session=40003f258 count=1 flags=[00] savepoint=0
        LIBRARY OBJECT HANDLE: handle=40057f620
        name=delete from col$ where obj#=:1
        hash=864ff862 timestamp=02-12-1999 16:51:11
        namespace=CRSR flags=RON/TIM/PN0/SML/[12010000]
        kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
        lwt=40057f650[40057f650,40057f650]
ltm=40057f660[40057f660,40057f660]
        pwt=40057f680[40057f680,40057f680]
ptm=40057f718[40057f718,40057f718]
        ref=40057f630[40057f630,40057f630]
          LIBRARY OBJECT: object=40057f2e8
          type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
          CHILDREN: size=16
          child#    table reference   handle
          ------ -------- --------- --------
               0 40057b288 40057b030 40057aed8
          DATA BLOCKS:
          data#     heap  pointer status pins change
          ----- -------- -------- ------ ---- ------
              0 40057f558 40057f3d8 I/P/A     0 NONE

********************************************************************************

PROCESS 15:
  ----------------------------------------
  SO: 40002e810, type: 1, owner: 0, flag: INIT/-/-/0x00
  (process) Oracle pid=15, calls cur/top: 40005d2d0/40005d2d0, flag: (0) -
            int error: 0, call error: 0, sess error: 0, txn error 0
  (post info) last post received: 0 0 0
              last post received-location: No post
              last process to post me: none
              last post sent: 0 0 0
              last post sent-location: No post
              last process posted by me: none
    (latch info) wait_event=27 bits=0
    O/S info: user: nvengurl, term: ttyp2, ospid: 5342
    OSD pid info: Unix process pid: 5342, image: oraclenitin
    ----------------------------------------
    SO: 40003fc40, type: 3, owner: 40002e810, flag: INIT/-/-/0x00
    (session) trans: 0, flag: (41) USR/- BSY/-/-/-/-/-
              DID: 0001-000F-00000001, short-term DID: 0000-0000-00000000
              txn branch: 0
              oct: 49, prv: 0, user: 0/SYS
    O/S info: user: nvengurl, term: ttyp2, ospid: 5250, machine: charlie1.us.oracle.com
              program: <svrmgrl>@charlie1.us.oracle.com (TNS V1-V3)
    waiting for 'checkpoint completed' seq=193 wait_time=0
```

```
761.
762.-------> process is waiting for checkpoint to complete
763.
764.                =0, =0, =0
765.       ----------------------------------------
766.       SO: 40026cb88, type: 24, owner: 40003fc40, flag: INIT/-/-/0x00
767.       LIBRARY OBJECT PIN: pin=40026cb88 handle=4003544a8 mode=S lock=400266b38
768.       user=40003fc40 session=40003fc40 count=1 mask=0041 savepoint=9 flags=[00]
769.       ----------------------------------------
770.       SO: 400266b38, type: 23, owner: 40003fc40, flag: INIT/-/-/0x00
771.       LIBRARY OBJECT LOCK: lock=400266b38 handle=4003544a8 mode=N
772.       call pin=0 session pin=40026cb88
773.       user=40003fc40 session=40003fc40 count=1 flags=PNS/[08] savepoint=7
774.       LIBRARY OBJECT HANDLE: handle=4003544a8
775.       namespace=CRSR flags=RON/PN0/[10010000]
776.       kkkk-dddd-llll=0000-0041-0041 lock=N pin=S latch=0
777.       lwt=4003544d8[4003544d8,4003544d8] ltm=4003544e8[4003544e8,4003544e8]
778.       pwt=400354508[400354508,400354508] ptm=4003545a0[4003545a0,4003545a0]
779.       ref=4003544b8[400392460,400392460]
780.         LIBRARY OBJECT: object=400354108
781.         type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
782.         DATA BLOCKS:
783.         data#     heap  pointer status pins change
784.         ----- -------- -------- ------ ---- ------
785.             0 4003543e0 4003542a0 I/P/A     0 NONE
786.             6 400354218 400396dd8 I/P/A     1 NONE
787.       ----------------------------------------
788.       SO: 40026d1f0, type: 23, owner: 40003fc40, flag: INIT/-/-/0x00
789.       LIBRARY OBJECT LOCK: lock=40026d1f0 handle=400392608 mode=N
790.       call pin=40026ee70 session pin=0
791.       user=40003fc40 session=40003fc40 count=1 flags=[00] savepoint=7
792.       LIBRARY OBJECT HANDLE: handle=400392608
793.       name=alter system checkpoint local  ----> This is the command issued
794.                                           ----> points to dbwr to complete
795.                                    ----> the write
796.       hash=a8feff29 timestamp=03-06-1999 16:04:44
797.       namespace=CRSR flags=RON/TIM/PN0/SML/[12010000]
798.       kkkk-dddd-llll=0000-0001-0001 lock=N pin=0 latch=0
799.       lwt=400392638[400392638,400392638] ltm=400392648[400392648,400392648]
800.       pwt=400392668[400392668,400392668] ptm=400392700[400392700,400392700]
801.       ref=400392618[400392618,400392618]
802.         LIBRARY OBJECT: object=40039a4a8
803.         type=CRSR flags=EXS[0001] pflags= [00] status=VALD load=0
804.         CHILDREN: size=16
805.         child#    table reference   handle
806.         ------ -------- --------- --------
807.              0 400392550 400392460 4003544a8
808.         DATA BLOCKS:
809.         data#     heap  pointer status pins change
810.         ----- -------- -------- ------ ---- ------
811.             0 400399398 40039a598 I/P/A     0 NONE
812.       ----------------------------------------
813.       SO: 40026d0b8, type: 23, owner: 40003fc40, flag: INIT/-/-/0x00
814.       LIBRARY OBJECT LOCK: lock=40026d0b8 handle=400702b58 mode=N
815.       call pin=40026ed08 session pin=0
816.       user=40003fc40 session=40003fc40 count=1 flags=[00] savepoint=1
817.       LIBRARY OBJECT HANDLE: handle=400702b58
818.       name=SYS.IDGEN1$
819.       hash=617ab9e8 timestamp=09-17-1997 15:56:24
```

Conclusion:

Process 3 (DBW0)	is waiting for I/O done
Process 8 (user)	holding a CI lock in 'X' mode
Process 9 (I/O slave)	waiting for i/o done
Process 13 (user)	waiting for enqueue in 'X' mode
Process 14 (user)	waiting for enqueue in 'X' mode
Process 15 (user)	waiting for checkpoint

Fortunately, in this scenario we have more information in the alert.log file and a generated DBW0 trace file that point to an async I/O problem. So investigate in that direction.

Case Studies 8: Rollback Segment Corruption Recovery

In the following case studies we explore rollback segment recovery in three scenarios:
A. Object Block Corruption
B. Rollback Segment Body Corruption
C. Rollback Segment Header Corruption

The three case studies were performed on Oracle Release 8.0.5. In tests performed on Oracle 7.3.3, diagnostics, identification and recovery were found to be the same.

The cases were re-created using the following procedure:
1. Start a transaction (inserting a row into a table).
2. In a separate session, identify the block to be corrupted.
3. Shutdown Abort.
4. Corrupt the block (zero block type using orapatch).
5. Start up and perform recovery.

	Object Block Corruption	**RBS Body Corruption**	**RBS Header Corruption**
Identification	1. The database opens. 2. SMON generates ORA-1578 in alert log. 3. Corrupt block belongs to table, cluster, or index.	1. The database opens. 2. SMON generates ORA-1578 in alert log. 3. Corrupt block belongs to rollback segment (xxx), but not header.	1. The database does not open. 2. ORA-1578 is reported in the alert log. 3. After setting event 10015, a trace file is generated on startup which ends abruptly with a message of the form "Recovering rollback segment xxx". xxx is the corrupt segment.
Supported recovery	1. Complete media recovery. 2. Drop object after salvaging data.	1. Complete media recovery.	1. Complete media recovery.
Unsupported recovery	Not required.	1. Shut down. 2. Edit the init.ora: • Remove xxx from rollback_segments. • Add xxx to _corrupted_ rollback_segments 3. Startup 4. Export, database rebuild, Import	1. Edit the init.ora: • Remove xxx from rollback_segments. • Add xxx to _corrupted_ rollback_segments 2. Start up 3. Export, database rebuild, Import

A. Object Block Corruption

A table, index, or cluster block is corrupted. A transaction which had updated the corrupted block is being rolled back, and in the process tries to read and update the block. From 7.3 onwards, if the transaction is being aborted as part of crash recovery, the database will open. Prior to 7.3, *_offline_rollback_segments* may have been required to open the database.

This is not rollback segment corruption at all; however, if a transaction is active in the block then rollback segments are involved.

Diagnostics
The alert.log file will contain the following messages on startup:

```
SMON: enabling cache recovery
SMON: enabling tx recovery
Tue Nov  2 13:20:19 1999
Errors in file /bugmnt1/tar10159734.6/dump/smon_20562.trc:
ORA-01578: ORACLE data block corrupted (file # 5, block # 3)
ORA-01110: data file 5: '/bugmnt1/tar10159734.6/dbs/users01.dbf'
Tue Nov  2 13:20:19 1999
Completed: alter database open
```

Note that the file# reported in the ORA-1578 error message is the relative file number. The file number reported in the ORA-1110 message is the absolute file number.

The following query was executed (substituting the absolute file number and block number) to identify the corrupt object:

```
SQL> set charwidth 10
SQL> select * from dba_extents
 2   where  file_id=5
 3   and    3 between block_id and block_id + blocks - 1;

OWNER  SEGMENT PARTITION SEGMENT TABLESPACE EXTENT FILE BLOCK BYTES BLOCKS RELATIVE
       _NAME   _NAME     _TYPE   _NAME          _ID  _ID   _ID                 _FNO
------ ------- --------- ------- ---------- ------ ---- ----- ----- ------ --------
ASHISH TEST              TABLE   USERS           0    5     2 10240      5        5
```

Identification
1. SMON is reporting a corrupt block, and
2. The above query indicates that the corruption lies in a user object.

Recovery
The preferred recovery method is complete media recovery.

The corruption will prevent the selection of data from the object. If media recovery is not possible, the following can be used to resolve the corruption. However, the data in the corrupt block may be lost.

1. Select data from the corrupt object using traditional block corruption recovery techniques (rowid range scans, event 10231, etc.).
2. Drop the corrupt object.

Summary
The active transaction cannot be aborted because the object block cannot be updated (corrupt). Selects will also fail on this block.

Media recovery is recommended. If this is not possible, data can be salvaged using traditional corruption recovery techniques. The object can then be dropped to resolve the problem. If media recovery is not possible, the data in the corrupt block will usually be lost.

B. Rollback Segment Body Corruption

A rollback segment body block (not the header) containing undo for an active transaction has become corrupted. In Oracle8, the database will open, but SMON will fail repeatedly when trying to abort the transaction.

Diagnostics
The alert.log file will have the following message:

```
ORACLE Instance TRBS (pid = 6) - Error 1578 encountered while recovering transaction
(4, 1).
Errors in file /bugmnt1/tar10159734.6/dump/smon_9513.trc:
ORA-01578: ORACLE data block corrupted (file # 32, block # 804)
ORA-01110: data file 2: '/bugmnt1/tar10159734.6/dbs/rbs01.dbf'

ORACLE Instance TRBS (pid = 6) - Error 1578 encountered while recovering transaction
(4, 1).
Errors in file /bugmnt1/tar10159734.6/dump/smon_9513.trc:
ORA-01578: ORACLE data block corrupted (file # 32, block # 804)
ORA-01110: data file 2: '/bugmnt1/tar10159734.6/dbs/rbs01.dbf'
```

Note that the file# reported in the ORA-1578 error message is the relative file number. The file number reported in the ORA-1110 message is the absolute file number.

Execute the following query (substituting the absolute file number and block number) to identify the corrupted object:

```
SQL> select * from dba_extents
 2   where file_id = 2 and
 3   804 between block_id and block_id + blocks - 1;

OWNER      SEGMENT_NA PARTITION_ SEGMENT_TY TABLESPACE EXTENT_ID FILE_ID
BLOCK_ID   BYTES      BLOCKS     RELATIVE_F
---------- ---------- ---------- ---------- ---------- --------- -------
---------- ---------- ---------- ----------
SYS        R01                   ROLLBACK   RBS                6       2
       782     133120         65         32
```

Identification
1. SMON reports ORA-1578 in alert log.
2. The corrupted block is in the body of a rollback segment.

Recovery
The preferred recovery method is complete media recovery. This is the only way to guarantee database consistency.

The corruption will probably prevent a successful export of the database. If media recovery is not possible, the following procedure will allow the corrupted rollback segment to be dropped. This will normally allow an export to complete successfully. Note, however, that this process will effectively "force commit" all active transactions in the corrupted rollback segment, so logical corruption is likely. This mode of recovery is unsupported.

1. Shut down.
2. Edit the init.ora
   - Remove R01 from *rollback_segments*.
   - Add *_corrupted_rollback_segments = R01*
3. Startup
4. Export/Database Rebuild/Import

Summary and Impact
The active transaction cannot be aborted because the undo block cannot be read (corrupt). Consistent reads which require this block will also fail.

Media Recovery is recommended. If this is not possible, the rollback segment can be corrupted allowing export/rebuild/import to be performed. However; this is unsupported and may result in logical corruption.

C. Rollback Segment Header Corruption

The header of an online rollback segment is corrupted. With all versions, this will prevent the database from opening.

Diagnostics
During startup you will get the following:

```
ORA-01578: ORACLE data block corrupted (file # 2, block # 132)
ORA-01110: data file 2: '/bugmnt1/tar10159734.6/dbs/rbs01.dbf
```

Set event 10015. This will dump the name, segment header and transaction table of every rollback segment during database open. The trace will be written to the trace file of the server manager shadow.

Identification
1. ORA-1578 reported in alert log and the database does not start, and
2. After setting event 10015, the trace file ends abruptly with a message of the form "Recovering rollback segment xxx" on startup; xxx is the corrupt segment.

Recovery
The preferred recovery method is media recovery, as this is the only way to guarantee database consistency.

If media recovery is not possible, the first priority is to open the database. The following procedure will allow the database to be opened and normally allow an export to complete successfully. Note, however, that this process will effectively "force commit" all active transactions in the corrupted rollback segment, so logical corruption is likely, possibly in the data dictionary.

Summary
The database cannot be started because the rollback segment header cannot be read during open (corrupt).

Media recovery is recommended. If this is not possible, the rollback segment can be corrupted allowing the database to be opened and export/rebuild/import to be performed. However; this is unsupported and may result in logical corruption.

Case Studies 9: Data Salvage Using DUL

In the following case studies we explore using DUL in the most common two scenarios:
A. Against a set of database files including the system tablespace files
B. Against a set of database files, but not including the system tablespace files

The case studies were performed on data files from an Oracle release 7.3.3 database running on Sun Solaris 2.5. using, for example, DUL version 3 (in the practices you will use version 8). Procedures are very similar when used against data files from other supported releases.

The cases were re-created using the following procedure:
1. Shut down the database abnormally (abort)
2. Remove access to the online redo logs so that the database cannot be rolled forward

Although it is quite possible that the database might be opened by using undocumented events, the above procedure will produce a set of data files which will suffice for the purposes of the case studies.

A. Against a set of database files including the system tablespace files

All attempts at opening the database have failed. The following files are available:
- User data files
- System tablespace data files
- A control file reflecting the current database configuration

Configuring DUL
First configure the init.dul file. The following is taken from the example in the *User's and Configuration Guide* (from the DUL Web page) and modified where appropriate for this database and platform.

```
# sample init.dul configuration parameters

# the cache must hold all entries from the dollar tables.
dc_columns = 200000
dc_tables = 10000
dc_objects = 10000
dc_users = 40

# OS specific parameters
# echo dul | od -x gives the following output:
# 00000000 6475 6c0a i.e. the port is big endian (not byte-swapped)
big_endian_flag = true
# SQL> select dump(chartorowid('0.0.1')) from dual gives the following:
# Typ=69 Len=6: 4,0,0,0,0,0
dba_file_bits = 6
# Not VAX/VMS -> align_filler=1
align_filler = 1
# Unix platform -> 1 extra leading block
db_leading_offset = 1

# database parameters
db_block_size = 2048

# loader format definitions
ldr_enclose_char = "
ldr_phys_rec_size = 81
```

Next configure control.dul. The control file is available, and so can be used as the basis for creating control.dul.

```
# SVRMGR> select * from v$dbfile gives the following output:
 1 /oracle/app/oracle/product/7.3.3/dbs/sysDUL.dbf
 4 /oracle/app/oracle/product/7.3.3/dbs/dataDUL.dbf
 5 /oracle/app/oracle/product/7.3.3/dbs/indexesDUL.dbf
# Note, I have removed the RBS and TEMP data files (file# 2 & 3)
# as they only will contain transient data
```

Unload the dictionary tables

```
$ dul dictv7.ddl

UnLoader: Release 2.2.0.2 - Very Restricted on Tue Sep  2 17:14:47 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.

. unloading table                      OBJ$     893 rows unloaded
. unloading table                      TAB$      85 rows unloaded
. unloading table                      COL$    3631 rows unloaded
. unloading table                     USER$      12 rows unloaded
```

Note that the following eight files are created in the default directory, representing SQL*Loader control and data files for USER$, OBJ$, TAB$, COL$: USER.ctl, USER.dat, OBJ.ctl, OBJ.dat, TAB.ctl, TAB.dat, COL.ctl, COL.dat

Unloading an individual table

```
$ dul

UnLoader: Release 2.2.0.2 - Very Restricted on Tue Sep  2 17:15:09 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.

Loaded 893 objects
Loaded 85 tables
Loaded 3631 columns
Loaded 12 users
DUL> unload table jbarlow.emp;
About to unload specified tables ...
. unloading table                       EMP      14 rows unloaded
DUL> quit
```

Note that the above command creates a SQL*Loader control and data file for the unloaded object in the default directory:

```
$ ls *EMP*
JBARLOW_EMP.ctl JBARLOW_EMP.dat
```

Unloading a user

```
$ dul

UnLoader: Release 2.2.0.2 - Very Restricted on Tue Sep  2 17:15:45 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.

Loaded 893 objects
Loaded 85 tables
Loaded 3631 columns
Loaded 12 users
DUL> unload user jbarlow;
About to unload JBARLOW's tables ...

DUL: Warning: Recreating file "JBARLOW_EMP.dat"
. unloading table                       EMP      14 rows unloaded

DUL: Warning: Recreating file "JBARLOW_EMP.ctl"

. unloading table                      DEPT       4 rows unloaded
. unloading table                     BONUS       0 rows unloaded
. unloading table                   SALGRADE       5 rows unloaded
. unloading table                     DUMMY       1 rows unloaded
DUL> quit
```

Note that DUL overwrites existing identically named files with a warning.

Unloading the complete database

```
$ dul
```

```
UnLoader: Release 2.2.0.2 - Very Restricted on Tue Sep  2 17:16:11 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.

Loaded 893 objects
Loaded 85 tables
Loaded 3631 columns
Loaded 12 users
DUL> unload database;
About to unload PUBLIC's tables ...
About to unload CONNECT's tables ...
About to unload RESOURCE's tables ...
About to unload DBA's tables ...
About to unload SYSTEM's tables ...
. unloading table           PRODUCT_PROFILE       0 rows unloaded
. unloading table              USER_PROFILE       0 rows unloaded
. unloading table                 DEF$_CALL       0 rows unloaded
. unloading table                DEF$_ERROR       0 rows unloaded
. unloading table          DEF$_DESTINATION       0 rows unloaded
. unloading table             DEF$_CALLDEST       0 rows unloaded
. unloading table          DEF$_DEFAULTDEST       0 rows unloaded
. unloading table       DBMS_LOCK_ALLOCATED       0 rows unloaded
. unloading table           DBMS_ALERT_INFO       0 rows unloaded
About to unload EXP_FULL_DATABASE's tables ...
About to unload IMP_FULL_DATABASE's tables ...
About to unload SNMPAGENT's tables ...
About to unload DBSNMP's tables ...
About to unload JBARLOW's tables ...

DUL: Warning: Recreating file "JBARLOW_EMP.dat"
. unloading table                        EMP      14 rows unloaded

DUL: Warning: Recreating file "JBARLOW_EMP.ctl"

DUL: Warning: Recreating file "JBARLOW_DEPT.dat"
. unloading table                       DEPT       4 rows unloaded

DUL: Warning: Recreating file "JBARLOW_DEPT.ctl"

DUL: Warning: Recreating file "JBARLOW_BONUS.dat"
. unloading table                      BONUS       0 rows unloaded

DUL: Warning: Recreating file "JBARLOW_BONUS.ctl"

DUL: Warning: Recreating file "JBARLOW_SALGRADE.dat"
. unloading table                    SALGRADE       5 rows unloaded

DUL: Warning: Recreating file "JBARLOW_SALGRADE.ctl"

DUL: Warning: Recreating file "JBARLOW_DUMMY.dat"
. unloading table                      DUMMY       1 rows unloaded

DUL: Warning: Recreating file "JBARLOW_DUMMY.ctl"
About to unload _NEXT_USER's tables ...
DUL> quit
```

B. Against a set of database files, but not including the system tablespace files

All attempts at opening the database have failed. Only the user datafiles are available.

Configuring DUL
Init.dul is identical to the one for the first case. Control.dul differs only in there is no system tablespace datafile. Furthermore because there is no controlfile, there is no automated way of identifying the datafiles. An old control file would help here, but it would have to match the physical structure of the database. Otherwise, it is possible to identify the file number by dumping the first Oracle header block in the file.

So for this case control.dul looks like this:

```
  4 /oracle/app/oracle/product/7.3.3/dbs/dataDUL.dbf
  5 /oracle/app/oracle/product/7.3.3/dbs/indexesDUL.dbf
```

Scanning the database for segment headers and extents

```
$ dul

UnLoader: Release 2.2.0.2 - Very Restricted on Tue Sep  2 17:49:47 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.

DUL> scan database;
data file 4 250 blocks scanned
data file 5 512 blocks scanned
DUL> quit
```

Scanning the found tables for column statistics

```
$ dul

DUL> scan tables;
Scanning tables with segment header

Oid 983 fno 4 bno 2 table number 0

UNLOAD TABLE T_O983 ( C1 NUMBER, C2 UNKNOWN, C3 UNKNOWN, C4 NUMBER, C5 DATE
        , C6 NUMBER, C7 NUMBER, C8 NUMBER )
    STORAGE ( TABNO 0 EXTENTS( FILE 4 BLOCK 2));
 Colno   Seen    MaxIntSz   Null%    C75%    C100     Num%    NiNu%    Dat%    Rid%
     1     14           3      0%      0%      0%     100%     100%      0%      0%
     2     14           6      0%    100%    100%     100%       0%      0%     21%
     3     14           9      0%    100%    100%     100%       0%      0%      0%
     4     14           3      7%      0%      0%     100%     100%      0%      0%
     5     14           7      0%      0%      0%       0%       0%    100%      0%
     6     14           3      0%      0%      0%     100%     100%      0%      0%
     7     14           2     71%      0%      0%     100%     100%      0%      0%
     8     14           2      0%      0%      0%     100%     100%      0%      0%
"7369" "SMITH" "CLERK" "7902" "17-DEC-1980 AD 00:00:00" "800" "" "20"
"7499" "ALLEN" "SALESMAN" "7698" "20-FEB-1981 AD 00:00:00" "1600" "300" "30"
"7521" "WARD" "SALESMAN" "7698" "22-FEB-1981 AD 00:00:00" "1250" "500" "30"
"7566" "JONES" "MANAGER" "7839" "02-APR-1981 AD 00:00:00" "2975" "" "20"
"7654" "MARTIN" "SALESMAN" "7698" "28-SEP-1981 AD 00:00:00" "1250" "1400" "30"

Oid 984 fno 4 bno 7 table number 0

UNLOAD TABLE T_O984 ( C1 NUMBER, C2 UNKNOWN, C3 UNKNOWN )
    STORAGE ( TABNO 0 EXTENTS( FILE 4 BLOCK 7));
 Colno   Seen    MaxIntSz   Null%    C75%    C100     Num%    NiNu%    Dat%    Rid%
     1      4           2      0%      0%      0%     100%     100%      0%      0%
     2      4          10      0%    100%    100%     100%       0%      0%      0%
     3      4           8      0%    100%    100%     100%       0%      0%     50%
"10" "ACCOUNTING" "NEW YORK"
"20" "RESEARCH" "DALLAS"
"30" "SALES" "CHICAGO"
"40" "OPERATIONS" "BOSTON"

Oid 986 fno 4 bno 17 table number 0
```

```
UNLOAD TABLE T_O986 ( C1 NUMBER, C2 NUMBER, C3 NUMBER )
    STORAGE ( TABNO 0 EXTENTS( FILE 4 BLOCK 17));
 Colno   Seen    MaxIntSz   Null%    C75%    C100     Num%    NiNu%    Dat%    Rid%
     1      5           2      0%      0%      0%     100%     100%      0%      0%
     2      5           3      0%      0%      0%     100%     100%      0%      0%
     3      5           3      0%      0%      0%     100%     100%      0%      0%
"1" "700" "1200"
"2" "1201" "1400"
"3" "1401" "2000"
"4" "2001" "3000"
"5" "3001" "9999"

Oid 987 fno 4 bno 22 table number 0

UNLOAD TABLE T_O987 ( C1 NUMBER )
    STORAGE ( TABNO 0 EXTENTS( FILE 4 BLOCK 22));
 Colno   Seen    MaxIntSz   Null%    C75%    C100     Num%    NiNu%    Dat%    Rid%
     1      1           1      0%      0%      0%     100%     100%      0%      0%
"0"
DUL> quit
```

Meanings of column statistics

Column Name	Description
Colno	Column number
Seen	How often the column is seen in a data block
MaxIntSz	The maximum internal column length
Null%	How often the column is NULL
C75%	How often the column consists of at least 75% printable ASCII
C100	How often the column consists of 100% printable ASCII
Num%	How often the column is a valid Oracle number
NiNu%	How often the column is a nice number (not many leading or trailing zeros)
Dat%	How often the column is a valid date
Rid%	How often the column is a possible valid rowid

It is easier to run this in the background and redirect the output to a file to be viewed later.

```
$ echo scan tables \; | dul > scan.out&
```

Identifying and unloading the scanned tables
From the above column statistics it is possible to identify object T_O983 (object id 983) is the EMP table.

```
$ dul

UnLoader: Release 2.2.0.2 - Very Restricted on Tue Sep  2 17:51:56 1997
Copyright (c) 1994/95 Oracle Corporation, The Netherlands. All rights reserved.

Loaded 5 segments
Loaded 4 extents
Extent map sorted
DUL> UNLOAD TABLE emp ( C1 NUMBER, C2 UNKNOWN, C3 UNKNOWN, C4 NUMBER, C5 DATE, C6
NUMBER, C7 NUMBER, C8 NUMBER ) STORAGE ( TABNO 0 EXTENTS( FILE 4 BLOCK 2));
. unloading table                       EMP      14 rows unloaded

DUL> quit
```

Note, the table number will be non-zero if the table to be unloaded is clustered.

B
Practices

Practice 2: Source Code, the Ultimate Oracle

Mainly working with the Oracle 8.1.5 source code as a reference, answer the following questions:

1. What functions return the error ORA-1245?

2. Name a function that returns the error ORA-1578. The file number is also logged with this error. Is this the relative or absolute file number?

3. What functions return the error ORA-600 [12025] in Oracle 8.0.5? Explain the conditions that would cause this error to be logged.

4. What function returns the ORA-600 [12025] in 8.1.5?

5. What functions return the error ORA-600 [kfhcfh1_01] in Oracle 8.0.5? What other information is returned with this error?

6. In which file is the *_corrupted_rollback_segments* parameter defined for Oracle 8.1.5? What is the internal identifier of this parameter? Where is this parameter referenced in the code?

7. On which fixed (X$) tables is the *v$process* view ultimately defined in Oracle 8.1.5?
   Which internal structures is/are used to derive the *serial#* column?

8. In which file is the *redo allocation* latch defined in Oracle 8.1.5?
   What is its internal identifier and where is it used?

9. Which function controls the processing of the *alter database* command in Oracle 8.1.5?
   How is *open resetlogs* handled?

Practice 3: Using Diagnostic Events

1. Log on to the database as user SCOTT and do a process-state dump.

2. Log on as user SCOTT in one session, start SQL*Plus in another session, and do a system state dump from the SCOTT session using the ORADEBUG utility.

3. Set the event with the alter session command so that an error stack will be produced whenever the error "table or view does not exist" is reported. Verify that the trace file is created by running the following query:
   ```
   SQL> select * from a;
   ```

4. Activate the event from another session using the dbms_system package.

5. Repeat step 4 using the ORADEBUG utility.

6. Repeat step 4 by editing the parameter file.

7. Create a small table EMP, do some INSERTS and then do a shutdown abort. Add the event to dump rollback segment information during recovery, and then startup the database. What happens?

Practice 4: Hang, Loop, and Crash Diagnostics

1. In this exercise you will attempt to crash a database and see the trace files generated. From the trace files, you will try to identify the problem and eventually fix it.

   Take the following steps:
   1. Bounce the database.
   2. From the operating system prompt, run the script *l3crsh.sh*.
   3. Log in as internal using SQL*Plus.
   4. Run a simple query like `select sysdate from dual;`

2. In this exercise, you will run a short PL/SQL script which loops infinitely. The aim of this exercise is to demonstrate the difference between a "process looping" situation and a "process hanging" situation. The script to run in this exercise is *l3loop.sql* which should be in your team directory. This PL/SQL script will loop infinitely and to exit out of the script, issue a ^*C*.

3. In this exercise you will create a "process hung" situation. You will have two concurrent user sessions connected. From one session, you will create a table, insert some rows, commit and finally, update a particular row. From the other session, you will try to update the same row. You will then notice that the process will not respond (or it will be hung). This is because the update from the first session was not committed.

   Scripts to run in this case are *l3hang1.sql* and *l3hang2.sql*. Run *l3hang1.sql*. This script creates a table, inserts some rows and updates a particular row. Run *l3hang2.sql* from another session. This script tries to update the same row.

Practice 5: Heap Corruption

1. Run the *l5heap1.sh* script and then diagnose the corruption.
   Take the following steps:

   1. Shut down and start up your database.
   2. From the UNIX command prompt run the *l4heap1.sh* command;
      this will introduce a heap corruption into the database.
   3. Connect to the database and query an object - this will fail and produce diagnostics.
   4. Inspect the alert.log file for any information related to the failure and determine the cause of the failure.

Practice 6: Block Dump Analysis

Run the *blocklab.sql* script in your schema to create a table.

1. Perform a UNIX binary dump on the BLOCK_LAB table.
2. Select the ROWID from BLOCK_LAB and dump the Oracle8 formatted block based on the ROWID.
3. Dump the Oracle8 formatted block of the BLOCK_LAB segment header.

Practice 7: Block Corruption, Diagnostics and Recovery

The following practice attempts to illustrate an in-memory block corruption. Follow the steps to introduce the corruption. Look for any errors produced and inspect the alert.log and trace files. Research the errors on tao and formulate steps to correct the error.

Perform the following steps:

1. Start SQL*Plus and connect as internal.
2. Create a table called BLOCKCOR, insert some values and commit.

```
SQL> create table blockcor (a number);
SQL> insert into blockcor values (1234);
SQL> commit;
```

3. Find the object number for BLOCKCOR and use it to find the block address.

```
SQL> select * from blockcor;     /* load table data into the cache */
SQL> select obj# from obj$
  2  where  name = 'BLOCKCOR';

SQL> select class,ba
  2  from x$bh where obj = xxx;    /* xxx: the object number found */
```

4. Corrupt the block in memory using oradebug.

```
SQL> oradebug poke 0xADDRESS1 1 0
SQL> oradebug poke 0xADDRESS2 1 0
```

   ADDRESS1 is the block address + 6 and *ADDRESS2* is the block address + 7. This corrupts bytes 6 and 7 in the block's cache layer. You should remember from lesson 6 that these bytes represent the two low order bytes of the RDBA of a block. The 1 and 0 at the end of the poke command represent the number of bytes to write and the value to write, respectively.

5. Insert some more values into BLOCKCOR.

```
SQL> insert into blockcor values (5678);
```

6. Observe any errors. Determine what has happened and how to repair it.

Practice 8: Rollback Segment Corruption

Startup the database used for this lesson. This instance has been purposely corrupted for the purpose of this excercise. Your instructor will provide the necessary information. Identify what is wrong with the database, take the necessary actions and bring the database back up.

Practice 9: Data Salvage Using DUL

1. A customer file system has suffered from a failure resulting in unrecoverable loss of all files in the database. The database consists of four user data files: two storing tables data, and two storing index data. The customer's backup strategy is flawed. The system tablespace files and user data files are backed up on different days. Furthermore, the database is open when the backups are taken, and the database is not in archivelog mode.

   The available data files are:

```
Rfile#  File#  File Name
     0      1  /u05/home/dsi/dsi301/TEAMB/systemTEAMB01.dbf
     1      2  /u05/home/dsi/dsi301/TEAMB/rbsTEAMB01.dbf
     2      3  /u05/home/dsi/dsi301/TEAMB/tempTEAMB.dbf
     3      4  /u05/home/dsi/dsi301/TEAMB/data01TEAMB.dbf
     4      5  /u05/home/dsi/dsi301/TEAMB/data02TEAMB.dbf
     5      6  /u05/home/dsi/dsi301/TEAMB/index01TEAMB.dbf
     6      7  /u05/home/dsi/dsi301/TEAMB/index02TEAMB.dbf
```

   The customer needs to unload the data from two tables, EMP and DEPT. Both tables belong to user PRODUSER. All the customer knows is that these two tables are stored in user tablespaces.

   Configure DUL and unload the two tables from the database into Export format files. The following default init.dul file is provided, but may need to be modified to reflect the hardware, operating system, and database configuration:

```
# sample init.dul configuration parameters
# these must be big enough for the database in question
# the cache must hold all entries from the dollar tables.
dc_columns = 200000
dc_tables = 10000
dc_objects = 10000
dc_segments=10000
dc_users = 400

# OS specific parameters
osd_big_endian_flag = false
osd_dba_file_bits = 6
osd_c_struct_alignment = 32
osd_file_leader_size = 1
osd_word_size = 32

# database parameters
db_block_size = 2048

export_mode=true
```

To modify the init.dul, see the following platform-specific parameters:

PORT	PLATFORM	ENDIAN	FBITS67	FBITS8	CSTR	LEAD	WORD
1	Digital VAX OpenVMS	FALSE	8	-	0	0	32
2	HP Series 9000 HP-UX	FALSE	8	-	0	0	32
21	Novell Netware	FALSE	8	8	0	1	32
22	MS DOS	FALSE	8	-	16	512	16
168	Silicon Graphics UNIX	TRUE	6	?	32	1	32
87	Digital Unix	FALSE	6	10	32	1	32
89	Alpha Vms	FALSE	8	10	32	0	32
198	Sequent Dynix/PTX	FALSE	6	?	32	1	32
319	IBM RS 600 AIX	TRUE	8	?	32	1	32
453	Sun Sparc Solaris	TRUE	6	10	32	1	32
912	MS Windows NT Intel	FALSE	8	10	32	1	32

Explanation of the values:

Title	Description
ENDIAN	OSD_BIG_ENDIAN_FLAG
FBITS67	OSD_DBA_FILE_BITS (Oracle version6/Oracle7)
FBITS8	OSD_DBA_FILE_BITS (Oracle8)
CSTR	OSD_C_STRUCT_ALIGNMENT
LEAD	OSD_FILE_LEADER_SIZE
WORD	OSD_WORD_SIZE

The dictv8.dll file is also available.

C
Solutions

Practice Solution 2: Source Code, the Ultimate Oracle

Mainly working with the Oracle 8.1.5 source code as a reference, answer the following questions:

1. What functions return the error ORA-1245?

   Use findora to find the files which refer to 1245:

```
tao-sun% findora 1245
Searching for OER(1245)
File:   /export/home/ssupport/815/rdbms/src/server/rcv/kcv.c
Line #: 13291
       ksesec1(OER(1245), ksenrg(fno));           /* file will be lost */
```

   View the file and use /1245 to scan down to the error:

```
tao-sun% view /export/home/ssupport/815/rdbms/src/server/rcv/kcv.c

/* report an error */
       kcfrfn(fno, ctxp, (kmid)0);    /* record file name in err frame */
       kccacx(ctxp);
       ksesec1(OER(1245), ksenrg(fno));           /* file will be lost */
```

   Now scan upwards in the file (^U) to find the function that contains this code:

```
/*
** Verify Reset Allowed
** Scan all the data file headers to insure that none are in a state that would
** indicate a reset logs would leave a corrupted data base. This checks for
** files that are in the middle of recovery from a hot backup. It insures that
** all of the files are at the same checkpoint SCN as the end of incomplete
** recovery, or at the resetlogs SCN (incase the header was updated by an
** earlier RESETLOGS that crashed)
*/
STATICF word kcvvra(ctxp, dip, ckp, maxrel, cfckp)
reg0 kcccx  *ctxp;                                      /* ctrl xaction ptr */
reg3 kccdi  *dip;                                     /* current database info */
reg4 kcvcp  *ckp;                                      /* new reset checkpoint */
```

   The only function that logs error ORA-1245 is kcvvra( ).

2. Name a function that returns the error ORA-1578. The file number is also logged with this error. Is this the relative or absolute file number?

   Use findora to find 1578:

```
tao-sun% findora 1578
Searching for OER(1578)
File:   /export/home/ssupport/815/rdbms/src/server/cache/if/kcb.h
Line #: 1078
#define KCBCRRPT  OER(1578)                            /* data block corrupted */
```

   OER(1578) is defined as the macro KCBCRRPT so we must search for the macro instead. Using cscope:

```
C  symbol:  KCBCRRPT

   File    Function            Line
1  kcb.h   <global>            1074  #define KCBCRRPT OER(1578)
2  kcb.h   <global>            1074  #define KCBCRRPT OER(1578)
3  ktfb.h  KTFBERR_FILE_SKIP    414  ((err) == OER(372) || (err) == OER(376)
                                                        ||(err) == KCBCRRPT)
4  ktfb.h  KTFBERR_BLOCK_CORRUPT 417      #define KTFBERR_BLOCK_CORRUPT(err)
                                                          ((err) == KCBCRRPT)
5  kcb.c   kcbcrr              3786  ksesec2(KCBCRRPT, ksenrg( KRD2RFN(b->kcbbhrdba)),
6  kcb.c   kcbema              8507  kserec2(KCBCRRPT, ksenrg(afn),
                                      ksenrg(KRD2BLK(rdba)));
7  kcb.c   kcbsor              9082  kserec2(KCBCRRPT, ksenrg(b->kcbbhafn),
8  kcb.c   kcbdrget           11703  ksesec2(KCBCRRPT, ksenrg(afn),
9  kdi.c   kdifxs              2775  if (err == KCBCRRPT &&

Lines 1-9 of 12, press the space bar to display next lines
```

kcbcrr( ) is an example of a function that returns ORA-1578. Let's take a closer look at kcbcrr( ) to see what else it logs with the error:

```
ksesec2(KCBCRRPT, ksenrg( KRD2RFN(b->kcbbhrdba)),
            ksenrg( KRD2BLK(b->kcbbhrdba)));
```

As well as the error we also log KRD2RFN(..) and KRD2BLK(..). Find out what they are, using cscope:

```
C symbol: KRD2RFN

  File    Function         Line
1 ktst.c <global>         1048 row_result->seghrfno_ktstxt =
                               KRD2RFN(sort_descriptor->segdba_ktstssd);
2 k.h     KRDBA_DMPF       630 ksdwrf("rdba: 0x%08lx (%u/%lu)", (unsigned long )(r_dba),
                               KRD2RFN((r_dba)), \
3 k.h     KDBAFR_DMPF      635 (unsigned long )(r_dba), KRD2RFN((r_dba)), KRD2BLK((r_dba)))
4 k.h     KDBAFA_DMPF      639 (unsigned long )(r_dba), KRD2RFN((r_dba)), KRD2BLK((r_dba)))
5 k.h     KIECBBAS         695 #define KRD2RFN(rdba) ((krfil)(DBAFOLD(rdba) |
                               (DBAFNEW(rdba) << SDBAOFBITS)))
6 k.h     KIECBBAS         698 #define KRD2RFN(rdba)  ((krfil)((ub4)(rdba) >> DBABBITS))
7 kd4.h   kd_ubrid_physini 426 f_local = (ub2)KRD2RFN(d_local); \
8 kd4.h   kd_sbri2rid      591 ridini_restricted(r, (ub2)KRD2RFN(d), KRD2BLK(d),
                               (ub2)kd_sbrigsno(b)); \
9 kd4.h   kd_lbri2rid      611 (ub2)KRD2RFN(d), KRD2BLK(d), (ub2)kd_lbrigsno(b)); \
```

Choosing 5:

```
** KRFN2RD - relative file number to relative DBA
**
**   krdba KRFN2RD(krfil rfn, kblk bno);
**     Parameters:
**       rfn - (IN) relative file number
**       bno - (IN) block number
**     Returns:
**       relative DBA
```

The file number in the ORA-1578 error message is the relative file number.

3. What functions return the error ORA-600 [12025] in Oracle 8.0.5? Explain the conditions that would cause this error to be logged.

Use find600 to locate the error:

```
tao-sun% find600 12025
Searching for OERI(12025) or OERINM("12025")
File:   /export/home/ssupport/815/rdbms/src/server/sqlexec/sort/srsima.c
Line #: 668
             OERI(12025),                              /* incorrect block type */

File:   /export/home/ssupport/815/rdbms/src/server/sqlexec/sort/srsima.c
Line #: 905
             OERI(12025),                              /* incorrect block type */
```

The error is logged in two places from srsima.c. Let's check this out:

```
tao-sun% view /export/home/ssupport/815/rdbms/src/server/sqlexec/sort/srsima.c
```

In srsget( ):

```
ASSERT(stsgbt(&sorp->sorstc, sdbp) == K_BTSOKY,
             OERI(12025),                              /* incorrect block type */
             KSESVSGN);
```

And in srsgnd1( ):

```
ASSERT(stsgbt(&sorp->sorstc, sdbp) == K_BTSOKY,
             OERI(12025),                              /* incorrect block type */
             KSESVSGN);
```

The error is logged if the value returned from stsgbt() is not equal to K_BTSOKY. Let's check out stsgbt( ) using cscope815:

```
#define stsgbt(ctx,bp)          kcbgbt((ptr_t)bp)
```

And kcbgbt( ):
```
#define kcbgbt(h) (kcbhgbt(kcbhhp(h)))
```

And kcbhgbt( ):
```
#define kcbhgbt(p) (((kcbh *)(p))->type_kcbh)
```

And type_kcbh:
```
/* This ub1 contains the block type. This is used for calling the correct
 * block dump/edit routine. It is also used for insuring that the block
 * is still the same format when there is the possibility that the block
 * has been reused. */
   ub1     type_kcbh;
```

Also, K_BTSOKY:
```
#define K_BTSOKY  8                                                  /* sort key */
```

Conclusion for Oracle 8.0.5: This error is logged by srsget() and srsgnd1() if a block is found with the wrong block type.

4. What function returns the ORA-600 [12025] in 8.1.5?

   Change the source tree to 815. Then use find600 to find where it is logged.

```
$ find600 12025
Searching for OERI(12025) or OERINM("12025")
```

   Conclusion for 8.1.5: The ORA-600 [12025] is no longer logged in Oracle 8.1.5.

5. What functions return the error ORA-600 [kfhcfh1_01] in Oracle 8.0.5? What other information is returned with this error?

   Use find600 to find the code:
```
tao-sun% cdb 815
tao-sun% find600 kfhcfh1_01
Searching for OERI(kfhcfh1_01) or OERINM("kfhcfh1_01")
File:   /export/home/ssupport/805/rdbms/src/server/rcv/kfh.c
Line #: 2027
 ksesin(OERINM("kfhcfh1_01"), 1, ksenrg(version));
```

   View the file and go to line 2027:

```
else
  {
     /* unknown version */
     if (sigerr)
        ksesin(OERINM("kfhcfh1_01"), 1, ksenrg(version));
     else
        return FALSE;
  }
```

   Scan backwards to get the function that returns the error:

```
#define v7p ((kfhfh7*)fhp)          /* pointer to file header as a v7 header */
  word   version = 0;
  kfhfh7 v7po;                      /* saved v7 file header                  */
                                 /* Try to determine the file header version. */
  if (v6p->kfhfh6dbi == kccgdd()                        /* correct DB id ... */
      && v6p->kfhfh6typ == 3                      /* and v6 datafile type ... */
      && (VSNNUMVSN(v6p->kfhfh6swv)&0x7f) == 6  /* and created by version 6 .*/
      && (VSNNUMVSN(v6p->kfhfh6cvn)&0x7f) == 6)/* and compatible with vern 6 */
    version = 6;
```

So, v7p( ) is the macro that returns the error. Something called "version" is also returned. The version is set to 0, and not changed if the error is logged so logging the "version" adds no value.

6. In which file is the *_corrupted_rollback_segments* parameter defined for Oracle 8.1.5? What is the internal identifier of this parameter? Where is this parameter referenced in the code?

   Using cscope, perform an egrep for corrupted_rollback_segments:

```
Egrep pattern: corrupted_rollback_segme

1 ktucts.h  895 KSPPARDV("_corrupted_rollback_segments",
                ktucrs_, ktucrs_s, ktucrs_l, ktucrs_p,
2 ktucts.h  895 KSPPARDV("_corrupted_rollback_segments",
                ktucrs_, ktucrs_s, ktucrs_l, ktucrs_p,
3 ktu.c    1320 /* cannot specify SYSTEM in _corrupted_rollback_segments */
4 ktu.c    1322 ksesec1(OER(1596), ksesrgl
                ((text *)"_corrupted_rollback_segments"));
```

   The definition lies in ktucts.h:

```
#define ktucrs KSPPARDN(ktucrs_)
KSPPARDV("_corrupted_rollback_segments", ktucrs_, ktucrs_s, ktucrs_l, ktucrs_p,
         LCCMDSTR, 0, NULLP(CONST text), UB4MAXVAL, KSPLS1(NULLP(text)),
         KSPLS1(UB4MAXVAL), 0, KSPLS1(NULLP(word)),
         "corrupted undo segment list")
```

   The internal identifier is ktucrs. Using cscope815 again, search for this symbol:

```
C symbol: ktucrs

  File     Function Line
1 ktucts.h KSMSGADV  869 #define ktucrs KSPPARDN(ktucrs_)
2 ktu.c    ktunfy    961 ncor = kspgvc(ktucrs)+1;
3 ktu.c    ktuiof   1160 count = kspgvc(ktucrs);
4 ktu.c    ktuiof   1163 us.kidnlen = min((ub2)M_IDEN,
                         (ub2)kspgsp(ktucrs, i, &argp));
```

   The parameter is referenced by ktunfy( ) and ktuiof( ).

7. On which fixed (X$) tables is the *v$process* view ultimately defined in Oracle 8.1.5?
   Which internal structures is/are used to derive the *serial#* column?

   Find *v$process* in kqfv.h:

```
KQFVIEW("V$PROCESS" , kqfv005_c,
"select addr, pid,spid,username,serial#,terminal,program,background,latchwait,\
latchspin from gv$process where inst_id = USERENV('Instance') ", KQFOB005, 4)
KQFVCOL("ADDR")                     /* address of process state object (raw) */
KQFVCOL("PID")                         /* oracle process identifier (number) */
KQFVCOL("SPID")                /* operating system process identifier (char) */
KQFVCOL("USERNAME")             /* operating system process user name (char) */
KQFVCOL("SERIAL#")                                  /* process serial number */
KQFVCOL("TERMINAL")           /* operating system terminal identifier (char) */
KQFVCOL("PROGRAM")                             /* program in progress (char) */
KQFVCOL("BACKGROUND")               /* 1 - background process, null - normal */
KQFVCOL("LATCHWAIT")      /* address of latch waiting for, null - none (raw) */
KQFVCOL("LATCHSPIN")     /* address of latch spinning for, null - none (raw) */
KQFVEND()
```

   Note that *v$process* is based on *gv$process*. This is the definition of *gv$process*:

```
KQFVIEW("GV$PROCESS" , kqfv344_c,
"select inst_id, addr,indx,ksuprpid,ksuprunm,ksuprser,ksuprtid,ksuprpnm,\
 decode(bitand(ksuprflg,2),0,null,1),\
 decode(ksllawat,hextoraw('00'),null,ksllawat),\
 decode(ksllaspn,hextoraw('00'),null,ksllaspn)\
```

```
 from x$ksupr where bitand(ksspaflg,1)!=0", KQFOB344, 1)
KQFVCOL("INST_ID")
KQFVCOL("ADDR")                     /* address of process state object (raw) */
KQFVCOL("PID")                         /* oracle process identifier (number) */
KQFVCOL("SPID")                /* operating system process identifier (char) */
KQFVCOL("USERNAME")             /* operating system process user name (char) */
KQFVCOL("SERIAL#")                                 /* process serial number */
KQFVCOL("TERMINAL")           /* operating system terminal identifier (char) */
KQFVCOL("PROGRAM")                              /* program in progress (char) */
KQFVCOL("BACKGROUND")               /* 1 - background process, null - normal */
KQFVCOL("LATCHWAIT")      /* address of latch waiting for, null - none (raw) */
KQFVCOL("LATCHSPIN")     /* address of latch spinning for, null - none (raw) */
KQFVEND()
```

As you can see, *gv$process* is based on *x$ksupr*. The *serial#* parameter is derived from the KSUPRSER column. The definition of X$ tables is elsewhere, guess ksu.h:

```
KQFTABL(ksupr, ksupr_c, "X$KSUPR", ksusga.ksusgpra, ksusga.ksusgprl,
        KQFOB093, 6)
KSSPATY(ksupr, ksuprgen)
KQFCINT(ksupr, ksuprflg, "KSUPRFLG")
KQFCINT(ksupr, ksuprser, "KSUPRSER")
KSLLATY(ksupr, ksuprlki)
KQFCSTR(ksupr, ksuprpri.ksuospid, "KSUPRPID", SPOSPIDLEN,  ksuprpri.ksuospdl)
KQFCSTR(ksupr, ksuprpri.ksuosunm, "KSUPRUNM", SPUSRNMLEN,  ksuprpri.ksuosunl)
KQFCSTR(ksupr, ksuprpri.ksuosmnm, "KSUPRMNM", SPMACHNMLEN, ksuprpri.ksuosmnl)
KQFCSTR(ksupr, ksuprpri.ksuospnm, "KSUPRPNM", SPPRGNMLEN,  ksuprpri.ksuospnl)
KQFCSTR(ksupr, ksuprpri.ksuostnm, "KSUPRTID", SPTNMLEN,    ksuprpri.ksuostnl)
KSSRCTY(ksupr, ksuprsrc)
KSASTTY(ksupr, ksuprmsg)
KQFENDT(ksupr)
```

The KSUPRSER parameter is derived from the ksuprser symbol.

8. In which file is the *redo allocation* latch defined in Oracle 8.1.5?
   What is its internal identifier and where is it used?

   Using cscope815, egrep for “redo allocation”. In kcrf.h:

```
# define kcrfal KSLLATDN(kcrfal_)
KSLLATDV(kcrfal_, "redo allocation", FADDR(kcralc), KSLLMXLV-2)
```

   The latch identifier is kcrfal. Using cscope again, search for this symbol:

```
C symbol: kcrfal

1 kcrf.h  <global> 1033 #define kcrfal KSLLATDN(kcrfal_)
2 kcrf.h  <global> 1033 #define kcrfal KSLLATDN(kcrfal_)
3 kcrf.h  KSMUGFDV 1103 KSLLAWHRDV(kcral10_, "kcrfwr: redo allocation",
                        "pass #", kcrfal)
4 kcrf.h  KSMUGFDV 1111 KSLLAWHRDV(kcral12_, "kcrfml", "", kcrfal)
5 kcrf.h  KSMUGFDV 1131 KSLLAWHRDV(kcral14_, "kcrfgr", "", kcrfal)
6 kcrf.h  KSMUGFDV 1135 KSLLAWHRDV(kcral15_, "kcrfwi: before write",
                        "kcrfs addr", kcrfal)
7 kcrf.h  KSMUGFDV 1147 KSLLAWHRDV(kcral16_, "kcrfwi: no space", "", kcrfal)
8 kcrf.h  KSMUGFDV 1151 KSLLAWHRDV(kcral17_, "kcrfwi: more space",
                        "new bufs", kcrfal)
9 kcrf.h  KSMUGFDV 1159 KSLLAWHRDV(kcral08_, "kcrfwfl", "", kcrfal)

Lines 1-9 of 48, press the space bar to display next lines
```

9. Which function controls the processing of the *alter database* command in Oracle 8.1.5? How is *open resetlogs* handled?

Let's start by looking opiexe.c and searching for alter database:

```
case OCTADB:                                        /* alter database */
        /* security check in dbsdrv */
        adbdrv();
        break;
```

The OCT code is OCTADB. The implementation is in adbdrv(). Next, find adbdrv( ) using cscope.

```
1 drv.h    <global>   90 void adbdrv();
2 drv.h    <global>   90 void adbdrv();
3 dbsdrv.c <global>  836 void adbdrv()
4 opiexe.c opiexe   2041 adbdrv();
```

Let's look in drv.h:

```
case ADBORESETL:                                    /* alter db open resetlogs */
```

We have a special identifier ADBORESETL. Where we find this identifier we will see how resetlogs is performed. Using cscope find where the identifier ADBORESETL is implemented.

```
if (adbp->adbopt != ADBOOPEN)
       kcvorl((adbp->adbopt == ADBORESETL), FALSE);
```

kcvorl( ) performs the open with resetlogs.

Practice Solution 3: Using Diagnostic Events

1. Log on to the database as user SCOTT and do a process-state dump.

```
SQL> connect SCOTT/TIGER
Connected.

SQL> alter session set events 'immediate trace name PROCESSSTATE level 10';
Session altered
```

In the user_dump_dest directory, a file called <SID>_ora_<PID>.trc will be created which contains the following process state dump:

```
Dump file /kurse/dba4/dba40/UDUMP/d40_ora_5583.trc
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
ORACLE_HOME = /oracle81
System name:    SunOS
Node name:      trgsunm3
Release:        5.6
Version:        Generic_105181-05
Machine:        sun4u
Instance name: D40
Redo thread mounted by this instance: 1
Oracle process number: 8
Unix process pid: 5583, image: oracle@trgsunm3 (TNS V1-V3)

*** 1999.11.24.18.02.07.000
*** SESSION ID:(7.788) 1999.11.24.18.02.07.000
==========================================================
PROCESS STATE
-------------
Process global information:
     process: 2203b49c, call: 220651d0, xact: 0,
     curses:  22049a10, usrses: 22049a10
...
```

2. Log on as user SCOTT in one session, start SQL*Plus in another session, and do a system state dump from the SCOTT session using the ORADEBUG utility.

a. Find the process id of the session you need to dump. Query v$process:

```
SQL> select spid, program, terminal,username from v$process;

SPID      PROGRAM                          TERMINAL   USERNAME
--------- -------------------------------- ---------- ---------------
          PSEUDO
3256      oracle@trgsunm3 (PMON)           UNKNOWN    dba40
3258      oracle@trgsunm3 (DBW0)           UNKNOWN    dba40
3260      oracle@trgsunm3 (LGWR)           UNKNOWN    dba40
3262      oracle@trgsunm3 (CKPT)           UNKNOWN    dba40
3264      oracle@trgsunm3 (SMON)           UNKNOWN    dba40
3266      oracle@trgsunm3 (RECO)           UNKNOWN    dba40
6015      oracle@trgsunm3 (TNS V1-V3)      pts/3      dba40
6026      oracle@trgsunm3 (TNS V1-V3)      pts/4      dba40
9 rows selected.
```

b. Connect as SYSDBA and execute the following ORADEBUG commands:

```
SQL> oradebug setospid 6015
Oracle pid: 8, Unix process pid: 6015, image: oracle@trgsunm3 (TNS V1-V3)
SQL> oradebug unlimit
Statement processed.
SQL> oradebug dump systemstate 10
Statement processed.
```

c. In the user_dump_dest directory, a file called <SID>_ora_6015.trc will be created which contains the following process state dump:

```
dba40 D40> more d40_ora_6015.trc
Dump file /kurse/dba4/dba40/UDUMP/d40_ora_6015.trc
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
ORACLE_HOME = /oracle81
System name:    SunOS
Node name:      trgsunm3
Release:        5.6
Version:        Generic_105181-05
Machine:        sun4u
Instance name: D40
Redo thread mounted by this instance: 1
Oracle process number: 8
Unix process pid: 6015, image: oracle@trgsunm3 (TNS V1-V3)

*** 1999.11.25.11.28.05.000
*** SESSION ID:(7.1037) 1999.11.25.11.28.05.000
==================================================
SYSTEM STATE
------------
System global information:
     processes: base 22039c9c, size 50, cleanup 2203a29c
allocation: free sessions 2204a1ec, free calls 0
     control alloc errors: 0 (process), 0 (session), 0 (call)
     system statistics:
          0         14 logons cumulative
          0          8 logons current
          0       1585 opened cursors cumulative
          0          1 opened cursors current
          0          0 user commits
          0          0 user rollbacks
          0        367 user calls
          0      45365 recursive calls
          0        172 recursive cpu usage
          0    2308964 session logical reads
          0          0 session stored procedure space
          0        255 CPU used when call started
          0        259 CPU used by this session
          0 1147216481 session connect time
...
```

3. Set the event with the alter session command so that an error stack will be produced whenever the error "table or view does not exist" is reported. Verify that the trace file is created by running the following query:

```
SQL> select * from a;

SQL> alter session set events '942 trace name errorstack forever';
Session altered.

SQL> select * from a;
select * from a
              *
ERROR at line 1:
ORA-00942: table or view does not exist
```

In the user_dump_dest directory, a file called <SID>_ora_<PID>.trc will be created which contains the following process state dump:

```
dba40 D40> more d40_ora_6120*
Dump file /kurse/dba4/dba40/UDUMP/d40_ora_6120.trc
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
ORACLE_HOME = /oracle81
```

```
System name:    SunOS
Node name:      trgsunm3
Release:        5.6
Version:        Generic_105181-05
Machine:        sun4u
Instance name: D40
Redo thread mounted by this instance: 1
Oracle process number: 9
Unix process pid: 6120, image: oracle@trgsunm3 (TNS V1-V3)

*** SESSION ID:(9.80) 1999.11.25.12.06.57.000
*** 1999.11.25.12.06.57.000
ksedmp: internal or fatal error
ORA-00942: table or view does not exist
Current SQL statement for this session:
select * from a
----- Call Stack Trace -----
...
```

4. Activate the event from another session using the dbms_system package.

   Query v$session:

```
SQL> select program,sid, serial#, process from v$session;

PROGRAM                                          SID SERIAL# PROCESS
---------------------------------------------- ----- ------- -------
oracle@trgsunm3 (PMON)                             1       1 3256
oracle@trgsunm3 (DBW0)                             2       1 3258
oracle@trgsunm3 (LGWR)                             3       1 3260
oracle@trgsunm3 (CKPT)                             4       1 3262
oracle@trgsunm3 (SMON)                             5       1 3264
oracle@trgsunm3 (RECO)                             6       1 3266
sqlplus@trgsunm3 (TNS V1-V3)                       7    1039 6144
sqlplus@trgsunm3 (TNS V1-V3)                       9      80 6119
```

   Execute the dbms_system.set_ev procedure:

```
SQL> execute dbms_system.set_ev(9,80,942,10,'errorstack');
PL/SQL procedure successfully completed.
```

   After querying a nonexisting table, check out the UDUMP or $ORACLE_HOME/rdbms/log directory and verify that the trace file is created.

5. Repeat step 4 using the ORADEBUG utility.

```
SQL> select spid, program from v$process;

SPID      PROGRAM
--------- --------------------------
          PSEUDO
3256      oracle@trgsunm3 (PMON)
3258      oracle@trgsunm3 (DBW0)
3260      oracle@trgsunm3 (LGWR)
3262      oracle@trgsunm3 (CKPT)
3264      oracle@trgsunm3 (SMON)
3266      oracle@trgsunm3 (RECO)
6145      oracle@trgsunm3 (TNS V1-V3)
6120      oracle@trgsunm3 (TNS V1-V3)

SQL> oradebug setospid 6120
Statement processed.
SQL> oradebug unlimit
Statement processed.
SQL> oradebug session_event 942 trace name errorstack forever
Statement processed.
```

After querying a nonexisting table, check out the UDUMP or $ORACLE_HOME/rdbms/log directory and verify that the trace file is created.

6. Repeat step 4 by editing the parameter file.

```
event = "942 trace name errorstack forever"
```

After querying a nonexisting table, check out the UDUMP or $ORACLE_HOME/rdbms/log directory and verify that the trace file is created.

7. Create a small table EMP, do some INSERTS and then do a shutdown abort. Add the event to dump rollback segment information during recovery, and then startup the database. What happens?

init<SID>.ora:

```
event = "10015 trace name context forever"
```

In the user_dump_dest directory or $ORACLE_HOME/rdbms/log directory the following trace file will be created:

```
/oracle81/rdbms/log/d40_ora_6264.trc
Oracle8i Enterprise Edition Release 8.1.5.0.0 - Production
With the Partitioning and Java options
PL/SQL Release 8.1.5.0.0 - Production
ORACLE_HOME = /oracle81
System name:    SunOS
Node name:      trgsunm3
Release:        5.6
Version:        Generic_105181-05
Machine:        sun4u
Instance name: D40
Redo thread mounted by this instance: 0 <none>
Oracle process number: 0
6264

skgm warning: EINVAL creating segment of size 0000000004506000
fix shm parameters in /etc/system or equivalent
skgm warning: EINVAL creating segment of size 0000000004504000
fix shm parameters in /etc/system or equivalent
skgm warning: EINVAL creating segment of size 0000000004500000
fix shm parameters in /etc/system or equivalent
skgm warning: EINVAL creating segment of size 00000000044ec000
...
*** SESSION ID:(7.1) 1999.11.25.12.53.01.000
Acquiring rollback segment SYSTEM
Recovering rollback segment RBS01
Recovering rollback segment RBS02
Recovering rollback segment RBS03
Recovering rollback segment RBS04
Acquiring rollback segment RBS01
Acquiring rollback segment RBS02
Acquiring rollback segment RBS03
Acquiring rollback segment RBS04
```

Practice Solution 4: Hang, Loop, and Crash Diagnostics

1. In this exercise you will attempt to crash a database and see the trace files generated. From the trace files, you will try to identify the problem and eventually fix it.

   Take the following steps:
   1. Bounce the database.
   2. From the operating system prompt, run the script *l3crsh.sh*.
   3. Log in as internal using SQL*Plus.
   4. Run a simple query like `select sysdate from dual;`

   You should see the following message:

```
SQL> connect internal
Connected.
SQL> select sysdate from dual;
select sysdate from dual
                    *
ORA-00600: internal error code, arguments: [kcfrbd_1],[1],[0],[],[],[],[],[]
```

   Diagnostics:
   Next, look in the alert.log for some more information You should see a pointer to a trace file for more detailed information. You might see something similar to the following:

```
Successful open of redo thread 1.
Tue Nov 30 02:25:35 1999
SMON: enabling cache recovery
SMON: enabling tx recovery
Tue Nov 30 02:25:42 1999
Completed: alter database open
Wed Dec  8 14:50:29 1999
Starting ORACLE instance (normal)
Wed Dec  8 15:16:41 1999
Errors in file /u01/app/oracle/admin/TEAMM/udump/teamm_ora_16134.trc:
ORA-00600: internal error code, arguments: [kcfrbd_1],[1],[0],[],[],[],[],[]
```

   Looking in <SID>_ora_16134.trc file:

```
*** 1999-12-08 15:16:41.406
ksedmp: internal or fatal error
ORA-00600: internal error code, arguments: [kcfrbd_1],[1],[0],[],[],[],[],[]
Current SQL statement for this session:
select ts#,file#,block#,nvl(bobj#,0),nvl(tab#,0),intcols,nvl(clucols,0),audit$,
flags,pctfree$,pctused$,initrans,maxtrans,rowcnt,blkcnt,empcnt,avgspc,chncnt,
avgrln,analyzetime,samplesize,cols,property,nvl(degree,1),nvl(instances,1),
avgspc_flb,flbcnt,kernelcols,nvl(trigflag,0),nvl(spare1,0),nvl(spare2,0),
spare4,nvl(spare3,0) from tab$ where obj#=:1

----- Call Stack Trace -----
calling           call     entry              argument values in hex
location          type     point              (? means dubious value)
----------------- -------- ------------------ ----------------------------
ksedmp()+168      CALL     ksedst()+0         540 ? 0 ? EFFF6ED4 ?
                                              EFFF6978 ? EFFF695C ? 0 ?
kgerinv()+184     PTR_CALL 00000000           3 ? 0 ? 0 ? 16BF700 ? 2 ?
                                              EFFF7484 ?
kgeasnmierr()+28  CALL     kgerinv()+0        17E7A9C ? 1872BBC ? 146F980 ?
                                              2 ? EFFF7484 ? 17D6D08 ?
kcfrbd()+1116     CALL     kgeasnmierr()+0    17E7A9C ? 1872BBC ? 16BF700 ?
                                              2 ? 0 ? 1 ?
kcbzib()+920      CALL     kcfrbd()+0         0 ? EFFF7D08 ? 1 ? 5C8 ? 0 ?
                                              0 ?
kcbgtcr()+3188    CALL     kcbzib()+0         80007F18 ? EFFF8C5C ? 4 ? 0 ?
                                              0 ? 1 ?
```

This trace gives us the root of the problem which was a recursive statement (querying the data dictionary) failing with an ORA-00600: [kcfrbd_1], error. Log into WebIV and go to the admin folder on your menu. Choose Oracle7/Oracle8 Stack Interpreter from the Support Tools folder. Diagnose the problem using information gathered here. If neded, you may research kcfrbd_1 in the source code.

To get the database back up, shutdown the database using `SHUTDOWN ABORT` to clean up any resources and startup again.

2. In this exercise, you will run a short PL/SQL script which loops infinitely. The aim of this exercise is to demonstrate the difference between a "process looping" situation and a "process hanging" situation.
The script to run in this exercise is *l3loop.sql* which should be in your team directory. This PL/SQL script will loop infinitely and to exit out of the script, issue a *^C*.

Diagnostics:
After firing off the script, open another session at operating system level and issue a *ps* command to monitor processes. On most platforms *ps -ef* would suffice. In order to trace the offending process, you will have to *grep* for the client process from where you fired off this script (*sqlplus* process). Please note that you have to use the server process and not client-side process.

```
ps -ef|grep sqlplus
  oracle 16196 16114  0 15:56:03 pts/16  0:00 sqlplus
  oracle 16241 16232  0 15:59:49 pts/17  0:00 grep sqlplus

$ ps -ef|grep 16196
  oracle 16197 16196 48 15:56:03 ?       1:35 oracleTEAMM (DESCRIPTION=(LOCAL=)
  oracle 16196 16114  0 15:56:03 pts/16  0:00 sqlplus

$ ps -ef|grep 16196
  oracle 16197 16196 49 15:56:03 ?       1:51 oracleTEAMM (DESCRIPTION=(LOCAL=)
  oracle 16196 16114  0 15:56:03 pts/16  0:00 sqlplus

$ ps -ef|grep 16196
  oracle 16197 16196 49 15:56:03 ?       2:00 oracleTEAMM (DESCRIPTION=(LOCAL=)
  oracle 16196 16114  0 15:56:03 pts/16  0:00 sqlplus
  oracle 16251 16232  0 16:00:40 pts/17  0:00 grep 16196
```

Look at the TIME statistic for `oracleTEAMM` process above, it shows an upward trend uniformly. This implies that the process is consuming CPU and hence looping. In a HUNG situation this statistic should remain static.

3. In this exercise you will create a "process hung" situation. You will have two concurrent user sessions connected. From one session, you will create a table, insert some rows, commit and finally, update a particular row. From the other session, you will try to update the same row. You will then notice that the process will not respond (or it will be hung). This is because the update from the first session was not committed.

   Scripts to run in this case are *l3hang1.sql* and *l3hang2.sql*. Run *l3hang1.sql*. This script creates a table, inserts some rows and updates a particular row. Run *l3hang2.sql* from another session. This script tries to update the same row.

   Diagnostics:
   Use the ps command to determine if the process is gaining CPU time. If the process is not gaining CPU time then it must be hung. Query V$LOCK and see if you can determine if this hung process is waiting on some resource.

   Example output:

```
SQL> select sid, type, id1, id2, request, block from v$lock;

       SID TY         ID1         ID2    REQUEST      BLOCK
---------- -- ---------- ---------- ---------- ----------
         2 MR          7          0          0          0
         2 MR          1          0          0          0
         2 MR          2          0          0          0
         2 MR          3          0          0          0
         2 MR          4          0          0          0
         2 MR          5          0          0          0
         2 MR          6          0          0          0
         3 RT          1          0          0          0
         7 TX     458782         91          0          1
         7 TM      20697          0          0          0
        10 TM      20697          0          0          0
        10 TX     458782         91          6          0
```

   Look at SIDs 7 and 10. SID 10 is waiting on SID 7 (SID 10 has REQUESTed for a resource and SID7 is BLOCKing the resource). To resolve this problem issue a commit in the first session.

Practice Solution 5: Heap Corruption

1. Run the *l5heap1.sh* script and then diagnose the corruption.
   Take the following steps:

   1. Shut down and start up your database.
   2. From the UNIX command prompt run the *l4heap1.sh* command;
      this will introduce a heap corruption into the database.
   3. Connect to the database and query an object; this will fail and produce diagnostics.
   4. Inspect the alert.log file for any information related to the failure and determine the cause of the failure.

   Selecting from scott's EMP table (step 3) produces the following results:

```
SQL> select * from scott.emp;
select * from scott.emp
                    *
ERROR at line 1:
ORA-00600: internal error code, arguments: [kcfrbd_1],[1],[0],[],[],[],[],[]
```

   Looking in the alert.log:

```
Errors in file /u01/app/oracle/admin/TEAMM/udump/teamm_ora_29586.trc:
ORA-00600: internal error code, arguments: [kcfrbd_1],[1],[0],[],[],[],[],[]
```

   Looking in file /u01/app/oracle/admin/TEAMM/udump/teamm_ora_29586.trc you find that the same kcfrbd_1 is logged. Go to tao and research this error.

```
$ find600 kcfrbd_1
Searching for OERI(kcfrbd_1) or OERINM("kcfrbd_1")
File:   /export/home/ssupport/815/rdbms/src/server/rcv/kcf.c
Line #: 9552
   ASSERTNM2(fno > 0 && fno <= kcfdfk, OERINM("kcfrbd_1"), fno, kcfdfk);
```

   Using cscope, view kcf.c

```
/kcfrbd_1
   ub4      bno = strtbno;                /* block # corresponding to dba */
   word     nread;                           /* number of blocks read */
   ub4      base;                               /* time io is started */
   ub4      elapsed;                             /* time taken for io */
   kcfio   *kcfiop;                        /* pointer to io stastistics */
   kgfdbd  *bds;
   kgfdbd   bd;
   serc     se;                                    /* OSD error buffer */

                                                  /* check file number */
   ASSERTNM2(fno > 0 && fno <= kcfdfk, OERINM("kcfrbd_1"), fno, kcfdfk);
```

   After researching the error we find that the variable containing the value for the maximum number of datafiles for the database (kcfdfk) has been exceeded. This value was corrupted by the script but in live customer cases this kind of corruption can occur as a result of a client or background process inadvertantly overwriting the memory pointer storing kcfdfk.

Practice Solution 6: Block Dump Analysis

Run the *blocklab.sql* script in your schema to create a table.

blocklab.sql

```
drop table block_lab;

create table block_lab (
  id       number,
  name     varchar2(20),
  hire_dt  date);

insert into block_lab values (1,'Richard',to_date('1990-May-12','YYYY-Mon-dd'));
insert into block_lab values (2,'Andy',   to_date('1990-Jun-01','YYYY-Mon-dd'));
insert into block_lab values (3,'Sunitha',to_date('1995-Nov-10','YYYY-Mon-dd'));
insert into block_lab values (4,'VB',     to_date('1991-Feb-14','YYYY-Mon-dd'));
insert into block_lab values (5,'Ali',    to_date('1986-Sep-23','YYYY-Mon-dd'));
```

1. Perform a UNIX binary dump on the BLOCK_LAB table.

```
SQL> select file_id, block_id from dba_extents
  2  where  segment_name = 'BLOCK_LAB';

FILE_ID   BLOCK_ID
------- ----------
      1       5060

SQL> select file#, name from v$datafile;

FILE#      NAME
----- ----------------------------
    1 /u01/oradata/816/system.dbf
    2 /u01/oradata/816/temp.dbf
    3 /u01/oradata/816/rbs.dbf
    6 /u01/oradata/816/rbs2.dbf
    8 /u01/oradata/816/big.dbf

SQL> exit

% dd if=/u01/oradata/816/system.dbf bs=4k skip=5061 count=5 | od -x > ddbd3.txt

5+0 records in
5+0 records out
ccd-orsun3% more ddbd3.txt
0000000 0602 0000 0040 13c5 0000 af23 0000 0102
0000020 0000 0000 0100 0000 0000 0c2d 0000 af22
0000040 0000 eb50 0001 0300 0000 0000 0004 002a
0000060 0000 0038 00c0 134b 0023 1e14 2005 0000
0000100 0000 af23 0001 0005 ffff 001c 0f56 0f3a
0000120 0f3a 0000 0005 0fa2 0f8f 0f79 0f68 0f56
0000140 0000 0000 0000 0000 0000 0000 0000 0000
*
0007620 0000 0000 0000 0000 0000 2c01 0302 c106
0007640 0341 6c69 0777 ba09 1701 0101 2c01 0302
0007660 c105 0256 4207 77bf 020e 0101 012c 0103
0007700 02c1 0407 5375 6e69 7468 6107 77c3 0b0a
0007720 0101 012c 0103 02c1 0304 416e 6479 0777
0007740 be06 0101 0101 2c01 0302 c102 0752 6963
0007760 6861 7264 0777 be05 0c01 0101 af23 0601
0010000 0002 0000 0000 13c6 0000 0000 0000 0101
0010020 0000 0000 0000 0000 0000 0000 0000 0000
*
0017760 0000 0000 0000 0000 0000 0000 0000 0001
0020000 1702 0000 0040 13c7 0005 481b 0000 0300
0020020 0000 0000 0000 0000 0000 0000 0000 0000
0020040 0000 0000 0000 0004 0000 0014 0820 0000
0020060 0000 0003 0000 0009 0000 000a 0040 13f8
0020100 0000 0000 0000 0003 0000 0000 0000 0013
0020120 0000 0000 0000 0000 0000 0000 0000 0004
```

```
0020140 0000 0000 0000 0cc1 4000 0000 0040 13c9
0020160 0000 0002 0040 13d5 0000 0003 0040 13d8
0020200 0000 0005 0040 13ef 0000 000a 0000 0000
0020220 0000 0000 0000 0000 0000 0000 0000 0000
*
0024060 0000 0000 0002 0000 0000 0fe8 0000 0001
0024100 0000 0000 0000 0002 0300 0000 0000 0000
0024120 0000 0000 0000 0000 0000 0000 0000 0000
*
0027760 0000 0000 0000 0000 0000 0000 481b 1703
0030000 1902 0000 0040 13c8 0005 481b 0000 0100
0030020 0000 0000 0000 0000 0000 0000 0000 0000
0030040 0000 0000 feff ffff ffff ffff ffff ffff
0030060 ffff ffff ffff ffff ffff ffff ffff ffff
*
0037760 ffff ffff ffff ffff ffff ffff 481b 1901
0040000 1a02 0000 0040 13c9 0005 481b 0000 0200
0040020 0000 0000 0000 0012 0000 0000 0000 0000
0040040 0000 0000 0000 0000 0000 0000 ffff ffff
0040060 ffff ffff ffff ffff ffff ffff ffff ffff
*
0047760 ffff ffff ffff ffff ffff ffff 481b 1a02
0050000
```

2. Select the ROWID from BLOCK_LAB and dump the Oracle8 formatted block based on the ROWID.

```
SQL> select rowid, id, name, hire_dt from block_lab;

ROWID                  ID NAME            HIRE_DT
------------------ ---- --------------- -----------
AAAAwtAABAAABPFAAA    1 Richard         12-MAY-1990
AAAAwtAABAAABPFAAB    2 Andy            01-JUN-1990
AAAAwtAABAAABPFAAC    3 Sunitha         10-NOV-1995
AAAAwtAABAAABPFAAD    4 VB              14-FEB-1991
AAAAwtAABAAABPFAAE    5 Ali             23-SEP-1986

SQL> select dbms_rowid.rowid_block_number('AAAAwtAABAAABPFAAA') from dual;

DBMS_ROWID
----------
      5061

SQL> select dbms_rowid.rowid_relative_nfno('AAAAwtAABAAABPFAAA') from dual;

DBMS_ROWID
----------
         1

SQL> alter system dump datafile 1 block 5061;

Dump file /u01/app/oracle/admin/816/udump/816_ora_21094.trc
Oracle8 Enterprise Edition Release 8.0.3.0.0 - Production
With the Partitioning and Objects options
PL/SQL Release 8.0.3.0.0 - Production
ORACLE_HOME = /u01/app/oracle/product/8.0.3
System name:     SunOS
Node name: ccd-orsun3-le0
Release:   5.5.1
Version:   Generic
Machine:   sun4m
Instance name: 816
Redo thread mounted by this instance: 1
Oracle process number: 10
Unix process pid: 21094, image: oracle816

Mon Sep 22 23:45:46 1997
*** SESSION ID:(9.707) 1997.09.22.23.45.46.000
Start dump data blocks tsn: 0 file#: 1 minblk 5061 maxblk 5061
buffer tsn: 0 rdba: 0x004013c5 (1/5061)
scn:0x0000.0000af23 seq:0x01 flg:0x02 tail:0xaf230601
```

```
       frmt:0x02 chkval:0x0000 type:0x06=trans data

Block header dump: rdba: 0x004013c5
 Object id on Block? Y
 seg/obj: 0xc2d  csc: 0x00.af22  itc: 1  flg: O  typ: 1 - DATA
     fsl: 0  fnx: 0x0 ver: 0x01

 Itl           Xid                  Uba         Flag  Lck        Scn/Fsc
0x01   0x0004.02a.00000038  0x00c0134b.0023.1e  --U-    5  fsc 0x0000.0000af23

data_block_dump
===============
tsiz: 0xfb8
hsiz: 0x1c
pbl: 0x00ff8f44
bdba: 0x004013c5
flag=---------
ntab=1
nrow=5
frre=-1
fsbo=0x1c
fseo=0xf56
avsp=0xf3a
tosp=0xf3a
0xe:pti[0] nrow=5 offs=0
0x12:pri[0]      offs=0xfa2
0x14:pri[1]      offs=0xf8f
0x16:pri[2]      offs=0xf79
0x18:pri[3]      offs=0xf68
0x1a:pri[4]      offs=0xf56
block_row_dump:
tab 0, row 0, @0xfa2
tl: 22 fb: --H-FL-- lb: 0x1 cc: 3
col  0: [ 2]  c1 02
col  1: [ 7]  52 69 63 68 61 72 64
col  2: [ 7]  77 be 05 0c 01 01 01
tab 0, row 1, @0xf8f
tl: 19 fb: --H-FL-- lb: 0x1 cc: 3
col  0: [ 2]  c1 03
col  1: [ 4]  41 6e 64 79
col  2: [ 7]  77 be 06 01 01 01 01
tab 0, row 2, @0xf79
tl: 22 fb: --H-FL-- lb: 0x1 cc: 3
col  0: [ 2]  c1 04
col  1: [ 7]  53 75 6e 69 74 68 61
col  2: [ 7]  77 c3 0b 0a 01 01 01
tab 0, row 3, @0xf68
tl: 17 fb: --H-FL-- lb: 0x1 cc: 3
col  0: [ 2]  c1 05
col  1: [ 2]  56 42
col  2: [ 7]  77 bf 02 0e 01 01 01
tab 0, row 4, @0xf56
tl: 18 fb: --H-FL-- lb: 0x1 cc: 3
col  0: [ 2]  c1 06
col  1: [ 3]  41 6c 69
col  2: [ 7]  77 ba 09 17 01 01 01
end_of_block_dump
End dump data blocks tsn: 0 file#: 1 minblk 5061 maxblk 5061
```

3. Dump the Oracle8 formatted block of the BLOCK_LAB segment header.

```
SQL> select header_file, header_block from dba_segments
  2  where  segment_name = 'BLOCK_LAB';

HEADER_FILE HEADER_BLOCK
----------- ------------
          1         5060

SQL> alter system dump datafile 1 block 5060;

Dump file /u01/app/oracle/admin/816/udump/816_ora_21149.trc
Oracle8 Enterprise Edition Release 8.0.3.0.0 - Production
With the Partitioning and Objects options
PL/SQL Release 8.0.3.0.0 - Production
ORACLE_HOME = /u01/app/oracle/product/8.0.3
System name:     SunOS
Node name: ccd-orsun3-le0
Release:   5.5.1
Version:   Generic
Machine:   sun4m
Instance name: 816
Redo thread mounted by this instance: 1
Oracle process number: 10
Unix process pid: 21149, image: oracle816

Mon Sep 22 23:49:28 1997
*** SESSION ID:(9.709) 1997.09.22.23.49.28.000
Start dump data blocks tsn: 0 file#: 1 minblk 5060 maxblk 5060
buffer tsn: 0 rdba: 0x004013c4 (1/5060)
scn:0x0000.0000af22 seq:0x01 flg:0x00 tail:0xaf221001
       frmt:0x02 chkval:0x0000 type:0x10=DATA SEGMENT HEADER - UNLIMITED

  Extent Control Header
  -----------------------------------------------------------------
  Extent Header:: spare1: 0      tsn: 0      #extents: 1    #blocks: 2
                  last map rdba: 0x00000000  #maps: 0     offset: 2080
      Highwater:: rdba: 0x004013c6  ext#: 0     blk#: 1    ext size: 2
  #blocks in seg. hdr's freelists: 1
  #blocks below: 1
  mapblk rdba: 0x00000000  offset: 0
                   Unlocked
     Map Header:: next rdba: 0x00000000  #extents: 1  obj#: 3117 flag: 0x40000000

  Extent Map
  -----------------------------------------------------------------
  rdba: 0x004013c5  length: 2

  nfl = 1, nfb = 1 typ = 1 nxf = 0
  SEG LST:: flg: USED   lhd: 0x004013c5 ltl: 0x004013c5

End dump data blocks tsn: 0 file#: 1 minblk 5060 maxblk 5060
```

Practice Solution 7: Block Corruption, Diagnostics, and Recovery

The following practice attempts to illustrate an in-memory block corruption. Follow the steps to introduce the corruption. Look for any errors produced and inspect the alert.log and trace files. Research the errors on tao and formulate steps to correct the error.

Perform the following steps:

1. Start SQL*Plus and connect as internal.
2. Create a table called blockcor, insert some values and commit.

```
SQL> create table blockcor (COLUMN1 number);
SQL> insert into blockcor values (1234);
SQL> commit;
```

3. Find the object number for BLOCKCOR and use it to find the block address.

```
SQL> select * from blockcor;     /* load table data into the cache */
SQL> select obj# from obj$
  2  where  name = 'BLOCKCOR';

   OBJ#
-------
  20704

SQL> select class,ba from x$bh
  2  where  obj = 20704;

CLASS BA
----- --------
    1 82070000
    4 82072000
```

   This query returns two rows. Use the row having a CLASS value of 1 as this is a data block. Type 4 refers to a segment header.

4. Corrupt the block in memory using oradebug.

```
SQL> oradebug poke 0x82070006 1 0
SQL> oradebug poke 0x82070007 1 0
```

   ADDRESS1 is the block address + 6 and *ADDRESS2* is the block address + 7. This corrupts bytes 6 and 7 in the block's cache layer. You should remember from lesson 6 that these bytes represent the two low order bytes of the RDBA of a block. The 1 and 0 at the end of the poke command represent the number of bytes to write and the value to write, respectively.

5. Insert some more values into BLOCKCOR.

```
SQL> insert into blockcor values (5678);
```

6. Observe any errors. Determine what has happened and how to repair it.

   a. After the insert you should see the following:

```
SQL> insert into blockcor values (5678);
ORA-00600: internal error code,
arguments: [2032],[4216626],[2],[4194304],[6],[1],[2],[1]
```

   b. Inspecting the alert.log you find:

```
Bad header found during preparing block for write
Data in bad block - type:6. format:2. rdba:0x00400000
last change scn:0x0000.0004b313 seq:0x1 flg:0x02
consistancy value in tail 0xb3130601
```

```
check value in block header: 0x0, check value not calculated
spare1:0x0, spare2:0x0, spare2:0x0
Mon Dec 13 04:46:51 1999
Errors in file /u01/app/oracle/admin/TEAMM/bdump/teamm_dbw0_29386.trc:
ORA-00600: internal error code,
arguments: [kcbzpb_1], [4216626], [4], [1], [], [], [], []
DBW0: terminating instance due to error 600
Instance terminated by DBW0, pid = 29386
```

c. Research kcbzpb_1 on tao.

```
$ find600 kcbzpb_1
Searching for OERI(kcbzpb_1) or OERINM("kcbzpb_1")
File:   /export/home/ssupport/815/rdbms/src/server/cache/kcbz.c
Line #: 4014
        ASSERTNM3(0, OERINM("kcbzpb_1"), d, kind, chk);
```

d. View kcbz.z using cscope. Searching for kcbzpb_1, you find:

```
/* Call kcbh to prepare this block for write. If it does not like this
 * block then we signal an internal error. The only way it should be bad
  * is if a stray store into memory destroyed the header or tail */
   kind = kcbhpbw(p, kcbbkl, d, chk);
   if (kind != KCBH_GOOD)
   {
      /* Block got corrupted in memory */
      kcbzrc(p, afn, d, kind, "preparing block for write");
      ASSERTNM3(0, OERINM("kcbzpb_1"), d, kind, chk);
   }
}
```

You can now verify that an in-memory corruption affecting the block cache layer has occured. Since this corruption is in memory, you can correct the corruption by bouncing the instance. However, you should also find the process that corrupted the memory location if this were an actual customer case.

Practice Solution 8: Rollback Segment Corruption

Start up the database used for this lesson. This instance has been purposely corrupted for the purpose of this excercise. Your instructor will provide the necessary information. Identify what is wrong with the database, take the necessary actions and bring the database back up.

Perform the following steps:

1. Start up the database.

```
SQL> startup pfile=initL7RBS1A.ora
ORACLE instance started.
Total System Global Area                          4750864 bytes
Fixed Size                                          48656 bytes
Variable Size                                     4210688 bytes
Database Buffers                                   409600 bytes
Redo Buffers                                        81920 bytes
Database mounted.
ORA-01578: ORACLE data block corrupted (file # 1, block # 3815)
ORA-01110: data file 1: '/u05/home/dsi/dsi301/L7RBS1A/systemL7RBS1A01.dbf'
```

   Note the ORA-1578 returned at startup. Block 3815 in file 1 is corrupted. Looking into the alert.log we find:

```
Corrupt block relative dba: 0x00400ee7 file=1. blocknum=3815.
Fractured block found during buffer read
Data in bad block - type:14. format:2. rdba:0x00400ee7
last change scn:0x0000.0000ffff seq:0x1 flg:0x00
consistancy value in tail 0x23510e01
check value in block header: 0x0, check value not calculated
spare1:0x0, spare2:0x0, spare2:0x0
```

   The information here shows that this block is a type 14. We know that this is not a data block. From lesson 2 you know that block type is defined in k.h. Go to tao and view k.h:

```
* the next few block types are used to distinguish segment headers which are
* set up in the unlimited extents format from ones which are 7.2 compatible
* the new types are used for unlimited extents format.
* also the type for an extent map block

    #define K_BTUUNH 14                          /* unlimited undo segment header
*/
```

   So you know that block 3815 is an undo header block. Which rollback segment contains this block? You could bring the database up using the *offline_rollback_segments* parameter if we knew which segment the bad block was in. You cannot query dba_segments, so start with R01 and work up the list. Editing our init.ora:

```
sequence_cache_hash_buckets = 10 # SMALL
max_dump_file_size = 10240       # limit trace file size to 5 Meg each
global_names = FALSE
_offline_rollback_segments = (r01)
```

2. Restart the database.

```
SQL> startup pfile=initL7RBS1A.ora
ORACLE instance started.
Total System Global Area                  4750864 bytes
Fixed Size                                  48656 bytes
Variable Size                             4210688 bytes
Database Buffers                           409600 bytes
Redo Buffers                                81920 bytes
Database mounted.
Database opened.
```

3. So the bad block was in R01. Now the database is available and recovery can be performed in accordance with the guidelines found in lesson 8.

Practice Solution 9: Data Salvage Using DUL

1. A customer file system has suffered from a failure resulting in unrecoverable loss of all files in the database. The database consists of four user data files: two storing table data, and two storing index data. The customer's backup strategy is flawed. The system tablespace files and user data files are backed up on different days. Furthermore, the database is open when the backups are taken, and the database is not in archivelog mode.

   The available data files are:

```
Rfile#  File#  File Name
     0      1  /u05/home/dsi/dsi301/TEAMB/systemTEAMB01.dbf
     1      2  /u05/home/dsi/dsi301/TEAMB/rbsTEAMB01.dbf
     2      3  /u05/home/dsi/dsi301/TEAMB/tempTEAMB.dbf
     3      4  /u05/home/dsi/dsi301/TEAMB/data01TEAMB.dbf
     4      5  /u05/home/dsi/dsi301/TEAMB/data02TEAMB.dbf
     5      6  /u05/home/dsi/dsi301/TEAMB/index01TEAMB.dbf
     6      7  /u05/home/dsi/dsi301/TEAMB/index02TEAMB.dbf
```

   The customer needs to unload the data from two tables, EMP and DEPT. Both tables belong to user PRODUSER. All the customer knows is that these two tables are stored in user tablespaces.

   Configure DUL and unload the two tables from the database into Export format files. The following default init.dul file is provided, but may need to be modified to reflect the hardware, operating system, and database configuration:

```
# sample init.dul configuration parameters
# these must be big enough for the database in question
# the cache must hold all entries from the dollar tables.
dc_columns = 200000
dc_tables = 10000
dc_objects = 10000
dc_segments=10000
dc_users = 400

# OS specific parameters
osd_big_endian_flag = false
osd_dba_file_bits = 6
osd_c_struct_alignment = 32
osd_file_leader_size = 1
osd_word_size = 32

# database parameters
db_block_size = 2048

export_mode=true
```

To modify the init.dul, see the following platform-specific parameters:

PORT	PLATFORM	ENDIAN	FBITS67	FBITS8	CSTR	LEAD	WORD
1	Digital VAX OpenVMS	FALSE	8	-	0	0	32
2	HP Series 9000 HP-UX	FALSE	8	-	0	0	32
21	Novell Netware	FALSE	8	8	0	1	32
22	MS DOS	FALSE	8	-	16	512	16
168	Silicon Graphics UNIX	TRUE	6	?	32	1	32
87	Digital Unix	FALSE	6	10	32	1	32
89	Alpha Vms	FALSE	8	10	32	0	32
198	Sequent Dynix/PTX	FALSE	6	?	32	1	32
319	IBM RS 600 AIX	TRUE	8	?	32	1	32
453	Sun Sparc Solaris	TRUE	6	10	32	1	32
912	MS Windows NT Intel	FALSE	8	10	32	1	32

Explanation of the values:

Title	Description
ENDIAN	OSD_BIG_ENDIAN_FLAG
FBITS67	OSD_DBA_FILE_BITS (Oracle version6/Oracle7)
FBITS8	OSD_DBA_FILE_BITS (Oracle8)
CSTR	OSD_C_STRUCT_ALIGNMENT
LEAD	OSD_FILE_LEADER_SIZE
WORD	OSD_WORD_SIZE

The dictv8.dll file is also available.

a. Configure control.dul and init.dul.

```
# control.dul:
0     1 /u05/home/dsi/dsi301/TEAMB/systemTEAMB01.dbf
1     2 /u05/home/dsi/dsi301/TEAMB/rbsTEAMB01.dbf
2     3 /u05/home/dsi/dsi301/TEAMB/tempTEAMB.dbf
3     4 /u05/home/dsi/dsi301/TEAMB/data01TEAMB.dbf
4     5 /u05/home/dsi/dsi301/TEAMB/data02TEAMB.dbf
5     6 /u05/home/dsi/dsi301/TEAMB/index01TEAMB.dbf
6     7 /u05/home/dsi/dsi301/TEAMB/index02TEAMB.dbf

#init.dul:
dc_columns = 200000
dc_tables = 10000
dc_objects = 10000
dc_segments=10000
dc_users = 400
control_file= /u05/home/dsi/bin/control.dul

# OS specific parameters
osd_big_endian_flag = true
osd_dba_file_bits = 10
osd_c_struct_alignment = 32
osd_file_leader_size = 1
osd_word_size = 32

# database parameters
db_block_size = 2048
```

```
compatible=8.0
export_mode=true
```

b. Unload the dictionary tables

```
$ dul dictv8.ddl

Data UnLoader: Release 8.0.5.4.0 - Internal Use Only - on Wed Dec  1 07:42:33 1999
Copyright (c) 1994/1999 Bernard van Duijnen All rights reserved.

Parameter altered
Parameter altered
Parameter altered
Parameter altered

. unloading table                      OBJ$    1516 rows unloaded
. unloading table                      TAB$     113 rows unloaded
. unloading table                      COL$    8227 rows unloaded
. unloading table                     USER$      15 rows unloaded
. unloading table                  TABPART$       0 rows unloaded
. unloading table                      IND$     118 rows unloaded
. unloading table                     ICOL$     225 rows unloaded
. unloading table                      LOB$       4 rows unloaded

Life is DUL without it
```

c. Unload tables EMP and DEPT.

```
$ dul
Data UnLoader: Release 8.0.5.4.0 - Internal Use Only - on Wed Dec  1 07:45:11 1999
Copyright (c) 1994/1999 Bernard van Duijnen All rights reserved.

DUL: Warning: Recreating file "dul.log"
Loaded 15 entries from USER.dat
Loaded 1516 entries from OBJ.dat
Loaded 113 entries from TAB.dat
Loaded 8227 entries from COL.dat
Loaded 0 entries from TABPART.dat
Loaded 118 entries from IND.dat
Loaded 4 entries from LOB.dat
Loaded 225 entries from ICOL.dat

DUL> unload table produser.emp;
. unloading table                       EMP      14 rows unloaded
DUL> unload table produser.dept;
. unloading table                      DEPT       4 rows unloaded
```

The DUL commands creates the following Export files:

- PRODUSER_EMP.dmp
- PRODUSER_DEPT.dmp