

ROLLBACK SEGMENT AND UNDO INTERNALS

Daniel W. Fink

UNDO INTERNALS

There are two areas of focus for this paper. The first section is a discussion of the block, rollback segment/undo entry and transaction architecture of Oracle8. Due to the relatively static nature of rollback segments over the past few releases, I have chosen not to discuss how they operate or the common problems. There is a wealth of documentation, notes, papers, etc. available to the industrious and web-enabled user.

The second section examines Oracle9 automatic undo management. This has been widely publicized in marketing materials and is a default feature in the new versions. However, there is little actual documentation on the mechanics and management of automatic undo from a DBA's perspective. This paper will shed light on the actual operations of automatic undo management.

One important concept to keep in mind is that much of what we will review is related to undoing actions. As such, there is a great deal of 'reverse' terminology and presentation. For example, undo is read in reverse order...from the last entry to the first. Instead of reading the next entry, we read the previous. Another example is that undo entries are the opposite of the transaction. An insert containing a large amount of data is represented by a minimal amount of undo. .sense makes that hope I

METHODOLOGY

I utilized the scientific method of testing a hypothesis within a controlled environment. In several cases, the testing contradicted my hypothesis. For example, I assumed, as many DBAs do, that a read consistent query used the data in the rollback segment as its source. After testing, I found this to be an incorrect hypothesis. How does a read consistent query operate? Have patience faithful reader, for all shall be revealed in due time.

In some regards, Oracle is a 'black box' in that most users, DBAs, developers do not have access to the design and source code. However, the effect of a single change can be seen. If we know that state of an object (or objects) before the change and the state after the change and no other actions would have been involved, we can infer the operation from it's impacts. For example, if I record the state of the data block and all undo blocks before a transaction and the state of those objects after the transaction and I know that there have been no other transactions, I can infer that all changes made to the blocks are the result of the transaction and only the transaction. With a little knowledge about the architecture of a transaction, I can identify the actual operations.

CAVEATS

This paper and subsequent presentation contain undocumented or lightly documented features and commands. While the author has tested the commands on some platforms and releases, not all systems have been involved. Do not use these commands on production or other valuable databases without a solid understanding of their function and experience in their use. Each release of Oracle may cause the commands to change in behavior or fail completely. Not all commands and permutations have been tested.

Additionally, over the course of testing, certain anomalies have been discovered for which there is no verifiable explanation. Some are mere puzzles (why does the rollback segment skip the 2nd block of the first extent initially?) while others are contrary to what Oracle has published (rollback segment blocks occasionally containing undo from current and previous transactions). As always, constructive, verifiable explanations are appreciated. Testing will continue past the submission of this paper and additional information will be available in the presentation, which can be downloaded.

TOOLS TO EXTRACT INFORMATION

To extract data from the database we may use a simple select statement. However, this only retrieves the actual data or a read consistent copy and does not provide any structural information for the storage of the data. It is possible to use a block level

viewer to examine the actual bytes of the data files. However, this would be rather tedious and not all that enlightening, unless we know exactly which byte means what. Fortunately, Oracle provides a method for extracting a formatted dump of the data blocks using the `alter system` command in the following example

```
alter system dump datafile 7 block 1569;
```

These commands will generate a formatted block dump and write the information to a trace file. The block dump will contain both structural data and a terse description. For example, in the following output some fields are self-documenting, while others are very cryptic.

```
Start dump data blocks tsn: 7 file#: 7 minblk 1569 maxblk 1569
buffer tsn: 7 rdba: 0x01c00621 (7/1569)
scn: 0x0000.00047616 seq: 0x02 flg: 0x00 tail: 0x76161002
frmt: 0x02 chkval: 0x0000 type: 0x10=DATA SEGMENT HEADER - UNLIMITED
Extent Control Header
-----
Extent Header:: spare1: 0      spare2: 0      #extents: 1      #blocks: 15
                last map 0x00000000 #maps: 0      offset: 2080
                Highwater:: 0x01c00623 ext#: 0      blk#: 1      ext size: 15
#blocks in seg. hdr's freelists: 0
```

WHERE EXACTLY IS THE DATA?

In order to dump the data blocks, the file and blocks for the particular extent are needed. This information is available with the `dba_extents` view only. The `user_extents` view does not contain the `block_id` and cannot be used to determine the block address values. Remember that the first block of a data segment is the segment header and does not contain actual data.

```
select segment_name, file_id, block_id, blocks
from dba_extents
where owner = 'DWFINK'
      and segment_name = 'TEST1';
```

```
SEGMENT_NAME FILE_ID BLOCK_ID BLOCKS
-----
TEST1          4         5         4
TEST1          4         9         4
TEST1          4        13         4
```

One of the most powerful features of the `alter system dump datafile # block #` commands is that you can actually dump multiple blocks at a time! This is not even documented in the 'undocumented' documents. A quick comment from a fellow dba, a hard look at a trace file and a wild guess revealed the ability to execute the following command:

```
alter system dump datafile 4 block min 5 block max 16;
```

This command dumped the entire contents of the table. In previous efforts, this was accomplished by dumping the first block, changing the statement to dump the second block, changing the statement to dump the third block...see where I'm going with this? In order to determine the max block, you take the `block_id + blocks - 1`. In the above example, the last extent occupies 4 blocks (13, 14, 15, 16) beginning at block 13. $13 + 4 - 1 = 16$.

WHERE EXACTLY IS THE UNDO?

Undo is only generated by a transaction and is only relevant in the context of that transaction. There are two steps to identifying where undo related to a specific transaction is saved. First is to identify the session performing the transaction in the `v$session` view. Until the transaction is started, the `TADDR` column is null and there are no entries in the `v$transaction` view.

```
SADDR      TADDR      USERNAME      OSUSER      TYPE
-----
7A9193B4           SYS           DANIELWFINK\dwfink  USER
7A919CCC 7A69F5F0 DWFINK       DANIELWFINK\dwfink  USER
```

Second, locate the related record(s) in `v$transaction`.

```
SES_ADDR  ADDR      START_UBAFIL  START_UBABLK  UBAFIL  UBABLK
```

```
-----
7A919CCC 7A69F5F0          2          37095          2          37095
```

There are two components to the undo chain that are represented in this statement, the head and the tail. The 'tail' or current block of the undo chain are represented by the ubafil and ubablk columns. The head or original block of the undo chain is represented by the start_ubafil and start_ubablk fields. UBA is an acronym for Undo Block Address, FIL is file and BLK is block.

It is important to note that the blocks in the undo chain are not guaranteed to be adjacent to each other. Because an extent may be shared by transactions, a block may be inserted into the chain that is not the next physical block in the extent. The only sure method to determine which blocks contain undo from a particular transaction is to dump the blocks and follow the undo chain. However, if the difference in v\$transaction.start_ubablk and v\$transaction.ubablk is equal to v\$transaction.used_ublk, there are no 'holes' in the blocks in the undo chain.

ORACLE8 ROLLBACK SEGMENT GENERAL CONCEPTS

WHAT IS A ROLLBACK SEGMENT

OFFICIAL DESCRIPTIONS

"A rollback segment records the old values of data that was changed by each transaction (whether or not committed). Rollback segments are used to provide read consistency, to roll back transactions, and to recover the database." (Oracle8i Concepts)

"They [rollback segments] capture the "before image" of data as it existed prior to the start of a transaction... A rollback segment entry is the set of before-image data blocks that contain rows that are modified by a transaction." (Oracle8 DBA Handbook)

" The rollback segment does not store the whole data block – only the before image of the row or rows that were modified. Information in the rollback segment consists of several rollback entries called *undo*. For example, if a row is inserted into a table, the undo created by that transaction would include the ROWID of that row, among other information. This is because the undo operation of an insert is a delete, and all you need to delete a row is the ROWID. If a delete operation is performed on a table, the complete row will be part of the undo. For update transactions, we store the old value of the updated columns. If the transaction modifies an index as well, then the old index keys will also be stored as part of the undo. Rollback segments guarantee that the undo information is kept for the life of a transaction." (Oracle8 Backup & Recovery Handbook)

POPULAR VIEW

- ?? Copies of blocks changed by transactions, both uncommitted and committed
- ?? Copies of database rows that are being updated, perhaps at the block level
- ?? Snapshot of transactions that have occurred since the last commit
- ?? Logical information stored in memory and not physically stored in a file
- ?? System undo data allowing non-committed transactions to be rolled back
- ?? Before image copy of data in a transaction

DEFINED

Information necessary to reverse a transaction or to reconstruct a read consistent view. The contents of an undo entry include: data column(s) address, transaction operation and old value(s) of the data.

TRANSACTIONS AND ROLLBACK SEGMENT BLOCKS

A rollback segment is composed of two or more blocks. In Oracle8, 8 blocks is the minimum rollback segment size, regardless of minimum extent and optimal settings. The first block of the segment is reserved for the undo header with the first transaction for a new rollback segment starting in the third block. Initially, the second block of a rollback segment is not used. During normal usage, the rollback segment will extend into this block.

Each block in a rollback segment may be assigned to one and only one transaction. If the transaction undo information will not fit within a single block, a second block is assigned to the transaction. This second block becomes the new 'head' block for the undo chain. Within the second block, the value of the previous block address is retained. When this occurs, the transaction table is updated to point to the current 'head' block.

Each rollback segment block may contain entries that point to different data/index blocks. Part of the undo entry is the datablock address of both the data/index segment header and block that stores the previous values.

INTERNALS OF ROLLBACK SEGMENTS

Each rollback segment is composed of two distinct components, the undo header and undo entry. The first block of a rollback segment is reserved for the undo header. When a rollback segment is created, only the header is created. Initially, the remaining blocks are allocated to the segment, but no other structures are created.

UNDO HEADER

TRANSACTION TABLE

One of the main features in the rollback segment header is the Transaction Table. The transaction table is made up of a fixed number of 'slots' or entries. The number of slots is block size dependent. Each of these slots is assigned to a transaction and contains information about that action. The slots are initially used in order, but are reused in a round robin fashion with one exception. A slot referencing an uncommitted transaction will not be reused. It is possible to fill up all slots with active transactions. If this occurs, the transaction waits until a slot becomes available.

Key components are the **state**, **scn**, and **dba**. A state of 10 indicates that the transaction is not committed. The SCN represents the System Change Number for uncommitted transactions and System Commit Number for committed transactions. The dba is the data block address of the *last* data block containing undo entries for the transaction. (Please note that parent txid has been removed for display purposes)

```

TRN TBL::
index  state  cflags  wrap#    uel          scn          dba          nub
-----
0x00   10    0xc0   0x0010  0x0000  0x0000.00075fec  0x02000004  0x00000001
0x01    9    0x00   0x000f  0x0002  0x0000.00075f77  0x00000000  0x00000000

```

UNDO ENTRIES

The undo entry for a transaction contains certain transaction level information and the contents of the column or columns before the change occurred. It does not contain a before image of the block or row.

Undo entries are chained together to form a single transaction undo action. A single undo chain contains the necessary information to transform a block from one coherent state to another coherent state. If the transaction contains updates to multiple blocks, each block has its own chain, though the chain may share space within a rollback segment block with other chains and a single chain may span multiple rollback segment blocks. The undo entry contains either a pointer to the previous link in the chain or an indicator that this is the end of the chain or an indicator that this is the start of the chain. If the undo entry is the tail of the chain, it may contain information about a previous transaction that has modified the block.

UPDATE A ROW WITH A SINGLE COLUMN

The undo entry for a row is the previous value of the column and the operation necessary to undo the action. The opposite of an update is an update to the previous value. The example below is an undo entry for a single column update. Note that the entire row is not copied, just the old column values and sufficient information to undo the action.

```

*-----*
* Rec #0x3  slt: 0x00  objn: 22101(0x00005655)  objd: 22101  tblspc: 6(0x00000006)
*          Layer: 11 (Row)  opc: 1  rci 0x02
Undo type: Regular undo  Last buffer split: No
Temp Object: No
Tablespace Undo: No
rdba: 0x00000000

```

```
*-----
KDO undo record:
KTB Redo
op: 0x02 ver: 0x01
op: C uba: 0x02000004.0005.02
KDO Op code: URP xtype: XA bdba: 0x01c00007 hdba: 0x01c00006
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 1(0x1) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 1 size: 0
col 5: [ 3] c2 1d 33
```

UPDATE A ROW WITH MULTIPLE COLUMNS

The undo entry for a row is the previous values of the columns.

```
*-----
* Rec #0xb slt: 0x01 objn: 22101(0x00005655) objd: 22101 tblspc: 6(0x00000006)
* Layer: 11 (Row) opc: 1 rci 0x0a
Undo type: Regular undo Last buffer split: No
Temp Object: No
Tablespace Undo: No
rdba: 0x00000000
*-----
KDO undo record:
KTB Redo
op: 0x02 ver: 0x01
op: C uba: 0x02000004.0005.0a
KDO Op code: URP xtype: XA bdba: 0x01c00007 hdba: 0x01c00006
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 1(0x1) flag: 0x2c lock: 0 ckix: 0
ncol: 8 nnew: 2 size: -1
col 5: [ 3] c2 20 24
col 7: [ 2] c1 1f
```

INSERT A ROW

The undo entry for a row is the row information and the operation necessary to undo the action. The opposite of an insert is a delete.

```
*-----
* Rec #0x12 slt: 0x02 objn: 22101(0x00005655) objd: 22101 tblspc: 6(0x00000006)
* Layer: 11 (Row) opc: 1 rci 0x00
Undo type: Regular undo Begin trans Last buffer split: No
Temp Object: No
Tablespace Undo: No
rdba: 0x00000000
*-----
uba: 0x02000004.0005.0a ctl max scn: 0x0000.00075f77 prv tx scn: 0x0000.00075f77
KDO undo record:
KTB Redo
op: 0x04 ver: 0x01
op: L itl: scn: 0x0003.001.00000010 uba: 0x02000004.0005.11
flg: C--- lkc: 0 scn: 0x0000.00075ff0
KDO Op code: DRP xtype: XA bdba: 0x01c00007 hdba: 0x01c00006
itli: 1 ispac: 0 maxfr: 4863
tabn: 0 slot: 8(0x8)
```

DELETE A ROW

The undo entry for a row is the row information, all column values and the operation necessary to undo the action. The opposite of a delete is an insert.

```
*-----
* Rec #0x13  slt: 0x03  objn: 22101(0x00005655)  objd: 22101  tblspc: 6(0x00000006)
*          Layer:  11 (Row)  opc: 1  rci 0x00
Undo type:  Regular undo  Begin trans  Last buffer split:  No
Temp Object:  No
Tablespace Undo:  No
rdba: 0x00000000
*-----
uba: 0x02000004.0005.12  ctl max scn: 0x0000.00075f77  prv tx scn: 0x0000.00075f77
KDO undo record:
KTB Redo
op: 0x04  ver: 0x01
op: L  itl: scn:  0x0003.002.00000010  uba: 0x02000004.0005.12
          flg: C---  lkc:  0  scn: 0x0000.00075ff2
KDO Op code: IRP  xtype: XA  bdba: 0x01c00007  hdba: 0x01c00006
itli: 1  ispac: 0  maxfr: 4863
tabn: 0  slot: 8(0x8)  size/delt: 33
fb: --H-FL--  lb: 0x0  cc: 8
null: ---N--N-
col  0: [ 3]  c2 64 64
col  1: [ 4]  54 45 53 54
col  2: [ 4]  54 45 53 54
col  3: *NULL*
col  4: [ 7]  78 65 01 03 0f 12 35
col  5: [ 2]  c2 0b
col  6: *NULL*
col  7: [ 2]  c1 0b
```

DATA BLOCK AND ROW STRUCTURE

For each table, there are two types of segment blocks. The first is the segment header, which contains information related to the segment as a whole, primarily concerned with space management. The second type is the data/index block (for discussion purposes, we will refer to this as a data block for the article). Each data block contains a header, footer and row information. Each row contains header information and the actual values of the columns.

BLOCK HEADER

In each block header is a structure called the Interested Transaction List (ITL). The number of initial and maximum entries in the ITL is determined by the `initrans` and `maxtrans` parameters. (For display purposes, all ITL components are not shown).

```
Itl          Xid          Uba          Flag
0x01  xid: 0x0003.000.00000010  uba: 0x02000004.0005.09  ----
```

The `XID` is the pointer to the rollback segment transaction table slot (`undoseg#.slot#.wrap`). The `undoseg#` is the rollback segment, the `slot#` refers to a slot in the transaction table, and `wrap` refers to the number of times that the slot has been used (i.e. incarnation or version) in a transaction.

The `UBA` is the pointer to the tail of the undo chain for the block (`undodatablockaddr.sequence#.record#`).

The `FLAG` indicates the status of transaction.

ROW HEADER

When a row is updated, one byte in the row header, called the lock byte (`lb`), is updated to indicate the transaction's ITL entry. If the value is `0x0`, there is not an ITL entry.

```

tab 0, row 0, @0x1e4c
tl: 44 fb: --H-FL-- lb: 0x1 cc: 8
col 0: [ 3]  c2 4b 64
col 1: [ 5]  41 4c 4c 45 4e
col 2: [ 8]  53 41 4c 45 53 4d 41 4e
col 3: [ 3]  c2 4d 63
col 4: [ 7]  77 b5 02 14 01 01 01
col 5: [ 3]  c2 12 3d
col 6: [ 2]  c2 04
col 7: [ 2]  c1 1f

```

BLOCK CLEANOUT

When a data block is altered, important transaction information is written to the block along with the data changes. When the transaction is committed, the block is not rewritten to reflect the new transaction state. The next transaction, whether a SELECT or INSERT/UPDATE/DELETE, will 'discover' the out of date information and update the block to reflect the committed state of the previous transaction. This is called 'delayed block cleanout.' It is possible that the block will never be touched again and contain transaction information that is days, weeks, months or even years old.

INTERNALS OF THE READ CONSISTENT PROCESS

One of the hallmarks of the Oracle database model is the ability to create a view of the data consistent with the start of the query, regardless of the changes that have been made since the query began. If the data cannot be reconstructed, the query will fail rather than return dirty data. The mechanism to create a read consistent view, especially accounting for multiple transactions to a single piece of data, is rather complex. Unfortunately, this is also a component that is lightly documented. The information contained below is based upon tests and attempts at logical recreation of the process. As such, it may not be entirely correct, but it does explain all the scenarios that I tested, both single and multiple block and transaction changes.

CREATING A READ CONSISTENT VIEW

STEPS

1. Read the Data Block.
2. Read the Row Header.
3. Check the LockByte to determine if there is an ITL entry.
4. Read the ITL to determine the Transaction ID.
5. Read the Transaction Table. If the transaction has been committed and has a System Commit Number less than the query's System Change Number, cleanout the block and move on the next data block (if required) and Step 1.
6. Read the last undo block indicated
7. Compare the block transaction id with the transaction table transaction id. If the Transaction ID in the undo block does not equal the Transaction ID from the Transaction Table, then signal an ORA-1650, Snapshot Too Old.
8. If the txids are identical, clone the data block in memory. Starting with the head undo entry, apply the changes to the cloned block.
9. If the tail undo entry (the last one read) indicates another data block address, read the indicated undo block into memory. Repeat 7 & 8 until the first record does not contain a value for the data block address.
10. When there is no previous data block address, the transaction has been undone.
11. If the undo entry contains
 - a) a pointer to a previous transaction undo block address, read the TxID in the previous transaction undo block header and read the appropriate Transaction Table entry. Return to step 5.
 - b) an ITL record, restore the ITL record to the data block. Return to step 4.

ORACLE 9i AUTOMATIC UNDO MANAGEMENT

Oracle 9i has introduced a new method of managing rollback segments, referred to as automatic undo management. Instead of manually creating rollback segments, an undo tablespace is created, which contains system generated rollback segments. When the `undo_tablespace` parameter is set to the new tablespace, the rollback segments are automatically brought online and are available for use. In testing the new Automatic Undo Management feature, I have found that the basic architecture of undo has remained the same. The main changes are related to space management and transaction allocation.

The focus of this paper is not to rehash what is available in the available documentation or on the support web sites. The focus is to review the actual architecture, highlight undocumented (or poorly documented) features and tasks. Much of the architectural discussions are based upon testing, block dumps and theory. When in doubt, Ockham's Razor (essentially the simplest explanation is the best) was applied. I will also make unsupported conjecture about performance problems. By the time this paper is presented, I plan to have supporting or contradictory data regarding these problems in the actual presentation.

9i UNDO ARCHITECTURE

The basic structure of the undo segment remains the same from 8i to 9i. Blocks, extents, segments, transaction tables, slots, undo entries are substantially unchanged. Internally, several items in the block structure of the undo segment have changed. The segment header now contains a retention table. In testing, I found that this table was populated for the undo tablespace that was created with the database and not for undo tablespaces created and used afterwards. At this point in time, the testing is incomplete so an explanation is not forthcoming. It is interesting to note that the undo segments that had no times in the retention table also had no times in `dba_undo_extents.commit_wtime`. Another new component is the `stmt_num` column in the transaction table. In testing, this column always contained 0x00000000. I am not certain what the future holds for this column.

CREATION OF UNDO TABLESPACE

The undo tablespace is created as a normal tablespace, but the only method for extent management is 'LOCAL AUTOALLOCATE'. Dictionary management and Local – Uniform Size is not supported at this time. Attempting to create the undo tablespace in any other manner results in the "ORA-30024: Invalid specification for CREATE UNDO TABLESPACE" error. Oracle documentation indicates that LOCAL UNIFORM is a valid extent management algorithm for UNDO tablespaces. In practice, the only method of extent allocation that does not return an error is LOCAL AUTOALLOCATE, which is the default.

The tablespace can be created in two ways. First, the `CREATE DATABASE` command can include the undo tablespace clause. This method creates the undo tablespace and associated datafile(s), creates the system managed undo segments and sets the `undo_tablespace` parameter to the value of the undo tablespace. One advantage is that the system starts running in automatic undo management mode and no manual intervention by the DBA is required.

The second method is to issue the `CREATE UNDO TABLESPACE` command when the database is open. This creates a new undo tablespace with undo segments, which are initially offline. They can be brought online for usage by setting the `undo_tablespace` parameter to the new undo tablespace name. This can be done without bouncing the database by issuing the `alter system set undo_tablespace = NEW_UNDO_TS` command. Of course, you can always use the tried and true method of changing the `init.ora` and restarting the instance (unless you are using the `sfile` feature).

In testing, I found that the database used undo segments in a similar fashion, regardless of the method of creation. However, only those undo segments generated with the `CREATE DATABASE` command would populate the retention table and appropriate data dictionary views. Dropping the original undo tablespace and stopping/starting the instance did not have any impact on these data fields. This anomaly is still under investigation.

CREATION OF UNDO SEGMENTS

When the undo tablespace is created, a fixed number of undo segments are created. Based upon testing, each undo tablespace will contain 10 undo segments. These are sized according to the autoallocate algorithm for locally managed tablespaces. The basic algorithm is that the first 16 extents are 64k in size. The subsequent allocation method is the next 63 extents of 1m, the next 120 extents of 8m and all additional extents at 64m.

The first block of the first undo extent is reserved as the header and not indicated in the data dictionary views. As with Oracle 8, the second block of the undo segment is not used initially and may or may not be allocated at a later time. This is one of those anomalies that has no consistent pattern and is therefore hard to explain.

USAGE OF UNDO SEGMENTS

Once the undo tablespace has been created it is available for use by any instance attached to the database. The current undo tablespace is determined by the `undo_tablespace` parameter. Additional undo tablespaces may be created, but will not be used while another undo tablespace is active. In the `dba_rollback_segs` view, these undo segments have a status of 'OFFLINE'.

A new undo tablespace can be activated while the database is running. Any transactions that have allocated space in the old undo tablespace are allowed to gracefully complete. All new transactions are assigned to undo segments in the new tablespace. Once the pending transactions have completed, the undo segments are no longer available for use.

The number of undo segments online for the instance is a function of the `SESSIONS` parameter. The algorithm is roughly 1 undo segment for each 5 sessions. All 10 undo segments are placed online when the sessions parameter is set to 46 or greater. The lower limit was not tested, as the minimum value for the sessions parameter in the test database was 16, which was derived from the minimum number of processes (10) for the database.

9i UNDO ADMINISTRATION

Administration of automatic undo is 'simpler' in 9i. No longer do you need to issue the commands to create or alter rollback segments. The creation of the undo tablespace and the assignment of the undo tablespace to the instance are the only steps required. However, the lack of configuration options are bound to cause performance problems.

AUTOMATIC UNDO PARAMETERS

UNDO_MANAGEMENT

Setting this parameter to `AUTO` invokes the automatic undo management algorithms. In order to allow for transition, `MANUAL undo` management is still available. In this case, the rollback segments are created and managed in the same fashion as Oracle8.

UNDO_TABLESPACE

The undo tablespace that is currently used by the instance is determined by this parameter. Only 1 undo tablespace can be active at the instance level at any given time. This parameter is dynamic and can be set at the system level without restarting the instance. In the Real Application Cluster (RAC) configuration, each instance can have its own undo tablespace active.

One of the 'features' that Oracle is mentioning is the ability to have different undo tablespaces for different processing needs. For example, one undo tablespace could be used for normal transaction processing, one for batch processing, one for large data loads, etc. There are two significant issues that contradict this 'feature'. First, at this time, only the autoallocate algorithm can be used for the undo tablespace. This guarantees that all undo segments, regardless of the tablespace, will have the exact same storage characteristics. The second issue is that only one undo tablespace can be associated with an instance at any given time. The old method of creating a `large_rbs`, onlining it and allocating it for a single transaction are not permissible in the new architecture.

UNDO_RETENTION

This parameter controls the number of seconds that UNDO is retained. In actual practice, this is the minimum amount of time to be retained. Depending upon the activity in the database and the number/size of undo segments, undo from committed transactions may be retained for a longer period of time. It is important to remember that the undo is retained across instance shut downs, which is key to the concept of a flashback query.

Oracle does not guarantee that undo will be retained for any specific amount of time once a transaction is committed. Yup, you read it right. If the system needs space for active undo and there is no free space available, it will overwrite committed undo, even if the amount of time since the commit is less than the `undo_retention` value. This could cause a flashback query to fail. Remember, Oracle never overwrites uncommitted undo.

UNDO_SUPPRESS_ERRORS

If an application contains code relating to rollback segments, this tells the system to ignore the errors that are returned by these calls. For example, issuing the `set transaction use rollback segment` command results in "ORA-30019: Illegal rollback segment operation in Automatic Undo mode" error. If the parameter is set to FALSE, the command appears to succeed and no error is returned.

CREATION OF ROLLBACK SEGMENTS

The rollback segments are automatically created when the undo tablespace is created. These are named using the `_SYSSMU<nn>$` algorithm. Because of the new name, some commands (`alter system dump undo header`) require that the undo segment name be enclosed in double quotes. The segment space allocation is dependent upon the space allocation method of the undo tablespace. The number of segments within a given tablespace is a function of the sessions parameter. Based upon testing, the ratio of sessions to tablespace is roughly 5 to 1 up to a sessions value of 46. In reality, very few databases will be running with such a small value for sessions.

NEW 9i UNDO DICTIONARY AND PERFORMANCE VIEWS

DBA_UNDO_EXTENTS

This is a new view for the undo extents. This view will show only the system managed undo segments, the SYSTEM and any other rollback segments are not included. Most of the columns are similar to the other segment/extent views, with 2 exceptions. First, the `block_id` for each segment's `extent_id` of 0 is off by 1 block. For example, in `dba_rollback_segs`, the `block_id` column contains the first block (segment header) of the first extent (`extent_id` of 0). In `dba_undo_extents`, the `block_id` column contains the second block (skips the segment header) of the first extent.

There is an error in the Oracle Documentation. The STATUS column is not mentioned. This column will contain the values 'ACTIVE' or 'EXPIRED'. According to the view definition, 'UNEXPIRED' is another value, though in testing, this value was never encountered.

V\$UNDOSTAT

This is a new view for Oracle9i. This contains statistical information as to the performance of the undo subsystem for the life of the instance. Although this is oriented towards the statistics related to the new mechanisms, the information will be useful for standard rollback segment monitoring. As with all V\$ views, once the instance is stopped all of the data is lost.

9i UNDO ADMINISTRATION CHALLENGES

There are three areas that may pose challenges to query performance. First, space allocation and management is fixed at the tablespace level. Second, only one undo tablespace can be online at any one time at the instance level. Third, `undo_retention` determination will be guesswork until the actual requirements are known. Essentially, the ability of the DBA to address undo related performance problems is significantly diminished. Will this be a severe problem...only time will tell.

SPACE ALLOCATION AND MANAGEMENT

When an undo tablespace is created, a fixed number (10 in 9.0.1) of rollback segments are created within the tablespace. Since the only extent management algorithm is local autoallocate, each of the segments is composed of 2 extents of 16k each in size. As the segment grows, the size of the extent will increase according to the algorithm. When the segment allocates its 16th extent, the new size is 1024 k (1 meg).

With fixed space allocation, the flexibility to use a few large rollback segments, many small rollback segments or a mix with transaction level allocation is gone. The small size and gradual growth of the segment could cause excessive space management operations and risk for statement failure. Multipurpose databases may encounter both situations.

An additional factor to consider is the balancing act with block size. A larger block means more transaction table slots, so more simultaneous transactions can be active at any given time. However, since only one transaction can use a given block at any time, there may be significant waste of space within the undo segments. This seems to present a choice between undo header contention and undo block contention with the strong possibility of statement failure due to an inability to allocate undo resource.

ONE AND ONLY ONE UNDO TABLESPACE

With only one undo tablespace online, all users will be accessing blocks in the same file or set of files. Physical I/O contention becomes a real possibility. Even with multiple data files for the undo tablespace, the undo segment headers and initial space allocation are in the first data file.

UNDO_RETENTION DETERMINATION

The system now has the ability to retain transaction information for a number of seconds since the transaction was committed. The number of seconds required for the database will initially be a guesstimate based on past experience. The effect of UNDO_RETENTION on the space requirements will be determined based upon the performance of the system. Oracle documentation notes that UNDO_POOL can be set at the resource level to prevent 'runaway' undo usage.

An interesting note recently uncovered during research indicated that the UNDO_RETENTION value is not a guarantee of preservation. If the system needs to reclaim the space for another transaction, it will overwrite old undo entries even if they are not 'expired'.

CONCLUSIONS

The exploration of the internal architecture and mechanics of the undo process has been at once both enlightening and frustrating. Incorrect documentation, contrary results and anomalies have provided ongoing challenges to the determined DBA mind. Oracle's ability to generate read consistent queries is a marvel in complicated simplicity.

The lack of configurability and flexibility in Automatic Undo Management are a cause of concern and impetus for more research for this production DBA. The current level of documentation and research is sparse. As more systems are converted to Oracle9, more information will be forthcoming. As long as there is the option to return to the standby mode of manual undo management, databases can be converted from one method to another to prevent performance degradation.