

**Oracle9i Automatic Undo Management
System Managed Undo**

**Nitin Vengurlekar
Frank Kobylanski
04/01/2001**

I. INTRODUCTION	3
II. REVIEW ROLLBACK SEGMENTS	3
III. INTRODUCTION TO SYSTEM MANAGED UNDO	4
A. HOW IS SYSTEM MANAGED UNDO IMPLEMENTED?	4
<i>Operational SMU</i>	4
B. UNDO SEGMENT STRUCTURES.....	7
<i>Transaction tables</i>	7
C. SPACE MANAGEMENT	9
D. READ CONSISTENCY	11
<i>Basics</i>	11
<i>UNDO_RETENTION</i>	12
APPENDIX A.....	16

I. Introduction

With the enterprise environment enabling newer technology sooner and faster, coupled with greater cost to maintain the technology (people and machinery) the industry is pushing for lowering the total cost of ownership (TCO). Oracle, with its introduction of Oracle9i, has incorporated several new features that will help organizations sustain a lower TCO. The System Managed Undo feature is one such feature and will be discussed in this document.

This document will explain the architecture/design of system managed undo and how it differs from the traditional rollback segments used in pre-9i.

Throughout this document, the term rollback segments (or RBS) will be used in reference to the pre-9i objects and structures. The term undo segments will be used to refer to the system managed undo (SMU) structures. Additionally, transaction tables generically refer to both structures.

II. Review Rollback segments

Rollback segments are traditionally used to store undo transaction information. This information can be used for various purposes; such as building read consistent images of a block, performing “rollback” of a transaction in the event of a transaction failure (either user explicit rollback or implicit failure), or assisting database recovery. Rollback segments are similar to regular segments, e.g. data or index segments, in that they have the same “layered” structure; i.e. they all have cache and transaction structures (kcb, kte, and kd structures). However, the management and usage of rollback segments differs greatly from these other segments.

For example, RBS must have a minimum of two extents. Each transaction will write to only one extent at a time, although several transactions can share the same extent. Rollback segment extents are used in a circular fashion with transactions moving from one extent to the next after the current extent is filled. To manage transactions, a “head and tail” mechanism is employed such that the head is the pointer location of the current uncommitted transaction record and the tail is the pointer location to the oldest uncommitted transaction record. The RBS mechanism will always try to reuse the next extent in the ring unless there is an active transaction occupying that extent. Note Oracle will not re-use an RBS extent while it contains active transactions. If an active transaction is found in the next extent, a new extent will be allocated and brought into the ring. As each transaction commits, its transaction space (or undo information) will be available for other transactions. This space will not be reused, however, until the head of the RBS attempts to move into the extent and all active transactions in the extent have completed.

III. Introduction to System Managed Undo

The current rollback segment architecture has some shortcomings in that it requires a considerable amount of time to manage and monitor. For example, RBS have to be created, sized appropriately and continually tuned. Routine RBS issues include space management (out of space and “snapshot too old” errors) or transaction management issues (making sure enough rollback segments exist for the transactions to bind to). Successfully managing RBS requires a detailed understanding of not only the application logic, but also the inner workings of undo administration.

The Oracle 9i feature System Managed Undo simplifies the process of managing undo segments by internalizing the interface of these segments. This inherently reduces (or eliminates) the number of kernel parameters (init.ora) and user commands (create/alter)¹ required to manage and manipulate the undo segments. Note physically and structurally, there is little difference between RBS and SMU.

The following are advantages/benefits of the System Managed Undo feature (SMU):

- Eliminates the customer interface to rollback segments.
- Reduces the number of parameters for managing rollback segments.
- Reduces (and in many cases eliminates) the occurrence of various ORA-15xx errors that generally appear with rollback segment management; e.g., ORA 1555, and 1561.
- Reduces or eliminates space management issues/tasks associated with rollback segments.

A. How is System Managed Undo Implemented?

Operational SMU

In pre-9i, the most common method used to create and size rollback segments was based on certain transaction characteristics. For example, DBAs usually built several small and medium rollback segments and one large rollback segment to accommodate known transaction volumes and transaction types (OLTP versus batch).

However, pre-9i Oracle did not assign rollback segments based on transaction size since it had no way of knowing how large the transactions would be and thus treated all RBS the same in regards to assignment and usage. For example, a large batch process generating a considerable amount of undo would require a large RBS, however, this transaction might bind to a small RBS and may incur space management problems. Therefore, to accommodate the amount of undo generated from the process, a DBA might perform a “set transaction rollback segment” to bind to a specific large rollback segment.

With the advent of SMU, the statically built and sized RBS in previous versions are replaced with a single storage pool of undo segments. System Managed Undo introduces the concept of dynamic undo segments, which are very much like rollback segments in that they house transaction tables and undo blocks. However, the manner in which undo segments are managed (created/alter/dropped) is quite different from previous versions.

¹ In 9i, DDL operations against SMU, except drop rollback segment, will be considered no-operations. The init.ora parameter UNDO_SUPPRESS_ERRORS can be used to suppress errors when pre-9i commands are used against SMU objects.

SMU is enabled by the following in the init.ora parameters:

- `UNDO_MANAGEMENT=auto`
- `UNDO_TABLESPACE=<undo tablespace name(,undo tablespace name,...)>`
- `COMPATIBLE = 9.0.0`

For compatibility reasons, users can still use RBS in 9i, however the two undo types cannot co-exist; i.e., RBS and SMU cannot be onlined simultaneously.

Before the Oracle database can make use of SMU, an undo tablespace needs to be built² as shown below.

Figure 1

```
Create UNDO tablespace undo_tabs_1
  datafile '/u01/oradata/undo_tab_01.dbf' size 800 M;
```

[Note: If the Oracle Managed Files feature is enabled, then the UNDO tablespace can be created using the following syntax:

```
Create UNDO tablespace undo_tabs_1;]
```

Upon creation of the undo tablespace(s), an entry will be placed in the TS\$ base table for this tablespace and a new flag (KTT_UNDO) is set reflecting its UNDO status. An undo tablespace is essentially a bitmapped tablespace, therefore its space-extent is managed via bitmaps that reside in the file header. The advantage of bitmap tablespaces is that space transaction and management is performed using bitmaps versus performing expensive recursive calls (which also consume undo space).

The extent size of undo tablespace is determined by the system at tablespace creation time and changes as the undo segment grows. This extent size can be between 64K and 1Mb, depending on the current size of the segment. Thus, the size of undo extents is similar to that of any other system managed (bit-mapped) tablespace in that, if the undo segment is less than 1M, its extent size will be 64K. As the segment grows beyond 1M, its extent size will become 1M. As the segment extends, its extents are managed by an extent map.

² As in pre9i, a system undo segment is still created for the initial database build and for subsequent recursive calls.

Figure 2 displays the output of the TS\$ table and DBA_TABLESPACES after creating two undo tablespaces. Both reflect the “undo” tablespace type.

Figure 2

```
SQL> select ts#, name, flags from ts$;
```

TS#	NAME	FLAGS
0	SYSTEM	0
2	TEMP	0
3	USERS	0
7	UNDO_TABS_1	17
8	UNDO_TABS_2	17

or

```
SQL> select TABLESPACE_NAME, STATUS, CONTENTS from dba_tablespaces;
```

TABLESPACE_NAME	STATUS	CONTENTS
SYSTEM	ONLINE	PERMANENT
TEMP	ONLINE	TEMPORARY
USERS	ONLINE	PERMANENT
UNDO_TABS_1	ONLINE	UNDO
UNDO_TABS_2	ONLINE	UNDO

At instance startup, an enqueue will be acquired on the undo tablespace(s) specified by the init.ora parameter *UNDO_TABLESPACE*, and thus binding the undo tablespace(s) to that instance. Therefore, in an OPS environment, each instance will acquire and bind to its own undo tablespace. The alert log will display the undo segments and the undo tablespace online. Below is an excerpt from the alert log illustrating this.

```
SMON: enabling cache recovery
Fri Mar 23 00:44:23 2001
SMU Undo Segment 11 Onlined
SMU Undo Segment 12 Onlined
SMU Undo Segment 13 Onlined
SMU Undo Segment 14 Onlined
SMU Undo Segment 15 Onlined
SMU Undo Segment 16 Onlined
SMU Undo Segment 17 Onlined
SMU Undo Segment 18 Onlined
SMU Undo Segment 19 Onlined
SMU Undo Segment 20 Onlined
Successfully onlined Undo Tablespace 8.
```

The undo tablespace is used to house the undo segments only; i.e., no user objects can be created in these tablespaces³. The undo tablespace is essentially a collection of undo segments and free space, from which extent space is allocated.

³ If no undo tablespace is specified in SMU mode, then the system undo segment will be used and a message in the alert log will be filed, alerting the DBA that the system undo segment is used and a *undo_tablespace* should be created.

B. Undo Segment structures

Transaction tables

Similar to RBS, each 9i undo segment contains an undo segment header and undo blocks. The undo segment header includes a transaction table area, transaction control area, and an extent map area. A transaction table contains transaction slots and describes space allocated by transactions and the transaction status. These slots are used to store transactional information required to rollback the transaction. Each transaction slot is associated with a local bounded transaction, also known as a transaction thread, as identified by a transaction id (XID)⁴. In pre-9i, the number of transactions that could bind to a particular RBS was based on the size of the undo segment header (determining the number of slots in the transaction table).

Upon creation of the undo tablespace, a minimum number of transaction tables (or undo segments) will be dynamically created⁵. The number of transaction tables created initially is based on several internal system statistics. Although a given number of these transactions tables get built, only a subset of these tables will become onlined and usable for transactions at instance startup. The offlined undo segments will be onlined as needed, when transaction volume dictates. Dynamically created undo segments are named `_SYSSMU usn` where usn is the undo segment number (USN). All dynamically created (onlined and offlined SMU) will be defined in the UNDO\$ base table (Figure 3 and Figure 5). Each started transaction will bind to only one transaction table and thus one undo segment⁶.

In the following illustration (Figure 3), the instance is currently running in RBS mode (this can be done either by not setting `UNDO_MANAGEMENT` or by setting `UNDO_MANAGEMENT=manual`) with newly created SMU objects. All system generated undo names are SMU undo segments, having a status of '2', indicating they are offline.

The possible values for status are:

- 1 - invalid
- 2 - defined but not necessarily usable (depending on init.ora setting and type of undo segment)
- 3 - in-use/online
- 4 - offline
- 5 - needs recovery
- 6 - partially available

⁴ Recursive transactions will always be bounded to the same undo segment as its parent call.

⁵ The initial number of transaction tables is $(1.1 * \text{sessions} / 5)$.

⁶ As in pre-9i, parallel dml transaction slaves will each get their own segment.

Figure 3

```
SQL> select us#, name, ts#, status$ from undo$;
```

US#	NAME	TS#	STATUS\$
0	SYSTEM	0	3
1	RBS6	5	3
2	RBS5	5	3
3	RBS4	5	3
4	RBS3	5	3
5	RBS2	5	3
6	RBS1	5	3
7	RBS0	5	3
8	_SYSSMU8\$	6	2
9	_SYSSMU9\$	6	2
10	_SYSSMU10\$	6	2

US#	NAME	TS#	STATUS\$
11	_SYSSMU11\$	6	2
12	_SYSSMU12\$	6	2
13	_SYSSMU13\$	6	2
14	_SYSSMU14\$	6	2
15	_SYSSMU15\$	6	2
16	_SYSSMU16\$	6	2
17	_SYSSMU17\$	6	2

[Note: The SMU segments cannot be brought online if the system is currently running in RBS mode (*UNDO_MANAGEMENT=manual*).]

Figure 4

```
SQL> alter system set undo_tablespace = undotbs1;
alter system set undo_tablespace = undotbs1
*
ERROR at line 1:
ORA-02097: parameter cannot be modified because specified value is
invalid
ORA-30014: operation not supported in non-SMU mode
```

In Figure 5, the instance was started with *UNDO_MANAGEMENT=auto*. Notice that the status flag for the SMU changed from off-line (in Figure 3) to online; i.e., STATUS\$ change from '2' to '3'. The RBS status changed from '3' to '2' indicating offline and un-usable. The System Undo Segment still exists for various tasks such as bootstrap transactions during initial database build, recursive transactions and transaction table creation.

Figure 5

```
SQL> select us#,name, ts#, status$ from undo$;
```

US#	NAME	TS#	STATUS\$
0	SYSTEM	0	3
1	RBS6	5	2
2	RBS5	5	2
3	RBS4	5	2
4	RBS3	5	2
5	RBS2	5	2
6	RBS1	5	2
7	RBS0	5	2
8	_SYSSMU8\$	6	3
9	_SYSSMU9\$	6	3
10	_SYSSMU10\$	6	3

US#	NAME	TS#	STATUS\$
11	_SYSSMU11\$	6	3
12	_SYSSMU12\$	6	3
13	_SYSSMU13\$	6	3
14	_SYSSMU14\$	6	3
15	_SYSSMU15\$	6	3
16	_SYSSMU16\$	6	3
17	_SYSSMU17\$	6	3

C. Space Management

As in previous versions of Oracle, once a transaction starts, it will bind to a rollback segment and obtain an entry in the segment's transaction table. In pre-9i however, the acquisition of the rollback segment was on a round robin basis and on a least loaded transaction table basis. Given that there was a finite number of rollback segments defined, there was usually more than one transaction per rollback segment. If the number of incoming transactions exceeded the maximum number of transactions per rollback segment, a wait for "undo segment header" would occur.

With SMU, the transaction-to-undo segment bind algorithm will generally yield only one transaction (XID) to a transaction table⁷. Therefore, a transaction thread⁸ will always attempt to bind to an online transaction table with no current active transactions. If none exist then "offlined" undo segments will be "onlined". If all the offlined segments have been onlined, more segments will be created dynamically, and the transaction

⁷ The rare times when this is an exception, are when space within the undo tablespace is constrained and all the transaction tables are consumed. In this case there can be more than one transaction per transaction table.

⁸ A transaction thread includes all its nested recursive calls as well.

will be bound to one of these. When no more undo segments can be created due to lack of space in the undo tablespace, the bind algorithm will then allow multiple transactions per transaction table.

This dynamic bind to undo segments helps to assuage the following issues with RBS.

- **Simplifies the approach to managing undo segments** The DBA will be freed from having to predict undo transaction sizes or rates, and not be required to assign specific transactions to specific undo segments.
- **Distributes transactions evenly across undo segments** Transactions will acquire a transaction table more quickly i.e., reducing contention for undo segment headers. Moreover, this limits the contention between transactions for undo blocks within the confines of an undo segment i.e., reduced waits for “undo blocks”.
- **Reduces space management issues** Undo segments will not be confined to a pre-defined storage area (*initial/next* extent sizes and *maxextents*).

The implementation of RBS had several issues with regards to space constraints. For example, a given transaction was constrained to the space boundaries (maximum extents defined and free space in the assigned tablespace) available in the bounded rollback segment, and thus incurred various ORA-1562 errors whenever all space was exhausted within the bounded RBS, regardless of how much space was remaining in other RBS or the RBS tablespace.

SMU supports the dynamic transfer of space across transaction tables. Therefore, when space constraint issues within the undo segment arise, instead of returning the ORA-1562 exception, Oracle will attempt to do the following in successive order:

1. Reclaim expired extents from the current transaction’s committed changes.
2. Acquire a free extent from the undo tablespace for the current undo segment, assuming the tablespace has free extents.
3. Acquire expired extent space from another undo segment for the current transaction, if no free space exists in the tablespace.
4. Acquire more space by datafile auto extension.
5. Reclaim space from un-expired (but committed) transactions from the current undo segment.
6. Reclaim space from un-expired (but committed) transactions from other undo segments
7. Raise ORA-1562

Occurrences of ORA-1555 errors or stealing of extents to satisfy space requirements are logged against the V\$UNDOSTAT table. This table can be used to determine the efficiency of space management (discussed later).

SMU space constraint management is developed such that a transaction needing undo space to complete the call will be accommodated. However, this could lead to runaway usage of undo space by uncontrolled transactions. To prevent this, a new Resource Manager directive, *UNDO_POOL*, can be setup to limit the amount of undo space consumed by a transaction.

SMON is responsible for managing space within the undo tablespace, periodically (proactively) performing shrinks on idled undo segments. However, SMON reactively performs space management when foreground processes need to steal extents.

D. Read Consistency

Basics

Although read consistency is loosely tied to space management issues, it is discussed here as a separate topic from space management to illustrate a feature within SMU.

Read consistency allows the construction of a valid and consistent result set or “before image” (via CR operations) for any given transaction for a given point in time. Read consistency is enforced using the SCN or System Commit Number. A query will use the rollback segments and a given SCN to build a “before image” when a changed but not yet committed data block is encountered after the said query started. This is especially important for large queries reading blocks during concurrent change requests from other transactions, such that undo information is required to produce older data block images. The management of read consistency and its consistent set is maintained within the constructs of the undo segments.

Management of read consistency and the resolution of “ORA-1555: snapshot too old” error messages have always been daunting tasks for DBAs. These messages occur for various reasons, but the underlying reason involves the main property of undo space management; reuse of committed undo blocks and its associated reuse of undo transaction slot entries. The reuse of committed undo blocks is a necessary evil in that it helps to manage undo storage.

Recall that undo segments are like “doughnuts”, in that they are circular objects that require at least two extents. Transactions start in one extent (the head) and when completely filled, attempt to go the next extent. If the next extent has an active transaction, the extent cannot be used (Oracle does not allow rolling into active extents) and a new extent is allocated and inserted into the circular doughnut. However, if the next extent does not have an active transaction (but may have committed transactions), Oracle will roll into this extent, slowly overwriting the committed undo data blocks. Note, all blocks are not overwritten at once; they are “re-newed” as they are needed.

If an existing transaction fails in its attempt to rollback needed changes (made by another transaction) or build a read consistent view from the newly overwritten undo data, then it will incur the ORA-1555 error. To remedy this situation, DBAs may have several proactive plans in place. These include pre-allocating a large number of extents (usually 20) at RBS definition or directing transactions to specific RBS via “set transaction rollback segment”. The set transaction command is usually targeted at a pre-defined large RBS. However, these measures may not always relieve the problem.

UNDO_RETENTION

With SMU, an optional init.ora parameter called *UNDO_RETENTION* allows the DBA to specify how long the system will attempt to retain undo information for a committed transaction without being overwritten or recaptured.⁹ This parameter, based in units of wall-clock seconds, is defined universally for all undo segments.

The *UNDO_RETENTION* value is dynamic and can be changed via “alter system set undo_retention = <time-interval-in-seconds>;”. For example, if the *UNDO_RETENTION* parameter is set to 1200 seconds, Oracle will attempt to keep 20 minutes worth of undo available for transactions that run 20 minutes or less (assuming no space constraint issues arise). Moreover, this retention period is valid across instance recycles; i.e., it is persistent information. However, if the instance’s inactivity period exceeds the *UNDO_RETENTION*, then SMON (as needed) will slowly perform reclaims after the next startup.

UNDO_RETENTION pertains to extents that are reclaimed and not individual blocks. Thus, the extent commit time (retention period) is equal to the maximum commit time of all the transactions within that extent. To illustrate, recall the earlier example using SMU. When a transaction attempts to move into an extent with an active transaction, the undo segment will grow by allocating a new extent. If the next extent is not active but has committed transactions, then it cannot be reclaimed (overwritten) if the maximum commit time is less than the defined retention period. If the retention period has expired, then the extent can be reclaimed and used.

Use of *UNDO_RETENTION* can potentially increase the size of the undo segment for a given period of time, so the retention period should not be arbitrarily set too high. The UNDO tablespace still must be sized appropriately. The following calculation can be used to determine how much space a given undo segment will consume given a set value of *UNDO_RETENTION*.

$$\text{Undo Segment Space Required} = (\text{undo_retention_time} * \text{undo_blocks_per_seconds})$$

As an example, an *UNDO_RETENTION* of 5 minutes (default) with 50 undo blocks/second (8k blocksize) will generate:

$$\text{Undo Segment Space Required} = (300 \text{ seconds} * 50 \text{ blocks/ seconds} * 8\text{K/block}) = 120 \text{ M}$$

The retention information (transaction commit time) is stored in every transaction table block and each extent map block. When the retention period has expired, SMON will be signaled to perform undo reclaims, done by scanning each transaction table for undo timestamps and deleting the information from the undo segment extent map. Only during extreme space constraint issues will retention period not be obeyed.

⁹ Note, in case of space pressure, these extents will still be overwritten.

The V\$UNDOSTAT view can be used in determining space issues and *UNDO_RETENTION* settings. This view holds undo statistics for 10 minute intervals¹⁰. Note, this view represents statistics across instances, thus each begin time, end time, and undotsn will be a unique interval per instance. This view contains the following columns aiding in this effort:

BEGIN_TIME	- The beginning time for this interval check
END_TIME	- The ending time for this interval check
UNDOTSN	- The undo tablespace number.
UNDOBLKS	- The total number undo blocks consumed during the time interval
TXNCOUNT	- the maximum number of transactions during that interval
MAXQUERYLEN	- The maximum duration of a transaction within that time period.
MAXCONCURRENCY	- The maximum number of concurrent transactions during that interval
UNXPSTEALCNT	- The number of incidences when un-expired extents were stolen from other undo segments to satisfy space requests .
UNXPBLKRELCNT	- The count of un-expired extents stolen from other undo segments to satisfy space requests.
UNXPBLKREUCNT	- The number of incidences when un-expired extents were stolen to satisfy space requests in the current transaction.
EXPSTEALCNT	- The number of incidences when expired extents were stolen from other undo segments to satisfy space requests .
EXPBLKRELCNT	- The number of expired extents were stolen from other undo segments to satisfy a space request.
EXPBLKREUCNT	- The number of incidences when expired extents were stolen to satisfy space requests in the current transaction.
SSOLDERRCNT	- The number of ORA-1555, “snapshot too old” errors that occurred during the time interval.
NOSPACEERRCNT	- The number of “Out of Space” occurrences.

When the columns UNXPSTEALCNT through EXPBLKREUCNT reflect non-zero values, there is some indication of space constraints. If column SSOLDERRCNT is non-zero, then *UNDO_RETENTION* is not properly set. If column NOSPACEERRCNT is non-zero, then there are serious space issues.

¹⁰ An SGA array has 144 slots to store 24 hours worth of 10 minute histogram statistics.

SMU also provides the capability to perform flashback queries against committed data; i.e., perform a snapshot query from a given point in time. This new feature, called Oracle Flashback, leverages the System Managed Undo functionality by performing reads against the retained undo blocks. Thus, a user can execute arbitrary queries or PL/SQL packages queries as of the specified time/SCN, and see the data in the database as of that time¹¹. The package *DBMS_FLASHBACK* is provided to enable Flashback and requires passing of a SCN or timestamp as a reference point. In the case of timestamp, SMON keeps an internal table to map system time to an SCN.

Note, the Flashback functionality will succeed only if the queries address a time within the retention interval; i.e., enough undo information must exist to reconstruct the snapshot.

Figure 6

```
[otcsol1]/proj/tar13202642.600/nitin> date
Wed Jan 24 10:55:14 EST 2001

SQL> update scott_emp set deptno = 00 where deptno = 44;

3 rows updated.

SQL> commit;
Commit complete.

[otcsol1]/proj/tar13202642.600/nitin> date
Wed Jan 24 10:59:24 EST 2001

SQL> select deptno, count(*) from scott_emp
        group by deptno;

DEPTNO  COUNT(*)
-----  -
      0      3
     20      5
     30      6

REM wait couple of minutes .....

[otcsol1]/proj/tar13202642.600/nitin> date
Wed Jan 24 11:09:30 EST 2001

SQL> @point_n_time.sql

PL/SQL procedure successfully completed.

DEPTNO  COUNT(*)
-----  -
     20      5
     30      6
     44      3

PL/SQL procedure successfully completed.
```

¹¹ Flashback may be used to perform DML as well.

Figure 7 Point_n_time.sql

```
EXECUTE dbms_flashback.enable_at_time('24-JAN-01 10:55:14');
  select deptno, count(*) from scott_emp
    group by deptno;
REM To Perform using SCN use ENABLE_AT_SYSTEM_CHANGE_NUMBER
EXECUTE dbms_flashback.disable;
```

Appendix A

To test the capabilities of SMU's bind algorithm, a test was performed using 5 concurrent sessions, with each session updating a 9 partition table in parallel. The total number of concurrent transactions executing was 45. The maximum number of *PROCESSES*, *PARALLEL_MAX_SERVERS*, and table parallelism was increased to accommodate the update run.

When the database was shutdown and restarted, there were 11 undo segments (including the system undo segment) available, as shown in the query below:

```
SQL> select usn from v$rollstat;
```

```
-----
USN
-----
0
1
2
3
4
5
6
7
8
9
10
```

Five tables were created (Sysobjx_part, where x is 1 to 5) with 9 partitions each. A sample structure is shown below:

```
CREATE TABLE sysobj_part
tablespace users
STORAGE (INITIAL 100K NEXT 100K)
PARTITION BY RANGE (obj#)
(PARTITION sysobj_p1 VALUES LESS THAN (500),
PARTITION sysobj_p2 VALUES LESS THAN (1000),
PARTITION sysobj_p3 VALUES LESS THAN (1500),
PARTITION sysobj_p4 VALUES LESS THAN (2000),
PARTITION sysobj_p5 VALUES LESS THAN (2500),
PARTITION sysobj_p6 VALUES LESS THAN (3000),
PARTITION sysobj_p7 VALUES LESS THAN (3500),
PARTITION sysobj_p8 VALUES LESS THAN (4000),
PARTITION sysobj_p9 VALUES LESS THAN (MAXVALUE))
as select * from sys.obj$;
```

A typical transaction is shown below:

```
TXN 1: ALTER SESSION ENABLE PARALLEL DML;

UPDATE /*+ PARALLEL(sysobj_part,9) */ sysobj_part
SET name = 'AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA';
```

Once the TXNS were started, the following query was performed to check on the allocated undo segments:

```
select substr(vs.username,1,10) "USERNAME",vt.start_time "TXN START
TIME",
vt.used_ublk "# UNDO BLOCKS", substr(vr.name,1,10) "RBS NAME"
from v$transaction vt,v$session vs,v$rollname vr
where vt.addr=vs.taddr
and vt.xidusn = vr.usn
order by vs.username;
```

As each update process started, the number of undo segments allocated steadily increased. At the end of the transaction, there were 44 undo segments allocated; however, due to space constraints, 6 slaves were "doubled up" on 6 undo segments.

USERNAME	TXN START TIME	# UNDO BLOCKS	RBS NAME
PJ1908	03/09/01 15:31:09	1	_SYSSMU4\$
PJ1908	03/09/01 15:31:10	1	_SYSSMU6\$
PJ1908	03/09/01 15:31:10	1	_SYSSMU8\$
PJ1908	03/09/01 15:31:10	71	_SYSSMU9\$
PJ1908	03/09/01 15:31:10	1	_SYSSMU8\$
PJ1908	03/09/01 15:31:10	71	_SYSSMU9\$
PJ1908	03/09/01 15:31:10	67	_SYSSMU10\$
PJ1908	03/09/01 15:31:10	68	_SYSSMU1\$
PJ1908	03/09/01 15:31:10	36	_SYSSMU2\$
PJ1908	03/09/01 15:31:10	69	_SYSSMU3\$
PJ1908	03/09/01 15:31:10	67	_SYSSMU5\$
PJ1908	03/09/01 15:31:10	67	_SYSSMU7\$
PJ1908	03/09/01 15:31:10	65	_SYSSMU12\$
PJ1908	03/09/01 15:31:12	1	_SYSSMU14\$
PJ1908	03/09/01 15:31:11	1	_SYSSMU11\$
PJ1908	03/09/01 15:31:14	61	_SYSSMU16\$
PJ1908	03/09/01 15:31:10	66	_SYSSMU13\$
PJ1908	03/09/01 15:31:14	58	_SYSSMU15\$
PJ1908	03/09/01 15:31:14	58	_SYSSMU17\$
PJ1908	03/09/01 15:31:16	26	_SYSSMU24\$
PJ1908	03/09/01 15:31:14	58	_SYSSMU21\$
PJ1908	03/09/01 15:31:16	22	_SYSSMU26\$
PJ1908	03/09/01 15:31:17	17	_SYSSMU40\$
PJ1908	03/09/01 15:31:17	19	_SYSSMU48\$
PJ1908	03/09/01 15:31:17	17	_SYSSMU40\$
PJ1908	03/09/01 15:31:17	19	_SYSSMU48\$
PJ1908	03/09/01 15:31:16	24	_SYSSMU33\$
PJ1908	03/09/01 15:31:14	58	_SYSSMU22\$
PJ1908	03/09/01 15:31:14	60	_SYSSMU18\$
PJ1908	03/09/01 15:31:16	22	_SYSSMU30\$
PJ1908	03/09/01 15:31:17	17	_SYSSMU36\$
PJ1908	03/09/01 15:31:14	56	_SYSSMU19\$
PJ1908	03/09/01 15:31:14	57	_SYSSMU23\$
PJ1908	03/09/01 15:31:14	31	_SYSSMU20\$
PJ1908	03/09/01 15:31:16	18	_SYSSMU27\$
PJ1908	03/09/01 15:31:16	24	_SYSSMU43\$
PJ1908	03/09/01 15:31:16	27	_SYSSMU34\$
PJ1908	03/09/01 15:31:16	25	_SYSSMU50\$
PJ1908	03/09/01 15:31:16	16	_SYSSMU25\$
PJ1908	03/09/01 15:31:17	26	_SYSSMU45\$
PJ1908	03/09/01 15:31:17	19	_SYSSMU47\$
PJ1908	03/09/01 15:31:17	26	_SYSSMU45\$
PJ1908	03/09/01 15:31:17	19	_SYSSMU47\$
PJ1908	03/09/01 15:31:17	21	_SYSSMU35\$
PJ1908	03/09/01 15:31:17	17	_SYSSMU29\$
PJ1908	03/09/01 15:31:17	25	_SYSSMU44\$
PJ1908	03/09/01 15:31:17	23	_SYSSMU41\$
PJ1908	03/09/01 15:31:19	20	_SYSSMU39\$
PJ1908	03/09/01 15:31:19	24	_SYSSMU32\$
PJ1908	03/09/01 15:31:19	20	_SYSSMU37\$
PJ1908	03/09/01 15:31:19	21	_SYSSMU28\$
PJ1908	03/09/01 15:31:19	30	_SYSSMU49\$
PJ1908	03/09/01 15:31:19	21	_SYSSMU42\$

```

PJ1908      03/09/01 15:31:19      24  _SYSSMU38$
PJ1908      03/09/01 15:31:19      36  _SYSSMU31$
PJ1908      03/09/01 15:31:19      20  _SYSSMU46$

```

50 rows selected.

Here is also the out put from the v\$undostat view:

```

-----
BEGIN_TIME          END_TIME          UNDOTSN    UNDOBLKS    TXNCOUNT
-----
MAXQUERYLEN  MAXCONCURRENCY  UNXPSTEALCNT  UNXPBLKRELCNT  UNXPBLKREUCNT
EXPSTEALCNT
-----
EXPBLKRELCNT  EXPBLKREUCNT  SSOLDERRCNT  NOSPACEERRCNT
-----
03/08/2001 15:59:53 03/08/2001 16:04:27          1          478          21
9          21          0          0
0          0
0          0          0          0
03/08/2001 15:49:53 03/08/2001 15:59:53          1          507          63
9          21          0          0
8          0
0          0          0          0

```