



Oracle9i: Understanding Automatic Undo Management and Flashback Query



By Kirtikumar Deshpande

This article provides an overview of simplified Automatic Undo Management and the Flashback Query features in Oracle 9i.



Introduction

Automatic Undo Management (AUM) simplifies rollback segment management with the introduction of Automatic Undo Segments. Oracle9i calls the rollback segments undo segments. An Oracle9i database can automatically create, allocate, and manage these undo segments. AUM is also referred to as System Managed Undo (SMU). If the Oracle9i database is operating in the AUM mode, the DBA is not responsible for creating and managing the undo segments. If the database does not use AUM, then it is said to be operating in Manual Undo Management (MUM) mode, or Rollback Undo mode (RBU). As in the prior Oracle versions, the DBA will be responsible for the management of rollback segments.

Flashback Query leverages the AUM feature to provide a mechanism to view data as it existed sometime in the past. It is possible to query past data by specifying date and time, or a System Change Number (SCN). Using Flashback Query it is possible to recover data from minor mishaps, such as data deletion or modification, if they are detected in a timely fashion and the undo information is available in the undo segments. However, Flashback Query is not a data recovery tool.

Automatic versus Manual Undo Management

You can operate your Oracle9i databases in either Automatic or Manual Undo Management mode. If you created your database using the GUI tool, Database Creation Assistant (DBCA), Oracle will default to using AUM. AUM is available only when the database compatibility is set for Oracle9i or greater. That is, the compatible parameter in the init.ora file has a value of 9.0.0 or higher. MUM is available when the database compatibility is set for Oracle9i or Oracle8i. You can switch from AUM to MUM mode, and vice

versa. However, each such switch will need a change to an initialization parameter and an instance restart. In addition, when switching from AUM to MUM mode, you must create good old rollback segments if they were not already present. Changing undo management mode is still not automatic. However, Oracle does not encourage operating an Oracle9i database in MUM mode.

Automatic Undo Mode

The initialization parameter, UNDO_MANAGEMENT, must be set to AUTO to use AUM. The default value for this parameter is MANUAL.

When operating the database in AUM mode, you are not permitted to create, allocate, and manage undo segments. Oracle takes over that responsibility. The undo segments it creates are referred to as Automatic Undo Segments (AUS). It is not necessary to list them in the init.ora file using the ROLLBACK_SEGMENTS parameter. This parameter will simply be ignored during database startup if it is specified in the init.ora file. Operations – such as offline, online, or assigning a specific undo segment to a transaction – will result in statement failure errors. These operations are not permitted in AUM mode.

Oracle controls the physical characteristics of these segments. You will not be able to specify or change any storage parameters of AUS. To prevent you from influencing the storage parameters of these undo segments, Oracle creates them in a special type of tablespace called an UNDO Tablespace. You cannot change the storage characteristics of the undo tablespace either.

Undo Tablespace

When using AUM, it is the DBA's responsibility to create the undo tablespace. The undo tablespace can be created using one of two methods – during the creation of the database using the CREATE DATABASE command, or in an existing Oracle9i database using the CREATE TABLESPACE command. As a part of creating the undo tablespace, Oracle will also create a predetermined number of undo segments that will be assigned to this undo tablespace.

The CREATE DATABASE command has been enhanced to offer an option to create an undo tablespace. The following code shows how this option is used to create an undo tablespace titled undo_tbs:

```
create database KED9
  controlfile reuse
  datafile '/u01/oradata/KED9/system_01.dbf' size 250M
  undo tablespace undo_tbs
  datafile '/u02/oradata/KED9/undo_tbs_01.dbf' size 500M
  logfile
  group 1
  ('/u10/oradata/KED9/redo_g1m1.log') size 25M,
  .....
/
```

The CREATE TABLESPACE command has been enhanced to include syntax to create an UNDO tablespace. The following code shows how this syntax is used to create an undo tablespace titled undo_tbs:

```
create undo tablespace undo_tbs
  datafile '/u02/oradata/KED9/undo_tbs_01.dbf' size 500M
/
```

In an undo tablespace, the extents of the undo segments are managed locally, as this tablespace is created as a Locally Managed Tablespace using the autoallocate (or SYSTEM) policy for sizing the undo extents. Oracle returns an ORA-30024 error if you specified any storage parameter option with CREATE UNDO TABLESPACE command. Also, you cannot create any

permanent objects in an undo tablespace. ORA-30022 will be returned if an attempt is made to do so. However, you can resize the datafiles of an undo tablespace or change them to be autoextensible. You can also alter the undo tablespace to begin and end an on-line backup of its datafiles.

Oracle allows only one active undo tablespace per instance. The active undo tablespace cannot be taken offline. You can create more than one undo tablespace in the database, but only one can be active at any given time. In a Real Application Clusters (RAC) environment, each instance must have its own active undo tablespace.

The initialization parameter `UNDO_TABLESPACE`, specifies the name of the undo tablespace to use when the instance is started. The active undo tablespace can be changed using an `ALTER SYSTEM` command. This is a dynamic parameter and the change will take effect immediately. The transactions that started after such a change will begin using the new undo tablespace. Any active transactions prior to this change will continue to use the previously active undo tablespace, which will be marked `PENDING OFFLINE`. When those transactions end, either via a commit or a rollback, the old undo tablespace will go offline along with the undo segments it supported. However, an offline undo tablespace may contain critical undo information that is required for a successful creation of the read consistent image of the changed data. Active queries that cannot access required data in the offline tablespace would fail with an ORA-1555 error. It will also negatively affect any Flashback Query operation. So care must be taken when switching active undo tablespaces. The following command shows how to switch the active undo tablespace to `undo_tbs_2`:

```
SQL> alter system set undo_tablespace = undo_tbs_2;
```

You can drop an undo tablespace using the `DROP TABLESPACE` command. However, an undo tablespace can be dropped only if it is not currently used by any instance. If it has any active transactions, the command will fail. All associated undo segments will be dropped along with the undo tablespace when the tablespace is dropped. Dropping an undo tablespace is the only way to drop all automatic undo segments from the database.

A couple of minor issues with `UNDO_TABLESPACE` initialization parameter are worth mentioning. First, if the specified undo tablespace is not available (misspelled name, or dropped tablespace), Oracle will not issue an error or warning during instance startup. It will use one of the other available undo tablespaces. If no other undo tablespace was available, it will simply use the `SYSTEM` tablespace for undo segments. The only indication that this has occurred is that the following warning message will be reported in the alert.log file:

```
***Warning - Executing transaction without active Undo Tablespace
```

It is a good idea to change your alert.log monitoring mechanism to track this message.

The second point involves the creation of a new database. If you specified a name for an undo tablespace in the `init.ora` file, using `UNDO_TABLESPACE`, but forgot to specify the `UNDO TABLESPACE` directive in the `CREATE DATABASE` statement, the database creation statement will fail. Oracle will report following error to your session:

```
ORA-01092 ORACLE instance terminated. Disconnection forced
```

This vague error does not really tell you what happened. But an additional error in the alert.log will provide further explanation. The errors reported in the alert.log will state:

```
ORA-01501: CREATE DATABASE failed  
ORA-30045 No undo tablespace name specified
```

Both of these errors will be reported in the trace file that will be generated in the user dump directory.

Use of AUM mode is not mandatory for creating undo tablespaces. The automatically created undo segments are not used until AUM is activated.

Automatic Undo Segments

When you create an undo tablespace, Oracle automatically creates a predetermined number of undo segments that will be assigned to this tablespace. Each undo segment is created with two 64KB extents. The value of `MINEXTENTS` is set to a value of 2 while the value of `MAXEXTENTS` set to unlimited. The author has observed that 10 undo segments are created in undo tablespaces on most platforms.

The names for the automatic undo segments are, of course, system generated and are of the form `_SYSSMU n $`. The underscore at the beginning of the name should indicate that this structure is off-limits to DBAs, just like any other underscore initialization parameter. The `SYS` is for system generated and `SMU` for System Managed Undo. The `n` denotes the undo segment number, or the `USN`, followed by the dollar sign. The `USN` is the next available sequentially numbered `USN` for the database. So, if your database had, for example, five existing rollback segments and you created a new undo tablespace, then the newly created undo segments will have names starting with `_SYSSMU6$` and ending with `_SYSSMU15$`. The `USN` is reported in `V$ROLLNAME`, `V$ROLLSTAT` views. It is exposed as a `SEGMENT_ID` in the `DBA_ROLLBACK_SEGS` view. If you wanted the automatic undo segments to have names starting with `_SYSSMU1$`, `_SYSSMU2$` and so on, then you will need to drop all existing rollback segments prior to creating your first undo tablespace. In some rare situations, if you had to use the name of the automatic undo segment in a SQL script, you would need to enclose it in double quotation marks because of the way Oracle named these segments.

Should you run into undo segment corruption, and the database does not start or reports undo segment block corruption problems, you should contact Oracle Support. The hidden parameter `_corrupted_rollback_segments` is still available. However, please refrain from using it without the approval from Oracle Support.

Using Automatic Undo Segments

As the name suggests, the use of undo segments is completely automatic. Oracle will put online a certain number of undo segments when the instance starts. This number initially depends on the `SESSIONS` parameter in the `init.ora` file. As the number of active transaction increases, Oracle will try to put online any available offline undo segments. If all undo segments are already online, and there is enough space available in the undo tablespace, Oracle will create additional undo segments and set them online so that new transactions can use them. One undo segment per transactions is the desired goal. When space in the undo tablespace to create more undo segments has been exhausted, Oracle will begin assigning the new transactions to the existing undo segments. Approximately every 12 hours the `SMON` process shrinks idle undo segments. The undo information in those segments is considered to be obsolete.

continued on page 24

In the AUM mode, Oracle tries to minimize errors associated with undo segments. Attempts are made to minimize the out of undo space error (ORA-1562), and the snapshot too old error (ORA-1555). The former is achieved using a mechanism called Dynamic Extents Transfer, while the latter is achieved by providing the DBA with a mechanism to influence the retention period of committed information in the undo segments that can be used by long-running queries.

Dynamic Extents Transfer

In AUM mode, Oracle tries to manage available undo space more effectively. Effort is made to not let the transaction abort due to space exhaustion when the undo segment extends. If the datafile is autoextensible, Oracle will try to extend the file to acquire more space. If more space cannot be acquired, Oracle will deallocate extents from other undo segments and allocate those to the undo segment requiring additional space. Since MAXEXTENTS is set to unlimited, or to a very large number, the chance of encountering a MAXEXTENTS exceeded error is extremely remote. In a busy database, it is technically possible that one long-running transaction could steal extents from all other undo segments and consume almost all of the space in the undo tablespace. Only when Oracle cannot extend the datafile, and cannot steal extents from other undo segments will the query encounter an out of undo space error (actually an ORA-1562: failed to extend rollback segment number %s).

Minimizing ORA-1555 Snapshot too old Error

With proper sizing of undo tablespace (discussed later in this article) and a proper value for the UNDO_RETENTION initialization parameter, you can minimize the occurrence of this error. The value of the UNDO_RETENTION parameter sets the duration, in seconds, to preserve committed information in the undo segments. This information will be used by queries when they need to build a read consistent view of the changed data. The default value for UNDO_RETENTION is 900 seconds (15 minutes). It can be changed dynamically to affect the retention of subsequently committed undo information, as well as currently committed, but unexpired undo information. You can monitor the undo space usage, and how long the queries run in your database using the new undo views to derive an acceptable value for this parameter.

However, use of AUM will not eliminate ORA-1555 errors. You may still encounter one when least expected. Dynamic extents transfer will overwrite undo information that may be required to build a read consistent view of the changed data. The error may also occur if the active undo tablespace is changed, because the undo segments that contain the information to build a consistent view of the changed data are now offline.

One nice thing about Oracle9i is that it will report all ORA-1555 errors in the alert log. It will also list the SQL statement that encountered this error. It will also report how long the SQL scripts ran before receiving this error. This information can be used to adjust the UNDO_RETENTION parameter value. Following is an example of such an entry from the alert log. It shows that reported SQL statement (statement truncated in the example) executed for 10904 seconds before receiving ORA-1555 errors.

```
Tue Dec 23 14:19:28 2003
ORA-01555 caused by SQL statement below (Query Duration=10904 sec, SCN:
0x056f.b98b7787):
Tue Dec 23 14:19:28 2003
SELECT yr, clshdg, SUM(billed_amt), SUM(item_cnt), SUM(cust_cnt), SUM(pd_adv_cnt),
SUM(qce_tot) FROM ( SELECT con.DIR_PUB_YR yr, clhd.CLSHDG_STD_NAM clshdg, con.CUST_ID ||
con.SOURCE_SYSTEM_CDE custid, . . .
```

In this particular case, increasing UNDO_RETENTION period to more than 10904 seconds would have likely avoided the ORA-1555 error. Further investigation suggested that rescheduling the process at a different time was a much better choice.

Controlling UNDO Space Usage

The Dynamic Extents Transfer feature also makes it possible for a rogue transaction to run amuck and use all the undo space. This can adversely affect other legitimate transactions. To control undo space usage, Oracle introduced a new resource plan directive. The UNDO_POOL directive must be used to define undo space quota limits, in terms of bytes written to the undo tablespace. When sessions belonging to a consumer group exceed their undo quota limits, they will not be allowed to perform any more DML activity and the session that caused the undo quota to exceed will be terminated with an ORA-30027 error – Undo quota violation – failed to get %s (bytes). The default value of UNDO_POOL directive is UNLIMITED. This allows all sessions to use an unlimited amount of available undo space. Of course, you must use Resource Manager to make use of the UNDO_POOL directive.

Suppressing UNDO errors

In AUM mode, Oracle does not allow you to execute any manual management operations involving undo segments. You cannot manually offline or online them. Attempts to do so will result in an ORA-30019 error - Illegal rollback Segment operation in Automatic Undo mode.

Oracle will also disregard the command to use a specific undo segment for a transaction. An application using the command 'SET TRANSACTION USE ROLLBACK SEGMENT' will generate an ORA-30019 error. Although it is an option, changing the application code to avoid the ORA-30019 errors may be an impractical idea, and possibly impossible, if you have third party application code that cannot be changed.

Oracle provides an initialization parameter, UNDO_SUPPRESS_ERRORS, to address this issue. The default value is FALSE; meaning manual management operations against undo segments will terminate unsuccessfully. Setting this parameter to TRUE will cause Oracle to report success with all such operations without really carrying them out. The parameter can be set to the desired value dynamically at system and session levels. It is suggested that you set UNDO_SUPPRESS_ERRORS to TRUE when operating in AUM mode. However, leaving it set to its default value of FALSE can be an effective method to locate, and possibly correct, use of any such rollback segment handling in your application code, particularly during testing and acceptance scenarios.

Monitoring of Undo Segments and Undo Space

Even if you used AUM, you will still be performing tasks of monitoring the undo tablespace usage, adjusting undo retention time, sizing undo tablespace, etc., to support your workload. This is yet to be fully automated to eliminate the DBA's involvement. Oracle does provide you with some tools to perform these tasks efficiently.

Two new views, DBA_UNDO_EXTENTS, V\$UNDOSTAT, and an OEM display will offer information, such as, undo extents usage, undo generation rate and query execution times. Some of this information will be required to properly size the undo tablespace and set undo retention time to help minimize ORA-1555: snapshot too old errors.

You can also use existing views for undo usage monitoring. The V\$ROLLSTAT view provide information about undo segment extends, wraps and shrinks. The V\$TRANSACTION view provides information about the undo segments in use by transactions.

DBA_UNDO_EXTENTS View

This is a sister view of DBA_EXTENTS. It displays all the information that DBA_EXTENTS view provides, but exclusively for undo tablespaces. The view has following columns:

Name	Null?	Type
OWNER		CHAR(3)
SEGMENT_NAME	NOT NULL	VARCHAR2(30)
TABLESPACE_NAME	NOT NULL	VARCHAR2(30)
EXTENT_ID		NUMBER
FILE_ID	NOT NULL	NUMBER
BLOCK_ID		NUMBER
BYTES		NUMBER
BLOCKS		NUMBER
RELATIVE_FNO		NUMBER
COMMIT_JTIME		NUMBER
COMMIT_WTIME		VARCHAR2(20)
STATUS		VARCHAR2(9)

The columns COMMIT_JTIME and COMMIT_WTIME were meant for displaying the transactions' commit times in Julian and Wall Clock format. However, as of Oracle9i Release 2, these columns have been deprecated and contain NULL values. This information is internal to Oracle and may confuse others (see Metalink Note# 231509.1). The STATUS column indicates if the transaction using the extent is ACTIVE or not. When the transaction commits, the STATUS becomes UNEXPIRED to indicate that the UNDO_RETENTION period has not elapsed since the commit time. Once the UNDO_RETENTION time has elapsed since the commit time, the STATUS changes to EXPIRED.

All EXPIRED undo extents can be overwritten when undo space is required. However, all undo extents, EXPIRED or UNEXPIRED, can and will be dynamically transferred to other undo segments when Oracle requires the undo space for active transactions.

In some of my tests I found that the STATUS column reported the extent as EXPIRED as soon the short transaction was committed. The extent STATUS changed from ACTIVE to EXPIRED, rather than changing to UNEXPIRED prior to EXPIRED.

V\$UNDOSTAT View

The V\$UNDOSTAT view provides statistics for monitoring and tuning undo space. The view has following columns:

Column_Name	Type	Comment
BEGIN_TIME	DATE	-- Sample start date/time
END_TIME	DATE	-- Sample end date/time
UNDOTSN	NUMBER	-- Last Active Undo TS Number
UNDOBLKS	NUMBER	-- Undo blocks used
TXNCOUNT	NUMBER	-- Number of Transactions in sample
MAXQUERYLEN	NUMBER	-- MAX Query Length
MAXCONCURRENCY	NUMBER	-- Nbr of Concurrent transactions
UNXPSTEALCNT	NUMBER	-- Attempts to steal un-expired blocks
UNXPBLKRELCNT	NUMBER	-- Un-expired blocks released
UNXPBLKREUCNT	NUMBER	-- Un-expired blocks reused
EXPSTEALCNT	NUMBER	-- Attempts to steal expired blocks
EXPBLKRELCNT	NUMBER	-- Expired blocks released
EXPBLKREUCNT	NUMBER	-- Expired blocks reused
SSOLDERRCNT	NUMBER	-- Snapshot too old Error Count
NOSPACEERRCNT	NUMBER	-- No Space Left Error Count

The information provided by this view is critical to estimating the size of the undo tablespace, and in determining the undo retention time. According to

the Oracle Reference Guide, this view uses a 10-minute sample interval to display data. The BEGIN_TIME and END_TIME columns show the sample start and end time. The view displays data for the past seven days.

For sizing undo tablespace, you must monitor columns UNDOBLKS, TXNCOUNT, MAXQUERYLEN and all the columns from UNXPSTEALCNT to NOSPACEERRCNT. Column UNDOBLKS shows the number of new undo blocks used in the reported time interval and column TXNCOUNT shows the number of transactions counted in the reported time interval. Column MAXQUERYLEN shows the maximum length of a query in seconds, which completed during the sampled time interval. Oracle suggests that you set UNDO_RETENTION to a value equal to or greater than the value of MAX(MAXQUERYLEN) to minimize ORA-1555 errors. Any non-zero value in columns from UNXPSTEALCNT to NOSPACEERRCNT is an indication that the undo tablespace may be too small to support the workload during the sampled time interval.

According to the Oracle Concepts Guide, this view is available in both the AUM and MUM modes. However, in Oracle9i Release 1 this view returns one useless row in MUM mode. In Oracle9i Release 2 this view is not populated if you use MUM mode. So, to be able to view undo usage statistics using this view, you must operate the database in AUM mode. Although this view provides critical information to use in undo tablespace sizing calculations, the view has a couple of minor problems that are worth a mention.

First, in Oracle9i Release 2 this view does not display rows at exactly 10-minute sample intervals. I noticed that the 10-minute sample interval is followed only if Oracle detected at least one transaction in the sample interval. This may have been done intentionally to avoid loading rows with all zero column values.

Second, the value displayed by column TXNCOUNT does not correspond to the 10-minute sample interval if the value is non-zero. Instead, what is reported in this column is a cumulative value since the instance startup. So, you must do the math to find out how many transactions were detected in any given sample interval. This is a known issue (bug #3130916, #2506744) and there are no workarounds or patches as of this writing.

The following sample output from my Oracle 9.2.0.4 database on an AIX platform shows the above two problems. I observed this behavior on HP-UX and Windows 2000 platforms as well.

BEGIN_TIME	END_TIME	UNDOBLKS	TXNCOUNT
10/12/03 23:44:44	10/12/03 23:54:44	4	838684
10/12/03 22:24:44	10/12/03 23:44:44	0	0
10/12/03 22:14:44	10/12/03 22:24:44	2	838633
10/12/03 20:34:44	10/12/03 22:14:44	0	0
10/12/03 20:24:44	10/12/03 20:34:44	1	838609
10/12/03 20:14:44	10/12/03 20:24:44	2	838599

If you were to write queries to find the number of transactions for a given time interval, you must subtract the current value of TXNCOUNT from the previous interval, ignoring the rows reporting a zero in the TXNCOUNT column. To simplify this computation, I created a new view for V\$UNDOSTAT as shown below. This view can be used in queries to monitor undo generation rates and maximum query run times.

continued on page 26

```
CREATE OR REPLACE VIEW vw_undostat AS
SELECT *
FROM v$undostat
WHERE txncount != 0
/
```

Sizing the Undo Tablespace

As briefly mentioned in the previous section, the view V\$UNDOSTAT provides critical information for monitoring and sizing the undo tablespace. The goal is to have enough undo space to support the transaction loads as well as to retain committed information long enough so that the queries will not encounter ORA-1555 errors.

By monitoring the V\$UNDOSTAT view you can find out the number of transactions, the number of undo blocks used and the maximum query execution time. Once you have this information, you can use the following formula to find out how much undo space will be required to support the current workload.

```
Undo Space in Bytes = (UR * UDBPS * DB_Block Size) + Overhead
where
UR = Undo Retention Time in Seconds
UDBPS = Undo Blocks used Per Second
Overhead = One DB block for metadata
```

The following query shows the undo blocks per seconds (UDBPS) and max query run time from all the rows reported by V\$UNDOSTAT in the seven day period. Please note that the query uses a custom view discussed above:

```
SELECT
to_char(min(begin_time), 'MM/DD/YYYY HH24:MI:SS') "Begin Time",
to_char(max(end_time), 'MM/DD/YYYY HH24:MI:SS') "End Time",
(max(end_time)-min(begin_time))*24*60*60 "Seconds",
sum(undoblks) "UndoBlks",
ceil(sum(undoblks)/((max(end_time)-min(begin_time))*24*60*60)) "UDBPS",
(max(txncount) - min(txncount)) "Xactions",
max(maxquerylen) "MaxQryLen"
FROM
vw_undostatat;

Begin Time      End Time      Seconds UndoBlks UDBPS Xactions MaxQryLen
-----
10/06/2003 19:12:25 10/13/2003 18:32:25 602400 1814189 4 841975 19141
```

So, using an undo retention value of 20000 seconds (slightly more than MaxQryLen), an average undo blocks per second rate of 4, the size for the undo tablespace for an 8K (8192 bytes) block database using the above formula is computed to be $(20000 * 4 * 8192) + 8192 = 655368192$ bytes or about 625 MB.

You can use following query to quickly find out how much undo space will be required to support the current workload and current undo retention time:

```
SELECT rd AS "Retention",
(rd * (udbps * overhead) + overhead) as "Bytes"
FROM
(SELECT value AS RD
FROM v$parameter
WHERE name = 'undo_retention'),
(SELECT (sum(undoblks) /
sum((end_time - begin_time) * 86400)) as UDBPS
FROM v$undostat),
(SELECT value AS OVERHEAD
FROM v$parameter
WHERE name = 'db_block_size');
```

Retention	Bytes
3600	88515698.5

In the above example, 88515698.5 bytes, or approximately 84 MB, will be required to support 3600 seconds of undo retention time based on current rate of undo generation.

Using Oracle Enterprise Manager (OEM) to monitor undo generation rate and undo space requirements for the current undo retention time is very quick and easy. The screen displays the currently used undo tablespace name, current undo retention time, and the average and maximum undo generation rates. The graph shows the undo space sizing information based upon these undo generation rates. To see this screen in OEM, drill down to Instance | Configuration and then click on the UNDO tab.

Flashback Query

Flashback Query feature leverages AUM to provide a mechanism to view data as it existed at some point in the past. Users can use either wall clock time or the System Change Number (SCN) to access data consistent to a point in time in the past. Use of AUM is highly recommended if you plan on using the Flashback Query feature. But please remember that the Flashback Query is possible only if the undo segments still contain the undo information to reconstruct the data, as it existed in the past.

Flashback Query is a read-only operation; it cannot undo any past changes. However, it can provide data, as it existed in the past, which can be used to undo such changes, if necessary.

In my tests, I was able to successfully use this feature in MUM mode. However, MUM mode does not provide any mechanism to control the retention time of the undo information that is very critical for Flashback Query to work properly. In a properly configured AUM environment Flashback Query can be useful.

When to use Flashback Query

Flashback Query can be useful in situations where recovery from an accidental data modification is required, and the error is detected in a timely fashion, namely before the undo information is overwritten. You can also use Flashback Query to compare current and past data values, or to track data changes.

The export utility has an option to export data as of a timestamp or SCN. This is very useful when testing application code where restoring data prior to some critical operation is required. It is not necessary to export/import the entire schema, or backup/restore the entire database. Just export the affected tables at desired timestamps or SCNs, even after the test cycle has completed.

How Flashback Query Works

So, what magic does Oracle use to make Flashback Query work? There is no magic. Flashback Query simply relies on Oracle's existing read consistency model. Oracle uses rollback, or undo, segments to build a read consistent view of the data to the point in time that a query was started. Users always saw the committed data as of the start of their query.

Flashback Query uses the same mechanism to build the snapshot of the data consistent to a point in time as specified by the user. So, it is critical to have sufficient undo information for a Flashback Query to succeed.

Although one can specify a timestamp (date and time) when using the Flashback Query feature, Oracle must use the SCN number to reconstruct the data. To convert the timestamp to the nearest SCN, Oracle uses an internal table, SMON_SCN_TIME. It is owned by SYS. The background process SMON updates the table approximately every five minutes to record the current timestamp and SCN. Once the SCN is identified, Oracle reconstructs data as of that SCN using the information from the undo segments. This SYS.SMON_SCN_TIME table is the key player in Flashback Query operation. The sql.bsq script in the \$ORACLE_HOME/rdbms/admin directory contains more information on this internal table.

In Oracle9i Release 1, the flashback mode is enabled at the session level. The user must have the execute privilege on a DBMS_FLASHBACK package. The user must execute a procedure in this package to enable the flashback mode for his session by specifying a timestamp or the SCN. All queries will then access data, as it existed as of this SCN, or the closest SCN as of the timestamp. No DML is allowed in the session once the flashback mode is enabled.

Oracle9i Release 2 introduced the FLASHBACK object privilege and FLASHBACK ANY TABLE system privilege. The DBMS_FLASHBACK package is still available to enable flashback mode at the session level. The SQL syntax has been enhanced to offer the AS OF SCN/TIMESTAMP flashback clause to trigger flashback operation. It is unnecessary to enable flashback at the session level. Users can execute Flashback Query against only the tables for which they have been granted the FLASHBACK object privilege (grant flashback on table to user1). Users with flashback system privilege (grant flashback any table to user2) can use Flashback Query against any table, except the data dictionary tables. Flashback against data dictionary tables is not possible. The DBA role has the flashback system privilege. Since flashback is not enabled at session level, DML operations work as usual. In addition, DML operations as a part of a Flashback Query are also possible.

DBMS_FLASHBACK Package

The DBMS_FLASHBACK package contains the following three procedures and one function. The procedures allow users to enable and disable flashback mode while the function allows them to display the current SCN.

- ENABLE_AT_TIME

This procedure enables flashback mode for the session as of the supplied date and time. Multiple formats for specifying the date and time are supported. The following examples show how to use this procedure.

```
exec dbms_flashback.enable_at_time (past_date);
exec dbms_flashback.enable_at_time (to_timestamp - ('10-
MAR-2002:11:47:00', 'DD-MON-YYYY:HH24:MI:SS'));
exec dbms_flashback.enable_at_time (sysdate - 1/24);
```

- ENABLE_AT_SYSTEM_CHANGE_NUMBER

This procedure enables flashback mode for the session as of the supplied SCN. Please note that the SCN is fully spelled out in the procedure name. I hope you love typing!

```
exec dbms_flashback.enable_at_system_change_number(23488);
```

- DISABLE

This procedure simply disables the flashback mode. No arguments are required.

```
exec dbms_flashback.disable;
```

- GET_SYSTEM_CHANGE_NUMBER

If you want to use SCN to enable flashback for the session, or use it in the SQL script for Flashback Query, you must use this function to find its value. The following example shows how to do this. Please notice that the SCN can get pretty large and Oracle will use scientific notation to display it. You must adjust the numwidth setting to see the decimal value of the SCN.

```
SQL> SELECT dbms_flashback.get_system_change_number
2 FROM dual;

GET_SYSTEM_CHANGE_NUMBER
-----
                    5.98E+12

SQL> set numwidth 18
SQL> /

GET_SYSTEM_CHANGE_NUMBER
-----
                    5976736332383
```

Although, this package is available in Oracle9i Release 2, it is not necessary to enable the session for flashback queries using the above procedures. The new flashback clause in the SELECT statement allows flashback queries based on SCN or TIMESTAMP.

Using Flashback Query

Listing 1 is a demonstration script that shows how you can use Flashback query to access data as it existed at some time in the past. The script uses various options available to access such data. It also shows various options to recover data that was accidentally deleted.

Listing 1

```
SQL> set pages 50
SQL> set numwidth 18
SQL> set lines 132
SQL> set time on
18:04:54 SQL>
18:04:54 SQL> REM - Create a test table - DEPT
18:04:55 SQL> create table dept
18:04:55 2 as select *
18:04:55 3 from demo.departments;
Table created.
18:04:55 SQL> commit;
Commit complete.
```

continued on page 28

```
18:04:55 SQL> REM - Create another table to store date and SCN.
18:04:55 SQL> REM - We will use this information to flashback to.
18:04:56 SQL> create table save_date (fb_date date, fb_scn number);
Table created.
18:04:57 SQL> REM - See the contents of DEPT table.
18:04:57 SQL> select * from dept;
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

10 rows selected.

```
18:04:58 SQL> REM - Now we have DEPT table, and SAVE_DATE table
18:04:58 SQL> REM - to store the timestamp and SCN to flashback to.
18:04:59 SQL> REM - Let us see the latest entry in the SMON_SCN_TIME table
18:05:00 SQL> @@get_max_smon_scn.sql
18:05:00 SQL> REM - get_max_smon_scn.sql
18:05:00 SQL> select to_char(max(time_dp), 'MM/DD/YYYY HH24:MI:SS') timestamp,
18:05:00 2 max(scn_wrp * 4294967295 + scn_bas) scn
18:05:00 3 from
18:05:00 4 sys.smon_scn_time
18:05:00 5 /
TIMESTAMP SCN
-----
02/22/2004 18:03:07 9641769
```

```
18:05:01 SQL> REM - Now, wait for >5 minutes to make sure a new timestamp is
18:05:01 SQL> REM - recorded in the smon_scn_time table after creating
18:05:01 SQL> REM - DEPT table with our test data in it.
18:05:01 SQL> exec dbms_lock.sleep(360)
PL/SQL procedure successfully completed.
18:11:10 SQL> REM - Check the new entry in the SMON_SCN_TABLE
18:11:16 SQL> @@get_max_smon_scn.sql
18:11:16 SQL> REM - get_max_smon_scn.sql
18:11:16 SQL> select to_char(max(time_dp), 'MM/DD/YYYY HH24:MI:SS') timestamp,
18:11:17 2 max(scn_wrp * 4294967295 + scn_bas) scn
18:11:17 3 from
18:11:17 4 sys.smon_scn_time
18:11:17 5 /
TIMESTAMP SCN
-----
02/22/2004 18:08:25 9642837
```

```
18:11:18 SQL> REM - Save current timestamp and SCN in our SAVE_DATE table
18:11:21 SQL> insert into save_date
18:11:21 2 (select curr_date, curr_scn
18:11:21 3 from select sysdate curr_date from dual),
18:11:21 4 (select dbms_flashback.get_system_change_number curr_scn from dual)
18:11:21 5 );
1 row created.
18:11:21 SQL> commit;
Commit complete.
18:11:21 SQL> select to_char (fb_date, 'DD-MON-YYYY:HH24:MI:SS') CURR_DATE,
18:11:21 2 fb_scn CURR_SCN
18:11:21 3 from save_date
18:11:21 4 ;
CURR_DATE CURR_SCN
-----
22-FEB-2004:18:11:21 9643279
```

```
18:11:21 SQL> REM - We have data in DEPT table as of the above TIMESTAMP,
18:11:21 SQL> REM - and there is a new SCN TIMESTAMP prior to this entry in
18:11:21 SQL> REM - the SMON_SCN_TIME table.
```

```
18:11:21 SQL> pause
18:11:24 SQL> REM - Let us simulate an accidental delete from DEPT table.
18:11:24 SQL> pause
18:11:25 SQL> delete from dept;
27 rows deleted.
18:11:25 SQL> commit;
Commit complete.
18:11:28 SQL> REM - But we wanted to delete just a few rows!
18:11:29 SQL> REM - Use Flashback to get them back..
18:11:31 SQL> REM - Enable flashback mode (Oracle 9i R1 and R2 Syntax)
18:11:32 SQL> @@enable_fb.sql
18:11:32 SQL> declare
18:11:32 2 reco_date date;
18:11:32 3 begin
18:11:32 4
18:11:32 5 -- Get Date to flashback to.
18:11:32 6 select fb_date into reco_date
18:11:32 7 from save_date;
18:11:32 8
18:11:32 9 -- Enable flashback at the saved date and time
18:11:32 10 dbms_flashback.enable_at_time (reco_date);
18:11:32 11
18:11:32 12 end;
18:11:32 13 /
PL/SQL procedure successfully completed.
```

18:11:32 SQL> REM - We are in flashback mode. Let us select data from DEPT.

```
18:11:33 SQL> select * from dept;
DEPARTMENT_ID DEPARTMENT_NAME MANAGER_ID LOCATION_ID
-----
10 Administration 200 1700
20 Marketing 201 1800
30 Purchasing 114 1700
40 Human Resources 203 2400
50 Shipping 121 1500
60 IT 103 1400
70 Public Relations 204 2700
80 Sales 145 2500
90 Executive 100 1700
100 Finance 108 1700
```

10 rows selected.

```
18:11:33 SQL> REM - How do we insert this data back into our DEPT table?
18:11:34 SQL> REM - There are two ways to do it (Oracle 9i Rel 1)
18:11:34 SQL> REM - 1. Open a Cursor while in flashback mode
18:11:34 SQL> REM - End flashback mode (No DML allowed while in FB session)
18:11:34 SQL> REM - Fetch and save data from cursor
18:11:36 SQL> @@recover_fb.sql
18:11:36 SQL> declare
18:11:36 2 cursor fb_cur
18:11:36 3 is
18:11:36 4 select *
18:11:36 5 from dept;
18:11:36 6
18:11:36 7 dept_fb_rec dept%rowtype;
18:11:36 8
18:11:36 9 begin
18:11:36 10 -- Open cursor to read Dept table data
18:11:36 11 open fb_cur;
18:11:36 12
18:11:36 13 -- Now end flashback. ---- This is required.
18:11:36 14 -- Data is captured in Cursor.
18:11:36 15 dbms_flashback.disable;
18:11:36 16
18:11:36 17 -- Read through the cursor to capture the data and insert
18:11:36 18 -- into the table.
18:11:36 19 loop
18:11:36 20 fetch fb_cur into dept_fb_rec;
18:11:36 21 exit when fb_cur%notfound;
18:11:36 22 insert into dept
18:11:36 23 values (dept_fb_rec.department_id,
18:11:36 24 dept_fb_rec.department_name,
18:11:36 25 dept_fb_rec.manager_id,
18:11:36 26 dept_fb_rec.location_id);
18:11:36 27 end loop;
```

```

18:11:36 28
18:11:36 29 -- Close the cursor and commit work
18:11:36 30 close fb_cur;
18:11:36 31 commit;
18:11:36 32
18:11:36 33 end;
18:11:36 34 /
PL/SQL procedure successfully completed.
18:11:36 SQL> REM - Check the contents of the DEPT table.
18:11:38 SQL> select * from dept;

```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

10 rows selected.

```

18:11:39 SQL> REM - So, we have recovered the data.
18:11:40 SQL> REM - Let us delete from DEPT again and check out the other
18:11:40 SQL> REM - method to recover the data
18:11:41 SQL> delete from dept;
27 rows deleted.
18:11:41 SQL> commit;
Commit complete.
18:11:41 SQL> REM - 2. Valid in 9i R1 and 9i R2.
18:11:41 SQL> REM - No need to disable flashback.
18:11:41 SQL> REM - Create another table using SQL*Plus COPY command
18:11:42 SQL> @enable_fb.sql
18:11:42 SQL> declare
18:11:42 2 reco_date date;
18:11:42 3 begin
18:11:42 4
18:11:42 5 -- Get Date to flashback to.
18:11:42 6 select fb_date into reco_date
18:11:42 7 from save_date;
18:11:42 8
18:11:42 9 -- Enable flashback at the saved date and time
18:11:42 10 dbms_flashback.enable_at_time (reco_date);
18:11:42 11
18:11:42 20 end;
18:11:42 21 /
PL/SQL procedure successfully completed.
18:11:42 SQL>
18:11:42 SQL> REM - Now, we will recover data while in flashback mode.
18:11:43 SQL> REM - Check if we can see the DEPT table rows.
18:11:44 SQL> select * from dept;

```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

10 rows selected.

```

18:11:44 SQL> REM - Use SQL*Plus COPY command to create a new version of
18:11:44 SQL> REM - of the DEPT table called OLD_DEPT with prior data.
18:11:45 SQL> copy to kirti/kirti@OR92 -
18:11:45 > create old_dept -
18:11:45 > using -

```

```

18:11:45 > select * -
18:11:45 > from dept;
Array fetch/bind size is 15. (arraysize is 15)
Will commit when done. (copycommit is 0)
Maximum long size is 80. (long is 80)
Table OLD_DEPT created.
27 rows selected from DEFAULT HOST connection.
27 rows inserted into OLD_DEPT.
27 rows committed into OLD_DEPT at kirti@OR92.
18:11:45 SQL> REM - Let us check what's in OLD_DEPT table.
18:11:48 SQL> select * from old_dept;
select * from old_dept
*
```

```

ERROR at line 1:
ORA-01466: unable to read data - table definition has changed
18:11:48 SQL> REM - Oops!!! Why is this?
18:11:50 SQL> REM - We are in the flashback mode, OLD_DEPT was created by
18:11:50 SQL> REM - another SQL session (COPY command). It will be visible
18:11:50 SQL> REM - when we get out of current flashback mode.
18:11:51 SQL> exec dbms_flashback.disable;
PL/SQL procedure successfully completed.
18:11:51 SQL> REM - Check the OLD_DEPT now...
18:11:56 SQL> select * from old_dept;

```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

10 rows selected.

```

18:11:57 SQL> REM - We have the previous data. It can now be inserted into the
18:11:57 SQL> REM - original table. We will leave DEPT table empty to show how
18:11:57 SQL> REM - the flashback clause in 9i R2 works.
18:11:58 SQL> REM -
----- Syntax Examples -----
18:11:58 SQL> REM - In Oracle 9i Rel 2, the SQL syntax 'AS OF' is introduced
18:11:58 SQL> REM - to read data as it existed sometimes in the past.
18:11:58 SQL> REM - No need to be in flashback mode.
18:11:58 SQL> REM - No need to have execute privilege on DBMS_FLASHBACK.
18:11:58 SQL> REM -
-----
18:11:58 SQL> conn sys/sys as sysdba
Connected.
18:11:58 SQL> revoke execute on dbms_flashback from kirti;
Revoke succeeded.
18:12:03 SQL> conn kirti/kirti
Connected.
18:12:04 SQL> REM - New SQL syntax in Oracle 9i Release 2
18:12:04 SQL>
18:12:04 SQL> /*
18:12:04 DOC>----- Syntax Examples -----
18:12:04 DOC>select *
18:12:04 DOC> from dept
18:12:04 DOC> as of scn 1234;
18:12:04 DOC>
18:12:04 DOC>select *
18:12:04 DOC> from dept
18:12:04 DOC> as of timestamp to_timestamp('02/10/2003 15:10:05','MM/DD/YYYY
HH24:MI:SS');
18:12:04 DOC>-----*/
18:12:04 SQL> REM - Subquery is not allowed with 'as of' syntax to derive SCN
18:12:04 SQL> REM - or timestamp. But, we can use variables.
18:12:07 SQL> col fbscn noprint new_value fb_scn
18:12:07 SQL> col timestamp noprint new_value fb_timestamp
18:12:07 SQL> select fb_scn fbscn,
18:12:07 2 to_char(fb_date,'MM/DD/YYYY HH24:MI:SS') timestamp
18:12:07 3 from save_date
18:12:07 4 /

```

continued on page 30

```
18:12:07 SQL> REM - So, now we have the SCN and timestamp in variables.
18:12:07 SQL> REM - Let us see the old data using timestamp and new SQL syntax
18:12:08 SQL> select *
18:12:08 2 from dept
18:12:08 3 as of timestamp to_timestamp('&&fb_timestamp','MM/DD/YYYY HH24:MI:SS');
old 3: as of timestamp to_timestamp('&&fb_timestamp','MM/DD/YYYY HH24:MI:SS')
new 3: as of timestamp to_timestamp('02/22/2004 18:11:21','MM/DD/YYYY HH24:MI:SS')
```

DEPARTMENT_ID	DEPARTMENT_NAME	MANAGER_ID	LOCATION_ID
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	121	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700

27 rows selected.

```
18:12:09 SQL> REM - Oracle 9i R2 offers following syntax for DML operation
18:12:09 SQL> REM - with Flashback query.
18:12:09 SQL> /*
18:12:09 DOC> -----
18:12:09 DOC> insert into dept
18:12:09 DOC> select *
18:12:09 DOC> from dept
18:12:09 DOC> as of scn <variable>;
18:12:09 DOC> -----*/
18:12:09 SQL> REM - Check the row count in DEPT table
18:12:10 SQL>
18:12:10 SQL> select count(*)
18:12:10 2 from dept
18:12:10 3 ;
```

COUNT(*)
0

```
18:12:11 SQL> insert into dept
18:12:11 2 select *
18:12:11 3 from dept
18:12:11 4 as of scn &&fb_scn
18:12:11 5 ;
old 4: as of scn &&fb_scn
new 4: as of scn 9643279
```

27 rows created.

```
18:12:11 SQL> commit;
Commit complete.
18:12:11 SQL> REM - We have recovered the data in DEPT table.
```

Flashback Query Limitations

When considering the use of Flashback Query in your database and applications, you should keep in mind the following limitations:

- You cannot use Flashback Query to access past data for more than five days of instance uptime. The internal table SYS.SMON_SCN_TIME can have a maximum of 1440 rows. The table is used in a circular fashion to keep the timestamp to SCN mappings for a maximum of five days of instance uptime. If the SCN required to reconstruct the data is not within the range of SCN numbers stored in the SYS.SMON_SCN_TIME table, Flashback Query will fail, even if the UNDO_RETENTION is set large enough and you have a very large undo tablespace to retain the undo information and you kept track of all the old SCNs. The five days

limitation is based on five days of instance uptime and not five calendar days. If you kept the instance down for a month, you can certainly go as far back as 30 days or so.

- DDL operations that change table structure (or contents) will invalidate all the stored undo information for the table data. Flashback queries trying to access data beyond the DDL time will result in ORA-1466 errors.
- Oracle records the timestamp to SCN mapping in approximately five-minute intervals after instance startup. When using timestamp in the flashback operation, Oracle will round down the flashback time by up to five minutes while retrieving the SCN. For example, suppose Oracle has 10:00:00AM mapped to SCN 100, and 10:05:00AM mapped to SCN 110. When you use Flashback Query for any time between 10:00:00AM and 10:04:59AM, it will be mapped to SCN 100. But a Flashback Query for 10:05:00AM will be mapped to SCN 110. This behavior may not find the data you were looking for. You may need to adjust your flashback time and try again. Using SCN in the Flashback Query is, therefore, strongly recommended.
- You cannot use Flashback Query against a remote table using database links.
- The SYS user cannot use DBMS_FLASHBACK to enable flashback mode for the session, but can use the GET_SYSTEM_CHANGE_NUMBER function to retrieve current SCN. SYS can, however, use the flashback clause, AS OF, in a SQL script to perform flashback queries.

Conclusion

Automatic Undo Management and Flashback Query are very nice new features in Oracle9i. As with any new features, AUM and Flashback Query have had their share of initial glitches. But these features seem to be working well in Oracle9i Release 2 in most environments. However, I have heard of a few problems in heavily used OLTP environments, where thousands of concurrent transactions overwhelmed the automatic undo management mechanism and caused ORA-7445 error. When configured correctly, AUM will certainly minimize the occurrence of ORA-1555 and out of undo space errors. I wish Oracle had designed another method for controlling the use of undo space instead of making it a part of Resource Manager. Flashback Query also provides the DBAs and Developers with a new method to correct minor errors. DBAs need not be involved in such minor data recovery operations. However, enough experimentation in your own environment is always beneficial to fully understand how these features work.



About the Author

Kirtikumar Deshpande is a Senior Oracle Database Administrator for Verizon Information Services (<http://www.superpages.com>). He has over 24 years of IT experience including over 10 years as an Oracle DBA. He co-authored Oracle Press books, titled *Oracle Performance Tuning 101*, published in May 2001, and *Oracle Wait Interface - A Practical Guide to Performance Diagnostics & Tuning*, published in June 2004. You can reach him at Kirtikumar_Deshpande@yahoo.com.