The Ins and Outs of Oracle Total Recall

Björn Rost
portrix Systems GmbH
Hamburg

Tags

Total Recall, flashback, flashback queries, flashback archives

Introduction

There are many use cases for historical versions of data. Which result would would my query return if I had issued it last week? How did this data change over the past? Who modified this data? PORTRIX implemented Total Recall in their GPM software for logistic service providers and enabled customers the unique ability to view cargo freight rates over a time dimension. By using Total Recall this feature was implemented fast, easy and transparent to the existing code.

But dealing with flashback archives poses some new challenges for DBAs and developers. One needs to consider generating historical "test" data and also take great care when applying DDL to flashback enabled tables among other things.

Setting Up Flashback Archives

Total Recall is an optional feature that can be enabled at the table level. When set up properly it will allow flashback queries to access historical data in the format of flashback archives in addition to the information stored in the UNDO tablespace.

The first step will be to set up your flashback archive tablespace and the flashback archive itself. This is very well documented so I will not go into too much detail. Just know that it is advisable to use a separate tablespace or even more than one for flashback archives and that there are options to specify and modify a quota and a retention time. One needs to be careful with the quota since no more DML will be allowed to your base tables once the flashback archive is full or has reached its quota. One can also designate a system wide default flashback archive.

```
SQL> create tablespace fba1;

Tablespace created.

SQL> create flashback archive fba1 tablespace fba1 retention 1 year;

Flashback archive created.
```

Now before actually enabling this feature for a table there is at least one more thing that should be checked. Once a table has flashback enabled, Oracle will automatically create a new partition for each of these tables for each single day. That is done transparently and helps the database maintain performance and manageability for accessing data from the past. Each of these tables will get created with an initial extent and the minimum size of these initial extents was raised from 64kB to 8MB in

version 11.2.0.2. That means that each flashback enabled archive will use 8MB of data for each day even if you never make modifications to that table. Normal tables or other objects can override this default value by specifying the initial size in the CREATE TABLE clause. Unfortunately, the initial extent size of flashback archives cannot be modified in such a way and neither can table partitions be shrunk. A single static schema with twenty flashback enabled tables will use 60GB of storage space for a single year worth of flashback data – even without every actually changing that data. Therefor I highly advise to review the hidden initialization parameter _index_partition_large_extents. With that parameter you can globally set the size of initial partition extents to less than 8MB. Since this is a hidden parameter, users should consult with Oracle Support before actually changing the parameter in a production environment and also review ramifications for other parts of the database. With all that out of the way, tables can now be created or altered to use flashback archives.

```
SQL> create table t_fba (id number(19), description varchar2(256), value
number(19)) flashback archive fba1;

Table created.
```

Again, the whole syntax can be found in the documentation but there is not much more to be specified at this point. It is propably a could idea to do some DML of any row in the table to fully initialize the flashback archive, a more detailed explanation can be found below. While release 11.1 did not allow a lot of DDL on flashback enabled tables like altering columns, 11.2. allows almost all DDL operations on a table. It is still advisable to think carefully before modifying columns of such a table. There are operations that can lead to a lot of confusion later like adding, removing and re-adding a column. Or adding a colun, dropping a column and renaming the new column to the old name. These operations are legal but a flashback query with “WHERE COLNAME=somevalue” might return unexpected results because while the name of this column might be the same at two points in time, such a query would only search in one of them. Also note that new columns with default values (and possibly a not null constraint) will show null values if queried for a point in time where that column did not exist yet.

Flashback Queries

It is very convenient that queries against flashback archives use the same flashback query and flashback transaction syntax that was introduces in Oracle 10 and which used data in the UNDO tablespace to query historical data. There are plenty of resources describing the syntax so I will not go into it in too much detail. One difference between total recall enabled flashback queries and “regular” ones is that regular flashback cannot span DDL changes to a table but total recall does allow this. So the number and names of columns can differ in total recall queries.

There are two basic ways to perform flashback queries. A very easy and convenient way is to set a point in time at which you want to perform all subsequent queries. Simply execute the procedure DBMS_FLASHBACK.ENABLE_AT_TIME with the desired time at which you want to execute queries. For more granularity, there is also a function that takes an SCN as parameter. All subsequent queries of this session will be performed against the point in time specified through that procedure without any additional modification. The available table columns will also be exactly like in the past, meaning that additional columns added after the flashback time will not exist in queries which can lead to errors in application code. Also, this will perform flashback queries against all tables, even if they do not have flashback enabled which could lead to ORA-01555 errors.

The second method is to specify a point in time during the query with the AS OF TIMESTAMP or AS OF SCN clauses like this:

```
SELECT COUNT(*) FROM A AS OF TIMESTAMP (SYSTIMESTAMP-INTERVAL '2' DAY);
```

There is also a VERSIONS BETWEEN clause that will let you query all versions a piece of data has gone through. By using these clauses manually, one has much finer control over which tables should be queried from the past and which ones should not. One can also join two tables in two different points in time.

A slightly more complex setup would be to create parametrized views using a session context. This allows for a lot of flexibility in joins and syntax and also provides an easy way to fill not-null columns that return null for times when the column had not existed yet. Using the nvl() function in the view definition these columns can be forced to return a value.

Debugging execution plans

Sometimes, strange things can happen with Flashback queries against some tables (that have had FBA enabled for quite some time now) would only use UNDO data and therefor fail if we wanted to go further back in time than a few days. This can be observed by looking at the execution plan. Good old regular flashback query (using only undo data) would show up like a regular query and only report to touch the base tables. The actual access to undo segments is hidden here but that just makes sense because all the database is doing is accessing a consistent image at some past scn which is not much different from getting a consistent image of current blocks which might also use undo.
But when flashback archives are involved, we see access to tables named SYS_FBA_12345 and partitioning clauses. Here are two examples, the first one does not touch flashback archives even though they are set up correctly:

```
SQL> SELECT COUNT(*) FROM A AS OF TIMESTAMP (SYSTIMESTAMP-INTERVAL
'2' DAY);
   COUNT(*)
 ----------
        378
 Execution Plan
 ----------
 Plan hash value: 321999946
 ---------------------------
 | Id  | Operation         | Name      | Rows  | Cost (%CPU)| Time
|
 ---------------------------
 |   0 | SELECT STATEMENT  |           |     1 |     1   (0)|
00:00:01 |
 |   1 |  SORT AGGREGATE   |           |     1 |            |
|
 |   2 |   INDEX FULL SCAN| A_PK      |   378 |     1   (0)|
00:00:01 |
 ---------------------------
```

So this one does not access a flashback archive even though it does exist. And the query fails if we try to go back in time for more than a few hours or days:

```
SQL> SELECT COUNT(*) FROM A AS OF TIMESTAMP (SYSTIMESTAMP-INTERVAl
'3' DAY);
```

```
 SELECT COUNT(*) FROM A AS OF TIMESTAMP (SYSTIMESTAMP-INTERVAl '3'
DAY)
                                    *
 ERROR at line 1:
 ORA-01555: snapshot too old: rollback segment number 6 with name
 "_SYSSMU6_2228977038$" too small
```

If things are looking good and the query does access the flashback archives correctly, the execution plan should look more like this example:

```
SQL> SELECT COUNT(*) FROM B AS OF TIMESTAMP (SYSTIMESTAMP-INTERVAL '3'
DAY);

  COUNT(*)
 ----------
         1

Execution Plan
 ----------------------------------------------------------
 Plan hash value: 2293674641
 ------------------------------------------------------------------------------
-----------------------------------------------
 | Id  | Operation                          | Name                | Rows  |
Bytes | Cost (%CPU)| Time     | Pstart| Pstop |
 ------------------------------------------------------------------------------
-----------------------------------------------
 |   0 | SELECT STATEMENT                   |                     |     1 |
|     8  (13)| 00:00:01 |       |       |
 |   1 |  SORT AGGREGATE                    |                     |     1 |
|            |          |       |       |
 |   2 |   VIEW                             |                     |     2 |
|     8  (13)| 00:00:01 |       |       |
 |   3 |    UNION-ALL                       |                     |       |
|            |          |       |       |
 |*  4 |     FILTER                         |                     |       |
|            |          |       |       |
 |   5 |      PARTITION RANGE SINGLE        |                     |     1 |
28 |     3   (0)| 00:00:01 |   KEY |     1 |
 |*  6 |       TABLE ACCESS FULL            | SYS_FBA_HIST_94168  |     1 |
28 |     3   (0)| 00:00:01 |   KEY |     1 |
 |*  7 |     FILTER                         |                     |       |
|            |          |       |       |
 |*  8 |      HASH JOIN OUTER               |                     |     1 |
2031 |     5  (20)| 00:00:01 |       |       |
 |*  9 |       TABLE ACCESS BY INDEX ROWID| B          |     1 |     3 |
1   (0)| 00:00:01 |       |       |
 |  10 |        INDEX FULL SCAN             | B_PK       |     1 |       |
1   (0)| 00:00:01 |       |       |
 |* 11 |       TABLE ACCESS FULL            | SYS_FBA_TCRV_94168  |     1 |
2028 |     3   (0)| 00:00:01 |       |       |
 ------------------------------------------------------------------------------
-----------------------------------------------
```

```
Predicate Information (identified by operation id):
 ---------------------------------------------------



   4 - filter("TIMESTAMP_TO_SCN"(SYSTIMESTAMP(6)-INTERVAL'+03 00:00:00'
DAY(2) TO SECOND(0))<689239504)
    6 - filter("ENDSCN"<=689239504 AND ("OPERATION" IS NULL OR
"OPERATION"<>'D') AND
               "ENDSCN">"TIMESTAMP_TO_SCN"(SYSTIMESTAMP(6)-INTERVAL'+03
00:00:00' DAY(2) TO SECOND(0)) AND ("STARTSCN" IS NULL
               OR "STARTSCN"<="TIMESTAMP_TO_SCN"(SYSTIMESTAMP(6)-
INTERVAL'+03 00:00:00' DAY(2) TO SECOND(0))))
    7 - filter("STARTSCN"<="TIMESTAMP_TO_SCN"(SYSTIMESTAMP(6)-INTERVAL'+03
00:00:00' DAY(2) TO SECOND(0)) OR
               "STARTSCN" IS NULL)
    8 - access("T".ROWID=CHARTOROWID("RID"(+)))
    9 - filter("T"."VERSIONS_STARTSCN" IS NULL)
   11 - filter(("ENDSCN"(+) IS NULL OR "ENDSCN"(+)>689239504) AND
("STARTSCN"(+) IS NULL OR
               "STARTSCN"(+)<689239504))

Note
 -----
    - dynamic sampling used for this statement (level=2)

So why does Table A not use the flashback archives? After digging around a
little bit, I noticed that the corresponding SYS_FBA_21345 table did not
exist even though we already have a name for it in the data dictionary:

SQL> select ARCHIVE_TABLE_NAME FROM DBA_FLASHBACK_ARCHIVE_TABLES WHERE
TABLE_NAME = 'TABLE_A';

ARCHIVE_TABLE_NAME
 ---------------------------------------------------
 SYS_FBA_HIST_94151

SQL> SELECT * FROM SYS_FBA_HIST_94151;
 SELECT * FROM SYS_FBA_HIST_94151
                       *
 ERROR at line 1:
 ORA-00942: table or view does not exist
```

Querying the FBA table directly like that works well for Table B...
The explanation seems to be that the flashback archive table is not created until we actually update (or delete) stuff from the base table. And all we have done to this test instance until this point in time was insert rows. We never actually changed or deleted any rows. So all I needed to do was "update" a single row from that table and wait a few minutes for FBDA to do its job. Since FBDA is running asynchronously you have to wait a few minutes aftter the update until the flashback archive table gets generated.

```
SQL> UPDATE A SET ID=ID where ID=8;
```

```
1 row updated.
 SQL> COMMIT;
```

After that flashback queries against A worked just like expected. So remember to perform just any simple update after you enable flashback archives for a table or you might find yourself debugging exactly the same issue later.

Kontaktadresse:

Björn Rost

portrix Systems GmbH

Friesenweg 2b

D-22763 Hamburg

Telefon: +49 (0) 40-39805319

Fax: +49 (0) 40-39805329

E-Mail b.rost@portrix-systems.de

Internet: www.portrix-systems.de