Materialized Views: Oracle9i New Features

Christian Antognini
Trivadis AG
Zürich, Switzerland

INTRODUCTION

In a data warehouse environment the materialized views are used to pre-compute and store large joins and/or aggregations. The aim is to avoid the overhead associated with such time consuming operations during working hours, therefore to speed up the queries executed on the database.

In a distributed database environment the materialized views are used to replicate data between the different databases.

This publication covers only the most important new features in Oracle9i for data warehouses, i.e. the following themes are not covered:

- basic information about materialized views
- new features for distributed databases

A complete coverage of the new features in Oracle9i is given in our course *New Features Oracle9i*.

The schema used for the examples is Sales History, which is documented in the Oracle manual *Sample Schemas*.

ANALYSIS FEATURES

Not all materialized views support the same query rewrite modes or the same refresh modes (also named the materialized view capabilities).

In Oracle8i it is difficult to know exactly which features are supported by which materialized view. Therefore in Oracle9i to simplify the analysis of the materialized views, the following procedures have been added to the package DBMS_MVIEW:

- EXPLAIN_MVIEW
- EXPLAIN_REWRITE

DBMS_MVIEWS.EXPLAIN_MVIEW

This procedure enables you to know which features are supported for a materialized view. The generated output can be:

- written in the table MV_CAPABILITIES_TABLE
  (created with $ORACLE_HOME/rdbms/admin/utlxmv.sql)

```
DBMS_MVIEW.EXPLAIN_MVIEW(mv           IN VARCHAR2,
                         statement_id IN VARCHAR2 := NULL)
```

o returned as parameter in a varray

```
DBMS_MVIEW.EXPLAIN_MVIEW(mv           IN  VARCHAR2,
                         msg_array    OUT SYS.ExplainMVArrayType)
```

Let's look at an analysis example…

The materialized view (specified as first parameter) is analyzed, the second parameter (the value "1") is an analysis identifier, i.e. it is only important if many users are concurrently storing the generated output in the same table:

```
SQL> EXEC DBMS_MVIEW.EXPLAIN_MVIEW('FWEEK_PSCAT_SALES_MV', 1)
```

List the materialized view capabilities:

```
SQL> SELECT capability_name, possible, related_text, msgtxt
  2  FROM   mv_capabilities_table
  3  WHERE  mvname = 'FWEEK_PSCAT_SALES_MV'
  4  AND    mvowner = user
  5  AND    statement_id = 1
  6  ORDER BY capability_name, related_text;

CAPABILITY_NAME               P RELATED_TEXT MSGTXT
----------------------------- - ------------ ---------------------------------
PCT                           N
PCT_TABLE                     N PRODUCTS     relation is not a partitioned
                                             table
PCT_TABLE                     N SALES        no partition key or PMARKER in
                                             select list
PCT_TABLE                     N TIMES        relation is not a partitioned
                                             table
REFRESH_COMPLETE              Y
REFRESH_FAST                  N
REFRESH_FAST_AFTER_ANY_DML    N SH.TIMES     mv log does not have sequence #
REFRESH_FAST_AFTER_ANY_DML    N              see the reason why
                                             REFRESH_FAST_AFTER_ONETAB_DML is
                                             disabled
REFRESH_FAST_AFTER_INSERT     N SH.PRODUCTS  the detail table does not have a
                                             materialized view log
REFRESH_FAST_AFTER_INSERT     N SH.SALES     the detail table does not have a
                                             materialized view log
REFRESH_FAST_AFTER_INSERT     N SH.TIMES     mv log is newer than last full
                                             refresh
REFRESH_FAST_AFTER_ONETAB_DML N DOLLARS      SUM(expr) without COUNT(expr)
REFRESH_FAST_AFTER_ONETAB_DML N              see the reason why
                                             REFRESH_FAST_AFTER_INSERT is
                                             disabled
REFRESH_FAST_AFTER_ONETAB_DML N              SUM(expr) without COUNT(expr)
REFRESH_FAST_AFTER_ONETAB_DML N              COUNT(*) is not present in the
                                             Select list
REFRESH_FAST_PCT              N              PCT is not possible on any of the
                                             detail tables in the materialized
                                             view
REWRITE                       Y
REWRITE_FULL_TEXT_MATCH       Y
REWRITE_GENERAL               Y
REWRITE_PARTIAL_TEXT_MATCH    Y
REWRITE_PCT                   N              general rewrite is not possible
                                             and PCT is not possible on any of
                                             the detail tables
```

As you can see the information generated is very helpful. You can know precisely whether a specific capability can be used and, if it is not possible, the reason is given. All capabilities are fully described in the Oracle manual *Data Warehousing Guide* (chapter 8).

It should also be possible to know which rewrite mode is used by selecting the data dictionary, but...

```
SQL> SELECT rewrite_capability FROM user_mviews WHERE mview_name = 'A';

REWRITE_CAPABILITY
------------------
TEXTMATCH

SQL> SELECT capability_name, possible
  2  FROM   mv_capabilities_table
  3  WHERE  mview_name = 'A' AND capability_name = 'REWRITE_GENERAL';

CAPABILITY_NAME                P
------------------------------ -
REWRITE_GENERAL                Y
```

This is bug number 1862397, notice that the data dictionary views are wrong (they should be fixed in 9.0.2).

DBMS_MVIEWS.EXPLAIN_REWRITE

This procedure enables you to know why the cost based optimizer failed to rewrite a query using a materialized view. The generated output can be:

- written in the table REWRITE_TABLE
  (created with $ORACLE_HOME/rdbms/admin/utlxrw.sql)

```
DBMS_MVIEW.EXPLAIN_REWRITE(query        IN VARCHAR2,
                           mv           IN VARCHAR2,
                           statement_id IN VARCHAR2)
```

- returned as parameter in a varray

```
DBMS_MVIEW.EXPLAIN_REWRITE(query        IN  VARCHAR2(2000),
                           mv           IN  VARCHAR2(30),
                           msg_array    OUT SYS.ExplainMVArrayType)
```

Let's take a look at an analysis example…

It is possible to rewrite a specific query (specified as first parameter) with the materialized view FWEEK_PSCAT_SALES_MV? As before, the last parameter (the value "1") is an analysis identifier. Notice that the query is analyzed, not executed!

```
SQL> VAR query VARCHAR2(1000)

SQL> EXEC :query := 'SELECT   p.prod_subcategory, '|| -
>                   '         sum(s.amount_sold) dollars '|| -
>                   'FROM     sales s, products p '|| -
>                   'WHERE    s.prod_id = p.prod_id '|| -
>                   'GROUP BY p.prod_subcategory'

SQL> EXEC DBMS_MVIEW.EXPLAIN_REWRITE(:query, 'FWEEK_PSCAT_SALES_MV', 1)
```

Analyze the result:

```
SQL> SELECT message
  2  FROM   rewrite_table
  3  WHERE  mv_name = 'FWEEK_PSCAT_SALES_MV'
  4  AND    mv_owner = user
  5  AND    statement_id = 1;

MESSAGE
------------------------------------------------------------
QSM-01071: a lossy join in materialized view, FWEEK_PSCAT_SALES_MV,
           from table, SALES, not found in query
QSM-01052: referential integrity constraint on table, SALES, not
           VALID in ENFORCED integrity mode
QSM-01086: dimension(s) not present or not used in ENFORCED
           integrity mode
```

Once more the information generated is very helpful. In this case the "problem" is that the INIT.ORA parameter QUERY_REWRITE_INTEGRITY is set to ENFORCED, but the foreign key between SALES and TIMES was enabled with NOVALIDATE and marked as RELY.

```
SQL> SELECT validated, rely
  2  FROM user_constraints
  3  WHERE constraint_name = 'SALES_TIME_FK';

VALIDATED     RELY
------------- ----
NOT VALIDATED RELY
```

Since RELY is not supported in ENFORCED mode, the configuration must be modified:

```
SQL> ALTER SESSION SET query_rewrite_integrity = TRUSTED;
```

Now if the analysis is re-executed the query can be rewritten as expected:

```
SQL> DELETE rewrite_table
  2  WHERE  mv_name = 'FWEEK_PSCAT_SALES_MV'
  3  AND    mv_owner = user
  4  AND    statement_id = 1;

SQL> EXEC DBMS_MVIEW.EXPLAIN_REWRITE(:query, 'FWEEK_PSCAT_SALES_MV', 1)

SQL> SELECT message
  2  FROM   rewrite_table
  3  WHERE  mv_name = 'FWEEK_PSCAT_SALES_MV'
  4  AND    mv_owner = user
  5  AND    statement_id = 1;

MESSAGE
--------------------------------------------------------------------------------
QSM-01033: query rewritten with materialized view, FWEEK_PSCAT_SALES_MV
```

_EXPLAIN_REWRITE_MODE

The undocumented INIT.ORA parameter _EXPLAIN_REWRITE_MODE allows additional messages to be generated during the execution of the procedure EXPLAIN_REWRITE. The default value is FALSE, set it to TRUE in order to generate all messages. It can be changed dynamically at session or system level.

```
SQL> ALTER SESSION SET "_EXPLAIN_REWRITE_MODE" = TRUE;
```

FAST REFRESH

Enhancements in Materialized View Logs

To support fast refresh after update statements for all types of materialized views, a sequence value providing additional ordering information must be stored in the materialized view log. This is done with the new option SEQUENCE.

```
SQL> CREATE MATERIALIZED VIEW LOG ON customers WITH SEQUENCE, ROWID;
```

Materialized View Logs in the Data Dictionary

Finally the data dictionary views USER / ALL / DBA_MVIEW_LOGS have been added. In fact in Oracle8i the materialized view logs can only be found in USER / ALL / DBA_SNAPSHOT_LOGS.

Materialized View Logs: Bug Number 1862565

According to the documentation, the primary key should not be stored implicitly if WITH ROWID is specified, but...

```
SQL> CREATE MATERIALIZED VIEW LOG ON sales
  2  WITH SEQUENCE, ROWID, (amount_sold, time_id) INCLUDING NEW VALUES;

CREATE MATERIALIZED VIEW LOG ON sales
*
ERROR at line 1:
ORA-12014: table 'SALES' does not contain a primary key constraint
```

This is documentation bug number 1862565. When the comma before the column list is omitted, it works ☺, i.e. using a filter list itself, it adds the primary key.

```
SQL> CREATE MATERIALIZED VIEW LOG ON sales
  2  WITH SEQUENCE, ROWID (amount_sold, time_id) INCLUDING NEW VALUES;
```

Fast Refresh Enhancements

In Oracle8i a materialized view containing aggregations and joins has the following limitations:

- fast refresh is only supported after direct load insert
- refresh ON COMMIT is not supported

In Oracle9i these limitations no longer exist ☺. Therefore if you want to execute a script containing materialized aggregate join views written for Oracle8i, the materialized view logs must be created or modified to support these new features. If the materialized view logs are not usable or do not exist, some errors will be generated.

Let's look at a fast refresh example for a materialized aggregate join view:

```
SQL> CREATE MATERIALIZED VIEW LOG ON sales
  2  WITH SEQUENCE, ROWID (amount_sold, time_id) INCLUDING NEW VALUES;

SQL> CREATE MATERIALIZED VIEW LOG ON times
  2  WITH SEQUENCE, ROWID (calendar_month_desc, time_id)
  3  INCLUDING NEW VALUES;

SQL> CREATE MATERIALIZED VIEW month_sales_mv
  2  REFRESH FAST ON COMMIT WITH ROWID
  3  ENABLE QUERY REWRITE
  4  AS
  5  SELECT   count(*) cnt, count(s.amount_sold) cnt_amount_sold,
  6           t.calendar_month_desc, sum(s.amount_sold) dollars
  7  FROM     sales s, times t
  8  WHERE    s.time_id = t.time_id
  9  GROUP BY t.calendar_month_desc;

SQL> ANALYZE TABLE month_sales_mv COMPUTE STATISTICS;

SQL> SELECT * FROM month_sales_mv WHERE calendar_month_desc = '2000-01';

CNT        CNT_AMOUNT_SOLD CALENDAR    DOLLARS
---------- --------------- -------- ----------
     38878           38878 2000-01    26898412

SQL> INSERT INTO sales VALUES(1080, 180430, '01-JAN-00', 'I', 9999, 17, 1234);

Elapsed: 00:00:00.02

SQL> COMMIT;

Elapsed: 00:00:00.13

SQL> SELECT * FROM month_sales_mv WHERE calendar_month_desc = '2000-01';

       CNT CNT_AMOUNT_SOLD CALENDAR    DOLLARS
---------- --------------- -------- ----------
     38879           38879 2000-01    26899646
```

As you can see the refresh is really fast. Of course it also works for UPDATE, DELETE and direct INSERT statements. Therefore one of the biggest problems of the materialized views in Oracle8i is solved.

QUERY REWRITE

In Oracle8i the general query rewrite is not always used, thus the full or partial text match query rewrite must be used. For complex materialized views this is correct, but in Oracle8i also some "simple" materialized views cannot use it, e.g. in the following materialized view the "problem" is the predicate in the WHERE clause:

```
SQL> CREATE MATERIALIZED VIEW big_sales_mv
  2  REFRESH COMPLETE WITH ROWID ENABLE QUERY REWRITE AS
  3  SELECT PROD_ID, CUST_ID, TIME_ID, CHANNEL_ID, PROMO_ID,
  4         QUANTITY_SOLD, AMOUNT_SOLD
  5  FROM SALES
  6  WHERE amount_sold > 1000000;
```

In Oracle9i the general query rewrite is used for all materialized views that are not complex, i.e. materialized views which do not contain one of the following constructs:

- set operators (i.e. UNION, UNION ALL, INTERSECT and MINUS)
- START WITH clause
- CONNECT BY clause
- inline views
- self-joins

Notice that for inline views and self-joins there are some exceptions when full text match query rewrite can be used.

Let's take a look at an example where in Oracle9i the general query rewrite is used instead of the text match query rewrite:

- For the materialized view BIG_SALES_MV (defined above) in Oracle8i the text match query rewrite is used, and because in the following query the keyword WHERE is written in lowercase, the cost based optimizer cannot rewrite the query. Remember that when a text match query rewrite is used the two queries must be the same (an exception to this is the SELECT clause and white spaces):

```
SQL> SELECT * FROM SALES where amount_sold > 1000000;

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE
  PARTITION RANGE (ALL)
    TABLE ACCESS (FULL) OF 'SALES'
```

- In Oracle9i for such a query the general query rewrite, which is not case sensitive, is used. Therefore the query is rewritten:

```
SQL> SELECT * FROM SALES where amount_sold > 1000000;

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE
  TABLE ACCESS (FULL) OF 'BIG_SALES_MV'
```

- In Oracle9i the general query rewrite can also rewrite more queries, in fact in the following example where a BETWEEN is used instead of a “>” the query can still be rewritten ☺:

```
SQL> SELECT *
  2  FROM SALES
  3  WHERE amount_sold BETWEEN 1500000 AND 2000000;

Execution Plan
----------------------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE
  TABLE ACCESS (FULL) OF 'BIG_SALES_MV'
```

PARTITION CHANGE TRACKING

In a data warehouse many tables are partitioned and rolling windows are used for the data. Therefore when new data must be loaded the following operation should be executed:

- a new partition must be added
- the data must be loaded in the new partition (e.g. via direct insert or EXCHANGE PARTITION)
- the oldest partition must be dropped
- all dependent materialized views must be refreshed

In Oracle8i the materialized view is considered FRESH or STALE as a whole. Therefore after a partition management operation like CREATE or DROP partition:

- only complete refresh is supported (except when manual maintenance is possible)
- query rewrite is not possible before the refresh (except in STALE_TOLERATED mode)

In Oracle9i it is possible to track freshness to a finer grain, i.e. it is possible to identify which rows in the materialized view are affected by each master table partition. When a master table partition is modified, only the affected rows become STALE. Therefore after a partition management operation:

- fast refresh is supported, even if no materialized view logs exists!
- query rewrite on partially STALE materialized views is supported, even in ENFORCED and TRUSTED mode

This new feature is called: Partition Change Tracking (PCT)

To use it, there are the following rules and limitations:

- at least one master table must be range or composite partitioned
- the partition key must be composed of a single column
- the materialized view must contain the partition key or a partition marker. Notice that for a PCT query rewrite only the partition key can be used ☹
- if a GROUP BY clause is used, the partition key or the partition marker must be present in the GROUP BY clause
- data modification only occurs on partitioned tables
- PCT is not supported if the materialized view references views or remote tables, or if it contains outer joins

Of course in many cases, adding the partition key into the materialized view substantially increase the number of rows stored within it. To avoid this problem it is possible to add a partition marker instead of the partition key. A partition marker is nothing other than a partition identifier generated by the function DBMS_MVIEW.PMARKER. To generate the partition marker the function uses the ROWID, which is passed as parameter. Since the function is called for each row, don‘t underestimate the time needed to call it.

PCT Fast Refresh

Here is an example:

- create a materialized view with PCT support:

```
SQL> CREATE MATERIALIZED VIEW q_sales_mv
  2  ENABLE QUERY REWRITE
  3  AS
  4  SELECT t.calendar_quarter_desc, sum(s.amount_sold) dollars,
  5         dbms_mview.pmarker(s.rowid) pmarker
  6  FROM   sales s, times t
  7  WHERE  s.time_id = t.time_id
  8  GROUP BY t.calendar_quarter_desc, dbms_mview.pmarker(s.rowid);
```

- partition management operations (drop the oldest partition and add a new one):

```
SQL> ALTER TABLE sales DROP PARTITION sales_q1_1998;

SQL> ALTER TABLE sales SPLIT PARTITION sales_q4_2000
  2  AT   (to_date('01-JAN-2001','DD-MON-YYYY'))
  3  INTO (PARTITION sales_q4_2000, PARTITION sales_q1_2001);
```

- which data contains the materialized view before the refresh?

```
SQL> SELECT * FROM q_sales_mv ORDER BY calendar_quarter_desc;

CALENDAR_QUARTER_DESC    DOLLARS    PMARKER
-------------------- ---------- ----------
1998-Q1                  671804      12060
1998-Q2                  817610      12061
...
2000-Q3                  773658      12070
2000-Q4                  688011      12071
```

- execute the fast refresh and check to see if it has been successful, i.e. instead of the first quarter of 1998 the first quarter of 2001 should be stored in the materialized view:

```
SQL> EXEC DBMS_MVIEW.REFRESH('q_sales_mv','f')

SQL> SELECT * FROM q_sales_mv;

CALENDAR_QUARTER_DESC    DOLLARS    PMARKER
-------------------- ---------- ----------
1998-Q2                  817610      12061
1998-Q3               638343.15      12062
...
2000-Q4                  688011      12136
2001-Q1                  688011      12137
```

PCT Query Rewrite

Here is another example:

- create a materialized view with PCT support (remember that the partition marker cannot be used for query rewrite, therefore the partition key must be used ☹):

```
SQL> CREATE MATERIALIZED VIEW q_sales_mv
  2  ENABLE QUERY REWRITE
  3  AS
  4  SELECT t.calendar_quarter_desc, sum(s.amount_sold) dollars,
  5         s.time_id
  6  FROM   mysales s, times t
  7  WHERE  s.time_id = t.time_id
  8  GROUP BY t.calendar_quarter_desc, s.time_id;
```

- partition management operations (drop the oldest partition and add a new one):

```
SQL> ALTER TABLE sales DROP PARTITION sales_q1_1998;

SQL> ALTER TABLE sales SPLIT PARTITION sales_q4_2000
  2  AT   (to_date('01-JAN-2001','DD-MON-YYYY'))
  3  INTO (PARTITION sales_q4_2000, PARTITION sales_q1_2001);
```

- also in trusted mode, the query can be rewritten if the WHERE clause contains a condition on the partition key (at the parse time the cost based optimizer should know which data is selected, i.e. from which partitions):

```
SQL> ALTER SESSION SET query_rewrite_integrity = trusted;

SQL> SET AUTOTRACE TRACE EXP

SQL> SELECT t.calendar_quarter_desc, sum(s.amount_sold) "$"
  2  FROM   mysales s, times t
  3  WHERE  s.time_id = t.time_id
  4  AND    s.time_id BETWEEN to_date('01.01.2000','DD.MM.YYYY')
  5                   AND     to_date('31.03.2000','DD.MM.YYYY')
  6  GROUP BY t.calendar_quarter_desc;

Execution Plan
----------------------------------------------
SELECT STATEMENT Optimizer=CHOOSE
  SORT (GROUP BY)
    TABLE ACCESS (FULL) OF 'Q_SALES_MV'
```

- but if the WHERE clause contains no condition on the partition key an error is generated ☹:

```
SQL> SELECT t.calendar_quarter_desc, sum(s.amount_sold) "$"
  2  FROM   mysales s, times t
  3  WHERE  s.time_id = t.time_id
  4  AND    t.calendar_quarter_desc = '2000-Q1'
  5  GROUP BY t.calendar_quarter_desc;
GROUP BY t.calendar_quarter_desc
    *
ERROR at line 5:
ORA-00918: column ambiguously defined
```

This is bug# number 1937020. To solve it, the parameter _SUBQUERY_PRUNING_MV_ENABLED must be set to FALSE (should be fixed in 9.0.1.1).

CONCLUSION

Anybody who is working or planning to introduce materialized views should use Oracle9i to take advantage of a:

- better query rewrite
- better fast refresh
- better manageability

So there is no good reason to use Oracle8i anymore, move on to Oracle9i! ☺