Maximizing Materialized Views

John Jay King
King Training Resources
john@kingtraining.com
Download this paper and code examples from:
http://www.kingtraining.com
Session Objectives

• Learn how to create and use Materialized Views and Materialized View logs.
• Set Materialized Views to refresh in a variety of ways.
• Understand Materialized View Groups.
• Exploit Oracle’s Query Rewrite capability.
• Increase application performance using Materialized Views.
What are Materialized Views?

- Materialized Views (MViews) allow a view query’s results to be physically stored in the database
  - Originally introduced in Oracle8i, based upon SNAPSHOTs
  - Normally only a view’s SELECT statement is stored in the database; the result set is “Materialized” (recreated) each time the view is accessed
  - Materialized Views are based upon a SELECT too; but the “materialized” result set is stored in the database as well as the defining SELECT
  - View materialization is refreshed periodically based upon time criteria, upon commit of changes, or upon demand
  - Materialized View (MView) data is "old" until the view is refreshed (good idea to use temporal names like daily_sales_summary)
  - MVViews provide substantial performance gains since the result set is only materialized once rather than for each repeated use
  - To further speed things, indexes may be defined for MVViews
MView Feature Overview

- Used like any Table or View, “transparent” to user
- Physically store data, can be indexed
- Frequently used to make local copies of remote data
- Dramatically improve performance of queries that make repeated use of non-volatile result sets (ideal for Business Intelligence and Data Warehouses)
- Usually used for aggregate/summary (GROUP BY) output
- Need to be refreshed:
  - Based upon date/time
  - When view data changes are committed
  - Upon Demand
- Created using CREATE MATERIALIZED VIEW
Three Types of MView

• Materialized Views may be separated into three basic types:
  – Materialized Views containing aggregate data
  – Materialized Views containing data from joins (but without aggregates)
  – Materialized Views querying from Materialized Views (nested MViews)
CREATE MATERIALIZED VIEW

create materialized view daily_dept_summary
  build immediate
  refresh complete
  enable query rewrite
as
select dname, count(empno) nbremps,
     sum(coalesce(sal,0))
     + coalesce(comm,0)) totpay
from emp e full join dept d
  on e.deptno = d.deptno
group by dname
order by nbremps desc, totpay desc

- Build query results immediately (may be deferred)
- Each refresh completely replaces data
- Query Rewrite is enabled
Refresh criteria

• When defining MViews; it is important to consider different factors impacting data refresh:
  – How should the data be refreshed?
    • ON COMMIT Refresh when underlying data changes (must be "fast refresh" capable)
    • ON DEMAND Refreshed using DBMS_MVIEW
    • START WITH Refresh using date/time calculation and/or NEXT
  – What type of refresh mechanism should be used?
    • COMPLETE Re-execute MView query
    • FAST Incremental changes using MView log
    • FORCE FORCE fast if possible; complete otherwise
    • NEVER MView does not get refreshed
  – May "trusted constraints" be used
    • QUERY_REWRITE_INTEGRITY = TRUSTED
    • QUERY_REWRITE_INTEGRITY = ENFORCED
Using DBMS_MVIEW

- DBMS_MVIEW includes several procedures including:
  - REFRESH Refresh named mview
  - REFRESH_ALL_MVIEWS Refresh all mviews
  - REFRESHDEPENDENT Refresh dependent mviews

```sql
begin
  dbms_mview.refresh('daily_dept_summary');
end;
/
```

- Be Careful! This packaged procedure COMMITs changes in the active transaction as part of its execution
Primary Key / ROWID

- MViews may specify the use of keys:
  - WITH PRIMARY KEY (default)
    - Base table must include primary key
    - All primary key columns must be used in MView query (without modification)
  - WITH ROWID
    - MView must be based upon single table
    - MView query may not use:
      - Aggregate functions or GROUP BY
      - DISTINCT
      - Distinct or aggregate functions
      - CONNECT BY
      - Joins
      - Subqueries
      - Set operations
Using ROLLBACK Segment

- This clause is still supported for backward compatibility; requires ROLLBACK segments
- Most installations use Undo Tablespaces and automatic undo mode making this clause irrelevant
Fast Refresh Restrictions

• Each MView query table must have Materialized View Log
• Fast Refresh is possible only for queries that do not have:
  – RAW or LONG RAW data
  – Non-deterministic data like SYSDATE
  – SELECT list subqueries
  – Analytic functions (e.g. RANK, DENSE_RANK)
  – MODEL clause
  – HAVING with subquery
  – Subqueries using ANY, ALL, or NOT EXISTS
  – START WITH / CONNECT BY
  – Tables from multiple sites
• Other (more complex) restrictions exist; see the Oracle Data Warehousing Guide and SQL Reference
Materialized View Logs

- Materialized view logs are required to perform FAST REFRESH or to use PCT (Partition Change Tracking) REFRESH
- Use CREATE MATERIALIZED VIEW LOG to define a log for each base table that might be changed (not on the MView)
- If FAST REFRESH is specified for nested Materialized Views; ROWID is normally required and all columns referenced in the nested MView must be included

```sql
CREATE MATERIALIZED VIEW LOG ON sales
  WITH ROWID
  (prod_id, cust_id, time_id,
   channel_id, promo_id,
   quantity_sold, amount_sold)
  INCLUDING NEW VALUES;
```
• Query Rewrite provides an added benefit to MViews; Oracle uses Materialized Views to “rewrite” queries
• When end user queries access tables and/or views used in a Materialized View; the query rewrite mechanism in the Oracle server can automatically rewrite the SQL query to use the MView instead
• Query Rewrite improves query result time transparently
• System-level or Session-level must specify: QUERY REWRITE ENABLED = TRUE
• If a data warehouse MView references data from a Dimension (“trusted” data) also required for rewrite; the System-level or Session-level must specify: QUERY REWRITE INTEGRITY = TRUSTED
Query Rewrite Restrictions

• To use Query Rewrite, an MView’s SELECT statement’s expressions must be repeatable and cannot include:
  – Non-deterministic user-defined functions
  – Oracle Sequence values
  – Current date/time variables (e.g. SYSDATE)
  – Other “current” values (e.g. USER)
  – SAMPLE

• DISABLE QUERY REWRITE is the default; each Materialized View should specify ENABLE QUERY REWRITE

• Query Rewrite is performed as part of statement optimization and requires that statistics exist for the Materialized View (use DBMS_STATS)
• What if the Materialized View's data is no longer current? (i.e. underlying tables/views have changed without a refresh of the Materialized View; the MView is “stale”)

• Oracle’s ability to rewrite a “stale” MView depends upon the value of QUERY_REWRITE_INTEGRITY:
  – ENFORCED (default): materialized view used if data is not stale and does not involve any “trusted” relationships (like Dimensions)
  – STALE_TOLERATED: materialized view used even if detail data has changed
  – TRUSTED: materialized view used if data is not stale but query rewrite might use “trusted” relationships like Dimensions that have not been validated
Given the following Materialized View definition:

```sql
create materialized view dept_summary_mview
  build immediate
  refresh complete
  enable query rewrite
as
select dname, count(empno) nbremps,
     sum(coalesce(sal, 0) +
         coalesce(comm, 0)) totpay
from emp e full join dept d
  on e.deptno = d.deptno
  
group by dname
order by nbremps desc, totpay desc
```
• This query is rewritten to use the Materialized View (DEPT_SUMMARY_MVIEW will be joined to DEPT rather than joining DEPT to the EMP table)

```sql
select dname, count(empno)
    from emp, dept
    where emp.deptno = dept.deptno
    group by dname
```
• If you expect a rewrite to occur but the optimizer chooses a different path, the DBMS_MVIEW.EXPLAIN_REWRITE procedure may be used (first, run <oraclehome>/rdbms/admin/utlxrw.sql to build a REWRITE_TABLE; see next page)

```
SQL> execute dbms_mview.explain_rewrite('select deptno,count(*) from emp group by deptno');
SQL> select message from rewrite_table:
MESSAGE
QSM-01009: materialized view, DEPT_SUMMARY_MVIEW2, matched query text
```
# Query Rewrite Table

<table>
<thead>
<tr>
<th>Name</th>
<th>Null?</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>STATEMENT_ID</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>MV_OWNER</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>MV_NAME</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>SEQUENCE</td>
<td></td>
<td>NUMBER (38)</td>
</tr>
<tr>
<td>QUERY</td>
<td></td>
<td>VARCHAR2 (2000)</td>
</tr>
<tr>
<td>MESSAGE</td>
<td></td>
<td>VARCHAR2 (512)</td>
</tr>
<tr>
<td>PASS</td>
<td></td>
<td>VARCHAR2 (3)</td>
</tr>
<tr>
<td>MV_IN_MSG</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>MEASURE_IN_MSG</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>JOIN_BACK_TBL</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>JOIN_BACK_COL</td>
<td></td>
<td>VARCHAR2 (30)</td>
</tr>
<tr>
<td>ORIGINAL_COST</td>
<td></td>
<td>NUMBER (38)</td>
</tr>
<tr>
<td>REWRITTEN_COST</td>
<td></td>
<td>NUMBER (38)</td>
</tr>
<tr>
<td>FLAGS</td>
<td></td>
<td>NUMBER (38)</td>
</tr>
<tr>
<td>RESERVED1</td>
<td></td>
<td>NUMBER (38)</td>
</tr>
<tr>
<td>RESERVED2</td>
<td></td>
<td>VARCHAR2 (10)</td>
</tr>
</tbody>
</table>
Enabling Query Rewrite

• If query rewrite is not set at the System level, it may be set at the Session level (if your userid is allowed to ALTER SESSION)

```
ALTER SESSION SET query_rewrite_integrity=TRUSTED;
ALTER SESSION SET query_rewrite_enabled=FORCE;
show parameters query
```
Disabling Query Rewrite

• Occasionally it might be useful disable the query rewrite capability for a Materialized View

```
ALTER MATERIALIZED VIEW dept_summary_mview
disable query rewrite;
```
Using Prebuilt Tables, Table

- Basing a materialized view upon an existing table (ON PREBUILT TABLE) allows the use of existing tables and indexes
- Here is some syntax to create a table upon which a view may be based, this creates a normal table with no special features

```sql
create table dept_summary_tab
as
    select dept.deptno,
        dname,
        count(*) nbr_emps,
        sum(nvl(sal, 0)) tot_sal
    from scott.emp emp,
        scott.dept dept
    where emp.deptno(+)= dept.deptno
    group by dept.deptno, dname;
```
create materialized view dept_summary_tab
  on prebuilt table
  with reduced precision
  refresh start with sysdate next sysdate + 1
 as
  select dept.deptno
    ,dname
    ,count(*) nbr_emps
    ,sum(nvl(sal,0)) tot_sal
  from scott.emp emp
    ,scott.dept dept
  where emp.deptno(+) = dept.deptno
  group by dept.deptno,dname;

• In this case, the MView uses the same query as the one used to create
  the original table, this is not required
• Table and Materialized View must use the same name and schema
• WITH REDUCED PRECISION allows a refresh to work properly even if
  some columns generate different precision than originally defined
Indexing

- Materialized Views are usually used for queries.
- Query execution may be improved if a single-column bitmap index is defined for each "key" column in the MVView.
- If an MVView containing aggregates is set for FAST refresh; an index is created automatically unless USING NO INDEX is specified in CREATE MATERIALIZED VIEW.
- Note: When a partitioned MVView is refreshed, indexes must be rebuilt before FAST Refresh will work.
Update

• Specify FOR UPDATE to allow update of a Materialized View:
  – Primary key
  – Rowid
  – Subquery
  – Object

• When using Advanced Replication the allowed changes are propagated to the master
Materialized View Query

- MView SELECT defines a query that creates the result set to be “materialized” and stored.
- The SELECT statement may reference:
  - Any number of tables joined together
  - Views, Inline views (subqueries in the FROM clause of a SELECT statement), Subqueries, and Materialized Views can all be joined or referenced in the SELECT clause.
- The SELECT statement may not:
  - Use a subquery in the SELECT list of the defining query (subqueries may be used elsewhere; for example in the WHERE clause).
MView Refresh Groups

- Oracle’s Advanced Replication features allow definition of Materialized View Refresh Groups.
- Oracle can refresh collections of MViews in "Refresh Groups" to maintain Referential Integrity and Read Consistency.
- When two (or more) MViews should be “in-synch” a “Refresh Group” should be used.
- After refreshing a "Refresh Group" all MViews in the group correspond to a consistent point in time.
BEGIN
    DBMS_REFRESH.MAKE (
        name => 'myschema.mymviewrefgroup',
        list => '',
        next_date => SYSDATE,
        interval => 'SYSDATE + 1',
        implicit_destroy => FALSE,
        rollback_seg => '',
        push_deferred_rpc => TRUE,
        refresh_after_errors => FALSE);
END;
/

BEGIN
    DBMS_REFRESH.ADD (  
        name => 'myschema.mymviewrefgroup',  
        list => 'sh.sales_mview',  
        lax => TRUE);  
END;
/

BEGIN
    DBMS_REFRESH.ADD (  
        name => 'myschema.mymviewrefgroup',  
        list => 'sh.countries_mview',  
        lax => TRUE);  
END;
/
EXECUTE DBMS_REFRESH.REFRESH ('myschema.mymviewrefgroup');
MViews in the Catalog

- The catalog provides support for MViews
  - ALL_BASE_TABLE_MVIEWS
  - ALL_MVIEWS
  - ALL_MVIEW_AGGREGATES
  - ALL_MVIEW_ANALYSIS
  - ALL_MVIEW_COMMENTS
  - ALL_MVIEW_DETAIL_PARTITION
  - ALL_MVIEW_DETAIL_RELATIONS
  - ALL_MVIEW_DETAIL_SUBPARTITION
  - ALL_MVIEW_JOINS
  - ALL_MVIEW_KEYS
  - ALL_MVIEW_LOGS
  - ALL_MVIEW_REFRESH_TIMES
  - ALL_REGISTERED_MVIEWS
Important ALL_MVIEW Columns

- MVIEW_NAME
- QUERY
- REWRITE_ENABLED
- REFRESH_MODE
- REFRESH_METHOD
- BUILD_MODE
- FAST_REFRESHABLE
- LAST_REFRESH_DATE
- STALENESS
- STALE_SINCE
The main performance gain of Materialized Views is obtained by NOT re-materializing result sets repetitively.

If a regular View and Materialized View use the same query:

```sql
SELECT   substr(country_name,1,20) country,
          substr(prod_name,1,15) product ,sales.prod_id prodid,
          calendar_year year ,SUM(amount_sold) tot_amt,
          SUM(quantity_sold) tot_qty ,COUNT(amount_sold) tot_sales
FROM sh.sales sales join sh.times times
    on sales.time_id = times.time_id
    join sh.products products
    on sales.prod_id = products.prod_id
    join sh.customers customers
    on sales.cust_id = customers.cust_id
    join sh.countries countries
    on customers.country_id = countries.country_id
GROUP BY country_name,prod_name,
          sales.prod_id,calendar_year
ORDER BY country,product,year;
```
The two queries below process greatly different numbers of rows:

```sql
select country, year, product, tot_sales
  from sales_view
  where year = '2003'
  order by tot_sales, country;
drop view sales_view;
```

- Reads thousands of rows to generate the results

```sql
select country, year, product, tot_sales
  from sales_mview
  where year = '2003'
  order by tot_sales, country;
drop view sales_view;
```

- Reads 50 rows to generate the results
Recent Development

- Oracle 11g Materialized View changes
  - Query Rewrite will support queries containing inline views (SELECT in FROM subquery)
  - Query Rewrite can now rewrite queries referencing remote tables
  - Refresh now supports:
    - Automatic index creation for UNION ALL materialized views
    - Query rewrite during a materialized view refresh (single)
    - Materialized view refresh with set operators
    - Partition Change Tracking (PCT) can track refresh of MViews with UNION ALL
    - Catalog views have been expanded to include partition staleness
- Oracle 10 Materialized View changes
  - Materialized view fast refresh may involve multiple tables (partitioned or not)
  - Materialized View Fast Refresh involving multiple tables no longer always requires Materialized View Log (use DBMS_MVIEW.EXPLAIN_REWRITE)
  - Query rewrite performance improved because Oracle 10g query rewrite may use multiple materialized views to rewrite a query
Wrapping it all Up

- Materialized Views reduce the impact of frequently executed queries by storing results and refreshing them on a selected basis
- Materialized Views may be indexed
- Materialized Views may be synchronized
- Materialized Views are best suited for a predominately read-only environment like Business Intelligence
Training Days 2008

Mark your calendar for:
February 13-14 2008!
Maximizing Materialized Views

To contact the author:
John King
King Training Resources
6341 South Williams Street
Littleton, CO 80121-2627 USA
1.800.252.0652 - 1.303.798.5727
Email: john@kingtraining.com

Today’s slides and examples are on the web:
http://www.kingtraining.com

Thanks for your attention!