



Welcome to BBN Technologies

Innovation that Matters

MAOP Brown Bag Lunch Series
Dec 6, 2007

 www.bbr.com

Oracle Materialized Views

- What is a Materialized View?
- How are Materialized Views Used?
- Refreshing Materialized Views
- Types of Materialized Views
- Materialized Views with Aggregates
- Materialized Views with Only Join
- Nested Materialized Views
- Summary
- Note: this talk references the following:
 - Oracle Database Data Warehouse Guide 10g Release 2
 - Oracle Warehouse Administration: Student Guide
 - Oracle's Demo SH schema

What is a Materialized View

- A materialized view is similar to a view except the database query result is cached as a concrete table.
- The SELECT clause in a materialized view creation statement defines the data that the materialized view is to contain. The following can be joined or referenced in a materialized view's SELECT clause:
 - Tables
 - Views
 - Inline views
 - Subqueries
 - Materialized views

How are Materialized Views Used

- Materialized view are used in:
 - Data Warehouses
 - Distributed Computing

MViews for Data Warehouses

- Data warehouses use materialized views to:
 - Pre-compute and store aggregated data
 - Pre-compute joins with or without aggregation
 - Eliminate the overhead associated with expensive joins and aggregations for a large or important class of queries.
 - Query rewrite

MViews for Data Warehouses

- Query rewrite example 1:
- Before materialized views were available:
 - `sql> select c.cust_id, sum(amount_sold)
from sales s, customers c
where s.cust_id=c.cust_id
group by c.cust_id;`
 - `sql> create table cust_sales_sum as
select c.cust_id, sum(amount_sold) as amount
from sales s, customers c
where s.cust_id = c.cust_id
group by c.cust_id;`
 - `sql> select * from cust_sales_sum;`

MViews for Data Warehouses

- Query rewrite example 1 (continued):
- With materialized views:
 - sql> create materialized view cust_sales_mv
enable query rewrite as
select c.cust_id, sum(amount_sold)
from sales s, customers c
where s.cust_id=c.cust_id
group by c.cust_id;
 - sql> select c.cust_id, sum(amount_sold)
from sales s, customers c
where s.cust_id = c.cust_id
group by c.cust_id;
 - **select statement**
table access (full) of cust_sales_mv

MViews for Data Warehouses

- Query rewrite example 2:
- With materialized views:
 - sql> create materialized view cust_sales_mv2
enable query rewrite as
select c.cust_id, s.channel_id, sum(amount_sold)
from sales s, customers c
where s.cust_id=c.cust_id
group by c.cust_id, s.channel_id;
 - sql> select c.cust_id, sum(amount_sold)
from sales s, customers c
where s.cust_id = c.cust_id
group by c.cust_id;
 - select cust_id,sum(amount)
from cust_sales_mv2 group by cust_id;

MViews for Data Warehouses

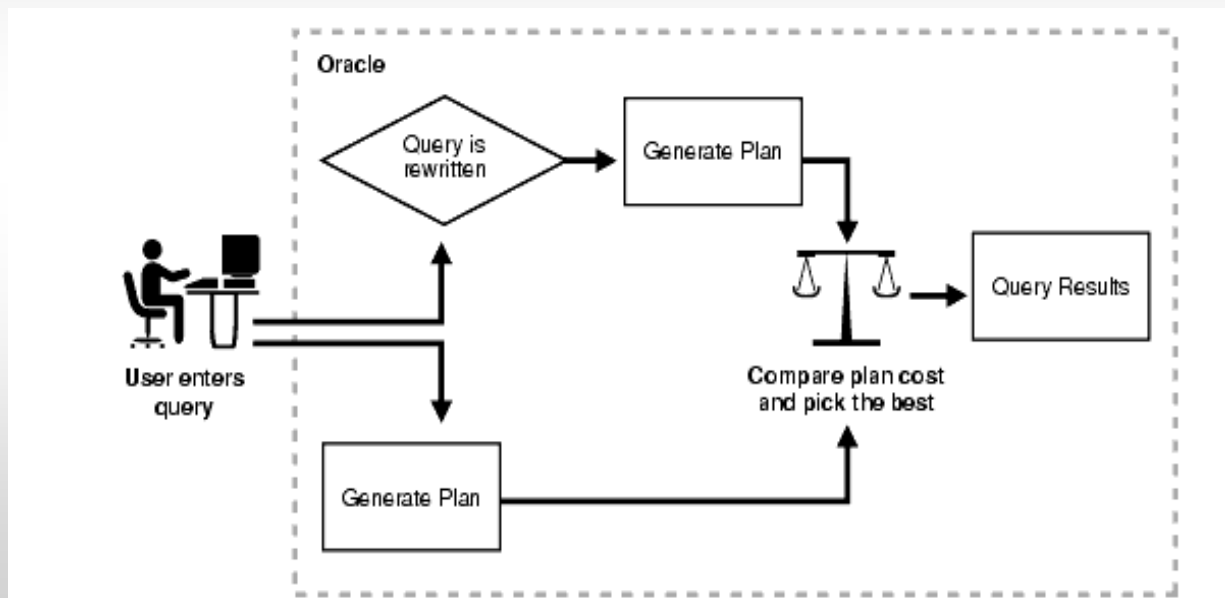
- Query rewrite example 2 explained:
- `Sql> select * from cust_sales_mv2;`

cust_id	channel_id	amount
2	2	13700.30
2	3	34036.78
2	4	17783.09
- `sql> select c.cust_id, sum(amount_sold)`
`from sales s, customers c`
`where s.cust_id = c.cust_id`
`group by c.cust_id;`

cust_id	amount
2	65520.17
- `select cust_id,sum(amount)`
`from cust_sales_mv2 group by cust_id;`

MViews for Data Warehouses

- Query rewrite example (continued)
- Figure from Oracle Database Warehouse Guide 10g Release 2:



MViews for Distributed Computing

- Distributing environments use materialized views to:
 - Replicate data at remote sites
 - Synchronize updates done at those site
 - Provide local access to data that otherwise would have to be accessed from remote sites

Refreshing Materialized Views

- Refresh Modes:
 - Commit
 - On demand (default)
- Refresh Methods:
 - Complete
 - Fast
 - Force (default)
 - Never

Types of Materialized Views

- Materialized Views with Aggregates
- Materialized Views Containing Only Joins
- Nested Materialized Views

Materialized Views with Aggregates

- Example 3: join with aggregate
 - Step 1: Create materialized view logs on products and sales tables:

```
sql> CREATE MATERIALIZED VIEW LOG ON products
      WITH SEQUENCE, ROWID
      (prod_id, prod_name, prod_desc, prod_subcategory,
       prod_subcategory_desc, prod_category, prod_category_desc,
       prod_weight_class, prod_unit_of_measure,
       prod_pack_size, supplier_id, prod_status, prod_list_price,
       prod_min_price)
      INCLUDING NEW VALUES;
```

```
sql> CREATE MATERIALIZED VIEW LOG ON sales
      WITH SEQUENCE, ROWID
      (prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold,
       amount_sold)
      INCLUDING NEW VALUES;
```

Materialized Views with Aggregates (cont)

- Example 3: materialized view log create statements explained:
- The create materialized view log statements create the following tables: MLOG\$_PRODUCTS and MLOG\$_SALES
- Terms Used in the materialized view log create statements above:
 - **SEQUENCE** – include the SEQUENCE clause in the materialized view log create statement to indicate that a sequence value providing additional ordering information should be recorded in the materialized view log. Sequence numbers are internally necessary to support fast refresh after some update.
 - **ROWID** – required for fast refresh for all types of materialized views. For join only materialized views, the rowid for every table used in the materialized view must be listed in the select statement.
 - **INCLUDING NEW VALUES** – for joins and aggregate materialized views to be fast refreshable after inserts, the materialized view logs must include every column referenced in the materialized view and the clause INCLUDING NEW VALUES

Materialized Views with Aggregates (cont)

- Example 3 continued:

- Step 2: create the materialized view product_sales_mv:

```
sql> CREATE MATERIALIZED VIEW product_sales_mv
      PCTFREE 0 TABLESPACE demo
      STORAGE (INITIAL 8k NEXT 8k PCTINCREASE 0)
      BUILD IMMEDIATE
      REFRESH FAST
      ENABLE QUERY REWRITE AS
      SELECT p.prod_name,
      COUNT(*) AS cnt,
      SUM(s.amount_sold) AS dollar_sales,
      COUNT(s.amount_sold) AS cnt_amt
      FROM sales s, products p
      WHERE s.prod_id = p.prod_id GROUP BY p.prod_name;
```

Materialized Views with Aggregates (cont)

- Example 3 explained:
- Terms used in create materialized view statement:
- **BUILD IMMEDIATE** – product_sales_mv is populated upon creation.
- **REFRESH FAST** – product_sales_mv can be fast refreshed
- **ENABLE QUERY REWRITE** – product_sales_mv can be used for query rewrite.
- The materialized view in Example 3 is fast refreshable because:
 - materialized view logs are defined on both products and sales tables and contain the following:
 - all columns used in the materialized view's defining query
 - sequence, rowid and including new values clauses
 - the materialized view contains count(*) and count(s.amount_sold)

Materialized Views with Aggregates (cont)

- Example 4: join with aggregate

```
sql> CREATE MATERIALIZED VIEW product_sales_mv
PCTFREE 0 TABLESPACE demo
STORAGE (INITIAL 16k NEXT 16k PCTINCREASE 0)
BUILD DEFERRED
REFRESH COMPLETE ON DEMAND
ENABLE QUERY REWRITE AS
SELECT p.prod_name, SUM(s.amount_sold) AS dollar_sales
FROM sales s, products p
WHERE s.prod_id = p.prod_id GROUP BY p.prod_name;
```

Terms used in create materialized view statement in Example 4:

BUILD DEFERRED – product_sales_mv does not contain data after creation. A complete refresh is needed in order to populate product_sales_mv.

REFRESH COMPLETE ON DEMAND – complete refresh is available for product_sales_mv.

ENABLE QUERY REWRITE – can be used by query rewrite (after initial refreshed)

Materialized Views with Aggregates (cont)

- Example 5: single table with aggregate

```
sql> CREATE MATERIALIZED VIEW LOG ON sales
WITH SEQUENCE, ROWID
(prod_id, cust_id, time_id, channel_id, promo_id, quantity_sold,
amount_sold)
INCLUDING NEW VALUES;
```

```
Sql> CREATE MATERIALIZED VIEW sum_sales
PARALLEL
BUILD IMMEDIATE
REFRESH FAST ON COMMIT AS
SELECT s.prod_id, s.time_id,
COUNT(*) AS count_grp,
SUM(s.amount_sold) AS sum_dollar_sales,
COUNT(s.amount_sold) AS count_dollar_sales,
SUM(s.quantity_sold) AS sum_quantity_sales,
COUNT(s.quantity_sold) AS count_quantity_sales
FROM sales s
GROUP BY s.prod_id, s.time_id;
```

Materialized Views with Aggregates (cont)

- Example 5 explained:
- Terms used in Example 5: create materialized view statement
- **PARALLEL** – parallel operations will be supported for this mview
- **BUILD IMMEDIATE** - product_sales_mv is populated upon creation
- **REFRESH FAST ON COMMIT** – if DML is applied against the sales table, then the changes will be reflected in the materialized view when the commit is issued.
- The materialized view in Example 5 is fast refreshable because:
 - the materialized view log contains all the columns used in the materialized view's defining query
 - The materialized view log contains the following clauses: sequence, rowid and including new values
 - the materialized view contains count(*) and count(s.amount_sold) count(s.quantity_sold)

Mviews Containing Only Joins

- Some materialized views contain only joins and no aggregates.
- The advantage of creating this type of materialized view is that expensive joins will be pre-calculated.
- Join only materialized views can be fast refreshed if:
 - A materialized view log is present for each detail table unless the table supports PCT (partition change tracking). Also, the ROWID column must be present in each materialized view log.
 - The rowids of all the detail tables appear in the SELECT list of the materialized view query definition.

Mviews Containing Only Joins

- If the materialized view has remote tables in the FROM clause, all tables in the FROM clause must be located on that same site.
- ON COMMIT refresh is not supported for materialized view with remote tables.
- Materialized view logs must be present on the remote site for each detail table of the materialized view and ROWID columns must be present in the SELECT list of the materialized view

Mviews Containing Only Joins

- Example 6:
 - sql> create materialized view log on sales with rowid;
 - sql> create materialized view log on times with rowid;
 - sql> create materialized view log on customers with rowid;
 - sql> create materialized view detail_salves_mv
parallel
build immediate
refresh fast as
select s.rowid "sales_rid", t.rowid "times_rid", c.rowid
"customers_rid", c.cust_id, c.cust_last_name,
s.amount_sold, s.quantity_sold, s.time_id
from sales s, times t, customers c
where s.cust_id = c.cust_id(+) and s.time_id=t.time_id(+);

Mviews Containing Only Joins

- Example 6 explained
- The materialized view is fast refreshable because the following conditions were met:
 - Materialized view logs with the rowid clause exist on all base tables found in the defining query of the materialized view.
 - The defining query of the materialized view contains the rowids of all the base tables referenced in the from clause.
 - Refresh Fast clause is used in the create materialized view statement.

Mviews Containing Only Joins

- Example 7
 - sql> create materialized view detail_sales_mv
parallel
build immediate
refresh force as
select s.rowid "sales_rid", c.cust_id, c.cust_last_name,
s.amount_sold, s.quantity_sold, s.time_id
from sales s, times t, customers c
where s.cust_id = c.cust_id(+) and s.time_id=t.time_id(+);
- Example 7 explained:
- Detail_sales_mv is fast refreshable only if the sales table is updated but not if the times or customers tables are updated because:
 - Defining query does not contain times_rid and customers_rid
 - The refresh method is refresh force.

Nested Materialized Views

- A nested materialized view is a materialized view whose definition is based on another materialized view.
- All parent and base materialized views must contain joins or aggregates.
- All underlying objects (materialized views and tables) must have materialized view logs with the rowid clause.

Nested Materialized Views

- Example 8:
- sql> create materialized view log on sales with rowid;
- sql> create materialized view log on customers with rowid;
- sql> create materialized view log on times with rowid;
- sql> create materialized view join_sales_cust_time
refresh fast on commit as
select c.cust_id, c.cust_last_name, s.amount_sold,
t.time_id, t.day_number_in_week, s.rowid srid, t.rowid
trid, c.rowid crid
from sales s, customers c, times t
where s.time_id = t.time_id and s.cust_id=c.cust_id;

Nested Materialized Views

- Example 8 (continued)
- `sql> create materialized view log on join_sales_cust_time with rowid (cust_last_name, day_number_in_week, amount_sold) including new values;`
- `sql> create materialized view sum_sales_cust_time refresh fast on commit as
select count(*) cnt_all,
sum(amount_sold) sum_sales,
count(amount_sold) cnt_sales,
cust_last_name,
day_number_in_week
from join_sales_cust_time
group by cust_last_name, day_number_in_week;`

Nested Materialized Views

- Example 8 explained
- The sum_sales_cust_time materialized view is fast refreshable because the following conditions have been met:
- Sum_sales_cust_time defining query contains count(*) and count(amount_sold).
- A materialized view log exists for the materialized view join_sales_cust_time. The log is created with rowid, including new values, and contains the column names that are referenced in the defining query of sum_sales_cust_time.
- Materialized view logs with rowid exist for the sales, customers and times tables.
- The join_sales_cust_time materialized view's defining query contains the rowids for sales, customers and time tables.
- REFRESH FAST used in join_sales_cust_time and sum_sales_cust_time.

Summary

- Materialized views are a powerful database object that can greatly increase query performance and replicate data from remote sites.
- Materialized views are used in data warehouse and distributing environments.
- There are 3 types of materialized views:
 - Materialized views with aggregates
 - Materialized views with only joins
 - Nested Materialized views
- Fast refreshable materialized views have many restrictions.