CSCI317 Database Performance Tuning

Materialized views

Dr Janusz R. Getta

School of Computing and Information Technology -
University of Wollongong

Materialized Views

Outline

Materialized view ? What is it ?

Materialized view is a relational table that contains the rows that would be returned by the view definition - usually `SELECT` statement of SQL

If we consider relational views as stored queries then materialized views can be considered as stored results

Materialized views are created and used to reduce an amount of time needed to compute `SELECT` statements, for example join materialized views eliminate the needs to join the relational tables

Materialized views can also be used to enforce a particular way of query processing, for example if we want to be sure that selection is computed before join then we create a materialize view that contains the results of selection

Materialized Views

Outline

- Materialized view ? What is it ?
- How do we use materialized view ?
- How do we create materialized view ?
- How do we refer to materialized view ?

How do we use materialized view ?

There are two ways how a materialized view can be used:

- brute force method
- transparent query rewrite

In brute force method SQL is written to explicitly access the view

Application programmer must know the names of the materialized views

Transparent query rewrite (materialized view routing) is performed when a query optimizer detects that a query can be computed against a materialized view instead of the source relational tables

Query optimizer determines that rewrite would be more efficient and materialized view data is sufficiently current

Materialized views can be automatically maintained such that it is always up to date, or it can be updated periodically

An optional materialized view log captures updates, deletes, and inserts against the source relational tables

How do we use materialized view ?

Performance related observations

Appropriate utilization of materialized views improves both I/O and query response time

Good candidates for materialized views are the results of frequent queries

Star schemas make materialized view applicable to a large family of queries

Frequent updates of materialized views decrease performance

Incremental updates within a fixed update window during off hours is a common strategy for updating materialized views

Materialized views can be indexed

Not every materialized view can be used for transparent query rewrite

Typical materialized views that can be used for transparent query rewrite include `SELECT` statement with simple `WHERE` clause and `SELECT` statement with `GROUP BY` clause and aggregation functions

Materialized Views

Outline

Materialized view ? What is it ?

How do we use materialized view ?

How do we create materialized view ?

How do we refer to materialized view ?

How do we create materialized views ?

Setting system initialization parameters (as `SYSTEM` user)

Setting system initialization parameters

```
ALTER SYSTEM SET OPTIMIZER_MODE=ALL_ROWS;
ALTER SYSTEM SET QUERY_REWRITE_ENABLED=TRUE;
ALTER SYSTEM SET QUERY_REWRITE_INTEGRITY=ENFORCED;
```

Granting necessary privileges (as `SYSTEM` user)

Granting privileges related to materialized views

```
GRANT CREATE MATERIALIZED VIEW TO CSCI317;
GRANT QUERY REWRITE TO CSCI317;
GRANT CREATE ANY TABLE TO CSCI317;
```

How do we create materialized views ?

A sample `SELECT` statement

SELECT statement with GROUP BY clause and aggregation function

```
SELECT N_NAME, COUNT(*)
FROM CUSTOMER JOIN NATION
              ON C_NATIONKEY = N_NATIONKEY
GROUP BY N_NATIONKEY, N_NAME;
```

has the following query processing plan

A query processing plan

```
-------------------------------------------------------------------------------------------
| Id  | Operation           | Name     | Rows  | Bytes | Cost (%CPU)| Time       |
-------------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT    |          | 44616 | 2309K | 292 (1)    | 00:00:01   |
|   1 |  HASH GROUP BY      |          | 44616 | 2309K | 292 (1)    | 00:00:01   |
|* 2  |   HASH JOIN         |          | 44616 | 2309K | 292 (1)    | 00:00:01   |
|   3 |    TABLE ACCESS FULL| NATION   |    25 | 1000  |   9 (0)    | 00:00:01   |
|* 4  |    TABLE ACCESS FULL| CUSTOMER | 44616 | 566K  | 282 (0)    | 00:00:01   |
-------------------------------------------------------------------------------------------
```

How do we create materialized views ?

A sample SELECT statement

SELECT statement with GROUP BY clause and aggregation function

```
SELECT N_NAME, COUNT(*)
FROM CUSTOMER JOIN NATION
              ON C_NATIONKEY = N_NATIONKEY
GROUP BY N_NATIONKEY, N_NAME;
```

Has the following results from AUTOTRACE

Results from AUTOTRACE

```
   0  recursive calls
   0  db block gets
1062  consistent gets
   0  physical reads
   0  redo size
1145  bytes sent via SQL*Net to client
 369  bytes received via SQL*Net from client
   3  SQL*Net roundtrips to/from client
   0  sorts (memory)
   0  sorts (disk)
  25  rows processed-
```

How do we create materialized views ?

A materialized view `CUSTNAT` is created in the following way

Creating a materialized view CUSTNAT

```
CREATE MATERIALIZED VIEW CUSTNAT AS
 SELECT N_NAME, COUNT(*)
 FROM CUSTOMER JOIN NATION
               ON C_NATIONKEY = N_NATIONKEY
 GROUP BY N_NATIONKEY, N_NAME;
```

Query processing plan from a direct access to a view `CUSTNAT`

SELECT statement using a materialized view

```
SELECT *
FROM CUSTNAT;
```

Query processing plan

```
-------------------------------------------------------------------------------
| Id  | Operation             | Name    | Rows  | Bytes | Cost (%CPU)| Time     |
-------------------------------------------------------------------------------
| 0   | SELECT STATEMENT      |         |   25  | 750   |    3    (0) | 00:00:01 |
| 1   |  MAT_VIEW ACCESS FULL| CUSTNAT|   25  | 750   |    3    (0) | 00:00:01 |
-------------------------------------------------------------------------------
```

Materialized Views

Outline

- Materialized view ? What is it ?
- How do we use materialized view ?
- How do we create materialized view ?
- How do we refer to materialized view ?

How do we refer to materialized views ?

A materialized view CUSTNAT is created in the following way

Creating a metrialized view CUSTNAT

```
CREATE MATERIALIZED VIEW CUSTNAT AS
 SELECT N_NAME, COUNT(*)
 FROM CUSTOMER JOIN NATION
              ON C_NATIONKEY = N_NATIONKEY
 GROUP BY N_NATIONKEY, N_NAME;
```

Results from AUTOTRACE of a direct access to a view CUSTNAT

Results from AUTOTRACE

```
     5  recursive calls
     0  db block gets
     6  consistent gets
     1  physical reads
     0  redo size
  1144  bytes sent via SQL*Net to client
   369  bytes received via SQL*Net from client
     3  SQL*Net roundtrips to/from client
     0  sorts (memory)
     0  sorts (disk)
    25  rows processed
```

How do we refer to materialized views ?

Preparing a view CUSTNAT for query rewrite

Enabling query rewrite

```
ALTER MATERIALIZED VIEW CUSTNAT ENABLE QUERY REWRITE;
```

A sample query

SELECT statement with GROUP BY clause and aggregatin function

```
SELECT N_NAME, COUNT(*)
FROM CUSTOMER JOIN NATION
              ON C_NATIONKEY = N_NATIONKEY
GROUP BY N_NATIONKEY, N_NAME;
```

A query processing plan

A query processing plan

```
------------------------------------------------------------------------------ ...
| Id | Operation                     | Name    | Rows | Bytes | Cost (%CPU)| ...
------------------------------------------------------------------------------ ...
| 0  | SELECT STATEMENT              |         | 25   | 750   |    3  (0)  | ...
| 1  |  MAT_VIEW REWRITE ACCESS FULL| CUSTNAT | 25   | 750   |    3  (0)  | ...
------------------------------------------------------------------------------...
```

How do we refer to materialized views ?

A materialized view CUSTNAT is created in the following way

Creating a metrialized view CUSTNAT

```
CREATE MATERIALIZED VIEW CUSTNAT AS
 SELECT N_NAME, COUNT(*)
 FROM CUSTOMER JOIN NATION
              ON C_NATIONKEY = N_NATIONKEY
 GROUP BY N_NATIONKEY, N_NAME;
```

Results from AUTOTRACE

Results from AUTOTRACE

```
     0  recursive calls
     0  db block gets
     5  consistent gets
     0  physical reads
     0  redo size
  1145  bytes sent via SQL*Net to client
   369  bytes received via SQL*Net from client
     3  SQL*Net roundtrips to/from client
     0  sorts (memory)
     0  sorts (disk)
    25  rows processed
```

How do we refer to materialized views ?

A materialized view MV_ORDERS is created in the following way

Creating a materialized view

```
CREATE MATERIALIZED VIEW MV_ORDERS
 REFRESH ON COMMIT
 ENABLE QUERY REWRITE
 AS( SELECT O_ORDERKEY, O_CUSTKEY, O_TOTALPRICE, O_ORDERDATE
     FROM ORDERS
WHERE O_ORDERDATE > TO_DATE('31-DEC-1986','DD-MON-YYYY') );
```

Direct access to a materialized view MV_ORDERS

Direct access to materialized view

```
SELECT *
FROM MV_ORDERS
WHERE O_ORDERDATE = TO_DATE('01-JAN-1992','DD-MON-YYYY');
```

Query processing plan

```
-------------------------------------------------------------------------------
| Id | Operation             | Name      | Rows | Bytes | Cost (%CPU)| Time      |
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT      |           | 187  | 4488  |  507  (1)  | 00:00:01  |
|* 1 |  MAT_VIEW ACCESS FULL | MV_ORDERS | 187  | 4488  |  507  (1)  | 00:00:01  |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
---------------------------------------------------
  1 - filter("O_ORDERDATE"=TO_DATE(' 1992-01-01 00:00:00', 'syyyy-mm-dd
              hh24:mi:ss'))
```

How do we refer to materialized views ?

A materialized view MV_ORDERS is created in the following way

Creating a materialized view

```
CREATE MATERIALIZED VIEW MV_ORDERS
 REFRESH ON COMMIT
 ENABLE QUERY REWRITE
 AS( SELECT O_ORDERKEY, O_CUSTKEY, O_TOTALPRICE, O_ORDERDATE
     FROM ORDERS
WHERE O_ORDERDATE > TO_DATE('31-DEC-1986','DD-MON-YYYY') );
```

Access to a materialized view MV_ORDERS through query rewriting

SELECT statement

```
SELECT O_ORDERKEY, O_CUSTKEY, O_TOTALPRICE, O_ORDERDATE
FROM ORDERS
WHERE O_ORDERDATE > TO_DATE('31-DEC-1986','DD-MON-YYYY');
```

Query processing plan

```
PLAN_TABLE_OUTPUT
-------------------------------------------------------------------------------
|  0 | SELECT STATEMENT             |           | 108K| 2539K| 507 (1)| 00:00:01 |
|* 1 |  MAT_VIEW REWRITE ACCESS FULL| MV_ORDERS | 108K| 2539K| 507 (1)| 00:00:01 |
-------------------------------------------------------------------------------
Predicate Information (identified by operation id):
1 - filter("MV_ORDERS"."O_ORDERDATE">TO_DATE(' 1986-12-31 00:00:00',
           'syyyy-mm-dd hh24:mi:ss'))
```

How do we refer to materialized views ?

A materialized view MV_ORDERS is created in the following way

Creating a materialized view

```
CREATE MATERIALIZED VIEW MV_ORDERS
 REFRESH ON COMMIT
 ENABLE QUERY REWRITE
 AS( SELECT O_ORDERKEY, O_CUSTKEY, O_TOTALPRICE, O_ORDERDATE
     FROM ORDERS
WHERE O_ORDERDATE > TO_DATE('31-DEC-1986','DD-MON-YYYY') );
```

A case when access to a materialized view MV_ORDERS through query rewriting is impossible

SELECT statement

```
SELECT O_ORDERKEY, O_CUSTKEY, O_TOTALPRICE, O_ORDERDATE
FROM ORDERS
WHERE O_ORDERDATE < TO_DATE('31-DEC-1996','DD-MON-YYYY');
```

Query processing plan

```
PLAN_TABLE_OUTPUT
------------------------------------------------------------------------
| 0  | SELECT STATEMENT  |        | 341K| 8341K| 1950 (1)| 00:00:01 |
|* 1 |  TABLE ACCESS FULL| ORDERS | 341K| 8341K| 1950 (1)| 00:00:01 |
------------------------------------------------------------------------
Predicate Information (identified by operation id):
1 - filter("MV_ORDERS"."O_ORDERDATE"
```

References

Cookbook, How to create materialized views and partitioned relational tables ?

S. Lightstone, T. Teorey, T Nadeau, Physical Database Design, Morgan Kaufman, 2007, chapter 5 Selecting materialized views

Cookbook 14 How to create materialized views and partitioned relational tables ?