Craig S. Mullins

Return to Home Page

April / May 2004

zData Perspectives

by Craig S. Mullins

Materialized Query Tables and the Death of Denormalization

DB2 V8 offers a very useful new feature known as Materialized Query Tables (MQTs). Though not exclusively for data warehousing, MQTs can improve the elegance and efficiency of DB2-based data warehouses. An MQT can be thought of as a view that has been materialized – that is, a view whose data is physically stored instead of virtually accessed when needed. Each MQT is defined as a SQL query (similar to a view), but the MQT actually stores the query

results as data. Subsequent user queries that require the data can use the MQT data instead of re-accessing it from the base tables. By materializing complex queries into MQTs and then accessing the materialized results, the cost of materialization is borne only once, when the MQT is refreshed.

So, you might consider using MQTs for your existing complex queries. But another approach is to consider using MQTs instead of denormalization. Instead of denormalizing, implement fully-normalized tables and then build MQTs where you would have denormalized. With MQTs you can get the best of both worlds: fully normalized tables to ensure data integrity during modification and MQTs for efficient querying.

But there are potential drawbacks to using MQTs, in terms of data currency, resource consumption, and administration. First of all, MQTs are not magic; they need to be refreshed when the data upon which they are based changes. Therefore, the underlying data should be relatively static. Additionally, MQTs consume disk storage. If your shop is storage-constrained you may not be able to create many

MQTs. Finally, keep in mind that MQTs need to be maintained. If data in the underlying base table(s) changes, then the MQT must periodically be refreshed with that current data.

There are two methods for creating an MQT: you can create it anew starting from scratch using CREATE TABLE or you can modify an existing table into an MQT using ALTER TABLE. The first method uses the CREATE TABLE statement using syntax that has been augmented to look like a view definition. Consider, for example:

```
CREATE TABLE DEPT_SAL
  (DEPT, TOTAL_SALARY, TOTAL_BONUS,
TOTAL_COMM, TOTAL_COMPENSATION,
   EMPLOYEES)
AS (SELECT   WORKDEPT, SUM(SALARY), SUM(BONUS)
SUM(COMM),
             SUM(SALARY+BONUS+COMM), COUNT(*)
    FROM     DSN8810.EMP
    GROUP BY WORKDEPT)
DATA INITIALLY DEFERRED     REFRESH DEFERRED
MAINTAINED BY SYSTEM        DISABLE QUERY
OPTIMIZATION;
```

The SELECT statement defines the MQT data. So far, so good - this statement looks like a CREATE VIEW statement, except we are creating a table. Following

the SELECT statement though, are several parameters that define the nature of the MQT. There are parameters for deferring data population and to specify whether the MQT is to be maintained by the system or the user. You can also specify how DB2 will use the MQT for query optimization. The choice you make will impact the type of SELECT that can be used by the MQT being defined. The default option is to ENABLE QUERY OPTIMIZATION.

But there are limits on the type of SELECT statement that can be used for query optimization. Optionally, you can DISABLE QUERY OPTIMIZATION. Of course, this means that the MQT cannot be used for query optimization, but it can be queried directly.

As of DB2 V8, when an MQT is created, data is not initially populated into the table. Most MQTs will be maintained by the system, and as such, data will be populated when the REFRESH TABLE statement is executed. If the MQT is specified as MAINTAINED BY USER though, it can also be refreshed using the LOAD utility, INSERT, UPDATE and DELETE, as well as by REFRESH TABLE.

MQTs are fascinating because the DB2 optimizer understands them. Your queries can continue to reference base tables, but DB2 may access an MQT instead. During access path selection, the optimizer examines your query to determine whether query cost can be reduced by replacing your table(s) with an MQT. The process undertaken by the DB2 optimizer to recognize MQTs and then rewrite the query to use them is called **automatic query rewrite** (AQR).

EXPLAIN will show whether AQR was invoked to use an MQT. If the final query plan comes from a rewritten query, the PLAN_TABLE contains the new access path using the name of the matched MQTs in the TNAME column. Also, TABLE_TYPE will be set to 'M' indicating an MQT was used.

The optimizer is somewhat aware of the freshness of system-maintained MQTs. AQR will be used for a system-maintained MQT only if a REFRESH TABLE has occurred. Of course, the MQT may not be up-to-date, but DB2 knows that the MQT was refreshed at least once.

This brief overview of MQTs should help to show how DB2 V8 can optimize access to complex data

structures. With MQTs, DBAs can create a fully normalized physical database implementation – and then create “denormalized” structures using MQTs. This brings the benefit of data integrity because the database is fully normalized, along with the speed of retrieval using materialized query tables.

From zJournal, April / May 2004

.

Home.