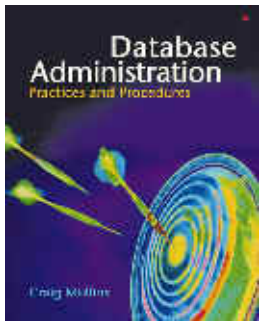
January 2004

DATABASE TRENDS AND APPLICATIONS

The DBA Corner

by Craig S. Mullins



The Death of Denormalization?

Ever since the first relational DBMS products were introduced, DBAs have fought the battle of normalization versus denormalization. Normalization is a design approach that minimizes data redundancy and optimizes data structures by systematically and properly placing data elements into the appropriate groupings. A normalized data model can be translated into a physical database that is organized correctly. In simple terms, normalization is the process of identifying the one best place each fact belongs.

E.F. Codd, the creator of the relational model, created normalization in the early 1970s. Like the relational model of data, normalization is based on the mathematical principles of set theory. Although normalization evolved from relational

theory, the process of normalizing data is generally applicable to any type of data.

Normalization is a logical process and does not necessarily dictate physical database design. A normalized data model will ensure that each entity is well formed and that each attribute is assigned to the proper entity. Of course, the best situation is when a normalized logical data model can be physically implemented without major modifications, but DBAs frequently had to divert from implementing a fully normalized physical database due to deficiencies in the DBMS in terms of performance or design.

So a normalized database implementation minimizes integrity problems and optimizes updating; but it may do so at the expense of retrieval. When a fact is stored in only one place, retrieving many different, but related facts usually requires going to many different places. This can slow the retrieval process. Updating is quicker, however, because the fact you're updating exists in only one place.

Many of our most critical applications drive transactions that require rapid data retrieval. Some applications require specific tinkering to optimize performance at all costs. To accomplish this, sometimes the decision is made to denormalize the physical database implementation, thereby deliberately introducing redundancy. This can speed up the data retrieval process, but at the expense of data modification.

Why is denormalization dying? First, the modern DBMS has been improved over the past twenty years. Today's most popular DBMSs (DB2, Oracle, SQL Server) have better

internal performance features and characteristics that can more quickly retrieve data. Another factor is better query optimization. With the in-depth, complex cost-based optimizers used by modern DBMSs, access paths are becoming more efficient. Finally, we have materialized query tables (MQTs), also known as automated summary tables (ASTs). These are new database objects supported by some of today's DBMSs that can be thought of as a materialized view. A table is created based on a SQL statement, and the DBMS manages the gathering of the data, which is then physically stored. And the optimizer "knows" about these objects so a query can be written against either the materialized query table or the underlying tables themselves. And the DBMS provides options to control data refresh rates and other use characteristics.

Using these features, the DBA can create a fully normalized physical database implementation - and then create "denormalized" structures using MQTs or ASTs. This brings the benefit of data integrity because the database is fully normalized, along with the speed of retrieval using the materialized query table.

Indeed, the death of denormalization is fast approaching. And who among us will really miss it when it finally kicks the bucket?

From Database Trends and Applications, January 2004.

Home.