Performance Tuning Considerations for DSS Systems

Steve Putman, Lemon Curry Solutions Ltd.

Dimensional modeling for efficient data warehouses is an iterative process, usually starting with little to no direction on data requirements aside from existing reports. Generally, these reports are produced from an operational perspective and likely are not appropriate for analytical applications such as performance dashboards. Therefore, the design team will take an educated guess at analytical data requirements. The team likely will design the warehouse structures accordingly, with the acknowledgment that redesign work will be required to make the structures more efficient as more concrete requirements emerge from warehouse usage experience.

Let's assume that the resources and time were provisioned for such a redesign effort, and an effective performance monitoring system is in place – a fairly large assumption in some organizations, but necessary to continue this discussion. Given this assumption, let's seek to answer the following:

- What situations warrant a change in data design to accommodate performance expectations or changing business requirements?
- What other avenues are available to the design team to improve the performance of existing structures?
- How does the team introduce these changes while mitigating disruption to existing reports?

The Issue

Imagine the following scenario, which should be easy considering it is far too common in data warehousing implementations:

The data design team designed a data structure that they believe is as flexible as possible and will maintain acceptable performance in testing environments. They implemented a dashboard-style user interface that accommodates summary views and drill-downs into detailed information. They took what is readily known about business processes and reports and created data structures that support these models.

They released the solution to the users, who found that some of the queries for detailed information are inefficient, returning results over several minutes or sometimes not returning results at all. The promise of quick (or even acceptable) performance is not realized. The system is in danger of falling into disuse because of lack of performance and the inability to double-check numbers for accuracy. The team is called back in to right the ship and restore system viability.

The Symptoms

At this point, the team must perform an analysis of the SQL being run against the database, plus any revised business requirements. This analysis likely will generate one or more common symptoms of poor performance:

- Outer joins to dimensions
- Queries that attempt to cross concepts by joining two fact tables, either directly or through non-designed dimensional connections
- Excessive snowflake joins to outer dimensions
- Poor table indexing strategy
- New requirements that do not conform to the existing data structures

None of these symptoms is surprising, given our scenario, and they are not fatal in themselves. However, in combination, they can bring a decision support system to its knees from a performance perspective, and cause the system to fall into disuse.

Dimensional Model Performance

This paper will not attempt to be a primer on dimensional data model design since there are a tremendous number of excellent sources for this information. However, returning to the basic tenets of design often goes a long way to solving performance issues. Let's review a few of these concepts:

- Minimize multi-column joins between tables by using surrogate keys
- Minimize multi-level joins, such as reference tables joined to dimensions
- Use bitmap indexes instead of B-tree indexes when permitted by your database engine
- Physically and logically partition data
- Minimize use of outer joins to dimensions by improving data quality through solid error processing in extract-transform-load (ETL) jobs that load data into the dimensional structure
- Reject joins between fact structures of different granularity and/or timing
- Realize the tradeoff between flexibility and performance

Use Surrogate Keys

A surrogate key is a single numeric value with no embedded intelligence that is used as the primary key of a table., By comparison, relational table keys can have intelligence embedded in them (such as transaction date) and usually have more than one database column. A surrogate key is assigned during ETL table loading, usually in a sequential manner, and resolves multi-column keys in one column.

Use of surrogate keys ensures reference to the time dimension inherent in data warehouses without explicitly using the time key in joins. This reduces the number of joins in SQL statements and makes these queries more efficient. This is a fairly radical solution for schemas that do not already use this concept. Both the data structures and the data loading programs that service them must change, This should be considered a long-term solution in future iterations of the warehouse.

Reference Table Denormalization

Transactional database design, which is optimized for individual core transaction integrity, utilizes reference tables for supporting information – codes, abbreviations, and the like. Decision support systems handle vast amounts of data and need to minimize joins between tables to retrieve data efficiently. The most efficient data retrieval strategy would be to denormalize dimensions to minimize table joins in queries.

However, the complexity of modeled data, even in the dimensional world, precludes the complete denormalization of dimensional data because of space limitations (in number of columns) for large dimensions – usually 255 columns. Identifying concepts that are good candidates for "snowflaking," or normalizing reference relationships in dimensions, rests on how often the concepts are used in user queries for data retrieval and filtering. Frequently used concepts such as product codes or marital status usually should be denormalized in the larger dimensions to reduce joins and improve performance.

This change can be introduced gradually over time, using care to regression-test each affected query and report.

Use Bitmap Indexes

A bitmap index, along with B-Tree indexing, is one of the main physical data table index techniques available to most database platforms. In essence, a map of bits is created within the database environment consisting of the distinct values within the table as columns, and each data row as rows, with either a one or a zero for a data value denoting the presence or absence of the column value. A bitmap index can be much smaller than its corresponding B-Tree index, enabling storage in system memory and making it much faster.

Bitmap indexes are inefficient for transactional systems. This is due to a requirement to physically rebuild the index when the underlying table rows change. Most database engines default to the B-Tree indexing structure when indexes are built. Bitmap indexing becomes viable in a decision support system since the underlying table changes only periodically, and can be more efficient for low-cardinality dimension tables (a small number of distinct values, such as a reference table). Recent research shows that they are no less efficient than B-Tree indexes for granular, high-cardinality fact tables, so they can be used throughout the system. Like any data structure change, analysis and experimentation in the testing environment often yields the correct indexing scheme.

This change also can be introduced gradually over time as more potential index gains are identified through query analysis by the technical team.

Partition Data in Large Tables

Most major database platforms allow for both logical and physical partitioning of data in high-volume tables to ease the load handled by the query engine. Logical partitioning is accomplished by adding keys to fact and dimension tables, such as date unit (i.e., year) and set keys. Query writers then use these keys to filter data into groups. Physical partitioning is accomplished within the database engine to separate data in a table into groups that are stored in different file systems using similar criteria. This effectively reduces the number of rows requiring retrieval during a query request. Although the database engine will reassemble the groups as necessary based on queries passed to the engine, the goal of this design is to support queries that do not span partitions.

Great care and analysis are required to effectively split tables into partitions. This requires advance knowledge of the reporting needs of the organization because the reassembly of partitions causes a great deal of overhead to the database system, which could move performance into unacceptable territory. Partitioning tables could limit the ability to perform ad-hoc analysis using the database, but the attainment of a proper balance can achieve large performance gains.

Minimize Outer Joins

Efficient database query design avoids the use of the inefficient outer join (where all records of one table are included regardless of whether matching rows in the second table are found) due to the fact that indexes cannot be used. In certain situations, outer joins cannot be avoided based on database design, and the cost of redesigning the data model is not worth the cost of query inefficiency. However, in some cases a query designer will specify an outer join when she is not confident of the quality of the data in the database, even though an inner join is warranted. When this is the case, establishment of and adherence to a strict data quality initiative that includes feedback loops can mitigate this concern when data is loaded. Query designers should have complete faith that the data in the database are complete and robust. A comprehensive data quality initiative will reinforce this belief.

These changes can be introduced slowly over time on a query-by-query basis, which poses low risk to the overall solution.

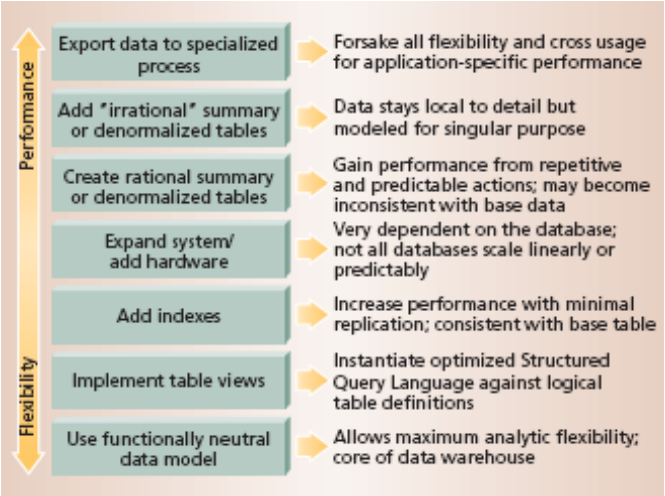
Joins Between Two Fact Tables

In classic dimensional modeling, each star-schema structure has one central fact table surrounded by supporting dimensions. Occasionally, there will be more than one fact table in the structure, built with careful consideration to the timing and grain or detail level of the data. Data warehouses can contain many seemingly related fact table structures in the same database. Query designers must resist the temptation to bridge fact tables that have similar keys, either directly or through a shared dimension, into a query for filtering or for other purposes. The database engine will accept this query and attempt to answer it, but often the result set is inconsistent and inefficient, sometimes to the point of inoperability.

The mitigation for this issue must involve the data architecture staff, which is the most familiar with the original purpose of the data models and the types of permissible joins within the design. Likewise, the temptation to join different structures should be curbed in favor of a redesign session with the architecture staff to ensure an efficient solution to the requirements. Changing requirements are a fact of life in a decision support system. The appropriate personnel must be involved when these changes are encountered.

This change can be introduced without affecting the existing structure, aside from reengineered queries and reports that take advantage of the new structures. It is low-risk to the overall solution, but it also takes longer than most changes due to the increased design work and creation of new data loading programs to populate the new structure.

Tradeoff between Flexibility and Performance



An inverse relationship exists between flexibility and performance in decision support systems. Flexibility in design exacts a price in performance, while high-performance systems are specialized and inflexible. It is apparent where each of these steps falls in the continuum, and incumbent on the design team to strike the appropriate balance between performance and flexibility to provide the most effective system to the business community.

Summary

There are many factors to consider when diagnosing performance issues in a decision support system. The solutions to these issues generally fall into either database-specific or design-oriented groups. Database-specific solutions are accomplished through a partnership with the technical support team. Design-oriented solutions target reducing the number of joined columns in user queries to enhance performance. Most of the solutions mentioned in this article can be summarized as maintaining the discipline of good modeling and data quality practice. The most radical solution is to add a fact table structure to the data model, which is sometimes necessary when changing requirements require revisiting the model's design assumptions.

References

Armstrong, Ron, *Seven Steps to Optimizing Data Warehouse Performance,* IEEE Computer, December 2001, pp 76-79.

Sharma, Vivek; Bitmap Index vs. B-Tree Index: Which and When?; Oracle Technology Network