OLAP, Relational, and Multidimensional Database Systems

George Colliat
Arbor Software Corporation
1325 Chesapeake Terrace,
Sunnyvale, CA 94089

Introduction

Many people ask about the difference between implementing On-Line Analytical Processing (OLAP) with a Relational Database Management System (ROLAP) versus a Mutidimensional Database (MDD). In this paper, we will show that an MDD provides significant advantages over a ROLAP such as several orders of magnitude faster data retrieval, several orders of magnitude faster calculation, much less disk space, and less programming effort.

Characteristics of On-Line Analytical Processing

OLAP software enables analysts, managers, and executives to gain insight into an enterprise performance through fast interactive access to a wide variety of views of data organized to reflect the multidimensional aspect of the enterprise data. An OLAP service must meet the following fundamental requirements:

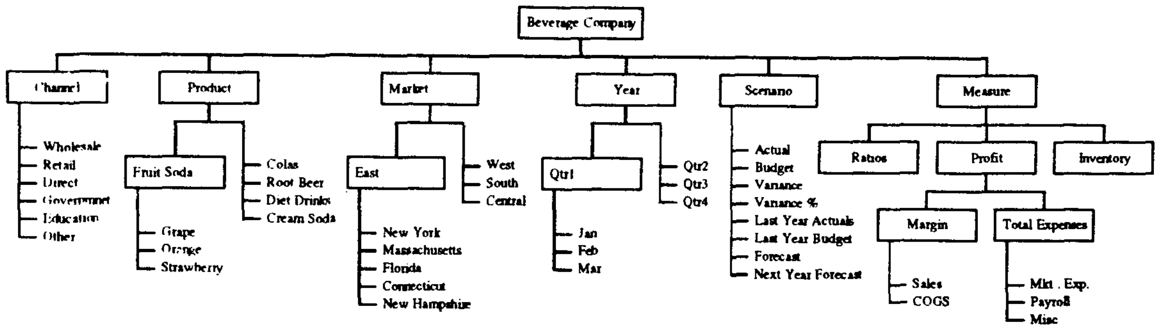
- The base level of data is summary data (e.g., total sales of a product in a region in a given period)
- Historical, current, and projected data
- Aggregation of data and the ability to navigate interactively to various level of aggregation (drill down)
- Derived data which is computed from input data (performance ratios, variance Actual/Budget,...)
- Multidimensional views of the data (sales per product, per region, per channel, per period...)
- Ad hoc fast interactive analysis (response in seconds)
- Medium to large data sets (1 to 500 Gigabytes)
- Frequently changing business model (Weekly)

As an illustration we will use the six dimension business model of a hypothetical beverage company. Each dimension is composed of a hierarchy of *members*; for instance the time dimension has months (January, February...) as the leaf members of the hierarchy, Quarters (Quarter 1, Quarter 2...) as the next level members, and Year as the top member of the hierarchy. We will assume the following number of members for each dimension:

• Channel	6 members
• Product	1500 members
• Market	100 members
• Time	17 members
• Scenario	8 members
• Measures	50 members

A simple OLAP scenario consists of getting the actual profit of the company for the current month and comparing it with the budget, then drilling down per market region and product family. Further drilling may be necessary in case of a large variation between Budget and Actual.

Multidimensional Model



Relational Approach

Given the popularity of relational DBMS's, it is tempting to implement the OLAP facility as a semantic layer on top of a relational store. This layer would provide a multidimensional view, calculation of derived data, drill down intelligence, and generation of the appropriate SQL to access the relational storage. The typical 3rd-normal form representation of data is completely inapplicable in this environment because of the overhead of processing joins and restrictions across a very large number of tables [3,4]. Instead a denormalized Star Schema is used to give acceptable performance.

The relational schema consists of a Fact table and one table per dimension. The Fact table contains one row for each set of measures and a *dimension-id* column for each dimension. Rollup summaries, such as East region, are precalculated and stored in the Fact table. Each dimension table represents the hierarchy of its dimension and contains the full name of the members of the dimension. The number of potential rows in the fact table is the cross product of the dimensions: Channels(6) * Products(1500) * Markets(100) * Time(17) * Scenario(8) = 122 million. With 80% sparsity, which is typical, the number of rows is about 24 million. With 50 columns in the Measure dimension the row size is about 500 bytes and the Fact table amounts to about 13 Gigabytes. Each column will need an index to execute the joins and restrictions with reasonable efficiency. If we assume a 4K block size, and a 30 byte average index entry, each index would require about 800 megabytes; 5 indexed dimensions correspond to about 4 Gigabytes of indexes. The total database size is about *17 Gigabytes*.

Fact Table

Chan-id	Prod-id	Mkt-id	Time-id	Scen-id	Sales	COGS	Margin	Mkt. Exp.	Payroll	Misc	Tot. Exp.	Profit
1	2	1	1	1	$245	$100	$145	$23	$45	$12	$80	$65
3	2	2	1	1	$123	$65	$58	$12	$23	$12	$47	$11
..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..
2	105	1	1	1	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..
3	105	100	1	1	..	..	..	..	..	..	..	..
..	..	..	..	..	..	..	..	..	..	..	..	..

Market Table

rollup	member	Mkt-id
East	New York	1
East	Florida	2
	....	
Market	East	100
Market	West	101
Market	South	102
Market	Central	103
Market	Market	500

Product Table

rollup	member	Prod-id
Fruit Soda	Grape	1
Fruit Soda	Orange	3
Colas	cola	2
	...	
Product	Fruit Soda	101
Product	Root Beer	102
Product	Cream Soda	103
Product	Diet drinks	104
Product	Colas	105
Product	Product	500

Retrieving all Measures for Actual and Budget, for each Month of the year at the corporate level (total products, total channels, total markets) can be expressed as:

```
SELECT Scenario.member, Time.member, Sales, COGS, Margin,..., Profit
         FROM Fact, Channel, Product, Market, Time, Scenario
         WHERE Fact.Chan-id = Channel.Chan-id and Channel.member ='Channel'
                  Fact.Prod-id = Product.Prod-id and Product.member = 'Product'
                  and Fact.Mkt-id = Market.Mkt-id and Market.member = 'Market'
                  and Fact.Time-id = Time.Time-id and Time.rollup = 'Year'
                  and Fact.Scen-id = Scenario.Scen-id and Scenario.member = 'Actual'
UNION  SELECT Scenario.member, Time.member, Sales, COGS, Margin,..., Profit
         FROM Fact, Channel, Product, Market, Time, Scenario
         WHERE Fact.Chan-id = Channel.Chan-id and Channel.member ='Channel'
                  Fact.Prod-id = Product.Prod-id and Product.member = 'Product'
                  and Fact.Mkt-id = Market.Mkt-id and Market.member = 'Market'
                  and Fact.Time-id = Time.Time-id and Time.rollup = 'Year'
                  and Fact.Scen-id = Scenario.Scen-id and Scenario.member = 'Budget'
```

It is reasonable to expect that at most one fourth of the top three levels of indexes (365 Megabytes) will stay in memory. In this case the 6 way join will result in an average about 10 I/O's per retrieved row or about 240 I/O's for this query.

Calculation

We will now examine the calculation of derived data using SQL and a 3GL. The roll-up columns are trivial to generate by using the column expression facility of SQL. As an example the following SQL statement computes the Margin, Tot Exp, and Profit columns:

```
UPDATE Fact SET        Margin = Sales - COGS,
                       Tot Exp = Mkt Exp + Payroll + Misc
                       Profit = Sales - COGS - (Mkt Exp + Payroll + Misc)
```

The roll up rows can be generated by SQL INSERT statements similar to the following ones that generate the roll-up's for the East region:

```
INSERT INTO Market (Market, East, '100')

INSERT INTO Fact  (Chan-id, Prod-id, Mkt-id, Time-id, Scen-id, Sales, COGS, ..., Profit)
         SELECT Chan-id, Prod-id, '100', Time-id, Scen-id, SUM(Sales), SUM(COGS), ..... SUM(Profit)
                  FROM Fact, Market
                  WHERE Fact.Mkt-id = Market.Mkt-id and Market.rollup = 'East'
                  GROUP BY Chan-id, Prod-id, Time-id, Scen-id
```

This roll up technique can only be applied for simple aggregations. Any computation which is not commutative and associative will require a 3GL program with cursors. For instance the computation of Variance as Actual - Budget will require a SELECT of an Actual row, a SELECT of the matching Budget row, a computation of the variance, and an INSERT of the Variance row. This will cost on the average of *17 I/O's per row*. To calculate and store all the Variance rows would take: 20% * Product* Market * Channel * Time * 17=*52 million I/O's (about 237 hours of I/O time)*!

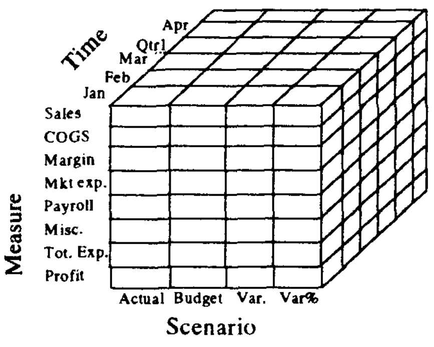
This process would have to be repeated for all derived values. It is clearly impractical to do this via SQL. The only practical solution is to write a special 3GL program which will do all the computation at the time the relational database is loaded with the base data. When all the base and derived data has been loaded the indexes can be built to provide acceptable efficiency.

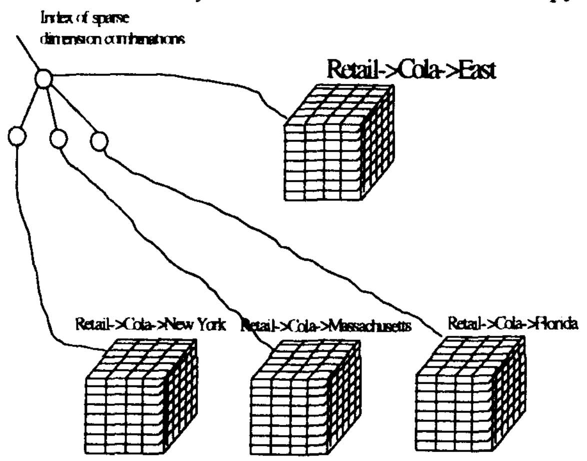
In spite of relational databases popularity in OLTP applications, this analysis shows that the relational model is not ideally suited for OLAP because of the large number of I/O's necessary to perform simple drill downs and computation. An alternative is to stage the OLAP data in a storage which is designed for multidimensional analysis.

Multidimensional Database Approach

We will use the same OLAP model with a server that is based upon a Multidimensional database such as Essbase. The data relevant to the analysis is extracted from a relational Data Warehouse or other datasources and loaded in a multidimensional database which looks like a hypercube with 6 dimensions (in this example). The following implementation of this hypercube is specific to the Essbase product. It is patented [7] and some of the advantages described may not apply to other multidimensional database implementations.

The dimensions which usually have data in every cell, such as Time, Scenario, and Measure are represented by a dense block symbolized in the following picture by a cube. The other dimensions are called sparse dimensions and for every combination of sparse dimensions where data exists (Retail->Cola->New York, Retail->Cola->Florida, Retail->Cola -> East, etc..), there is an entry in the index pointing to a dense block on disk. In the beverage company example, a block would consist of Time members * Scenario members * Measure members * 8bytes per cell=55K bytes, with 80% sparsity all the blocks would occupy 10 Gigabytes. The index would occupy 6 Megabytes,*because of its small size it would remain in memory*. The total database would occupy *10 Gigabytes*.




Index of sparse dimension combinations
Retail->Cola->East
Retail->Cola->New York
Retail->Cola->Massachusetts
Retail->Cola->Florida

A typical OLAP retrieve proceeds top down, i.e. starting from the highest level of aggregation in each dimension (Channel-> Market->Product->Year->Actual>Profit). The block corresponding to the combination of the top level members of the sparse dimensions is located via an index search and brought into memory with a *single I/O)* and the data is located by offset computation inside the block.

In summary the following conclusions can be drawn from this example:

1. Retrieval is very fast because
   - The data corresponding to any combination of dimension members can be retrieved with a single I/O.
   - Data is clustered compactly in a multidimensional array.
   - Values are calculated ahead of time (see Calculation below).
   - The index is small and can therefore usually reside completely in memory

2. Storage is very efficient because
   - The blocks contain only data
   - A single index locates the block corresponding to a combination of sparse dimension numbers.

- A single small index usually resides completely in memory.

Calculation

The Essbase provides a default calculation which is optimized for efficient roll-up and to take advantage of the clustering described previously. Typically all cells of a block containing input data such as Retail->Cola->Florida are calculated at once within the block, then a roll up block, such as Retail->Cola->East, is computed by summing the cells of all children blocks.

The calculation of the variance between Actual and Budget can be accomplished with 2 I/O's per block (Read & Write) for a total of 20% * Product * Market * Channel * 2 = *360,000 I/O's (about 2 hours of I/O time) for the whole database compared to 237 hours in the relational approach. All other derived data can be rolled up at the same time without requiring any more I/O's!*

The calculation is very efficient because:

- Only one read and one write I/O per block are necessary to roll-up a whole database
- The memory representation of a block is an array with efficient relative offset addressing.
- Roll-up's can be done by benefitting from the isomorphic nature of the multidimensional array representation of the data.

Comparison between the Relational and the Multidimensional models

The analysis of our example illustrates the following differences between the best Relational alternative and the Multidimensional approach.

	Relational	Multidimensional	Improvement
Disk space requirement (Gigabytes)	17	10	1.7
Retrieve the corporate measures, Actual vs Budget, by month (I/O's)	240	1	240
Calculation of Variance Budget/Actual for the whole database (I/O time in hours)	237	2*	110*

* This may include the calculation of many other derived data without any additional I/O.

The Multidimensional database in our example uses almost half the disk space, retrieves data between 8 and 200 times faster, and calculates the derived data at least 2 orders of magnitude faster than Relational. We have also seen that, when the best relational alternative was used, the generation and calculation of the Fact table and Dimension tables were a serious programming and maintenance challenge, requiring a complicated 3GL program for the calculation of the derived values and a sophisticated query processor to generate the proper SQL.

The fundamental reason for the large difference in performance comes from the data models. The Relational model assumes a table model where the only way to address a row is via the contents of one of its fields, requiring massive indexing. The Multidimensional model uses array addressing, with relative offsets. The *contents addressing* of the Relational model was created in the early 1970's to provide flexibility in data restructuring which did not exist in the popular databases of the time, IMS and CODASYL. It has proven over the years to be a very successful model for OLTP. However for OLAP applications, the Relational cursor that navigates through a multitude of indexes cannot compete with the Multidimensional array cursor that operates via relative offsets.

References

1. Beyond Decision Support, Edgar F. Codd, Sharon B.Codd, Clynch T. Salley, Computerworld July 26, 93
2. Structuring databases for analysis, Jeffrey P. Stamen, IEEE SPECTRUM, Oct. 1993
3. Two Steps Forward, One Step Back, David Vaskevitch, BYTE, May 1992
4. Why Decision Support Fails and How to Fix it, Ralph Kimball and Kevin Strehlo, Datamation June 1, 94
5. Multidimensional Databases, John Xenakis, Application Development Strategies, April 1994
6. Understanding the need for On line Analytical Servers, Richard Finkelstein
7. Arbor Software Corporation, Robert J. Earle, U.S. Patent # 5359724
8. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Jim Gray, Adam Bosworth, Andrew Layman, Hamid Piranesh, Microsoft Research Technical Report MSR-TR-95-22, July 17, 1995