Relational extensions for OLAP

by N. Colossi
W. Malloy
B. Reinwald

Enterprises have been storing multidimensional data, using a star or snowflake schema, in relational databases for many years. Over time, relational database vendors have added optimizations that enhance query performance on these schemas. During the 1990s many special-purpose databases were developed that could handle added calculational complexity and that generally performed better than relational engines. DB2® has added a number of features that make it more competitive with these special-purpose databases. In this paper, we define meta-data extensions that allow designers of multidimensional schemas to describe the structure of those schemas to multidimensional query and analysis tools. The SQL (Structured Query Language) extensions include a "cube" object that returns row sets that are "slices" of the cube. We also describe Web services for OLAP (on-line analytical processing) that provide meta-data for multidimensional data, as well as XML (Extensible Markup Language) query results.

On-line analytical processing (OLAP) is a term that was coined in an unpublished 1993 white paper, *Providing OLAP to User Analysts: An IT Mandate*, by E. F. Codd.[1] By introducing this new term as a play on the then-familiar term on-line transaction processing (OLTP), the paper signaled a shift in the paradigm for business analysis, in parallel with the shift that had already occurred for transaction processing. Instead of reviewing piles of static reports printed on green-bar paper, the OLAP analyst could explore business results interactively, dynamically adjusting the view of the data—asking questions and getting answers almost immediately. This freedom from static answers to fixed questions on a fixed schedule allows business analysts to operate more effectively and to effect improvements in business operations. In the white paper, the authors outlined 12 characteristics of an OLAP system. In a 1995 update to the white paper, six more characteristics were added. These rules provide one point of view on what constitutes an OLAP system, but are not definitive. One reason is that the development of the white paper was commissioned by Arbor Software (now Hyperion Solutions). Not surprisingly, the Arbor product, Essbase**, generally follows the rules outlined, whereas competitive products were often found lacking when measured against the provided yardstick. The ultimate meaning of OLAP is left to time and the growing collection of products that claim to be OLAP products, or have OLAP components.

Nigel Pendse, an analyst with Business Intelligence Ltd. who publishes *The OLAP Report*, provides another valuable point of view. In a Web page entitled "What Is OLAP?"[2] Pendse introduces a simpler model, FASMI, to characterize OLAP systems. Although no single definition is likely to receive universal support, Pendse's characterization is much simpler than the Codd rules. Briefly, the FASMI characteristics are:

0018-8670/02/$5.00 © 2002 IBM

- Fast. In keeping with the spirit of the "O" in OLAP, such systems need to provide results very quickly—usually in just a few seconds, and seldom in more than 20 or 30 seconds. This level of performance is key in allowing analysts to work effectively without distraction.
- Analytic. Considering the "A" in OLAP, OLAP systems generally must provide rich analytic functions appropriate to a given application, with minimal programming.
- Shared. An OLAP system is usually a shared resource. This means that there is a requirement for OLAP systems to provide appropriate security and integrity features. Ultimately, this can mean providing different access controls on each cell of a database.
- Multidimensional. Multidimensionality is the primary requirement for an OLAP system. OLAP products present their data in a multidimensional framework. Dimensions are collections of related identifiers, or attributes (product, market, time, channel, scenario, or customer, for example) of the data values of the system. The identifiers ("The Lord of the Rings—DVD," "San Jose, California," "2002," "Retail Rental," and "John Q. Public," for example) belonging to the collection for a particular dimension generally have some sort of structure—usually hierarchical. Sometimes there is more than one natural structure for these identifiers. The multidimensional characteristic means that an OLAP system can quickly switch among various orientations of dimensions, as well as among various subsets and structural arrangements of a dimension. Because of the multidimensional nature of OLAP systems, we often refer to the collections of data that they implement as *cubes*.
- Store and calculate information. OLAP systems must store and calculate information. Data for OLAP systems often come from one or more operational systems. Analytical models are applied to these data, and the results must either be stored in the system or generated at query time. The quantity of information that a particular OLAP system can manage is an important characteristic of that system.

The bottom line is that OLAP systems allow their users to do analysis at the speed of thought. They provide consistently fast response to queries expressed in terms of a business model that makes sense to the user. They provide flexible views of data without limiting the number of dimensions or levels of aggregation or allocation.

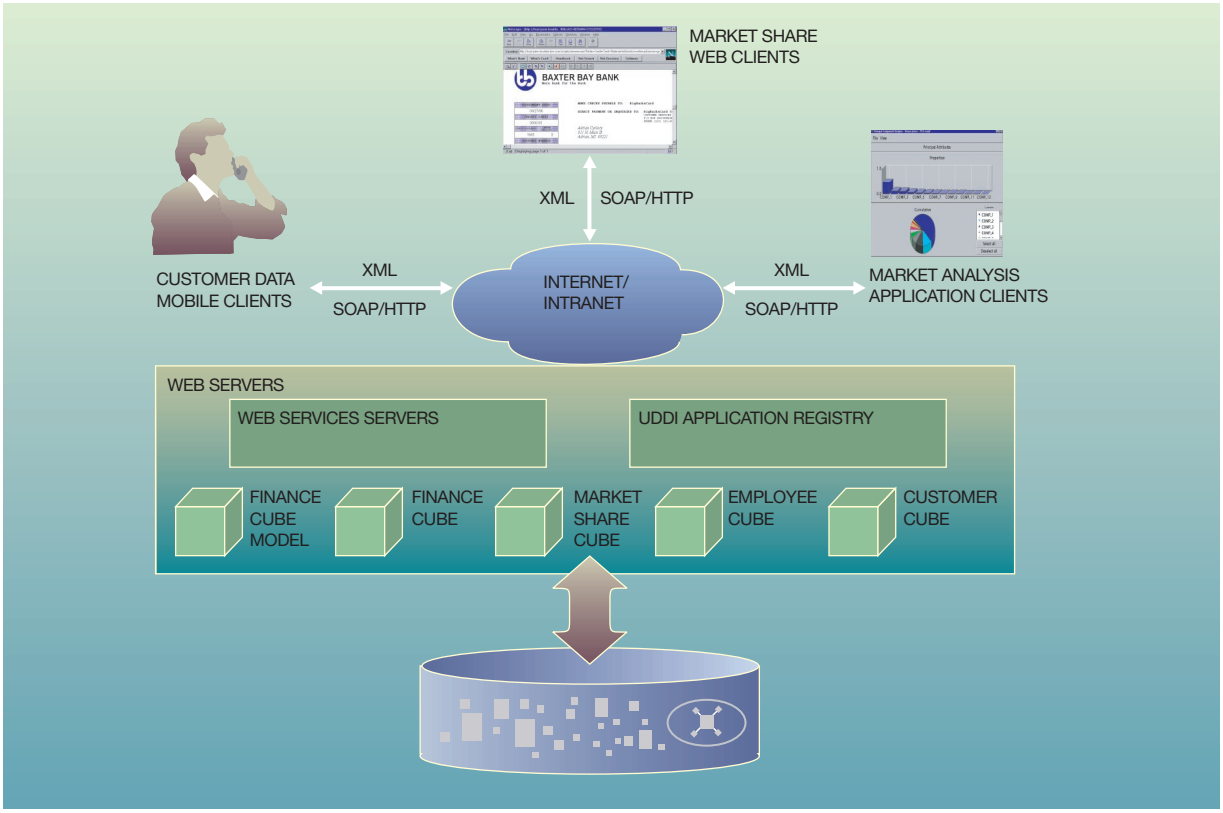
Although many products have claimed the OLAP label, no single product or model dominates strongly enough to result in widely accepted application programming interfaces (APIs), data definition languages (DDLs), or data manipulation languages (DMLs). This is in stark contrast to the world of relational databases in which ODBC (Open Database Connectivity) and SQL (Structured Query Language) are nearly universally accepted. This is a great burden to developers of OLAP systems and applications. However, the shift in the industry toward a service-oriented architecture, using a pervasive infrastructure that includes HTTP (HyperText Transfer Protocol) and XML (Extensible Markup Language), presents new opportunities for OLAP systems (see Figure 1).

A client or service requestor may access a server or service provider through Web services and their infrastructure. SOAP (Simple Object Access Protocol) essentially defines an RPC (remote procedure call)-like XML protocol over HTTP between service requestor and provider. A service provider may be registered in a UDDI (Universal Description and Discovery Interface) registry for requestors to find and discover services. In this paper we propose a set of Web services for accessing OLAP data and metadata. This set includes services to discover data cubes, explore the structure of a cube, and query data within a cube. Service requestors might reside in small devices such as cellular phones, in thin Web clients that deploy a browser interface, or in thick clients that perform some data analysis and visualization. The service requestors access the OLAP system through the Internet to retrieve OLAP data. An OLAP system might provide a finance cube, a market-share cube, an employee cube, and a customer cube. The strength of this set of services comes from its pervasive and ubiquitous infrastructure based on HTTP and XML, as well as the strict separation between the OLAP requestor and the provider.

OLAP basics

Figure 2 illustrates a simple, three-dimensional OLAP cube. Each *dimension* comprises a set of related members. In our example, product, time, and location are the dimensions of the cube. The *members* of the location dimension are location, USA, Canada, CA, NY, ON, SJC, PAS, YKT, TOR, 100, 200, 300, and 400. Members of a dimension are usually organized into a *hierarchy of levels* that show the parent-child relationships of the levels within a dimension. Levels are sets of members that share a number of common attributes. In our example, the elements

Figure 1 Application scenario

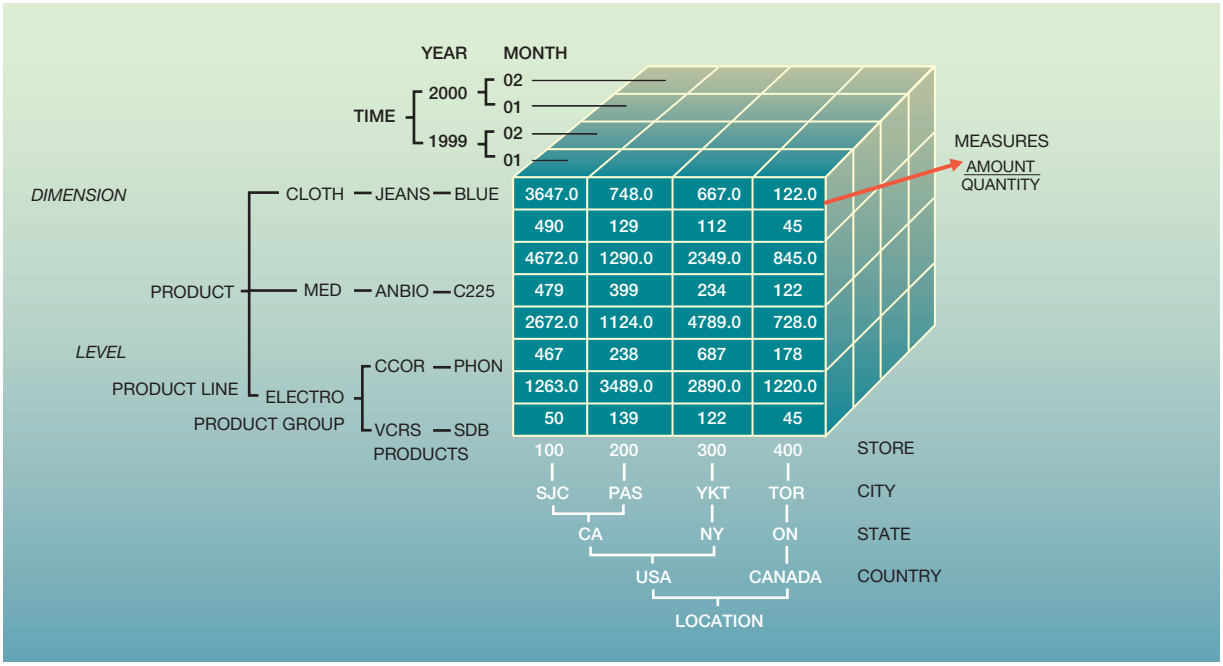


100, 200, 300, and 400 belong to the store level. Although not shown in Figure 2, this level has a *member attribute* of "square_foot"—the size of each store in square feet. The city level could also have attributes that include "mayor," while the state level could have attributes that include "governor." Note that if members at a particular level of a hierarchy are not unique (for example, 01 and 02 in the time dimension), they must be combined with parent identifiers to form unique member names (1999-01, for example). In order to show the hierarchical relationship of the members in the figure, aggregate members such as 1999 and SJC are not explicitly shown as identifiers of a slice of the cube. However, conceptually all members on an edge do identify such a slice. So, (1999, VCRS, SJC) describes a cell of the cube, in the same way that (1999-01, SDB, SJC) describes a cell. Each cell of the cube contains a vector of two values: amount and quantity. Sometimes it is convenient to consider these vectors of values as another dimension of a cube.

This view of an OLAP cube does not explicitly describe how values are computed. Usually, hierarchical relationships imply simple aggregation, such as CA = SJC + PAS. In many models, the mathematical relationships in a hierarchy are more complex, and they vary depending on the nature of the measurements. For example, in a cube measuring sales and inventory, sales values may be summed by time period, whereas average or final values are more appropriate for inventory. In many cases (particularly with costs), only aggregate values are available and an OLAP system is used to allocate those costs down to lower levels. The ability of OLAP systems to handle various types of calculations is a primary distinguishing factor among such systems.

Because the number of cells in a cube is the product of the size of each dimension, OLAP cubes can be very large. Consider the possibility of a large enterprise with 1 000 000 customers, 100 000 products, 2500

Figure 2 OLAP sales cube



time periods, and 25 measurements. An OLAP system implementing such a cube has to deal with the potential of 6250000000000000 cells. The number of potential cells is usually several orders of magnitude greater than the number of populated cells; thus OLAP systems must be designed to handle very sparsely populated matrices.

Figure 3 shows a view of our sample cube that is typical of grid-based OLAP viewers (query processors). From any such view, OLAP systems typically provide operations that allow an analyst to quickly switch to a related view. Operations typically provided include:

- Drill-down—navigate in a dimension from lesser detail to greater detail. Figure 3 shows the result of drilling down one level in the location dimension.
- Drill-up—navigate in a dimension from greater detail to lesser detail. Figure 3 shows the result of drilling up one level, to the top of the product dimension.
- Pivot—rearrange the ordering of dimensions in a result. Figure 3 shows the result of switching the product and time dimensions in the display grid.

Although the term OLAP is not yet ten years old, the development of OLAP technology is quite a bit older.[3] IRI's Express** and Comshare's System W are considered the progenitors of today's OLAP systems. Development of these systems began in the late 1960s to try to model business operations using multidimensional matrices and complex operations on them. Multidimensional analysis has also been done using SQL since the introduction of relational databases. The 1990s saw a burst of research and development in OLAP technology. OLAP systems today are generally classified by their general technological approach into one of the following classes:

- Multidimensional OLAP (MOLAP)
- Relational OLAP (ROLAP)

Multidimensional OLAP refers to the family of OLAP systems in which special-purpose file systems or indexes are used to store cube data. There is a great deal of academic research on, and numerous patents have been granted for, storage and index systems that optimize storage usage and query execution time for multidimensional data arrays. Express, Essbase, TM1**, and Pilot Suite** are a few exam-

Figure 3 Standard OLAP operations

INITIAL QUERY RESULT

		ELECTRO		MED		CLOTH	
		Amount	Quantity	Amount	Quantity	Amount	Quantity
1999-01	USA	16225	1703	8311	1112	5062	731
	Canada	1948	223	845	122	122	45
1999-02	USA	16902	1769	6449	807	5331	770
	Canada	1954	244	912	131	127	48

RESULT AFTER PIVOT

		1999-01		1999-02	
		Amount	Quantity	Amount	Quantity
ELECTRO	USA	16225	1703	16902	1769
	Canada	1948	223	1954	244
MED	USA	8311	1112	6449	807
	Canada	845	122	912	131
CLOTH	USA	5062	731	5331	770
	Canada	122	45	127	48

RESULT AFTER DRILL-UP

		Amount	Quantity
1999-01	USA	29598	3546
	Canada	2915	390
1999-02	USA	28682	3346
	Canada	2993	423

RESULT AFTER DRILL-DOWN

			ELECTRO		MED		CLOTH	
			Amount	Quantity	Amount	Quantity	Amount	Quantity
1999-01	USA		16225	1703	8311	1112	5062	731
		CA	8546	894	5962	878	4395	619
		NY	7679	809	2349	234	667	112
	Canada		1948	223	845	122	122	45
		ON	1948	223	845	122	122	45
1999-02	USA		16902	1769	6449	807	5331	770
		CA	8695	904	5992	777	4585	649
		NY	8207	865	457	30	746	121
	Canada		1954	244	912	131	127	48
		ON	1954	244	912	131	127	48

ples of products based on special-purpose storage and indexing technology. Microsoft's OLAP offering also includes a MOLAP engine. These systems are often read-only systems that are loaded with base data periodically, then derived results are calculated, stored, and indexed. Scalability of MOLAP systems is often limited by the size of the batch window within which derived results must be calculated and stored. To improve scalability, such systems often have a means for deferring calculation of some derived results until query time.

Relational databases have been used extensively for multidimensional analysis for many years. By far the most common approach is to organize the data in what is called a star schema. The simplest star schemas consist of a fact table joined with a number of dimension tables. The fact table contains one column for each dimension, plus one column for each measure. A great deal has been written on the techniques for designing star schemas and optimizing the calculation of derived results. A classic work on the subject is *The Data Warehouse Toolkit*, by Ralph Kimball.[4]

A simple example of a star schema is shown in Figure 4. The fact table has five columns. A row in the fact table contains one key value per dimension and two value columns—one for amount and one for quantity. Each of the dimension key values matches the key value in the key field of a single row of the corresponding dimension table. In the highlighted row of the fact table, the key value "1" of TID matches the first row of the time dimension table, so we know that the values in this row of the fact table are for month 1 of year 1999. Similarly, we can match key values of the PID and LID columns in the highlighted row to find matches in the product and location dimension tables of (ELECTRO, CCOR, PHON) and (USA, CA, SJC), respectively. We also know that the size of

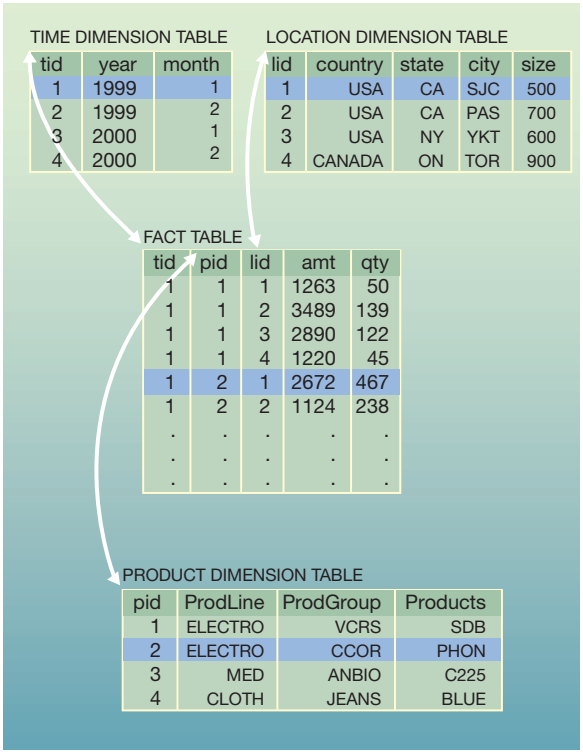
the store in SJC is 500, because of the dimensional attribute column in the location dimension table.

In the late 1980s and early 1990s, a number of products were designed to exploit this method of organizing multidimensional data. In 1986, Ralph Kimball launched Red Brick Systems to develop a relational database system designed specifically to handle star schema queries. However, the SQL language as implemented in the early 1990s had some serious performance problems in implementing large OLAP cubes. Some of the most serious problems were:

- GROUP BY could not be used to return the results for cells with different levels of aggregation. This meant that many different queries were required to obtain all cell values.
- Highly aggregated queries could reference virtually every row in the fact table in order to return very few cell values.
- There was generally no memory of the results of such huge calculations—the relational engine had to start "from scratch" with each query. As aggregates for various levels were computed, the rows of the fact table were read repeatedly.
- SQL supported only a few aggregation functions (SUM, COUNT, AVG, MIN, and MAX), although some relational database systems had extensions with additional analytic capabilities.
- The relational database catalogs did not contain meta-data about the structure of star schemas. Such meta-data were relegated to the domain of tools and applications. As a result, star schemas had to be "described" multiple times—once for each application or tool used.
- Because the meta-data were not part of the database catalog, key structural information was unavailable to query compilers.

Historically, the standard approach to solving the performance problems encountered in ROLAP systems was to maintain a number of summary fact tables and associated dimension tables. In such a system, whenever data in the fact table are modified, the corresponding summary tables must be adjusted, as well. Despite these problems, relational OLAP systems were widely implemented. At query time, if a summary table was not available with the desired aggregations, the OLAP system had to choose an appropriate summary table containing partially aggregated results, or query the base fact table. The cost of generating the summaries is paid only once, and then can be amortized over many queries.

Figure 4 A simple star schema



SQL extensions

During the 1990s, DB2* (Database 2*) responded to the competitive pressure of the MOLAP vendors by adding a number of features that addressed the difficulties of using DB2 as an OLAP engine. These features address some of the concerns just outlined. Improved grouping operations were added to reduce the number of queries required to access cube data and increase the efficiency of the queries that are executed. Additional summary functions were added to increase the analytical power of DB2. Automatic summary tables or materialized query tables were added to reduce query times and to reduce the extent of redundant query processing. With respect to both function and performance, these features make DB2 a competitive offering as a highly scalable OLAP engine. We describe these major additions to DB2 in more detail in the following subsections.

Aggregate computation. Until a few years ago, relational databases allowed the calculation of aggre-

Table 1 Result of ROLLUP query

country	state	revenue
—	—	235 329.24
CANADA	—	35 754.64
CANADA	ON	35 754.64
USA	—	199 574.60
USA	CA	103 910.41
USA	NY	95 664.19

gates at only a single level per query. This forced relational OLAP systems to run multiple queries against the database in order to calculate cells at varying levels. To facilitate OLAP-type query creation and provide more advanced optimizations, DB2 implemented three new operators that were added to the SQL standard to allow a single query to generate multiple aggregates: ROLLUP, CUBE, and GROUPING SETS. These operators are extensions to the GROUP BY clause and specify that aggregates be generated at multiple levels.[5]

It is important to note that these operators are more than mere shorthand for generating multiple grouping sets. Because multiple grouping sets are requested in a single statement, DB2 can build an execution plan that generates all the grouping sets in such a way that each input row needed for the calculation is referenced only once. This can result in performance improvements of orders of magnitude, especially when the set of input rows does not fit in the buffer pool (cache).

ROLLUP. The ROLLUP operator, an extension of the GROUP BY clause, generates multiple subtotal grouping clauses, based on a list of columns. This has the same effect, in OLAP terms, of a hierarchy calculation in a given dimension. Consider a dimension such as location, which has a hierarchy composed of country, state, and city. The ROLLUP(country, state, city) clause generates the grouping clauses that represent the calculation of the hierarchy. The general specification of a ROLLUP of n elements

$(c_1, c_2, \ldots, c_{n-1}, c_n)$

is equivalent to the following grouping clauses

$(c_1, c_2, \ldots, c_{n-1}, c_n)$
$(c_1, c_2, \ldots, c_{n-1})$
. . .
(c_1, c_2)
(c_1)
$( )$

Note that n elements in a ROLLUP clause translate to $n + 1$ grouping clauses. The empty grouping clause represents the grand total.

It is very common for an OLAP application to have multiple dimensions. A ROLLUP for each dimension returns results that represent an OLAP cube, in a relational way. The combination of more than one ROLLUP operator in a single statement results in the Cartesian product of the grouping clauses generated for each ROLLUP. For example, combining the following pair of ROLLUP operators in a single statement

```
ROLLUP (country, state), ROLLUP (year, month)
```

results in the generation of the following grouping clauses

```
(country, state, year, month)
(country, state, year)
(country, state)
(country, year, month)
(country, year)
(country)
(year, month)
(year)
( )
```

Queries that use ROLLUP operators include all the generated grouping clauses in a single result set. Hence, the result set includes the union of all grouping clause columns, plus the aggregated columns. In order to combine results of different grouping sets, DB2 returns nulls in any grouping columns in which a given row is not a member, as illustrated in the following example. See Table 1 for the result.

```
SELECT country, state, SUM (amt) AS revenue
FROM fact f, location l
WHERE f.lid = l.lid
GROUP BY ROLLUP (country, state)
```

In the example in Table 1, the row with the aggregate revenue for USA is designated by a null (shown as a dash) in the state column. The row with the aggregate revenue for all countries and states is designated by a null in both the country and state columns.

CUBE. The CUBE operator is an extension of the GROUP BY clause that generates *subtotals* for all the permutations of the grouping columns plus the *grand*

Table 2 Result of GROUPING SETS query

country	state	prodline	revenue
CANADA	ON	—	35 754.64
USA	CA	—	103 910.41
USA	NY	—	95 664.19
—	—	CLOTHING	53 912.31
—	—	ELECTRONICS	77 944.12
—	—	MEDICAL	103 472.81

total. OLAP cubes that have one attribute per dimension benefit from the CUBE operator because a complete cube can be calculated with a single statement using a single grouping clause. For example,

```
CUBE (time, product, location)
```

results in the following grouping clauses

```
(time, product, location)
(time, product)
(time, location)
(product, location)
(time)
(product)
(location)
( )
```

Because the CUBE operator generates all the permutations on the columns, the resulting number of grouping clauses for n columns is 2^n. The column order does not matter for the CUBE operator—CUBE (time, customer) and CUBE (customer, time) yield the same result set.

Ironically, because dimensions in real OLAP cubes nearly always have multilevel hierarchies, the CUBE operator is of very little use in large-scale OLAP systems. Applying CUBE to the columns forming the levels of a single, multilevel dimension results in grouping sets that are not generally of practical interest.

GROUPING SETS. The GROUPING SETS operator allows multiple grouping clauses to be specified in a single statement. This can be thought of as the union of two or more groups of rows into a single result set. It is logically equivalent to the union of multiple subselects, with the GROUP BY clause in each subselect corresponding to one grouping set. The following example makes use of two groupings in the GROUPING SETS definition. See Table 2 for the result.

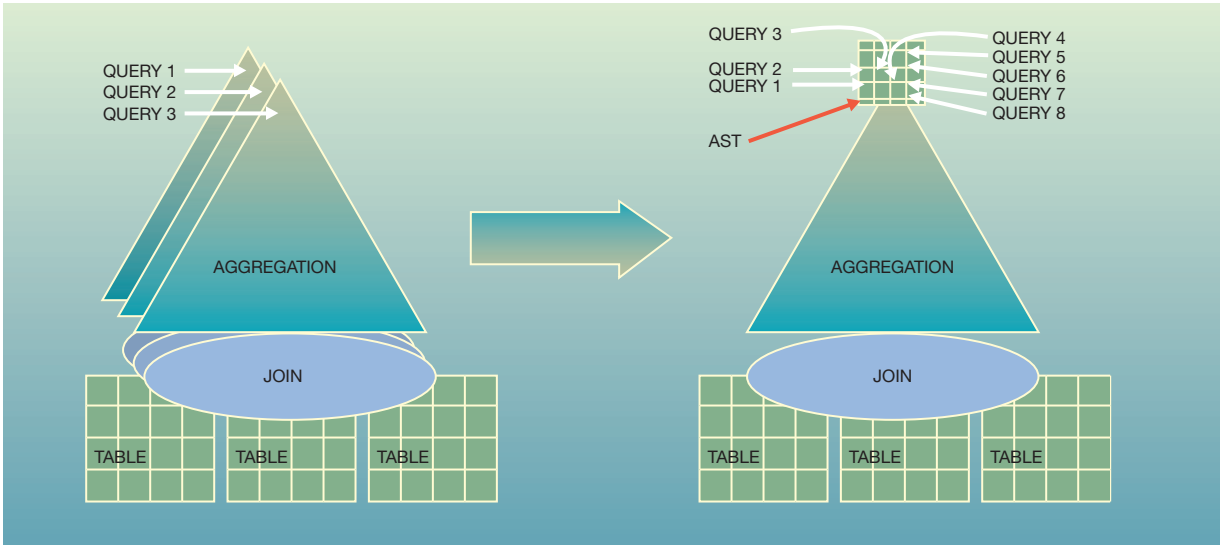
```
SELECT country, state, prodline,
  SUM (amt) AS revenue
FROM fact f, location l, product p
WHERE f.lid = l.lid AND
   f.pid = p.pid
GROUP BY GROUPING SETS ( (country, state),
  (prodline) )
```

In the context of OLAP systems, the primary benefit of the GROUPING SETS operator is in limiting the size of summary tables (discussed in the next subsection). OLAP queries typically have multiple ROLLUP clauses—one for each dimension. The number of grouping sets in a result set is the product of the number of grouping sets for each ROLLUP. The number of grouping sets for each ROLLUP is one more than the number of columns in the ROLLUP. This means that a summary table built for a five-dimensional OLAP cube in which each dimension had three levels would result in $(3 + 1)^5 = 1024$ grouping sets! The GROUPING SETS operator can be used to explicitly choose which grouping sets to use in such a summary table.

Automatic summary tables. Queries running against a data warehouse can easily become computationally expensive, thereby limiting the query throughput of the data warehouse. These queries are frequently aggregations of data, and the queries performed tend to have very similar aggregations. Automatic summary tables (ASTs) were introduced in DB2 to provide a means to precompute and store information that is repeatedly computed and accessed. ASTs are a special type of materialized query table, specialized for holding aggregates, or summaries. Figure 5 shows an example of how ASTs can reduce query time.

On the left side of the figure, similar queries have to go through all of the steps involved in joining fact and dimension tables and computing aggregates. On the right side, the AST stores the common work done

Figure 5 Using ASTs to reduce OLAP query time



for a large number of queries, thereby reducing the computation needed for subsequent queries. One of the most important design points is that DB2 is able to automatically reroute queries that can make use of an existing AST. This allows any application to exploit the advantages of ASTs without having to change how queries are written to the database.

Another key design point is that DB2 provides alternate currency models for AST maintenance. There are two basic modes:

1. *Refresh immediate*. DB2 watches for changes in any of the tables that affect values maintained in an AST. If an insert, update, or delete occurs in a table that is a source table for the AST, DB2 includes the appropriate changes to the AST as part of the originating transaction. In this mode, DB2 is responsible for keeping the AST consistent.
2. *Refresh deferred*. In this mode, changes to source tables do not trigger DB2's automatic AST maintenance. This gives the database administrator (DBA) full control of when the AST maintenance should be performed, and makes the DBA responsible for determining AST currency.

OLAP applications usually require many different aggregates to be computed when a user is manipulating a specific OLAP cube. DB2 provides special support for ASTs to handle OLAP cubes. ASTs can utilize statements that include the GROUP BY extensions. These extensions, such as ROLLUP, offer the database administrator a way to define an OLAP cube inside DB2. When an OLAP application queries the multidimensional schema, DB2 reroutes the query to the AST that has the result already calculated. This allows DB2 to significantly improve the query response time. Another important capability of the AST support allows DB2 to compute nonexisting aggregates by using more general computed aggregates. For example, if DB2 has an aggregate for sales by month, and the query requests sales by quarter, DB2's optimizer is able to calculate quarter figures by reading the month aggregate. This saves processing time, because the number of rows that must be read is smaller.

Normally, an OLAP cube includes multiple dimensions, with one hierarchy per dimension. The following example shows how a simple cube with three dimensions can be defined using an AST.

```
CREATE SUMMARY TABLE CUBE AS (
  SELECT l.country, l.state, l.city,
    p.prodline, p.prodgroup, p.product,
    t.year, t.month,
    SUM(f.amt) AS revenue,
    COUNT (*) AS count
  FROM fact f, location l, product p, time t
  WHERE f.lid = l.lid AND
```

```
    f.pid = p.pid AND
    f.tid = t.tid
  GROUP BY ROLLUP (l.country, l.state, l.city),
    ROLLUP (p.prodline, p.prodgroup, p.product),
    ROLLUP (t.year, t.month)
) AS DATA INITIALLY DEFERRED
      REFRESH IMMEDIATE
```

In this example, the AST includes 48 grouping clauses, due to the combination of the three ROLLUP clauses.

ASTs that define complete cubes tend to have a large number of grouping clauses. This usually results in large ASTs, which increases AST maintenance costs. However, the DBA can create ASTs by utilizing the GROUPING SETS operator. In this case, a smaller number of slices of a cube are defined in the AST, making the AST simpler and smaller. This reduces the maintenance cost for the AST, but defers some costs to query time. This is the DB2 form of the classical tuning exercise for OLAP systems—balancing up-front calculation time and storage expenses vs the performance of queries. Finding the optimal balance usually requires an understanding of the query patterns of users of a given system.

Analytic functions. Another area in which DB2 has improved over the years is in the calculation of analytic functions. OLAP applications usually require more complex calculations than those provided by basic aggregation functions, such as SUM and AVG. Recent extensions include ranking functions, statistical functions, and sliding window operators.

The ranking functions introduced by DB2 compute the ordinal rank of a row with respect to a specific ordering expression. If RANK is specified, the rank of a row is defined as 1 plus the number of rows that strictly precede the row. Thus, if two or more rows are not distinct with respect to the ordering, then there will be one or more gaps in the sequential rank numbering. If DENSE_RANK is specified, the rank of a row is defined as 1 plus the number of preceding rows that are distinct with respect to the ordering. This eliminates the gaps in the sequential rank numbering. The following example shows how to use RANK to rank states and provinces by sales revenue. See Table 3 for the result.

```
SELECT country, state, prodline,
  SUM (amt) AS revenue,
  RANK ( ) OVER
 (ORDER BY SUM (amt) desc) AS rank
FROM fact f, location l, product p
WHERE f.lid = l.lid AND
  f.pid = p.pid
GROUP BY country, state
```

Table 3 Result of query using RANK

country	state	revenue	rank
CANADA	ON	35 754.64	3
USA	CA	103 910.41	1
USA	NY	95 664.19	2

New statistical functions were introduced to improve the support to complex OLAP applications. These functions include correlation, variance, covariance, standard deviation, and a family of linear regression functions.

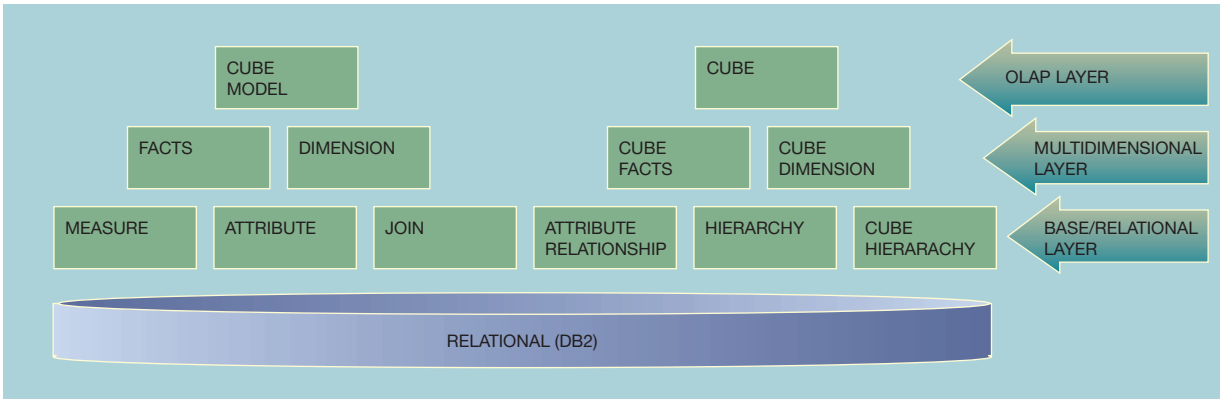
Sliding window operators give the ability to analyze data with respect to a window that is dynamically determined. The window, a set of rows, is determined according to the operation requested, for example, "ROWS BETWEEN 3 PRECEDING AND 3 FOLLOWING." Given such a window, the aggregation function is applied. Sliding window operators are useful in applications designed to calculate, for instance, the seven-day centered average of a stock for each day that it was traded.

Multidimensional meta-data

Multidimensional data have been stored in relational databases for many years. However, the information that describes the structure of these schemas has been stored only in proprietary repositories of specific OLAP tools. Keeping these meta-data apart from the data has prevented easy meta-data exchange among different OLAP tools and has forced DBAs to define the very same information in different places, with all kinds of complications. In this section we propose extensions to the relational database catalog in order to capture such structures, facilitating the exchange of information by having a central repository for the structural information. If this information were available to DB2, its optimizer would be able to use techniques specific to star schemas.

The multidimensional meta-data object model is designed to describe the schemas used in relational databases to represent multidimensional data. The most common way to organize such data is by using a star or snowflake schema (in snowflake schemas the dimension tables are normalized). The paradigm

Figure 6 Multidimensional meta-data objects



utilized to represent multidimensional data in the object model is the star schema. However, the model is flexible enough to handle more normalized schemas. To provide a better understanding of the model, multidimensional meta-data objects are categorized as belonging to one of the following three layers:

- *Base/relational layer*. This provides base infrastructure to other objects and encapsulates important concepts of the relational database.
- *Multidimensional layer*. Common multidimensional constructs are provided in this layer. The objects here reference objects in the base/relational layer, providing a multidimensional abstraction over the relational database.
- *OLAP layer*. This layer contains high-level objects that represent the main OLAP structures. By grouping objects from other layers, the OLAP layer provides OLAP cubes with different degrees of complexity.

Figure 6 shows the objects of the multidimensional meta-data model and the layering of those objects. We propose that these new conceptual objects become an integral part of the relational database catalog. Another key design point for the objects is that many are independently useful—they provide important information about the underlying relational schema, whether or not they are included in a more complex multidimensional structure.

OLAP layer: Cube model vs cube. In our proposal, OLAP concepts are represented in two different levels of complexity in the catalog, offering broader support to the various OLAP applications. A brief description of these objects follows:

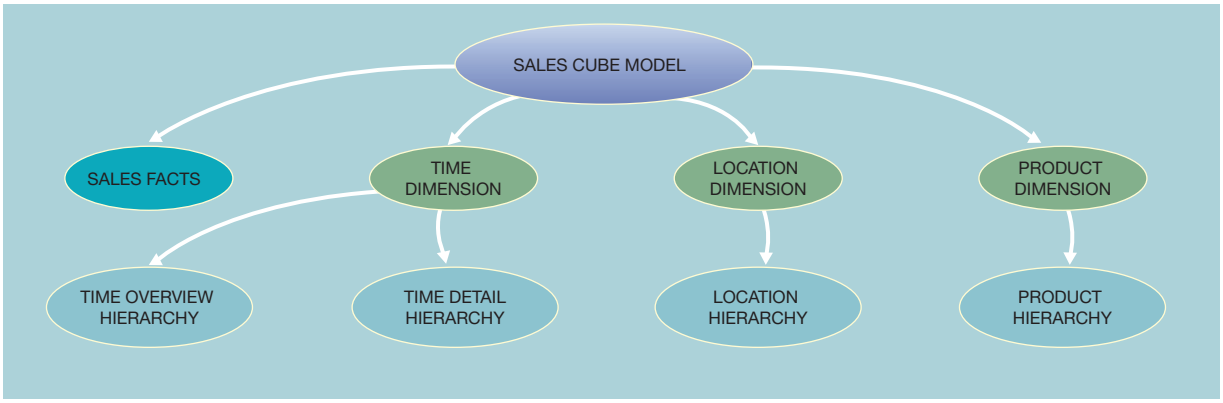
Cube model. The purpose of a *cube model* is to describe OLAP structures to a given application or tool. Cube models tend to describe all views that different users might want for the data that are being analyzed. A cube model groups *dimensions* and *facts*, and offers the flexibility of multiple *hierarchies* for dimensions. It conveys the structural information needed by query design tools and applications that generate complex queries on star schema databases.

Cube. The purpose of a *cube* is to define a standard relational view of an OLAP structure. Because a cube defines a view, it is the only object type that can be queried directly using SQL. In addition to the relational view, a cube provides an XML document that describes the roles of its columns in multidimensional terms.

Each cube is derived from a single cube model. In the process of defining a cube, the designer selects a subset of the possible elements, choosing a single hierarchy for each dimension. This ensures that the cube unambiguously defines a single relational result set. The simplicity of a cube makes it useful to less sophisticated OLAP applications, such as portable devices powered by Web services. Figure 7 shows a cube model created based on the star schema of Figure 4. Note that the time dimension references two different hierarchies.

Multidimensional layer. The objects in our proposed multidimensional layer are used as building blocks

Figure 7 Cube model example



to describe OLAP environments. Most objects refer to other objects in order to provide a higher level of abstraction. The main objects are described here.

The *facts* object plays the role of a fact table in a star schema. Just as a fact table does, a facts object gathers measurement entities, represented in the catalog by *measures*. These need not come from the same table, allowing the designer to group measures as required for any OLAP application.

The *dimension* object plays the role of a dimension table in a star schema. Dimensions group related *attributes*, which together describe some aspect of one or more measures. Dimensions also refer to *hierarchies* that can be applied, providing a way to navigate and make calculations.

Cube fact and *cube dimension* objects are used only in the context of a cube. They are used to reduce the scope of a cube model so that a cube contains only those attributes and measures that are required for a given application or report. A cube dimension also refers to a single *cube hierarchy*, which defines navigational and computational paths among the elements of the cube dimension.

Base/relational layer. In our proposal, the base/relational layer contains the most elementary objects. These are primarily used in objects of the multidimensional layer, though in some cases they are used in the OLAP layer.

A *hierarchy* describes parent-child relationships among attributes. This information is referred to by a dimension to indicate how dimension members can be browsed, and how to aggregate data in the dimension.

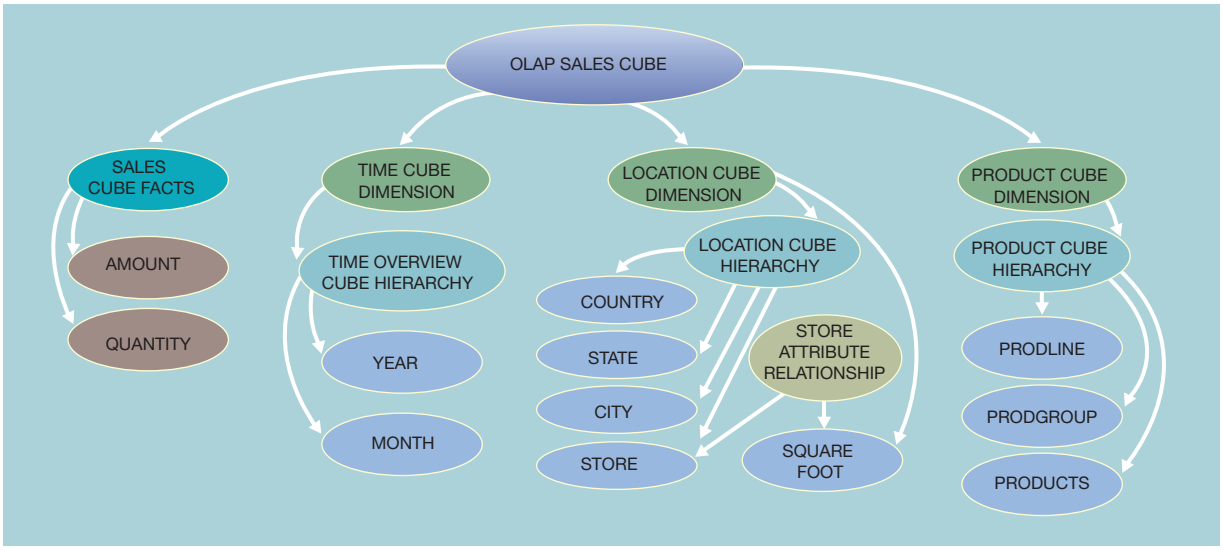
A *cube hierarchy* is very similar to a hierarchy; however, a cube dimension refers to only a single cube hierarchy. This restriction allows a single SELECT statement to calculate the cells of a cube.

Attribute and *measure* objects are abstractions of a relational database column. However, they are defined by an SQL expression that can include multiple columns. Measures are more specialized than attributes—they include aggregation functions (column functions) that are used to calculate higher-level summaries from lower-level data.

Join and *attribute relationship* objects define relationships between two attributes. The join object describes which relational joins are meaningful to the designer of the multidimensional model. Joins are used to describe the connections between tables in a database. They are referred to by cube model, facts, and dimension objects. An attribute relationship describes a functional relationship in which the value of the first attribute determines the value of the second attribute. For example: the city attribute determines the mayor attribute.

OLAP sales cube example. Figure 2 illustrates a simple OLAP cube with three dimensions and two measures. The sales cube is described in this section uti-

Figure 8 OLAP sales cube meta-data



lizing the multidimensional meta-data. Figure 8 shows the multidimensional objects created for the example and the relationship among those objects.

Using our meta-data schema, we might create a view as part of the definition of a cube object named sales. The data values for all of the cells in our example cube could then be calculated simply by SELECT * FROM sales. To do so, in the SELECT clause of the view, the fact and dimension tables are joined and ROLLUP operators are used to compute subtotals and totals of revenue and volume. Figure 9 shows the annotated result set illustrating the terminology used to describe the example cube shown in Figure 2. Note that in practice, querying all of the cells of a cube would be quite unusual—most real cubes are too large for such a query to be practical. Instead, the sales cube defined here would be queried using predicates to limit the data retrieved to the subset of interest for a report or analytical exercise.

```
CREATE VIEW sales AS (SELECT
    p.prodline, p.prodgroup, p.products,
    l.country, l.state, l.city,
    l.locid AS store, l.size AS square_foot,
    t.year, t.month,
    SUM (f.amt) AS amount,
    SUM (f.qty) as quantity
  FROM fact f,
    time t,
    product p,
    location l
  WHERE f.tid = t.tid AND
    f.pid = p.pid AND
    f.lid = l.lid AND
  GROUP BY
    ROLLUP (p.prodline, p.prodgroup, p.products),
    ROLLUP (l.country, l.state, l.city, (l.locid, l.size)),
    ROLLUP (t.year, t.month));
```

Web services for OLAP

The software industry is shifting toward a service-oriented architecture over existing infrastructure, including HTTP and XML. New applications run in small devices such as cellular phones, thin Web clients deploy a browser interface, and thick clients perform some data analysis and visualization. Easy access to remote data through the Web, without any client downloads, is crucial. The Web service framework provides standards for protocols such as SOAP over HTTP, interface descriptions through WSDL (Web Services Description Language), and service discovery through UDDI registries. In the context of the Web service framework, Web services for OLAP provide access to OLAP data, including its meta-data (cube model, cubes) as sketched in Figure 1. A client application can discover OLAP providers in UDDI registries, retrieve XML descriptions of cube models

Figure 9 Sales cube

PRODUCT DIMENSION LEVELS | LOCATION DIMENSION LEVELS | MEMBER ATTRIBUTE | TIME DIMENSION LEVELS

MEMBERS | MEASURES

ProdLine	ProdGroup	Products	Country	State	City	Store	Square_Foot	Year	Month	Amount	Quantity
-	-	-	-	-	-	-	-	-	-	129033	15422
-	-	-	-	-	-	-	-	2000	-	64845	7717
-	-	-	-	-	-	-	-	2000	2	31426	3712
-	-	-	-	-	-	-	-	2000	1	33419	4005
-	-	-	-	-	-	-	-	1999	-	64188	7705
-	-	-	-	-	-	-	-	1999	2	31675	3769
-	-	-	-	-	-	-	-	1999	1	32513	3936
...											
-	-	-	USA	NY	-	-	-	1999	-	20105	2171
-	-	-	USA	NY	-	-	-	1999	2	9410	1016
-	-	-	USA	NY	-	-	-	1999	1	10695	1155
-	-	-	USA	NY	YKT	-	-	1999	-	20105	2171
-	-	-	USA	NY	YKT	-	-	1999	2	9410	1016
-	-	-	USA	NY	YKT	-	-	1999	1	10695	1155
-	-	-	USA	NY	YKT	300	600	1999	-	20105	2171
-	-	-	USA	NY	YKT	300	600	1999	2	9410	1016
-	-	-	USA	NY	YKT	300	600	1999	1	10695	1155
...											
MED	-	-	-	-	-	-	-	1999	-	16517	2172
MED	-	-	-	-	-	-	-	1999	2	7361	938
MED	-	-	-	-	-	-	-	1999	1	9156	1234
...											
CLOTH	JEANS	BLUE	Canada	ON	TOR	-	-	1999	2	127	48
CLOTH	JEANS	BLUE	Canada	ON	TOR	-	-	1999	1	122	45
CLOTH	JEANS	BLUE	Canada	ON	TOR	400	900	1999	-	249	93
CLOTH	JEANS	BLUE	Canada	ON	TOR	400	900	1999	2	127	48
CLOTH	JEANS	BLUE	Canada	ON	TOR	400	900	1999	1	122	45

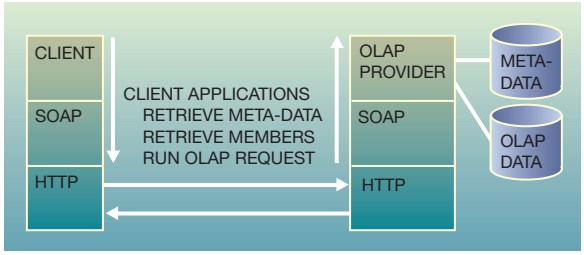
224 record(s) selected.

and cubes, and execute slicing and dicing queries on XML cubes.[6] The Web services approach for OLAP discussed in this paper focuses on XML cubes for ROLAP with XPath (XML Path Language) for slicing queries.

Web services in the context of OLAP. Figure 10 sketches the architecture of Web services for OLAP. An OLAP provider offers Web services to retrieve meta-data, to execute slice and dice queries, and to retrieve member data.

A client application composes a SOAP request envelope and sends it through SOAP and HTTP to the OLAP provider. To respond to the request, the OLAP provider computes a result and sends a SOAP response envelope back to the client application. In the subsequent sections, we describe the SOAP envelopes for the Web services for OLAP.

Figure 10 Web services for OLAP



DescribeCube Web service. The DescribeCube( ) Web service allows a client application to retrieve meta-data about cubes. The meta-data include all of the data necessary for a client application to con-

Figure 11 Virtual XML document of the DescribeCube( ) Web service

```
<Cubes>
  <Sales businessName="Sales Cube" description="Sales figures for Company IBM">
    <Product kind="dimension" businessName="Product Dimension" definer="John Doe">
      <ProductLine kind="level" description="Product lines for Company IBM">
        <ProductGroup kind="level" description="Product Groups for Company IBM">
          <Products kind="level" />
        </ProductGroup>
      </ProductLine>
    </Product>
    <Location kind="dimension">
      <Country kind="level">
        <State kind="level">
          <City kind="level">
            <Store kind="level" square_foot="integer" />
          </City>
        </State>
      </Country>
    </Location>
    <Time kind="dimension" businessName="Time Dimension" description="Timeline for Company IBM">
      <Year kind="level">
        <Quarter kind="level">
          <Month kind="level" />
        </Quarter>
      </Year>
    </Time>
    <Amount kind="measure" businessName="Sales Figures" datatype="float" aggregator="sum" />
    <Quantity kind="measure" businessName="Product Quantities" />
  </Sales>
  <Purchases businessName="Purchases Cube" description="Purchases by Company IBM">
    ...
  </Purchases>
</Cubes>
```

struct and submit slicing queries through the Execute( ) Web service (see next subsection). The meta-data do not include any members or measures data. For conceptual purposes, we assume that all of the meta-data are organized in a virtual XML document. The meta-data for the sales cube shown in Figure 2 are listed in the XML document in Figure 11. The XML document contains a sales XML element with three XML subelements for product, location, and time dimensions. Contained in the XML elements for dimensions are XML elements for the levels in the dimensions. Measures are represented in a similar way.

A client application may ask for particular information in the virtual XML document through an XPath query expression. For example, a client application may ask for a description of the product dimension in the sales cube as

```
DescribeCube ('/Cubes/Sales/Product')
```

In an informal way, the signature of the DescribeCube( ) Web service is defined as follows:

```
DescribeCube ( IN XPath-Predicate AS string,
               OUT Description AS XML )
```

This signature can be translated into a WSDL description of the Web service. The format of the output message is described in XML schema with an instance as described in Figure 11.

Execute Web service. The Execute Web service retrieves the XML representation of a cube. An XML cube contains member and measure data. Figure 12 shows the XML cube for the example in Figure 2. The Sales root element contains two top-level XML elements: Members and Measures. The Members el-

Figure 12 XML document from the Execute( ) Web service

```
<Sales>
  <Members>
    <Product>
      <ProductLine name="ELECTRO">
        <ProductGroup name="VCR">
          <Products name="SDB"/>
        </ProductGroup>
        <ProductGroup name="CCOR">
          <Products name="PHON"/>
        </ProductGroup>
      </ProductLine>
      <ProductLine name="MED">
        <ProductGroup name="ANBIO">
          <Products name="C225"/>
        </ProductGroup>
      </ProductLine>
    </Product>
  <Location>
    <Country name="USA">
      <State name="NY">
        <City name="YKT">
          <Store name="300" square_foot="600"/>
        </City>
      </State>
    </Country>
  </Location>
   ...
  </Members>

<Measures>
   <cell amount="129033" quantity="15422" />
   <cell year="2000" amount="64845" quantity="7717" />
   <cell year="2000" month="2" amount="31426"
     quantity="3712" />
   <cell year="2000" month="1" amount="23419"
     quantity="4005" />
   <cell year="1999" amount="64188" quantity="7705" />
   <cell year="1999" month="2" amount="31675"
     quantity="3769" />
   <cell year="1999" month="1" amount="127"
     quantity="48" />
        ...
   <cell  prodLineName="CLOTH"
     prodGroupName="JEANS" prodName="BLUE"
     country="Canada" state="ON" city="TOR"
     year="2000" amount="14.00"  quantity="2 " />
   <cell prodLineName="CLOTH"
     prodGroupName="JEANS" prodName="BLUE"
     country="Canada" state="ON" city="TOR"
     year="2000" month="2"
     amount="14.00" quantity="2 " />
 </Measures>
```

ement contains one XML element per dimension (using the dimension name as the element name) describing the levels of the dimension and their member values. The Measures element contains one XML cell element per row in the result set (see Figure 9). The column values in the result set are represented as attributes of the cell element. Attributes for null values are absent. Attributes with member values identify a cell in a cube. Attributes with measures, amount, and quantity, in our example, contain the aggregated values as shown in Figure 12.

The algorithm for mapping a relational result set to an XML cube works as follows. The root element of the XML cube is the name of the cube. Contained in the root element are the members and measures elements. The members XML element contains, per dimension in the cube, a dimension element with the same dimension name as the XML element name. Contained in the dimension elements are the level XML elements. The level names are used as XML element names. The member values are attribute values of the level XML elements. For example, level ProductLine in dimension Product has an attribute called name with member value "Electro." The levels are nested according to the cube definition. The measures XML element contains a "cell" XML element per row in the ROLAP result set. The columns in the result set become attributes with the column name as attribute name. Null values are absent attributes.

The Execute Web service supports two kinds of predicates: content restriction, with or without reachability, and shape subsetting.

Content-restriction predicates filter rows in the ROLAP result set, and consequently remove cell XML elements in the XML cube. Content restriction without reachability means that the corresponding members remain in the XML cube, whereas reachability means that only dimension members that are used to identify cells for measures remain. Shape subsetting allows identification of the dimensions and levels that should be retrieved in an XML cube.

The signature of the Execute( ) Web service is defined as follows:

```
Execute ( IN Cube-Identifier AS string
          [,IN Content-Restriction [WITH REACHABILITY]]
          [, IN Shape-Subsetting AS string]
          , OUT XMLCube AS XML)
```

For example, the query that returns the entire sales cube is specified as follows:

```
Execute ('Sales')
```

It returns the XML cube in Figure 12.

The query for product group "VCR" is expressed through an XPath predicate on the product group (with reachability):

```
Execute ('Sales',
        '/Sales/Product/Productline/Productgroup
                        [@ProdGroupName="VCR"]')
```

This XPath expression does not change the members results in the XML cube. However, the XPath predicate is semantically translated into a corresponding SQL predicate on the relational result set:

```
SELECT . . .
FROM Sales
WHERE ProdGroupName='VCRS'
```

The same Web service invocation with reachability means that members in the XML cube that are not used to identify measures are filtered out.

Shape subsetting in a Web service is necessary to select dimensions and levels in an XML cube with many dimensions and many levels. The following query retrieves the sales cube for product group "VCR" with location dimension to level "country" and time dimension to level "year":

```
Execute ('Sales',
        '/Sales/Product/ProductLine/ProductGroup
                        [@ProdGroupName="VCR"]',
        '/Sales/Time/Year' AND '/Sales/Location/Country')
```

Semantically this is equivalent to an SQL query, on the ROLAP result set, that filters rows for which everything is null except the year and country columns.

Members Web service. The Members( ) Web service returns just member data without any measure. The signature of the Members( ) Web service is defined as follows:

```
Members ( IN Cube-Identifier AS string
          [,IN Predicate AS String]
          [,OUT Result AS XML] )
```

For example, the call

```
Members ('Sales', '/Sales/Members/Product/
                ProductLine  [@Name="ELECTRO"]/
                                ProductGroup')
```

returns all product group names in product line "Electro" in the sales cube:

```
<Sales>
      <ProductGroup Name="VCR"/>
      <ProductGroup Name="CCOR"/>
</Sales>
```

Conclusion and a look at the future

The past decade has seen great strides in the functionality provided by DB2 to address multidimensional analysis. With respect to specialized OLAP systems, it has been transformed from a spectator to a viable competitor, incorporating comparable computational and management capabilities. Although some will certainly survive, DB2 and other relational engines seem to be on a path that will relegate specialized engines to the service of niche markets.

The introduction of the cube object provides a model that shifts the access paradigm of relational OLAP. Applications now can, in a single SQL query, retrieve multidimensional data at varying levels of aggregation, without any need to include code to manage summary tables and aggregation currency. DB2 is poised to assume responsibility for storage and calculation management activities that, until now, have been handled by relational OLAP products and applications.

The last decade has also seen a dizzying progression of changes in the Internet and the development of Web applications. Widespread acceptance of Web services, as the model for Web-based applications to locate and interact with each other, promises to provide the basis upon which virtually any electronic device could include analytical applications that deliver on the promise of OLAP—consistently fast queries, using a flexible framework, in terms familiar to its users.

Although much progress has been made, there is ample opportunity for improvement. Better performance in the relational engine, with less manual intervention, is still needed. The Web services described here provide a means to query a predefined OLAP cube. We are currently working on extensions to these services to allow more sophisticated Web-service applications to find dimensional data repositories and dynamically design, create, and query OLAP objects.

*Trademark or registered trademark of International Business Machines.

**Trademark or registered trademark of Hyperion Solutions Corporation, Oracle Corporation, Applix, Inc., or Pilot Software.

Cited references

1. E. F. Codd, *Providing OLAP to User Analysts: An IT Mandate*, unpublished manuscript, E. F. Codd and Associates (1993). This paper may be obtained from Hyperion Solutions Corporation at their Web site, http://www.hyperion.com.
2. N. Pendse, "What Is OLAP?" *The OLAP Report*, see http://www.olapreport.com/fasmi.htm.
3. E. Thomsen, *OLAP Solutions*, John Wiley & Sons, Inc., Hoboken, NJ (1997).
4. R. Kimball, *The Data Warehouse Toolkit*, John Wiley & Sons, Inc., Hoboken, NJ (1996).
5. J. Gray, A Bosworth, A. Layman, and H. Pirahesh, "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total," *Proceedings, 12th International Conference on Data Engineering*, New Orleans, LA (February 26–March 1, 1996), pp. 152–159.
6. *XML for Analysis Specification*, Version 1.0, Microsoft Corporation and Hyperion Solutions Corporation (April 24, 2001).

Accepted for publication August 19, 2002.

Nathan Colossi *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: colossin@us.ibm.com).* Mr. Colossi is a staff software engineer in the DB2 OLAP Server Development department at IBM's Silicon Valley Laboratory. He has been helping in the efforts to integrate OLAP and relational technology in DB2, specifically in the OLAP meta-data area. Mr. Colossi joined IBM in 2000 after receiving his master's degree in multidimensional indexing structures from the University of Campinas, Brazil.

William Malloy *IBM Software Group, Silicon Valley Laboratory, 555 Bailey Avenue, San Jose, California 95141 (electronic mail: malloy@us.ibm.com).* Mr. Malloy is a Senior Technical Staff Member in the Business Intelligence and Database Technology organization at IBM's Silicon Valley Laboratory. He is the architect for IBM's DB2 OLAP Server, and a leader in efforts to integrate OLAP and relational technology in DB2. Mr. Malloy has 30 years experience in business, scientific, and operating systems development. His work in decision support software began in 1988 when he joined Metaphor Computer Systems as a development manager for database middleware. He joined IBM in 1994 when Metaphor's operations were merged with IBM.

Berthold Reinwald *IBM Research Division, Almaden Research Center, 650 Harry Road, San Jose, California 95120 (electronic mail: reinwald@almaden.ibm.com).* Dr. Reinwald joined the IBM Almaden Research Center in 1993, after finishing his Ph.D. degree in computer science from the University of Erlangen-Nuernberg. His Ph.D. thesis on workflow management received the "best Ph.D. thesis" award from the university and was published as a book. At IBM Research, Dr. Reinwald contributed to SMRC (shared memory-resident cache) in DB2 Common Server, query explain tools, workflow management with Lotus Notes®, FlowMark®, and MQSeries, researched and delivered in DB2 Universal Database® support for OLE/COM, OLEDB, XML, and most recently Web services. Dr. Reinwald is active in the design, architecture, and implementation of SQL extensions for XML.