

53

Multidimensional Databases and OLAP

	53.1 Introduction.....	53-1
	53.2 Background	53-3
	Related Terminology • Multidimensional History	
	53.3 Spreadsheets and Relations	53-4
	53.4 Core Multidimensional Concepts.....	53-5
	Data Cubes • Dimensions • Facts • Measures	
	53.5 OLAP Functionality and Implementation	
	Techniques	53-9
	Querying • Multidimensional Versus Relational OLAP	
	• Relational OLAP Schemas • Achieving Good OLAP	
	Query Performance	
	53.6 Handling of Slowly Changing Dimensions	53-15
	The Problem • Solutions • Beyond Slowly Changing	
	Dimensions	
	53.7 Complex Multidimensional Data.....	53-18
	53.8 Survey of Multidimensional Data Models.....	53-19
	Requirements for Data Analysis • Existing	
	Multidimensional Models	
Christian S. Jensen	53.9 Commercial OLAP Systems	53-24
Aalborg University	53.10 Summary	53-25
Torben Bach Pedersen		
Aalborg University		

53.1 Introduction

The relational data model, which was introduced by Codd in 1970 and earned him the Turing Award a decade later, constitutes a significant part of the foundation of today's multi-billion-dollar database industry. During the 1990s, a new type of data model, the *multidimensional data model*, emerged that has since taken over from the relational model when the objective is to *analyze* data, rather than to perform on-line transactions.

Multidimensional data models are designed expressly to support data analyses. A number of such models have been proposed by researchers from academia and industry. In academia, formal mathematical models have been proposed, while the industrial proposals have typically been specified more or less *implicitly* by the concrete software tools that implement them.

Briefly, multidimensional models categorize data as being either *facts* with associated numerical *measures*, or as being *dimensions* that characterize the facts and are mostly textual. For example, in a retail business, *products* are sold to *customers* at certain *times* in certain *amounts* and at certain *prices*. A typical fact would be a *purchase*. Typical

measures would be the amount and price of the purchase. Typical dimensions would be the location of the purchase, the type of product being purchased, and the time of the purchase. Queries then aggregate measure values over ranges of dimension values to produce results such as the total sales per month and product type.

Multidimensional data models have three important application areas within data analysis. First, multidimensional models are used in *data warehousing*. Briefly, a data warehouse is a large repository of integrated data obtained from several sources in an enterprise for the specific purpose of data analysis. Typically, this data is modeled as being multidimensional, as this offers good support for data analyses.

Second, multidimensional models lie at the core of *On-Line Analytical Processing* (OLAP) systems. Such systems provide fast answers to queries that aggregate large amounts of so-called detail data to find overall trends, and they present the results in a multidimensional fashion. Consequently, a multidimensional data organization has proven to be particularly well suited for OLAP.

The widely acknowledged “OLAP Report” company [Rep07] provides an “acid test” for OLAP by defining OLAP as “Fast Analysis of Shared Multidimensional Information” (FASMI). In this definition, “Fast” refers to the expectation of response times that are within a few seconds, “Analysis” refers to the need for easy-to-use support for business logic and statistical analyses, “Shared” suggests a need for security mechanisms and concurrency control for multiple users, “Multidimensional” refers to the expectation that a data model with hierarchical dimensions is used, and “Information” suggests that the system must be able to manage all the required data and derived information.

Third, multidimensional data are increasingly becoming the basis for *data mining*, where the aim is to (semi-) automatically discover unknown knowledge in large databases. Indeed, it turns out that multidimensionally organized data are also particularly well suited for the queries posed by data mining tools.

Updating and extending a previous paper by the authors [PJ01], this chapter describes fundamental concepts in multidimensional data models. In doing so, the chapter aims to communicate the essence of the multitude of existing models in a clear and easily understood form. It is our hope that the chapter will serve as an introduction of the concepts and benefits of multidimensional databases, which are well-known in the database community by now, to the broader community of computer science. The chapter also covers more advanced issues, such as the handling of changing dimensions, termed slowly changing dimensions, and the handling of complex multidimensional data. Additionally, the chapter offers a survey of the design properties of a number of existing multidimensional data models. More in-depth coverage of the features of individual models may be found elsewhere (e.g., [PJD01, VS99]). The chapter also offers a brief coverage of implementation technologies such as bitmap indices, pre-aggregation, and star join query processing. Finally, the chapter includes an overview of the most prominent commercial OLAP systems.

The chapter outline is as follows. Section 53.2 presents relevant terminology and the history of multidimensional databases and OLAP. Section 53.3 motivates multidimensional databases by discussing the deficiencies of the immediate alternatives: spreadsheets and relations. Section 53.4 defines core multidimensional concepts such as cubes, dimensions, facts, and measures. Section 53.5 describes OLAP functionality and implementation techniques. Section 53.6 covers the handling of slowly changing dimensions, and Section 53.7 discusses complex multidimensional data. This is followed by Section 53.8 that presents a survey of multidimensional data models. Next, Section 53.9 describes commercial OLAP systems. Finally, Section 53.10 summarizes the chapter.

53.2 Background

53.2.1 Related Terminology

It is useful to be familiar with a few special terms when studying the literature on issues related to multidimensional databases.

OLAP:

OLAP abbreviates *On-Line Analytical Processing*. As opposed to the well-known OLTP (On-Line *Transaction* Processing), focus is on data analyses rather than transactions. Furthermore, the analyses occur “On-Line,” i.e., fast, “interactive” query response is required. OLAP systems always employ a multidimensional view of data.

Data Warehouse:

A data warehouse (DW) is a repository of integrated enterprise data that is used specifically for decision support, i.e., there is (typically, or ideally) only one data warehouse in an enterprise. The data in a DW is typically collected from a large number of sources within (and sometimes also outside) the enterprise.

Data Mart:

A data mart (DM) is a subset of a data warehouse that is specialized for the needs of a special user group, e.g., the marketing department.

ETL:

ETL (Extract-Transform-Load) is the three-step process that puts data into the DW. First, data is *extracted* from the operational source systems, e.g., ERP systems. Second, data is *transformed* from the source system formats into the DW format. This includes combining data from several sources and performing *cleansing* to correct errors such as missing or wrong data. Third, data is *loaded* into the DW. ETL is at times also referred to as ETT (Extract-Transform-Transport).

Business Intelligence:

Business Intelligence (BI) is the process of making “intelligent” business decisions by analyzing available data. From a technology point of view, BI covers the combined areas of data warehousing, reporting, OLAP, data mining, some data visualization, what-if analysis, and special-purpose analytical applications.

53.2.2 Multidimensional History

Multidimensional databases do not have their origin in database technology, but stem from multidimensional matrix algebra, which has been used for (manual) data analyses since the late 19th century.

During the late 1960s, two companies, IRI and Comshare, independently began the development of systems that later turned into multidimensional database systems. The IRI Express tool became very popular in the marketing analysis area in the late 1970s and early 1980s; it later turned into a market-leading OLAP tool and was acquired by Oracle. Concurrently, the Comshare system developed into System W, which was used heavily for financial planning, analysis, and reporting during the 1980s.

In 1991, Arbor was formed with the specific purpose of creating “a multiuser, multidimensional database server,” which resulted in the Essbase system. Arbor, now Hyperion,

later licensed a basic version of Essbase to IBM for integration into DB2. It was Arbor and Codd who in 1993 coined the term OLAP [Cod93].

Another significant development in the early 1990s was the advent of large *data warehouses* [Kim96], which are typically based on relational *star* or *snowflake* schemas, an approach to implementing multidimensional databases using relational database technology.

In 1998, Microsoft shipped its MS OLAP Server, the first multidimensional system aimed at the mass market. This has led to the current situation where multidimensional systems are increasingly becoming commodity products that are shipped at no extra cost together with leading relational database systems.

A more in-depth coverage of the history of multidimensional databases is available in the literature [Tho97].

53.3 Spreadsheets and Relations

Assume we want to analyze data about sales of music albums, for which we capture the number of albums sold, the album sold, and the city in which it was sold. A simple example with two dimensions is shown in Table 53.1.

Album	City			
	Aalborg	Berkeley	Tucson	Washington DC
USADSB	2072	16	5	32
Feels Like Home	3984	43765	11923	48959
Reise Reise	530	7834	5980	7825
Californication	4309	12567	15784	21348
Baduism	863	2345	879	1693

TABLE 53.1 Sales Data

When considering how to analyze such data, *spreadsheets* immediately come to mind as a possibility—Table 53.1 is just a (two-dimensional) spreadsheet.

Our first analysis requirement is that we do not just want to see sales by album and city, but also the two kinds of subtotals, sales by product and sales by city, and the grand total of sales. This means that formulas for producing the (sub)totals must be added to the spreadsheet, each requiring some consideration. It is possible, if rather cumbersome, to add new data to the spreadsheet, e.g., if new products are sold. Thus, for two dimensions, we can perhaps somehow manage with spreadsheets.

However, if we go to three dimensions, e.g., to include time, we have to consider carefully what to do. The obvious solution is to use separate worksheets to handle the extra dimension, with one worksheet for each dimension value. This will work only when the third dimension has few dimension values and only to some extent. Analyses involving several values of the third dimension are cumbersome, and with many thousands of, say, time dimension values, the solution becomes infeasible. The situation becomes even worse if we need to support four or more dimensions, which in any case will require a very complex set-up.

Another problem arises if we want to group, e.g., the album into higher-level categories like “R&B” and “Pop.” Then we must duplicate the grouping information across all worksheets, which results in a system that uses considerable extra space and is very hard to maintain. The essence of the problem is that spreadsheets tie the storage of data too tightly to the presentation—the *structure* and the *desired views* of the information are not separated.

However, spreadsheets are good for *viewing and querying* multidimensional data, e.g., using *pivot tables*. A pivot table is a 2-dimensional table of data with associated subtotals and totals. For example, if we add subtotals by City and Album and a City/Album grand total to Table 53.1, we have an example of a pivot table. To support viewing of more complex data, several dimensions may be nested on the x or y axes, and data may be displayed on multiple pages, e.g., one for each album. Pivot tables generally also offer support for interactively selecting subsets of the data and changing the displayed level of detail.

With spreadsheets falling short in meeting our requirements for the management of multidimensional data, we may then consider using an SQL-based, relational system for data management, as the relational model offers considerable flexibility in the modeling and querying of data. The problem is here that many desirable computations, including cumulative aggregates (sales in year to date), totals and subtotals together, and rankings (top 10 selling albums), are hard or impossible to formulate in standard SQL.

The main underlying issue is that *interrow* computations are difficult to express in SQL—only *intercolumn* computations are easy to specify. Additionally, transpositions of rows and columns are not easily possible, but rather require the manual specification and combination of multiple views. Although extensions of SQL, such as the *data cube operator* [GCB⁺97] and *query windows* [EM00], advanced by the standards bodies, will remedy some of the problems (see Section 53.5.1 for details), the concept of hierarchical dimensions remains to be handled satisfactorily.

To summarize, neither spreadsheets nor relational databases fully support the requirements posed by advanced data analyses. To be fair, these technologies may be adequate in more restricted circumstances. For example, if we have only few dimensions, do not need hierarchical dimensions, and the data volume is small, spreadsheets may provide adequate support. However, the only robust solution to the above problems is to provide data models and database technology that offer inherent support for the full range of multidimensional concepts.

53.4 Core Multidimensional Concepts

We first offer an overview of the concept of a multidimensional cube, then cover dimensions, facts, and measures in turn.

53.4.1 Data Cubes

Data cubes provide true multidimensionality. They generalize spreadsheets to any number of dimensions. In addition, hierarchies in dimensions and formulas are first-class, built-in concepts, meaning that these are supported without duplicating their definitions. A collection of related cubes is commonly referred to as a *multidimensional database* or a *multidimensional data warehouse*.

We obtain a higher dimensional cube for our music sales example by including additional dimensions. The most pertinent example of an additional dimension is a time dimension, but it is also possible to include other dimensions, e.g., an artist dimension that describes the artists associated with albums. In a cube, the combinations of a dimension value from each dimension define the *cells* of the cube. The actual sales counts are stored in the corresponding cells.

In a cube, dimensions are first-class concepts with associated domains, meaning that the addition of new dimension values is easily handled. Although the term “cube” implies 3 dimensions, a cube can have any number of dimensions. It turns out that most real-world

cubes have 4–12 dimensions [Kim96, Tho97]. Although there is no theoretical limit to the number of dimensions, current tools often experience performance problems when the number of dimensions is more than 10–15. To better suggest the high number of dimensions, the term “hypercube” is often used instead of “cube.”

Figure 53.1 illustrates a three-dimensional cube. We have assumed that the data in Table 53.1 contains aggregated sales for year 2006. The cube then adds a time dimension and contains sales counts for two cities, two albums, and two years.

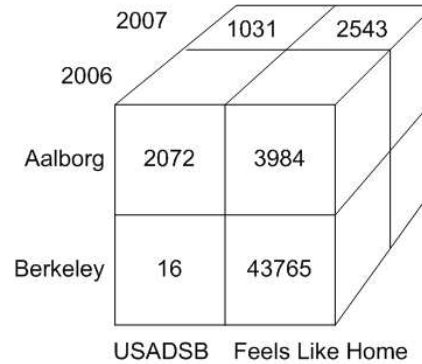


FIGURE 53.1: Sales Data Cube

Depending on the specific application, a highly varying percentage of the cells in a cube are non-empty, meaning that cubes range from *sparse* to *dense*. Cubes tend to become increasingly sparse with increasing dimensionality and with increasingly finer granularities of the dimension values.

A non-empty cell is called a *fact*. The example has a fact for each combination of time, album, and city where at least one sale was made. A fact has associated with it a number of *measures*. These are numerical values that “live” within the cells. In our case, we have one measure, the sales count.

Generally, only 2 or 3 dimensions may be viewed at the same time, although for low-cardinality dimensions, up to 4 dimensions can be shown by nesting one dimension within another on the axes. Thus, the dimensionality of a cube is reduced at query time by *projecting* it down to 2 or 3 dimensions via *aggregation* of the measure values across the projected-out dimensions. For example, if we want to view just sales by City and Time, we aggregate over the entire dimension that characterizes the sales by Album for each combination of City and Time. In our example, we get the total sales for Aalborg in 2006 by adding up the five numbers in the first column in Table 53.1.

An important goal of multidimensional modeling is to “provide as much context as possible for the facts” [Kim96]. The concept of *dimension* is the central means of providing this context. One consequence of this is a different view on *data redundancy* than in relational databases. In multidimensional databases, controlled redundancy is generally considered appropriate, as long as it considerably increases the information value of the data. One reason to allow redundancy is that multidimensional data is often *derived* from other data sources, e.g., data from a transactional relational system, rather than being “born” as multidimensional data, meaning that updates can more easily be handled [Kim96]. However, there is usually no redundancy in the facts, only in the dimensions.

Having introduced the cube, we describe its principal elements, dimensions, facts, and measures, in more detail.

53.4.2 Dimensions

The notion of a dimension is an essential and distinguishing concept for multidimensional databases. Dimensions are used for two purposes: the *selection* of data and the *grouping* of data at a desired level of detail.

A dimension is organized into a containment-like hierarchy composed of a number of *levels*, each of which represents a level of detail that is of interest to the analyses to be performed. The instances of the dimension are typically called *dimension values*. Each such value belongs to a particular level.

In some cases, it is advantageous for a dimension to have *multiple hierarchies* defined on it. For example, a Time dimension may have hierarchies for both *Fiscal Year* and *Calendar Year* defined on it. Multiple hierarchies share one or more common lowest level(s), e.g., Day and Month, and then group these into multiple levels higher up, e.g., Fiscal Quarter and Calendar Quarter to allow for easy reference to several ways of grouping. Most multidimensional models allow multiple hierarchies. A dimension hierarchy is defined in the metadata of the cube, or the metadata of the multidimensional database, if dimensions can be shared. This means that the problem of duplicate hierarchy definitions as discussed in Section 53.3 is avoided.

In Figure 53.2, the schema and instances of a sample *Location* dimension for the data in Table 53.1 are shown. The Location dimension has three levels, the City level being the

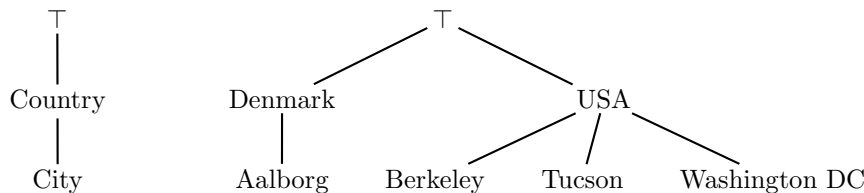


FIGURE 53.2: Schema and Instance for the Location Dimension

lowest. City level values are grouped into *Country* level values, i.e., countries. For example, Aalborg is in Denmark. The \top (“top”) level represents *all* of the dimension, i.e., every dimension value is part of the \top (“top”) value.

In some multidimensional models, a level may have associated with it a number of *level properties* that are used to hold simple, non-hierarchical information. For example, the duration of an album can be a level property in the Album level of the Music dimension. This information could also be captured using an extra Duration dimension. Using the level property has the effect of not increasing the dimensionality of the cube.

Unlike the linear spaces used in matrix algebra, there is typically no ordering and/or distance metric on the dimension values in multidimensional models. Rather, the only ordering is the containment of lower-level values in higher-level values. However, for some dimensions, e.g., the Time dimension, an ordering of the dimension values is available and is used for calculating cumulative information such as “total sales in year to date.”

Most models require dimension hierarchies to form *balanced trees*. This means that the dimension hierarchy must have uniform height everywhere, e.g., all departments, even small

ones, must be subdivided into project groups. Additionally, direct links between dimension values can only go between immediate parent-child levels, and not jump two or more levels. For example, all cities are first grouped into states and then into countries, cities cannot be grouped directly under countries (as is the case in Denmark which has no states). Finally, each non-top value has precisely one parent, e.g., a product must belong to exactly one product group. In Section 53.7 we discuss the relaxation of these constraints.

53.4.3 Facts

Facts are the objects that represent the *subject* of the desired analyses, i.e., the interesting “thing,” or event or process, that is to be analyzed to better understand its behavior.

In most multidimensional data models, the facts are *implicitly* defined by their combination of dimension values. If a non-empty cell exists for a particular combination, a fact exists; otherwise, no fact exists. (Some other models treat facts as first-class objects with a separate identity [PJD01].) Next, most multidimensional models require that each fact be mapped to precisely one dimension value at the lowest level in each dimension. Other models relax this requirement [PJD01].

A fact has a certain *granularity*, determined by the levels from which its combination of dimension values are drawn. For example, the fact granularity in our example cube is “Year by Album by City.” Granularities consisting of higher-level or lower-level dimension levels than a given granularity, e.g., “Year by Album Genre by City” or “Day by Album by City” for our example, are said to be *coarser* or *finer* than the given granularity, respectively.

It is commonplace to distinguish among three kinds of facts: *event* facts, *state* facts, and *cumulative snapshot* facts [Kim96]. Event facts (at least at the finest granularity) typically model *events in the real world*, meaning that a unique instance, e.g., a particular sale of a given product in a given store at a given time, of the overall real-world process that is captured, e.g., sales for a supermarket chain, is represented by one fact. Examples of event facts include sales, clicks on web pages, and movement of goods in and out of (real) warehouses (flow).

A snapshot fact models the *state* of a given process at a given point in time. Typical examples of snapshot facts include the inventory levels in stores and warehouses, and the number of users using a web site. For snapshot facts, the same object, e.g., a specific can of beans on a shelf, with which the captured real-world process, e.g., inventory management, is concerned may occur in several facts at different time points.

Cumulative snapshot facts are used to handle information about *a process up to a certain point in time*. For example, we may consider the total sales in year to date as a fact. Then the total sales up to and including the current month this year can be easily compared to the figure for the corresponding month last year.

Often, all three types of facts can be found in a given data warehouse, as they support complementary classes of analyses. Indeed, the same base data, e.g., the movement of goods in a (real) warehouse, may often find its way into three cubes of different types, e.g., warehouse flow, warehouse inventory, and warehouse flow in year-to-date.

53.4.4 Measures

A *measure* has two components: a *numerical property* of a fact, e.g., the sales price or profit, and a *formula* (most often a simple aggregation function such as SUM) that can be used to combine several measure values into one. In a multidimensional database, measures generally represent the properties of the chosen facts that the users want to study, e.g., with the purpose of optimizing them.

Measures then take on different values for different combinations of dimension values. The property and formula are chosen such that the value of a measure is meaningful for all combinations of aggregation levels. The formula is defined in the metadata and thus not replicated as in the spreadsheet example. Although most multidimensional data models have measures, some do not. In these, dimension values are also used for computations, thus obviating the need for measures, but at the expense of some user-friendliness [PJD01].

It is important to distinguish among three classes of measures, namely *additive*, *semi-additive*, and *non-additive* measures, as these behave quite differently in computations.

Additive measure values can be combined meaningfully along any dimension. For example, it makes sense to add the total sales over Album, Location, and Time, as this causes no overlap among the real-world phenomena that caused the individual values. Additive measures occur for any kind of fact.

Semi-additive measure values cannot be combined along one or more of the dimensions, most often the Time dimension. Semi-additive measures generally occur when the fact is of type snapshot or cumulative snapshot. For example, it does not make sense to sum inventory levels across time, as the same inventory item, e.g., a specific album, may be counted several times, but it is meaningful to sum inventory levels across albums and stores.

Non-additive measure values cannot be combined along any dimension, usually because of the chosen formula. For example, this occurs when averages for lower-level values cannot be combined into averages for higher-level values. Non-additive measures can occur for any kind of fact.

53.5 OLAP Functionality and Implementation Techniques

We first characterize the query functionality typically associated with on-line analytical processing, then proceed to consider aspects of the implementation of such functionality.

In particular, it is helpful to distinguish between two major approaches to implementing multidimensional databases, namely multidimensional versus relational on-line analytical processing. In Section 53.5.2, we contrast the two and then consider the latter in additional detail in Sections 53.5.3 and 53.5.4.

53.5.1 Querying

A multidimensional database naturally lends itself towards certain types of queries.

The queries known as *slice and dice* perform selections on a cube, thus reducing the cube. The effects of these on a cube are similar in spirit to the preparation of an onion for cooking. Envision a version of the two-dimensional cube in Table 53.1 that also has a time dimension, with the one in the table covering 2006 and additional two-dimensional cubes covering other years. We may then slice the three-dimensional cube by considering only those cells that concern “Tucson”; and we may further slice this resulting slice by considering only the cells for year “2006.” Doing this, we obtain sales counts for the different albums for a specific location and year. When selecting a single value in a dimension, we effectively reduce the dimensionality of the cube (strictly speaking, a degenerate dimension remains). In the example, only the Music dimension remains. We note that more general selections than on a single value are also possible.

The operations known as *drill-down* and *roll-up* are inverses of each other and make use of dimension hierarchies and measures to perform aggregations. Consider the location dimension in Figure 53.2 together with the cube, where the (additive) measure is the count of sales together with the function SUM. We may roll-up from the City level to the Country

level. This results in the measure values for all cities in the same country being combined into one by the associated formula. In our cube, values for “Berkeley,” “Tucson,” and “Washington DC” are added to obtain a single “USA” value. When applied to the cube in Table 53.1, the resulting cube has two columns instead of four, one for “Denmark” and one for “USA.”

Slicing and dicing may be combined with drill-down and roll-up. Rolling up to the top value in a dimension corresponds in a sense (different from above) to omitting the dimension.

When a multidimensional database consists of several cubes that share one or more dimensions, it is possible to combine the cubes via the shared dimensions, via a so-called *drill-across* operation. In terms of relational algebra, this operation performs a join.

Next, operations are also available that enable the user to manipulate the visualization of a cube, e.g., by *rotating* it to make a different set of dimensions “face the user,” i.e., to see the data grouped by other dimensions.

Queries involving order are very important in data analyses and are thus supported by all multidimensional data analysis tools. These may order cells in results, and they may return only those cells that appear at the top or bottom of the specified order. For example, one may wish to retrieve the two best selling albums in “Berkeley” in year 2006. Such queries are often referred to as *ranking* or *TOP N/BOTTOM N* queries [Tho97].

When it comes to the embedding into a query language of the functionality considered here, two directions exist: extensions to SQL and dedicated multidimensional languages.

OLAP SQL extensions were pioneered by the proposal of the data cube operators CUBE and ROLLUP [GCB⁺97]. The cube operator generalizes GROUP BY, crosstabs, and subtotals using the special “ALL” value that denotes that an aggregation has been performed over all values for one or more attributes, thus generating a subtotal, or a grand total. Consider Table 53.2, which contains a subset of the data in Table 53.1 in a standard table format.

City	Album	Sales
Aalborg	USADSB	2072
Berkeley	USADSB	16
Aalborg	Feels Like Home	3984
Berkeley	Feels Like Home	43765

TABLE 53.2 SalesTable

The SQL CUBE query below computes the total sales by city and album, with subtotals for both city and album, and a grand total of all sales. As the CUBE syntax varies slightly among vendors, please note that the query uses the syntax in [GCB⁺97].

```
SELECT City,Album,SUM(Sales) AS Sales
FROM SalesTable
GROUP BY CUBE City,Album
```

The query result is shown in Table 53.3. Note the ALL’s that represent the subtotals and the grand total. In most RDBMS’s, the ALL value is implemented as a NULL value, with a special function that makes it possible to distinguish it from “real” NULL values.

By now, the SQL standard has adopted the data cube operators [EM00, (IS01)] along with a range of OLAP functionality such as ranking, percentiles, and windowing, as well as various mathematical and statistical functions. The major RDBMS engines now also implement the data cube operators, along with some of the extra functionality mentioned above.

City	Album	Sales
Aalborg	USADSB	2072
Berkeley	USADSB	16
Aalborg	Feels Like Home	3984
Berkeley	Feels Like Home	43765
Aalborg	ALL	6056
Berkeley	ALL	43781
ALL	USADSB	2088
ALL	Feels Like Home	47749
ALL	ALL	49837

TABLE 53.3 Query Result

However, the syntax may vary slightly from engine to engine. The *iceberg cube* [BR99] is a further development of the data cube operator that only returns rows where the measure values exceed some threshold (corresponding to interesting “tip of the iceberg” values).

Within dedicated multidimensional query languages, the by far most prominent and widely language is MultiDimensional eXpressions (MDX) [TSC99, SHW⁺06], which originally was proposed by Microsoft, but is now also used in other OLAP products. Unlike for the SQL extensions, MDX statements directly produce pivot tables (see Section 53.3) as results, making the integration with OLAP client tools very easy.

In MDX, the dimension values are called *members*, and there is a special *Measures* dimension whose values range over the names of the cube measures. In this way, any measure value can be addressed using a combination of members (dimension values), including the measure dimension. Dimensions have one or more hierarchies, each with a number of levels. The top level in each dimension is called “(All)” and contains just one one value, like the \top value described earlier. MDX uses dot notation to refer to members. An MDX query uses a FROM clause for specifying the input cube. A SELECT clause enables the specification of so-called *axes*, the most important axes being the *query axes*. These specify the hierarchies from which the data are retrieved for multiple members. The notion used is similar to that of the GROUP BY clause in SQL. A query can have up to 128 query axes, although queries typically have only a few. There are five “standard” axes, called COLUMNS, ROWS, PAGES, SECTIONS, and CHAPTERS, that correspond to well-known report concepts. Additionally, a query may specify a *slicer axis* in the WHERE clause that specifies the hierarchies from where data is retrieved for a single member.

As an example, consider a cube, SalesCube, with two dimensions, City and Album, and one measure, Sales, built on the data in Table 53.2. Each dimension has just one hierarchy. The City hierarchy contains a (bottom) City level containing the city names and an (All) level containing the single value “All Cities.” The Album hierarchy contains a (bottom) Album level containing the album names and an (All) level containing the single value “All Cities.” The MDX query below shows the sales per city and album, including subtotals per city and album, and a grand total, i.e., it is the MDX version of the previous SQL CUBE query example.

```
SELECT
  { [Album].[Members],[Album].[All Albums] } ON COLUMNS,
  { [City].[Members],[City].[All Cities] } ON ROWS
FROM [SalesCube]
WHERE ( [Measures].[Sales] )
```

Note the use of the “{...}” notation to specify the set of members on each axis, the “.Members” notation to specify all members, and the use of the “[All ...]” members for referring to the top of the dimension hierarchies. The WHERE clause specifies that the Sales measure should be displayed. Table 53.4 shows the MDX query result.

Album	City		
	Aalborg	Berkeley	All Cities
USADSB	2072	16	2088
Feels Like Home	3984	43765	47749
All Albums	6056	43781	49837

TABLE 53.4 MDX query result

53.5.2 Multidimensional Versus Relational OLAP

Multidimensional OLAP (MOLAP) systems [Tho97, TSC99] store data on disk in specialized multidimensional structures. These typically include provisions for handling sparse arrays, and they apply advanced indexing and hashing to locate the data when performing queries [Tho97, TSC99].

In contrast, *Relational OLAP* (ROLAP) systems [Kim96] use relational database technology for storing the data. In order to achieve good query performance, ROLAP systems employ specialized index structures such as bit-mapped indices [GMUW00], as well as materialized views [Win98]. Section 53.5.4 offers more information regarding DW performance.

Generally, MOLAP systems provide faster query response and more space-efficient storage, while ROLAP systems scale better in the number of facts, are more flexible with respect to cube redefinitions, and provide better support for frequent updates. However, the boundaries are changing as MOLAP systems are becoming more scalable.

The virtues of the two approaches are combined in the *Hybrid OLAP* (HOLAP) approach, which generally stores higher-level summary data using MOLAP technology, while using ROLAP technology to store the detail data. Some HOLAP systems allow even more flexibility, giving users a choice between MOLAP and ROLAP at the level of individual cubes and/or materialized aggregates.

53.5.3 Relational OLAP Schemas

As ROLAP is the most commonly used technology for managing the detail data in a data warehouse, we proceed to describe the typical schemas found in ROLAP data warehouses.

ROLAP implementations typically employ *star* or *snowflake* schemas [Kim96], both of which store the data in *fact tables* and *dimension tables*. A fact table holds one row for each fact in the cube. It has a column for each measure. In a row, this column then contains the measure value for the fact that the row represents. A fact table also has one column for each dimension. In a row, these columns contain foreign keys that reference dimension tables.

The difference between star and snowflake schemas lie in their handling of dimensions. A star schema has one dimension table for each dimension. The dimension table has a key column, and it has one column for each level in the dimension. In a row, such a column holds a textual description of a dimension value at its level. Finally, the dimension table contains one column for each level property in the dimension. An example star schema instance for the Sales cube is shown in Table 53.5.

A row in the fact table of the star schema holds the sales count for one particular combination of “Album,” “City,” and “Day.” The fact table also has a foreign key column for each of the three dimensions, Music, Location, and Time. The dimension tables have corresponding key columns and one column for each of their levels, e.g., “CityID,” “City,” and “Country” for the table representing the Location dimension. No column is needed for the T level, as that column would always hold the same value. The key column in a dimension table is typically a “dumb” integer key without any semantics. This has several

AlbumID	Album	Genre	
7493	Baduism	Contemporary Soul	
9436	Feels Like Home	Pop-Jazz	

Music (dimension)

CityID	City	Country
876	Aalborg	Denmark
854	Berkeley	USA

Location (dimension)

DayID	Day	Month	Year
2475	May 11, 2006	May 2006	2006
3456	March 13, 2007	March 2007	2007

Time (dimension)

AlbumID	CityID	DayID	Sale
9436	854	2475	20
7493	854	2475	5
7493	876	3456	2
9436	876	3456	11
9436	876	2475	18

Sales (fact table)

TABLE 53.5 Star Schema For Sales Cube

advantages over the option of using information-bearing keys from the source systems, including better storage use, prevention of problems associated with key re-use, and better support for dimension updates [Kim96].

It can be seen that there will be redundancy in higher-level data. For example, “March 2007” will be present for each day during that month, meaning that it will be repeated 30 times (assuming that at least one album is sold during each day of that month). However, as dimensions typically take up only 1–5% of the total storage required for a cube, redundancy is not a problem space-wise; and since the updates of dimensions are handled centrally, it is also possible to ensure consistency. Thus, it is often a good idea to use redundant dimension tables because this supports simpler formulation of (and better-performing) queries.

Snowflake schemas contain several dimension tables for each dimension, namely one table for each (non- \top) level. This means that redundancy is avoided, which may be advantageous in some situations. The dimension tables contains a key, a column holding textual descriptions of the level values, and possibly columns for level properties. Tables for lower levels also contain a foreign key to the containing level.

Table 53.6 shows an instance of a snowflake schema that holds the same data as does the star schema in Table 53.5. For example, the Day table in Table 53.6 contains an integer key, a date, and a foreign key to the Month table. Note that with this schema, month values will not be replicated.

The choice of star versus snowflake schemas depends highly on the properties desired of the system being developed. For brevity, we omit a full discussion of this aspect.

53.5.4 Achieving Good OLAP Query Performance

The most essential performance-enhancing technique in multidimensional databases is *pre-computation* and its more specialized cousin, *pre-aggregation*. In ROLAP systems, this is referred to as *materialized views*. This technique enables the delivery of response times to queries involving potentially huge amounts of data that are fast enough to allow interactive data analysis.

As an example application of pre-aggregation, we may compute and store (materialize)

GenreID	Genre	
23	Contemporary Soul	
12	Pop-Jazz	

Genre (Music dimension)

AlbumID	Album	GenreID
7493	Baduism	23
9436	Feels Like Home	12

Album (Music dimension)

CountryID	Country	
147	USA	
783	Denmark	

Country (Location dimension)

CityID	City	CountryID
876	Aalborg	783
854	Berkeley	147

City (Location dimension)

YearID	Year	
78	2006	
88	2007	

Year (Time dimension)

MonthID	Month	YearID
45	March 2007	88
70	May 2006	78

Month (Time dimension)

DayID	Day	MonthID
2475	May 11, 2006	70
3456	March 13, 2007	45

Day (Time dimension)

AlbumID	CityID	DayID	Sale
9436	854	2475	20
7493	854	2475	5
7493	876	3456	2
9436	876	3456	11
9436	876	2475	18

Sales (fact table)

TABLE 53.6 Snowflake Schema For Sales Cube

the total sales of a product by country and month. This enables fast answers to queries that ask for the total sales, e.g., by month alone, by country alone, or by quarter and country in combination. These answers may be derived from the pre-computed results alone; access to the bulks of detail data in the data warehouse is unnecessary.

Pre-aggregation has attracted substantial attention in the research community [PJD99]; and recent versions of commercial relational database products, as well as the dedicated multidimensional systems, offer query optimization based on pre-computed aggregates, as well as automatic maintenance of the stored aggregates when the detail data on which the aggregates are based is updated [Win98].

Full pre-aggregation, where all combinations of aggregates are materialized, is infeasible as it takes too much storage and initial computation time. Instead, modern OLAP systems adopt the *practical*, or partial, pre-aggregation approach of materializing only select combinations of aggregates and then re-use these to efficiently compute other aggregates [BPT97, HRU96, SDN98, Tho97, TSC99]. This re-use of aggregates requires a

well-behaved structure of the multidimensional data.

With respect to indexing, standard B-trees are supplemented with so-called *bitmap indices* [GMUW00]. A bitmap index for a given dimension attribute is essentially a collection of bitmaps, one for each possible dimension value. Each bitmap contains one bit per fact table record. A bitmap for a given dimension value has set bits (1s) for the fact records that contain that dimension value; the rest of the bits are unset (0s). By employing fast bit-wise operations, bitmap indices allow very efficient searches for records with a given combination of dimension values. Using various compression schemes they can even work well for medium- or high-cardinality attributes [AYJ00, CI98, WOS04]. Chapter ?? covers bitmap indices in more detail.

Finally, aggregate queries over star schemas, often referred to as *star joins* involve joins of several dimension tables with the fact table, followed by aggregating measure values as specified by the dimension table attributes in the GROUP BY clause. It is very inefficient to process such queries using a standard pair-wise join strategy, as this procedure yields huge intermediary results. Instead, the major RDBMS engines generally recognize such queries and employ specialized star join evaluation algorithms [KTS⁺02, Wei02].

53.6 Handling of Slowly Changing Dimensions

Like any database, a multidimensional data warehouse models, or captures, selected aspects of some reality. Which specific aspects to capture depends on the intended uses of the data warehouse. Due to the dynamic nature of reality, the modeled reality as well as the uses of the data warehouse change over time. Time implies that the data warehouse must be able to evolve in order to continue to serve its purposes. In this section, we consider the handling of changes in a data warehouse that is represented by means of a star schema. As described in Section 53.5.3, such a data warehouse has a central fact table and a number of dimension tables.

53.6.1 The Problem

Recall that a data warehouse models some real-world process that we are interested in studying. The example we have used earlier concerns the sales process for music albums. As the process evolves, new rows are inserted into the fact table and into the dimension tables. In our example in Table 53.5, when sales come in for a new day, that day is entered as a row into the Time dimension table, and a row is entered into the Sales fact table for each combination of an album and a city for which there was at least one sale during that day. If a new album starts to be sold, that album is entered into the Music dimension table, and if sales start to occur in a new city, that city is entered into the Location dimension table. This scenario represents the desired evolution of a data warehouse. In practice, however, it is necessary to be able to cope with other types of change, including the so-called slowly changing dimensions, which occur when the existing rows in dimension tables need to be updated [Kim96]. The mental image intended by this naming is that although rows in dimension tables need to be updated occasionally, this happens infrequently.

Consider Table 53.5 with the slightly revised Music dimension table shown in Table 53.7. The table now includes a Rating column that indicates how well the customers like the album. Both the rating and the genre of an album may change over time. For example, the rating of “Baduism” may drop to “3 stars,” or the genre of “Reise Reise” may be refined to “Tanz-Metall.” The fundamental problem with simply updating the rows accordingly is that rows with old sales in the fact table already exist that refer to the existing rows with

AlbumID	Album	Rating	Genre
7493	Baduism	4 stars	Contemporary Soul
9436	Feels Like Home	5 stars	Pop-Jazz
9948	Reise Reise	5 stars	Metal
9967	Californication	5 stars	R&B

Music (dimension)

TABLE 53.7 Revised Music Dimension Table For The Sales Cube

their specific attribute values. When updating such attribute values, the old fact table rows refer to dimension table rows that have changed. For example, purchases of “Baduism” when rated as “4 stars” now appear to be purchases of an album with only a “3 stars” rating. This way, wrong information is created. As an aside, observe that an update of the genre of the Californication row from “R&B” to “Rock” is acceptable in that this change is a correction of an error. Next, the fundamental problem with not changing the rows to reflect changes of rating and genre classifications is that the data warehouse is outdated. New rows entered into the fact table really need to refer to the updated Music dimension table rows.

53.6.2 Solutions

We will consider three types of approaches to addressing the changes that may occur in dimension table rows.

The first approach is to simply overwrite the old attribute values. As pointed out already, old fact table rows will now refer to dimension table rows that have changed; and if the original state of the data warehouse was correct, the data warehouse now contains incorrect information. The good news is that new fact table rows refer to the right dimension table rows.

This approach is easy to implement, and if the dimension table updates are simply error corrections, as in the case of Californication being classified as an R&B album, the solution is ideal. Further, there may be cases where the inaccuracies introduced by the updates are considered as unimportant. For example, this might be the case for the genre refinement. However, the bottom line is that this approach basically ignores the fundamental problem.

The second approach is to version the rows in the dimension tables. So when a row is to be updated, that row is not modified, and a new row with the updated attribute values is inserted instead. This has the conceptual effect that the dimension tables go from recording rows to recording “version of rows.” The primary key column thus is to be generalized to capture versions of rows instead of rows. This generalization is straightforward when “dumb” keys are used, as is the recommended approach. With this approach, the old fact table rows continue to refer to the original dimension table rows, while new fact table rows refer to the new dimension table rows. In our example, if the rating of “Baduism” changes, we create a new row for this album that differs from the existing row only on the Rating value, and new sales of “Baduism” will refer to this row.

This approach enables the capture of correct information in the data warehouse. It has the effect that the dimension tables increase in size. Now that the dimension tables change over time, it may be relevant to capture accurately this change. This approach is not able to do so. In particular, it is generally not possible to determine when the changes to a dimension occurred.

In case a time dimension is present, which is typically the case in a data warehouse, the first time a fact table row refers to a row in a dimension table can be determined, and this time is an upper bound on when the dimension table row came into existence. The exact

time when a change happens in a dimension can be captured by inserting a special row in the fact table. This row refers to the new dimension row and to the row in the time dimension table that represents the time when the change happened. Another alternative is to introduce two time-valued columns in the dimensions where changes may occur. These columns record the times when the rows were or are valid.

The third approach is quite different. For each dimension table column that may change, an additional version of that column is introduced. This has the effect that we are able to record two values for columns that may change. We use one to record the current value, and we use the other to record the previously current value. When this scheme is applied to our example we obtain the dimension table shown in Table 53.8.

AlbumID	Album	Rating	OldRating	Genre	OldGenre
7493	Baduism	4 stars	3 stars	Contemporary Soul	Contemporary Soul
9436	Feels Like Home	5 stars	5 stars	Pop-Jazz	Pop-Jazz
9948	Reise Reise	5 stars	5 stars	Metal	Tanz-Metall
9967	Californication	5 stars	5 stars	Rock	R&B

Music (dimension)

TABLE 53.8 Music Dimension Table With Versioned Columns

With this approach, fact table rows from before and after the most recent change to an attribute refer to the same dimension table row; and as this row holds both the current value and the most recent previously current value, fact table rows refer directly to two values of the same attribute. This arrangement may be utilized to analyze data across changes. For example, one can study the difference in the distribution of sales across two different genre classifications of albums. This approach is limited by being able to capture only two values for each attribute that can change—this renders the approach useful only for certain special cases. It should also be noted that it is not possible to capture when the changes occur for the attributes.

In summary, the most versatile technique for handling slowly changing dimensions is the row versioning approach, and this approach is generally recommended.

53.6.3 Beyond Slowly Changing Dimensions

It is natural to ask when slowly changing dimensions become rapidly changing, and to consider how to address such cases. When changes to a dimension are frequent and we use row versioning, the effect is that the size of the dimension may get very large. Dimensions with many columns are generally most prone to frequent changes, as the more columns, the more changes may occur. This exacerbates the problem. However, with proper indexing, it is generally possible to contend with large, rapidly changing dimensions.

In special cases where versioning simply yields a dimension that is too large, one possible remedy is to split the dimension into two. One of the new dimensions then holds the static attributes that either do not change or do so only rarely. The other has fewer columns than the original dimension and is thus smaller. To further reduce the size of this dimension, the detailed attribute values may be replaced by ranges of values. For example, if a customer dimension originally recorded the annual income of the customers, the new dimension may instead record annual income ranges, using, e.g., 5 ranges. This limits the number of possible rows in the new table, and it renders changes less frequent. The negative consequences are that the data becomes less detailed, that the (very large) fact table needs an extra foreign-key column, and that some queries may become harder to formulate because the attributes

they refer to are not in two dimension tables rather than one.

53.7 Complex Multidimensional Data

The traditional multidimensional data models and implementation techniques assume that the data being modeled conforms to a quite rigid regime. Specifically, it is typically assumed that all facts map (directly) to dimension values at the lowest levels of the dimensions and only to one value in each dimension. Further, it is assumed that the dimension hierarchies are simply balanced trees. In many cases, this is adequate to support the desired applications satisfactorily. However, situations occur where these assumptions are too rigid for comfort.

In such situations, the support offered by “standard” multidimensional models and systems is inadequate, and more advanced concepts and techniques are called for. A more comprehensive treatment of complex multidimensional data is available in the literature [DPJ03]. We proceed to consider the impact of irregular hierarchies on pre-computation.

Complex multidimensional data are problematic as they are not summarizable [LS97, RS90, HGM05]. Intuitively, data is *summarizable* if the results of higher-level aggregates can be derived from the results of lower-level aggregates. Without summarizability, users will either get wrong query results, if they base them on lower-level results, or computation may be prohibitively time consuming because we cannot use pre-computed lower-level results to compute higher-level results. When it is no longer possible to pre-compute, store, and subsequently reuse lower-level results for the computation of higher-level results, aggregates must instead be calculated directly from base data, which is what leads to the increased computational costs.

It has been shown that summarizability is ensured if the aggregate functions are distributive and the ordering of dimension values are *strict*, *onto*, and *covering* [LS97, PJD01]. Informally, a dimension hierarchy is *strict* if no dimension value has more than one (direct) parent, *onto* if the hierarchy is balanced, and *covering* if no containment path skips a level. Intuitively, this means that dimension hierarchies must be balanced trees. If this is not the case, some lower-level values will be either double-counted or not counted when reusing intermediate query results for the computation of other results.

Figures 53.3 and 53.4 contain two dimension hierarchies: a Location dimension hierarchy that includes a State level, and a Music dimension hierarchy that captures one possible categorization of music into genres. The hierarchy in Figure 53.3 is *non-covering* because Denmark has no states and because Washington DC belongs to no state. If we pre-compute aggregates at the State level, we will have no values for Aalborg, Copenhagen, and Washington DC, which has the effect that facts mapped to these cities will not be taken into account when computing country aggregates from pre-computed State level aggregates.

The hierarchy in Figure 53.4 is *non-onto* because the Rock genre has no further subdivision. If we materialize aggregates at the lowest level, facts mapping directly to Rock will not be counted. The hierarchy is also *non-strict* because the genre Pop-Jazz (the primary genre of, e.g., the artist Norah Jones) is shared between genres Jazz and Pop. If we materialize aggregates at the middle level, data for Pop Jazz will be used twice, for both Jazz and Pop, which is what we want at this level. However, this means that the data will also be used twice if we combine these aggregates into the grand total, i.e., if we reuse these aggregates for further aggregation.

Irregular dimension hierarchies occur in many contexts, including organization hierarchies [ZS99], medical diagnosis hierarchies [Ser99], and concept hierarchies for web portals such as that of Yahoo! [Yah06].

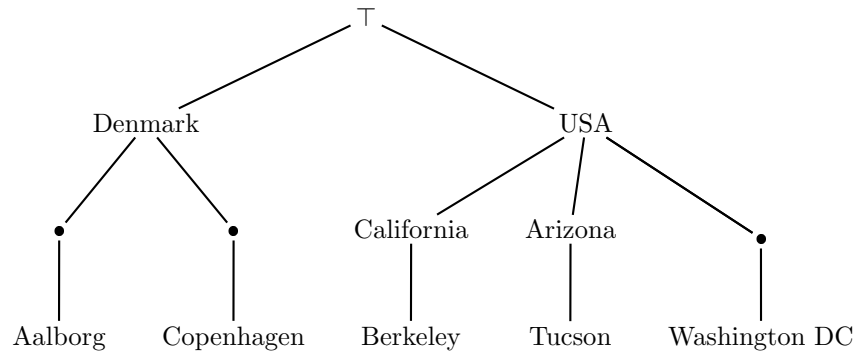


FIGURE 53.3: Irregular Location Dimension

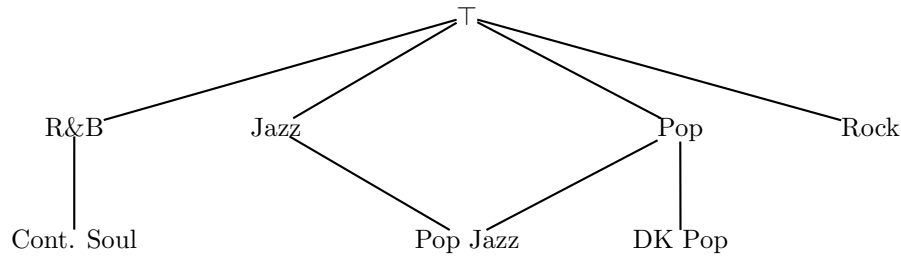


FIGURE 53.4: Irregular Music Dimension

A solution to the problems with irregular hierarchies is to *normalize* the hierarchies, a process that pads non-onto and non-covering hierarchies with “dummy” dimension values to make them onto and covering, and fuses sets of parents in order to remedy the problems with non-strict hierarchies. This transformation may be accomplished transparently to the user, rendering pre-aggregation applicable to also the more general types of dimension hierarchies discussed here [PJD99].

53.8 Survey of Multidimensional Data Models

We proceed to consider first present requirements to multidimensional data models. This is followed by a characterization of existing models with respect to these. The resulting survey is an condensed and updated version of an existing survey [PJD01]. The survey provide an overview of the levels of complexity in the data that the different models support. Another survey of multidimensional models can be found in [VS99].

53.8.1 Requirements for Data Analysis

We first describe requirements that a multidimensional data model should satisfy in order to fully support advanced applications.

1. *Explicit hierarchies in dimensions.* The hierarchies in the dimensions should be captured explicitly by the schema. This permits the user to drill-down and roll-up, as discussed in Section 53.5.1.

2. *Multiple hierarchies in each dimension.* A single dimension can have several paths for data aggregation. As an example, assume that we have a Time dimension that captures both the Calendar Year and the Fiscal Year. To model this, multiple hierarchies are needed.
3. *Support for aggregation semantics.* The data model should capture the aggregation semantics of the data and use this to provide a “safety net” that catches queries that might yield results that have no meaning to the user. Aspects of this include built-in support for avoiding double-counting of data and avoiding addition of non-additive data.
For example, when asking for the number of songs in different genres, Pop-Jazz songs will be counted as both Jazz songs and Pop songs. However, if the user then adds up the genre totals in an attempt to obtain the total number of songs, the system should prevent this, as Pop-Jazz songs will then be double counted. The user should also be able to specify which aggregations are considered meaningful for the different kinds of data available, and the model should provide a foundation for enforcing these specifications. As an illustration, it may not be meaningful to sum up inventory levels across time, while performing average calculations on them does make sense.
4. *Non-strict hierarchies.* As explained in Section 53.7 and illustrated in Figure 53.4, dimension hierarchies dimension may be non-strict, i.e., we can have many-to-many relationships between the different levels in a dimension. Because such dimensions make sense to the users, they should be supported by the data model.
5. *Non-onto hierarchies.* As also covered in the previous section, dimension hierarchies may be unbalanced, i.e., the path from the root to a leaf may have varying length for different leaves. This is illustrated in Figure 53.4.
6. *Non-covering hierarchies.* Another common feature of real-world hierarchies is that links between two nodes in the hierarchy “skip” one or more levels, as in Figure 53.3.
7. *Symmetric treatment of dimensions and measures.* The data model should allow measures to be treated as dimensions and vice versa. In our case, the Sales attribute would typically be treated as a measure, to allow for computations such as total sales, etc., but we should also be able to define an Age dimension which allows us to group sales into groups such as low, medium, and high.
8. *Many-to-many relationships between facts and dimensions.* The relationship between fact and dimension is not always the classical many-to-one one. In our case, some albums may be attached to several genres. For example, the album “Reise Reise” may be categorized as “Metal,” “Heavy Metal,” “Tanz-Metall,” “Rock,” “Hard Rock,” and “Industrial.”
9. *Handling change and time.* Although data change over time, it should be possible to perform meaningful analyses across times when data change. For example, genre hierarchies and the classification of songs into genres both change over time. It should be possible to easily combine data across changes. The problem of *slowly changing dimensions* [Kim96] discussed in Section 53.6 is part of this problem.
10. *Handling different levels of granularity.* Fact data might be registered at different granularities. For example, US sales might be reported per state, while Danish sales might be reported per city. It should still be possible to get correct analysis results when data is registered at different granularities.

11. *Handling imprecision.* Finally, it is very important to be able to capture directly the imprecision in the data and allow queries to take this into account. For example, the classification of songs into genres could have varying precision, as some songs are easy to classify, while others are not. It is important that this is captured and communicated to the users.

Many other requirements may be posed to multidimensional data models. We have chosen the eleven requirements above for several reasons. First, they are non-trivial and are not satisfied by all existing models. Second, they are “model” requirements that affect the core of a multidimensional data model. This contrasts less fundamental requirements that may be met by simply adding a new facility to the data model’s query language. Third, they derive from previous studies of the types of data and desired analyses that may be found in complex systems [PJ98, PJ99b].

53.8.2 Existing Multidimensional Models

We proceed to evaluate fifteen data models for data warehousing on the requirements just presented. We consider the models of Rafanelli & Shoshani [RS90], Agrawal et al. [AGS97], Gray et al. [GCB⁺97], Dyreson [Dyr96], Kimball [Kim96], Li & Wang [LW96], Gyssens & Lakshmanan [GL97], Cabbibo and Torlone [CT97], Datta & Thomas [DT97], Lehner [Leh98], Vassiliadis [Vas98], Jagadish et al. [JLS99], Mendelzon & Vaisman [MV00], Pedersen et al. [PJ99a, PJD01], and Microsoft’s Analysis Services data model [TSC99, SHW⁺06, Mic06a]. These models are representative of the current state of the art in both the research community and commercial systems. The models can be divided into *simple cube models*, *structured cube models*, *complex cube models*, and *statistical object models*.

The simple cube models [DT97, GCB⁺97, GL97, Kim96] treat data as n -dimensional cubes. Generally, the data is divided into *facts*, or *measures*, e.g., Sale, on which calculations should be performed, and *dimensions*, e.g., Music, which characterize the facts. Each dimension has a number of attributes, which can be used for selection and grouping. In our example, the Music dimension might have a Genre attribute and a Subgenre attribute that would be used to characterize the sales. The hierarchy between these attributes is not captured explicitly by the schema of the simple cubes, so these models do not “know” that sub-genres roll up to genres.

Kimball’s star schema model is the best example of the simple cube models. His model is based on plain SQL and does not embody multidimensional concepts per se. We include it here because it is the most widely used implementation model for multidimensional databases. Additionally, most relational OLAP tools assume a star schema structure of the database and cannot handle more complex designs. Thus, the evaluation of the star schema model is based on what can be achieved using a plain star schema design and the corresponding (simple) SQL queries, not on what can be done using full-fledged SQL.

The structured cube models [AGS97, CT97, Dyr96, JLS99, Leh98, LW96, MV00, TSC99, SHW⁺06, Mic06a, Vas98] capture the hierarchies in the dimensions explicitly, providing better guidance for the user navigating the cubes. This information may also be useful for query optimization [LR97]. The hierarchies are captured using either *grouping relations* [LW96], *dimension merging functions* [AGS97], measure graphs [Dyr96], roll-up functions [CT97, MV00], level lattices [Vas98], hierarchy schemas and instances [JLS99], or an explicit tree-structured hierarchy as part of the cube [Leh98, Mic06a, SHW⁺06, TSC99].

The complex cube models contain one member, namely the model by Pedersen et al. [PJ99a, PJD01]. This model was designed to support the eleven requirements presented above, and it contains constructs for handling these requirements.

The last group of models is the *statistical object models* [RS90]. For this group, a structured classification hierarchy is coupled with an explicit aggregation function on a single measure to produce a “pre-cooked” object that will answer a very specific set of queries. This approach is not as flexible as the others, but unlike most of these, it provides some protection, by using aggregation semantics, against getting query results that are incorrect or not meaningful to the user.

The results of evaluating the fifteen data models against the eleven requirements are shown in Table 53.9. If a model supports all aspects of a requirement, we say that the model provides *full* support, denoted by “√”. If a model supports some, but not all, aspects of a requirement, we say that it provides *partial* support, denoted by “p”. When it has not been possible to determine how support for a requirement should be accomplished in the model, we say that the model provides *no* support, denoted by “-”.

Model	Requirement										
	1	2	3	4	5	6	7	8	9	10	11
Rafanelli & Shoshani [RS90]	√	-	√	p	p	-	-	-	-	-	-
Agrawal et al. [AGS97]	p	√	-	p	-	-	√	-	-	-	-
Dyreson [Dyr96]	√	√	p	-	-	-	-	-	-	p	p
Gray et al. [GCB ⁺ 97]	-	√	p	-	-	-	√	-	-	-	-
Kimball [Kim96]	-	√	p	-	-	-	-	-	p	-	-
Li & Wang [LW96]	p	√	p	-	-	-	-	-	-	-	-
Gyssens & Lakshmanan [GL97]	-	√	p	-	-	-	√	-	-	-	-
Cabbibo & Torlone [CT97]	√	√	p	-	-	-	-	-	-	-	-
Datta & Thomas [DT97]	-	√	-	p	-	-	√	-	-	-	-
Lehner [Leh98]	√	-	√	-	-	-	-	-	-	-	-
Vassiliadis [Vas98]	√	√	√	-	-	-	-	-	-	-	-
Jagadish et al. [JLS99]	√	√	-	-	√	√	-	-	-	√	p
Mendelzon & Vaisman [MV00]	√	√	p	-	-	-	-	-	√	-	-
Pedersen et al. [PJ99a, PJD01]	√	√	√	√	√	√	√	√	√	√	√
MS Analysis Services [Mic06a, TSC99, SHW ⁺ 06]	√	√	p	-	√	√	-	-	p	-	-

TABLE 53.9 Evaluation of the Data Models

1. *Explicit hierarchies in dimensions:* The simple cube models [DT97, GCB⁺97, GL97, Kim96] do not capture the hierarchies in the dimensions explicitly. Some models provide partial support via a *grouping relation* [LW96] and a *dimension merging function* [AGS97], but do not capture the complete hierarchy together with the cube. This is done by the remaining models [CT97, Dyr96, JLS99, Leh98, MV00, TSC99, Mic06a, SHW⁺06, PJ99a, PJD01, RS90, Vas98], thus capturing the full cube navigation semantics in the schema.
2. *Multiple hierarchies in each dimension:* Some models [Leh98, RS90] require that the schema of dimension hierarchies is tree-structured. To support multiple hierarchies, a more general lattice structure is required. All the other models [AGS97, CT97, DT97, Dyr96, GCB⁺97, GL97, JLS99, Kim96, LW96, MV00, TSC99, Mic06a, SHW⁺06, PJ99a, PJD01, Vas98] allow multiple hierarchies.
3. *Support for aggregation semantics:* Most of the models [CT97, Dyr96, GCB⁺97, GL97, Kim96, LW96, MV00] support aggregation semantics partially, by implicitly requiring the dimension hierarchies to be *strict*, *onto*, and *covering*, i.e., the hierarchies should be balanced trees. This is one of the conditions of summarizability [LS97] and means that data will not be double-counted. Two of the models allow for non-strict hierarchies, while not addressing the issue of double-counting, thus providing no support [AGS97, DT97]. One model [JLS99] allows non-onto and non-covering hierarchies, but does not address the issue of

data not being counted, thus providing no support. One model [TSC99, Mic06a, SHW⁺06] allows non-onto and non-covering hierarchies and provides mechanisms to count data correctly in these situations, thus providing partial support. Two models [Leh98, RS90] place explicit conditions on both the hierarchies (strict, onto, and covering) and the aggregation functions used (only additive data may be added, etc.), thus providing full support for aggregation semantics. One model [Vas98] provides the support by always keeping a reference to the base data and computing from that when the aggregation semantics indicate the need to do so. Finally, one model [PJ99a, PJD01] allows non-strict, non-onto, and non-covering hierarchies, and also provides aggregation semantics.

4. *Non-strict hierarchies*: Most of the models [CT97, Dyr96, GCB⁺97, GL97, JLS99, Kim96, Leh98, LW96, MV00, Mic06a, TSC99, SHW⁺06, Vas98] implicitly or explicitly require that hierarchies be strict. Two models [AGS97, DT97] mention briefly that non-strict hierarchies are allowed, but do not explore the issues raised by allowing this, e.g., the possibility of double-counting and the use of pre-computed aggregates. One model [RS90] investigates the possible problems with allowing non-strict hierarchies and advises against using this feature. Finally, one model [PJ99a, PJD01] allows non-strict hierarchies, and prevents aggregation problems using aggregation semantics.
5. *Non-onto hierarchies*: One model [RS90] discusses the possibility of having non-onto hierarchies, but advises against using this feature. Three models [JLS99, Mic06a, SHW⁺06, TSC99, PJ99a, PJD01] allow non-onto hierarchies. All the other models do not allow non-onto hierarchies.
6. *Non-covering hierarchies*: Only three models [JLS99, Mic06a, SHW⁺06, TSC99, PJ99a, PJD01] allow hierarchies to be non-covering. All the other models disallow non-covering hierarchies.
7. *Symmetric treatment of dimensions and measures*: Most of the models [CT97, Dyr96, JLS99, Kim96, Leh98, LW96, MV00, Mic06a, TSC99, SHW⁺06, RS90, Vas98] distinguish sharply between measures and dimensions. An attribute designated as a measure cannot be used as a dimensional attribute and vice versa. This restricts the flexibility of the cube designs, e.g., if the Sales attribute of the example is a measure, it cannot be used to group albums into sales groups. The other models [AGS97, DT97, GCB⁺97, GL97, PJ99a, PJD01] do not impose this restriction. They either do not distinguish between measures and dimensions [GCB⁺97, GL97], allow for the conversion of measures to dimensions and vice versa [AGS97, DT97], or treat all data as dimensions on which aggregate computations can also be performed [PJ99a, PJD01].
8. *Many-to-many relationships between facts and dimensions*: Only one of the models [PJ99a, PJD01] allows many-to-many relationships between facts and their associated dimensions.
9. *Handling change and time*: Four models [Kim96, MV00, PJ99a, PJD01, Mic06a, SHW⁺06, TSC99] fully or partially support this issue, but only the models of Mendelzon & Vaisman [MV00] and Pedersen et al. [PJ99a, PJD01] fully support analyses across temporal changes in the dimensions. The other two models [Kim96, Mic06a, SHW⁺06, TSC99] support slowly changing dimensions. None of the other models support analysis across changes.
10. *Handling different levels of granularity*: Dyreson [Dyr96] specifies an *incomplete data cube* to be a union of *cubettes*. Each cubette may have a different data

granularity, thus providing some support for different levels of granularity. However, the granularity is fixed at the schema level, rather than at the data level, so the support is only partial. Two models [JLS99, PJ99a, PJD01] allow the granularity to vary at the data level. None of the other models handle different levels of granularity in the data.

11. *Handling imprecision*: For the reasons mentioned above, two models [Dyr96, JLS99] provide partial support for imprecision in the data, as varying granularities can provide a basis for handling imprecision. However, these models do not offer all the features necessary for handling the imprecision. One model [PJ99a, PJD01] provides full support for handling imprecision in the data. None of the other models provide explicit means for handling imprecise data.

To conclude, the models generally provide full or partial support for most of Requirements 1–3. Requirement 4 (non-strict hierarchies) is partially supported by three of the models and fully supported by one, while Requirement 5 (non-onto) is supported, partially or fully, by only four models. Requirement 6 is supported by three models, while Requirement 7 (symmetric treatment of dimensions and measures) is supported by five models. Requirement 9 (handling change and time) is fully supported by two models and partially supported by another two. Requirement 10 (handling different levels of granularity) is partially supported by one model and fully supported by two, while Requirement 11 (imprecision) is partially supported by two models and fully by one. Requirement 8 (many-to-many fact-dimension relationships) is only supported by one model.

53.9 Commercial OLAP Systems

As noted in Section 53.2.2, the development of multidimensional database systems began almost 40 years ago. However, it is only in the last 10 years that multidimensional technology has entered into the mainstream. Below, we introduce some of the most common commercial systems in the area. We focus on the offerings from the “big-3” software vendors (IBM, Microsoft, Oracle) as well as the systems having a market share of at least 5% in the 2004 OLAP market, as measured by the acknowledged, independent OLAP market research report “The OLAP Report” [Rep06].

Generally, commercial multidimensional systems fall in two groups: tools that are primarily multidimensional database servers and client-oriented analysis and reporting tools.

The multidimensional database server market is dominated by three products, two of which is owned by “big-3” vendors. The number one server is Microsoft’s Analysis Services [Mic06a, TSC99], which comes bundled with the MS SQL Server 2005 RDBMS. The number two server is Hyperion Essbase [Hyp06], long the market leader. A version of Essbase was for a number of years sold along with IBM DB2 as DB2 OLAP Services, but this re-selling agreement was ended in late 2005. The number three server is Oracle 10g OLAP R2 [Ora06], which is based on the Oracle RDBMS and the late Express MOLAP server.

All three servers offer flexible HOLAP technology, with flexible storage of data using ROLAP and/or MOLAP technology. All three servers generally provide good query and load performance, as well as advanced query functionality. After ending the agreement with Hyperion, IBM now only offers the ROLAP-based DB2 Data Warehouse Edition [IBM06], which is less feature-rich with respect to complex analytics.

The three largest vendors of client-oriented tools are Cognos [Cog06], Business Objects [BO06], and Microstrategy [Mic06b]. They all provide tools for “real” interactive OLAP analyses as well as for more traditional static reporting, including paper-based re-

ports. Another common functionality is support for building web portals that integrate analysis results with other data from the enterprise. They also include embedded server components that are primarily intended as back-ends for the client tools rather than as stand-alone OLAP servers.

The OLAP server vendors also have some client-side data analysis offerings. Microsoft's primary offerings are an Excel OLAP plug-in for OLAP analysis and Reporting Services for traditional reports. In Office 2007, these are supplemented with applications for performance management and scorecards. Also Excel 2007 features OLAP functionality.

Next, Hyperion (which has now been acquired Brio) has a large suite a client tools and a large market share in customized OLAP applications. Oracle offers both a range of general analysis tools and more specialized systems enabling easy analysis of data from the Oracle Applications ERP system.

Finally, SAP offers the similar SAP Netweaver Business Intelligence and SAP Business Information Warehouse (SAP BW) products [SAP06] that allow easy DW integration and subsequent analysis of data from the SAP ERP system, as well as from other source systems.

Open source software has achieved large market shares in several markets, but not in the OLAP market. A recent survey [TP05] reports that open souce OLAP products are generally not yet stable enough to be viable alternatives to commercial products. The most mature products are within ROLAP servers, where especially PostgreSQL [Pos06] (led by the Bizgres project [Biz06]) and MonetDB [Mon06] are prominent.

In summary, the most popular OLAP server and client tools generally offer sufficient functionality and performance to meet the needs of most customers. The choice of tools should thus be based on criteria such as existing system portfolio, the overall IT strategy, vendor support, and any specialized requirements, to name but a few criteria.

53.10 Summary

The continued advances in key information technology areas as well as the increasing electronic capture of business data have made possible the collection of very large volumes of business data. Advances in software technologies have contributed significantly to the provision of systems with response times that enable interactive analyses of such data.

At the core of these software technologies lies a type of data model that espouses a multi-dimensional view of data. While this type of model originally evolved from multidimensional matrix algebra, it has in recent years been influenced and enriched by the insights gained in the areas of semantic as well as scientific and statistical data models.

Multidimensional models view data as consisting of business facts, with associated measures, that are characterized by descriptive data values organized in multiple dimensions. Dimension values are organized in containment-type hierarchical structures that enable the computation of aggregate queries at different levels of granularity. This data organization lends itself towards graphical formulation of aggregate queries and graphical display of their results, and it is also conducive to the efficient evaluation of aggregate queries.

A wide variety of commercial systems exist that support multidimensional databases, both on the client and the server side. In recent years, multidimensional database functionality has become available as an integrated part of the leading RDBMS systems.

References

- [AGS97] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. In *ICDE*, pages 232–243, 1997.

- [AYJ00] S. Amer-Yahia and T. Johnson. Optimizing queries on compressed bitmaps. In *VLDB*, pages 329–338, 2000.
- [Biz06] Bizgres. Bizgres. <http://www.bizgres.org/>, Current as of November 29, 2006.
- [BO06] BO. Business objects corporation. <http://www.businessobjects.com/>, Current as of November 29, 2006.
- [BPT97] E. Baralis, S. Paraboschi, and E. Teniente. Materialized views selection in a multidimensional database. In *VLDB*, pages 156–165, 1997.
- [BR99] K. S. Beyer and R. Ramakrishnan. Bottom-up computation of sparse and iceberg cubes. In *SIGMOD*, pages 359–370, 1999.
- [CI98] C. Y. Chan and Y. E. Ioannidis. Bitmap index design and evaluation. In *SIGMOD*, pages 355–366, 1998.
- [Cod93] E. F. Codd. *Providing OLAP (on-line analytical processing) to user-analysts: An IT mandate*. E.F. Codd and Assoc, 1993.
- [Cog06] Cognos. Cognos corporation. <http://www.cognos.com/>, Current as of November 26, 2006.
- [CT97] L. Cabibbo and R. Torlone. Querying multidimensional databases. In *DBPL*, pages 319–335, 1997.
- [DPJ03] C. E. Dyreson, T. B. Pedersen, and C. S. Jensen. Incomplete information in multidimensional databases. In M. Rafanelli, editor, *Multidimensional databases: Problems and Solutions*. Idea Group Publishing, 2003.
- [DT97] A. Datta and H. Thomas. A conceptual model and algebra for on-line analytical processing in decision support databases. In *WOITS*, pages 91–100, 1997.
- [Dyr96] C. E. Dyreson. Information retrieval from an incomplete data cube. In *VLDB*, pages 532–543, 1996.
- [EM00] A. Eisenberg and J. Melton. Sql standardization: The next steps. *SIGMODR*, 29(1):63–67, 2000.
- [GCB⁺97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, M. Venkatrao D. Reichart, F. Pellow, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. *DMKD2*, 1(1):29–54, 1997.
- [GL97] M. Gyssens and L. V. S. Lakshmanan. A foundation for multi-dimensional databases. In *VLDB*, pages 106–115, 1997.
- [GMUW00] H. Garcia-Molina, J. Ullman, and J. Wido. *Database Systems: The Complete Book*. Prentice-Hall, 2000.
- [HGM05] Carlos A. Hurtado, Claudio Gutiérrez, and Alberto O. Mendelzon. Capturing summarizability with integrity constraints in olap. *TODS*, 30(3):854–886, 2005.
- [HRU96] V. Harinarayan, A. Rajaraman, and J. D. Ullman. Implementing data cubes efficiently. In *SIGMOD*, pages 205–216, 1996.
- [Hyp06] Hyperion. Hyperion corporation. <http://www.hyperion.com/>, Current as of November 26, 2006.
- [IBM06] IBM. Db2 data warehouse edition. <http://www.ibm.com/software/data/db2/dwe>, Current as of November 26, 2006.
- [(IS01] International Standards Organisation (ISO). *ISO/IEC 9075-1:1999/Amd 1:2001 : AMDT 1: On-Line Analytical Processing (SQL/OLAP) - Information technology - Database languages - SQL - Part 1: Framework (SQL/Framework)*. ISO, 2001.
- [JLS99] H. V. Jagadish, L. V. S. Lakshmanan, and D. Srivastava. What can hierarchies do for data warehouses? In *VLDB*, pages 503–541, 1999.
- [Kim96] R. Kimball. *The Data Warehouse Toolkit*. Wiley Computer Publishing, 1996.

- [KTS⁺02] N. Karayannidis, A. Tsois, T. K. Sellis, R. Pieringer, V. Markl, F. Ramsak, R. Fenk, K. Elhardt, and R. Bayer. Processing star queries on hierarchically-clustered fact tables. In *VLDB*, pages 730–741, 2002.
- [Leh98] W. Lehner. Modeling large scale olap scenarios. In *EDBT*, pages 153–167, 1998.
- [LR97] W. Lehner and T. Ruf. A redundancy-based optimization approach for aggregation in multidimensional scientific and statistical databases. In *DASFAA*, pages 253–262, 1997.
- [LS97] H. Lenz and A. Shoshani. Summarizability in olap and statistical data bases. In *SSDBM*, pages 39–48, 1997.
- [LW96] C. Li and X. S. Wang. A data model for supporting on-line analytical processing. In *CIKM*, pages 81–88, 1996.
- [Mic06a] Microsoft. Microsoft sql server: Analysis services. <http://www.microsoft.com/sql/technologies/analysis/default.msp>, Current as of November 26, 2006.
- [Mic06b] Microstrategy. Microstrategy. <http://www.microstrategy.com>, Current as of November 26, 2006.
- [Mon06] MonetDB. Monetdb. <http://monetdb.cwi.nl/>, Current as of November 29, 2006.
- [MV00] A. O. Mendelzon and A. A. Vaismann. Temporal queries in olap. In *VLDB*, pages 242–253, 2000.
- [Ora06] Oracle. Oracle database 10g olap option. http://www.oracle.com/technology/products/bi/olap/ds_bi_olap_10ir1_0104.pdf, Current as of November 26, 2006.
- [PJ98] T. B. Pedersen and C. S. Jensen. Clinical data warehousing—a survey. In *MEDICON*, page 20.3, 1998.
- [PJ99a] T. B. Pedersen and C. S. Jensen. Multidimensional data modeling for complex data. In *ICDE*, pages 336–345, 1999.
- [PJ99b] T. B. Pedersen and C. S. Jensen. Research issues in clinical data warehousing. In *SSDBM*, pages 43–52, 1999.
- [PJ01] T. B. Pedersen and C. S. Jensen. Multidimensional database technology. *COMP*, 34(12):40–46, 2001.
- [PJD99] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. Extending practical pre-aggregation in on-line analytical processing. In *VLDB*, pages 663–674, 1999.
- [PJD01] T. B. Pedersen, C. S. Jensen, and C. E. Dyreson. A foundation for capturing and querying complex multidimensional data. *INFSYS*, 26(5):383–423, 2001.
- [Pos06] PostgreSQL. Postgresql. <http://www.postgresql.org/>, Current as of November 29, 2006.
- [Rep06] The OLAP Report. 2004 market shares. <http://www.olapreport.com/Market.htm>, Current as of November 28, 2006.
- [Rep07] The OLAP Report. What is olap? <http://olapreport.com/fasmi.htm>, Current as of February 7, 2007.
- [RS90] M. Rafanelli and A. Shoshani. Storm: A statistical object representation model. In *SSDBM*, pages 14–29, 1990.
- [SAP06] SAP. Sap netweaver business intelligence. <http://www.sap.com/platform/netweaver/components/bi/index.epx>, Current as of November 26, 2006.
- [SDN98] A. Shukla, P. Deshpande, and J. F. Naughton. Materialized view selection for multidimensional datasets. In *VLDB*, pages 488–499, 1998.

- [Ser99] National Health Service. *Read Codes version 3*. NHS, 1999.
- [SHW⁺06] G. Spofford, S. Harinath, C. Webb, D. H. Huang, and F. Civardi. *MDX-Solutions: With Microsoft SQL Server Analysis Services 2005 and Hyperion Essbase*. Wiley, 2006.
- [Tho97] E. Thomsen. *OLAP Solutions: Building Multidimensional Information Systems*. Wiley, 1997.
- [TP05] C. Thomsen and T. B. Pedersen. A survey of open source tools for business intelligence. In *DAWAK*, pages 74–84, 2005.
- [TSC99] E. Thomsen, G. Spofford, and D. Chase. *Microsoft OLAP Solutions*. Wiley, 1999.
- [Vas98] P. Vassiliadis. Modeling multidimensional databases, cubes, and cube operations. In *SSDBM*, pages 53–62, 1998.
- [VS99] P. Vassiliadis and T. K. Sellis. A survey of logical models for olap databases. *SIGMODR*, 28(4):64–69, 1999.
- [Wei02] A. Weininger. Efficient execution of joins in a star schema. In *SIGMOD*, pages 542–545, 2002.
- [Win98] R. Winter. Databases: Back in the olap game. *Intelligent Enterprise Magazine*, 1(4):60–64, 1998.
- [WOS04] K. Wu, E. J. Otoo, and A. Shoshani. On the performance of bitmap indices for high cardinality attributes. In *VLDB*, pages 24–35, 2004.
- [Yah06] Yahoo! Yahoo! <http://www.yahoo.com>, Current as of November 26, 2006.
- [ZS99] T. Zurek and M. Sinnwell. Data warehousing has more colours than just black and white. In *VLDB*, pages 726–729, 1999.