Diplomarbeit

A set-based M/OLAP kernel with constraint-like queries

ausgeführt am

Institut für Softwaretechnik
der Technischen Universität Wien

unter Anleitung von

Prof. Dr.techn. A Min Tjoa
und DI Andreas Rauber

durch

Philipp Rudolf Tomsich
Wimmergasse 1/26
A–1050 Wien

September 13, 1999

_____

Ich habe zur Kenntnis genommen, daß ich zur Drucklegung meiner Arbeit unter der Bezeichnung **DIPLOMARBEIT** nur mit Bewilligung der Prüfungskommission berechtigt bin.

Ich erkläre weiters an Eides statt, daß ich meine Diplomarbeit nach den anerkannten Grundsätzen für wissenschaftliche Abhandlungen selbstständig ausgeführt habe und alle verwendeten Hilfsmittel, insbesondere die zugrunde gelegte Literatur genannt habe.

September 13, 1999 _____

Contents

List of Figures

List of Tables

1. Introduction

"We are learning to generate data more rapidly,
than we can move it."
Communications of the ACM, Nov. 1998

1.1. A Paradigm Shift in Database Systems

When the relational database model was introduced by Codd in 1970 [16], it addressed the major shortcomings of early database systems. The advantage of separating the data repository from the actual data processing application was quickly recognized; so was the benefit of a consistent data model and a standardized interface. The relational model provided an abstract and well-understood framework for a wide variety of commercial and scientific applications. Above all, it promised equal access. In the relational model it does not matter which questions are asked first. This was a major improvement over the flat file databases and hierarchical databases which forced the analyst to ask business questions in a pre-set order. Relational databases finally offered the ability to unlock a wealth of information hidden before—at least conceptually.

During the early 1980s, data bases underwent a rapid evolution, when businesses began to capture business data such as orders, invoices and other business transactions. Early relational database systems handled about one transaction per

second. Apparently, no large organization could survive such low transaction rates. This situation fueled the work effort dedicated to the improvement of these systems. Judging from their wide-spread use and the increasing inclusion of database systems in business critical applications, those work efforts succeeded. For example, Kimball [30] cites the *SABRE* system, the reservation system for American Airlines, which routinely handles workloads of 4000 transactions per second and experienced peak workloads in excess of 13000 transactions per second. A recent comparison published by Sequent [47] tested four systems capable of handling more than 30000 transactions per second. The fastest systems was found to execute almost 49000 transactions per second. The most recent benchmark available from the *Transaction Processing Performance Council's* web-site indicates that half a year later, the best-performing system scored more than 115000 transactions per second.

In parallel with the query and transactional performance of database systems, the amount of data processed by businesses and governments has increased steadily during the last decade from the megabyte range to the multi-gigabyte ranges common today. One example of a large databases is given by Watson [50]: Wal-Mart, the U.S. retailer, collects sales data at its 2800 stores into a single database, which amounts to 24 terabytes of raw data. In this context, Codd [17] remark that "business enterprises prosper or fail according to the sophistication and speed of their information systems, and their ability to analyze and synthesize information using those systems." With the amount of data processed, the number of individuals within organizations who need to access the data and perform analysis on it is growing. These people usually fill roles, that are not primarily concerned with data processing, but rather with the monitoring and planning of business processes. In contrast, the development of database technology aimed at the increase of data throughput, creating the powerful On-Line Transactional Processing (OLTP) systems available. As Kimball puts it poignantly, the database world had become so fixated on getting data into the databases, that we forgot to think about how to

get it out.

Almost thirty years after their introduction, relational database systems are used for a wide variety of applications requiring the storage, updating and retrieval of data. The application domains range from the tracking of business processes and electronic commerce to decision support systems. A variety of products are available, each with a special focus and non-standard extensions. However, all of them share a common interface through the *structured query language (SQL)*. Modern systems support concurrent transactions, authentification and multiple users.

Despite of all the progress made, it is surprising that the promises of simple user interfaces and equal access to all data never came true. On the one hand, queries on certain attributes offer a performance advantage and query optimization has become an art of its own. Worse, the available front-end applications are designed to support a limited number of *reasonable* queries. These front-end products have become the main limiting factor to supporting flexible user-views of the data. Notably absent are features to freely and dynamically aggregate, consolidate, summarize and view data. As a consequence, explorative data analysis and generating reports remains unnecessarily complicated.

The analysis and control of business processes is performed by specialists in their respective business areas who are not trained database experts. Their analytical process often involves data exploration to discover what questions should be asked, which requires interactive response times in face of unpredictable query patterns. In contrast to OLTP which requires instantaneously accurate data, data analysis works on data accurate as of a given point in time. The availability of historic data is deemed more important than the availability of the latest data. Data is viewed at various levels of details, passed trough complex analytical functions, summed up, averaged and presented to the user–analyst. Speculative *what-if* scenarios are evaluated. All this needs to take place within an interactive environment. Codd coined the term *On-line Analytical Processing* for this type of decision

support applications.

The process data stored in databases is only rarely flat, but usually multidimensional. Suppose a database that stores the sales of a grocery chain: Different *products* belonging to *product categories* are sold in different *stores* in different *regions* and a store's performance if measured over *time*. Apparently, three separate dimensions of the data emerge: the *regional* dimension, the *product* dimension and a *time* dimension (compare Figure 1.1). It is possible to imagine the resulting data base as a cube where each point resulting from the intersection of a particular product, store and time contains a measurement (i.e., the sales number). Such multidimensional databases are therefore referred to as *data cubes*. This is a far closer approximation to the way people see business enterprises than the purely relational model. In addition, information on the hierarchical grouping of the raw facts within the data cube can be added in the form of meta data. This additional data, which is not available in an explicit form in relational databases, expresses additional domain knowledge. It can be used to allow the easy navigation of data and also affects query optimization (i.e., queries usually exhibit a temporal locality in respect to the areas of the data cube referenced).

Data consolidation is the process of aggregating regions of this cube into essential knowledge. This process of generating summary information follows so-called *consolidation paths* which usually reflect the actual, natural structure of the data. For example, the aggregation of all of a year's months into the corresponding year is a typical aggregation path for a time dimension. The highest level of abstraction in any data aggregation path is referred to as dimension [17] of the data. However, this definition is somewhat misleading, since it defines the dimension through the only dimension-value of the coarsest granularity in a data aggregation path, which contains all dimension-values the entire dimension. Any given data dimension represents a specific perspective of the data. The simultaneous analysis of multiple data dimensions forms the base of multi-dimensional data analysis. Once data is consolidated according to one or multiple consolidation paths, it is possible to move

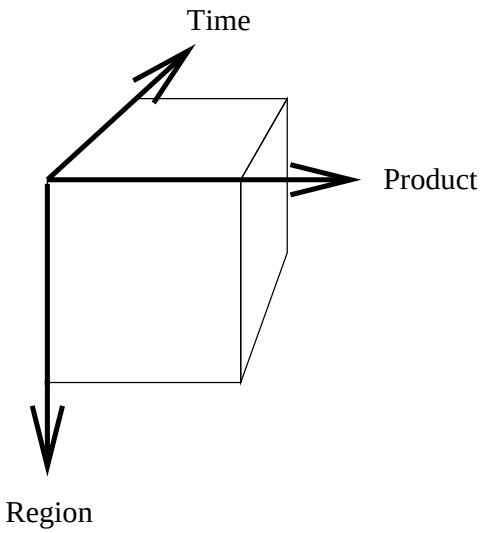

Figure 1.1.: A 3-dimensional data cube

from a less detailed to a more detailed view and vice versa. On-line Analytical Processing systems operate within this multi-dimensional data model to facilitate the data analysis and decision support process.

The data On-line Analytical Processing systems operate on will only rarely come from a single source. Instead multiple heterogenous data sources will provide the necessary raw data to conduct meaningful analysis. This is a consequence of the fact, that data is frequently collected in multiple locations and processed locally. Often data analysis will require the combination of one organization's data with another's. In such situations, different data models must be consolidated, redundancies eliminated, inconsistencies detected and resolved. The result of this data merging and cleaning is a homogeneous data repository with high data quality. Usually, these data bases are referred to as *data warehouses*. They provide a unified and consistent view at data originating from relational data base systems, flat files or other legacy systems. Combined with the multi-dimensional data model, data warehouses offer the possibility to pre-calculate views according to the con-

solidation paths present. This is one of the reasons why data warehouses tend to be orders of magnitudes larger compared to the associated operational databases. According to Chaudhuri and Dayal [15], data warehouses are targeted at and almost exclusively used for decision support applications.

We may conclude that database systems underwent a continuous evolution and multiple paradigm shifts over the course of the last decades. The first ad-hoc implementations solving specific problems, which later became generalized to general purpose applications, were replaced by relational database systems which provided a common abstract data model. During the 1980s, these relational database systems became the backbone of transactional systems recording business transactions almost in real-time. The addition of transactions and concurrent, multi-user access created On-Line Transactional databases. Only recently the latest paradigm shift occurred with the re-evaluation of database technology as a business tool and the re-focusing on data analysis, report generation and decision-support. The On-line Analytical Processing paradigm, which provides flexible multi-dimensional data analysis attempts to empower the user–analyst to be more productive. After all, a large repository of data is worthless without the tools necessary to derive knowledge from it.

1.2. On-Line Analytical Processing

It is impossible to provide a single, all-encompassing definition of On-line Analytical Processing (OLAP), because it inseparably depends on data warehousing: data warehouses provide the consistent data sources to operate on. Today, data warehouses are set up almost exclusively to support On-line Analytical Processing. Even for data mining on data warehouses, On-line Analytical Processing systems nowadays provide a flexible and intelligent data base access layer.

Essentially, On-line Analytical Processing provides interactive report generation and data analysis on data stored in a data warehouse. It evolved, when the

analytical data models and algorithms used in decision support systems evolved into a new category of database application and the first OLAP systems emerged. The initial characterization of these On-line Analytical Processing systems appeared in a white paper published by Codd *et. al.* [17]. This landmark paper provided the following twelve rules outlining the fundamental requirements for On-line Analytical Processing systems:

1. **Multi-dimensional Conceptional View.**

   As the user–analysts view of the organization is multi-dimensional in nature, the conceptual view during analysis should be multi-dimensional as well. This guarantees easy and intuitive data manipulation. *Slice and dice* (ie., selecting a cross-section of a data-cube and selecting a sub-cube, respectively) are simple operations within a multi-dimensional model, but require significantly more effort with other approaches. Furthermore, multiple hierarchies should be supported for each hierachy, as different user groups may have different understandings of what constitutes a natural grouping of the data (e.g., calendar years vs. fiscal years).

2. **Transparency.**

   Whether On-line Analytical Processing functionality is a built-in feature of the user's customary front-end product or an additional service, should be transparent to the user. If OLAP is provided in the context of a client–server application, this fact should be transparent as well. It should remain transparent where the data input into the system comes from, whether from a homogeneous or heterogenous database environment. Codd also states the additional goal that On-line Analytical Processing should be provided in "the context of a true open systems architecture, allowing the analytical tool to be embedded anywhere."

   Transparency is paramount to preserving and improving the user's produc-

tivity. One of the implementation goals is to assure that no additional complexity is introduced into the analytical process.

3. **Accessibility.**

   The analyst must be able to perform data analysis based on a common multidimensional abstraction, independent of the actual data sources and their internal organization. The On-line Analytical Processing tool needs to map a logical schema of the data, which is provided as meta data, to possibly heterogenous physical data stores, access this data, perform any necessary conversions and present the user with a single, coherent and consistent view. Only the data required to perform the indicated analysis should be accessed to eliminate unnecessary memory traffic.

4. **Consistent Reporting Performance.**

   The subjective query performance perceived by the user may not degrade significantly when the number of dimensions or the database size increases. Codd considers this as "critical to maintaining the ease-of-use and lock of complexity required in bringing OLAP to the end-user."

   This implies the (at least in my opinion) most important promise of On-line Analytical Processing: scalable systems which can provide interactive response time for data analysis tasks on very large data bases. However, this scalability promise has not been redeemed so far: Current systems depend mostly on the pre-materialization of the data cube to improve response times (requiring only a single database access) and slow down considerably when ad-hoc queries/aggregates/calculations are to be computed. The computation of ad-hoc aggregates requires the retrieval of a possibly large number of facts and the application of the consolidation function to them, requiring a time linear to the database size in the worst case.

5. **Client–Server Architecture.**

As most data requiring analysis is stored on dedicated servers, On-line Analytical Processing tools need to operate in a client-server environment. In effect, the architecture of the analysis tool will follow the database systems architecture: the server component of the analysis tool will interact with the various database servers (becoming a client in turn), while the user interacts with it through client applications. The structure of such an architecture is shown in Figure 1.2; it is particularly noteworthy, how different query tools—including decision support systems, report generators and statistical analysis packages—will access a common data warehouse through the OLAP services.

6. **Generic Dimensionality.**

   All dimensions are symmetric and equivalent regarding their operational capabilities. The basic data structures used, aggregation functions, query evaluation algorithms and reporting formats may not be biased toward any dimension.

7. **Dynamic Sparse Matrix Handling.**

   Sparseness is one of the most prominent features of multi-dimensional databases. It can be measured as the ratio between used cells and possible cells. Only the ability of On-line Analytical Processing systems to adapt to the various distributions of the source data and deal with the resulting sparseness makes compact storage and fast operation attainable. As the distribution is unpredictable for any given data model, it is necessary to dynamically adapt the physical storage format to environment. Any system will always be caught in a trade-off between compact storage in spite of sparse data and fast access. The physical access storage available include

   - (re-)calculation of aggregates from source data,
   - B-trees and derivates,
   - high-dimensional trees and index structures,

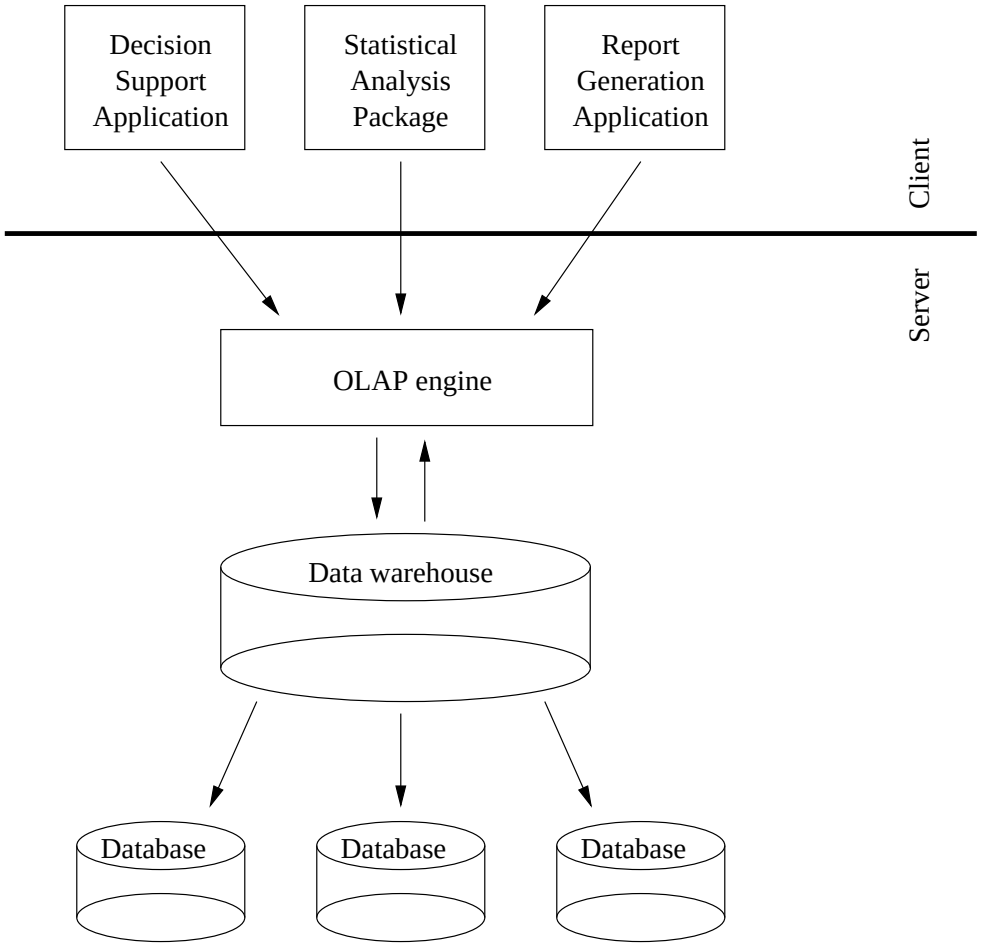

Figure 1.2.: On-line Analytical Processing as a client-server application

- arrays and grid files,
- hash tables and
- any combination of the above.

8. **Multi–User Support.**

   Multiple analysts within an organization may require access to the same data at the same time. For this reason, On-line Analytical Processing tools are required to provide concurrent access to the data and security features. If the data base can be updated by the analysts, integrity and transaction management has to be available as well.

9. **Unrestricted Cross-Dimensional Operations.**

   The various aggregation levels contain the relations from the source database in an implicit form. Accordingly, a multitude of consolidation paths can be derived from them. However, other calculations will be additionally necessary. These have to be formulated in an appropriate language. Such a language has to support calculations across any number of dimensions and granularities. This is necessary to model such business information as "*The overhead equals the percentage of total sales represented by the sales of each individual local office multiplied by the total corporate overhead.*"

10. **Intuitive Data Manipulation.**

    The user–analysts view of the system should reflect all the information necessary to effect the next analytical action, whether it is a drill-down (i.e., choosing a more detailed view), a roll-up (i.e., choosing a less detailed view) or a pivoting (i.e., exchanging the axis) operation. The current view needs to be displayed in a way natural to the application domain.

    Sauter [45] discusses the importance of choosing the appropriate presentation of data to help increase a decision-maker's intuition about trends in the

data. Regarding On-line Analytical Processing applications, the most important conclusion is the fact that data warehouses normally provide too much information to use satisfactorily, because users become lost in the possibilities. Only the reduction (i.e., consolidation) of data simplifies it in such a way, that a meaningful analysis can be conducted.

11. **Flexible Reporting.**

    The analysis of data can be simplified, if the presentation of the data can be arranged according to logical groupings occurring naturally within the data. A query tool has to be capable of rearranging data to capture this information, as well as to give feedback on the consolidation paths and aggregation levels of the displayed data.

    Although this is one of the original twelve rules, it becomes more and more irrelevant to the design and implementation of the On-line Analytical Processing system. In modern tools a clear separation of the query evaluator and the user-interface is attempted, splitting the functionality between the server and the client (which may or may not be considered part of the On-line Analytical Processing system).

12. **Unlimited Dimensions and Aggregation Levels.**

    Research indicating an empirical limit of nineteen distinct dimensions is cited by Codd [17]. However this number seems far too low. In many applications an even higher number of dimensions will be necessary, particularly if enumerated types are modeled using dimensions (e.g., a customer database which uses the customer's gender as a dimension).

    Furthermore, each of these dimensions has to support an unlimited number of aggregation levels and consolidation paths.

However, this first definition of On-line Analytical Processing as a separate category of database product has not remained undisputed. As Codd himself remarks,

all the functions in question except the data retrieval, could be expected from any reasonable spreadsheet program. Yet, spreadsheet vendors provide no or very limited support for OLAP so far. Many argue, that the requirements given are too fuzzy to differentiate On-line Analytical Processing from other techniques used in data mining and data analysis. As a consequence, On-line Analytical Processing is frequently mistaken for a type of time-series analysis or a statistical analysis package. A reason for criticism of Codd's paper is the fact that it contains a very favorable evaluation of *a specific commercial product*, which unsurprisingly satisfies all the rules laid forth. This casts a serious doubt on the objectivity of the entire report. Codd's clairvoyant treatment of the integration of transactional database systems with decision support proved that decision support systems would become a key element in the database solutions market.

Codd voices a concern for empowering the end-user within an organization with powerful tools to view, manipulate and animate data, but recognizes the necessity to remain compatible with legacy systems and lagacy databases. For this reason, he mandates a synergistic implementation consisting of separate, end-user tools that are outside and complementary to relational database products. A multi-level architecture is suggested, which includes the following [17]:

- access to the data in the DBMS or access method files;
- definitions of the data and its dimensions;
- tools to view, manipulate and animate the data models;
- integration with the end-users' customary interface.

In effect, this leads to an integrated business information system made up of three distinct layers: data warehousing, OLAP and decision support.

A similar, yet fundamentally different, definition is presented by by Chaudhuri and Dayal [14, 15]. Although this definition shared Codd's view that a multidimensional data model is a prerequisite to On-line Analytical Processing, it differs in the

fact that it does not require certain features, but rather a concept: Any application offering multidimensional analysis, an interactive user-interface and a set of OLAP operations (e.g., *roll-up*, *drill-down* and *pivot*) fulfills this definition. In contrast to the earlier model for On-line Analytical Processing, it does separate the responsibilities of the data warehouse and the decision support tool rather clearly—data storage, index selection and cleaning are the responsibility of the data warehouse, while the querying, administration and presentation of query results is part of the On-line Analytical Processing system. The process of pre-aggregating data is situated somewhere between. The data warehouse executes the calculations and stores both operands and the results, while the OLAP system maintains the meta data from which the appropriate consolidation paths and consolidation functions are determined.

Chaudhuri's definition proves to be far more pragmatic than Codd's twelve rules. However, this is not surprising, as it was written well after the first generation of OLAP tools was released. From the experiences gained with those early systems, a number of advances in data warehouse design, modeling and implementation resulted. As On-line Analytical Processing is inseparably connected to data warehousing, a short characterization of multi-dimensional data warehouses helps to understand its capabilities and limitations. Queries in decision support applications routinely access a large number of database entries and evaluate analytical functions against the resulting data sets. This is a harsh contrast to the transactions found in operational databases, where each transaction updates exactly one record. Data warehouse design attempts to negotiate this and other problems faced when analyzing data.

A *data warehouse* is a homogeneous, consistent data collection. Gardner [22] admits that there exists no single, accepted and all-encompassing definition of a data warehouse within the information technology industry, but asserts that "Data warehousing is a process, not a product, for assembling and managing data from various sources for the purpose of gaining a single, detailed view of part or

all of a business." In data warehousing, multidimensional data models are used to provide better support for decision support applications. The data is derived from multiple sources, merged, transformed and cleansed. Additional meta data is added to describe the structure and the internal relationships of the facts. The data represented is the data stored in the operational databases used for transaction processing with the addition of additional knowledge regarding its structure and semantics. However, the data warehouse is not current with the last committed transaction, but it is accurate as of the last time the operational database was fed into the data warehouse. The resulting delay between a change to the data base and the propagation of this information into the data warehouse is the *update window*. The data in plain data warehouses is consistent, but data is not aggregated into summary information.

Data marts are a special type of data warehouses. In a large organization it is often beneficial to the query performance to replicate frequently accessed parts of the database closer to the analyst. A data mart contains a replica of a subset to a data warehouse. Such a data mart is used in hierarchical organizations, where most analysts will access the data pertinent to their department and line of work far more frequently than other data. In such environments, data marts are an effective way to reduce the load on an organization's data warehouse. Possible configurations of data warehouses and data marts for a business enterprise are shown in Figure 1.3.

Kimball [30] blurs the difference between a data warehouse and an On-line Analytical Processing system in his introduction to the subject. The OLAP-typical, multi-dimensional operations, such as the *drill-down, roll-up* and *pivot* are presented as SQL-queries to a relational data warehouse. Following Codd's rules, this constitutes an On-line Analytical Processing system. However, Kimball precomputes a complete data cube including all aggregations, which certainly is beyond the scope of data warehouses alone: The metadata describing the consolidation paths—a fundamental feature of OLAP—is necessary for this. In an earlier publication by the same author [31], the concept of denormalized databases for

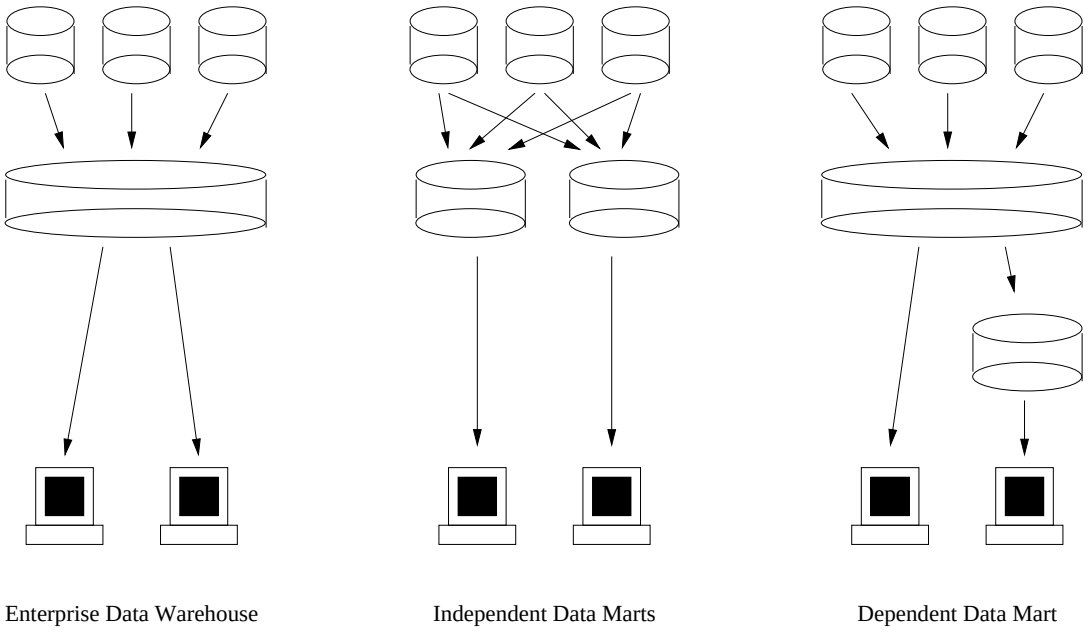

Figure 1.3.: Examples for various configurations of data warehouses.

decision support applications is emphasized. It is argued, that only the redundant storage of data in a normalized OLTP database and a denormalized data warehouse for decision support can provide the response times requirements of interactive decision support. This article details the benefits of dimensional modeling using the star schema (i.e., a fully denormalized data model; see Section 1.4.2 for a description) and cites one case, where the redesign of a database from 50 tables down to 5 tables, without a loss of expressiveness, resulted in twenty-fold increase in query throughput.

In contrast to the usage-oriented definitions given above, Agrawal [2] defines a data warehouse in a more abstract way: here it is a logically multi-dimensional data repository, which integrates data from different sources into one homogeneous database. The physical storage format used is left unspecified provided that a *data cube* can be represented. In this research report which focuses on the foundations of multidimensional databases, aggregated information is identified as an On-line Analytical Processing-specific extension tot he data warehousing concept. Chapter 2 contains a discussion of the modeling approach presented.

In summary, On-line Analytical Processing characterizes the requirements for summarizing, consolidating, viewing, applying formulae to and synthesizing data according to multiple dimensions and consolidation paths. It is an enabling technology which provides consistent multidimensional view on data, allows user–analysts to navigate their databases on different granularities, supports advanced data mining and tries to achieve interactive response times. However, the extent to which interactive responses can be achieved on *ad-hoc* queries, i.e., queries that require the calculation of aggregated values after the initial retrieval from the data warehouse, is largely dependent on optimization techniques such as caching. The prime difference between a simple data warehouse and an On-line Analytical Processing system is the ability of the latter to aggregate and pre-aggregate data.

1.3. On-Line Analytical Processing for Data Analysis

On-line Analytical Processing offers a number of benefits to user–analysts. Among the most prominent one is the ability to operate on very large data sets and still provide interactive responses for most operations. The use of this technology is often argued on the base of flexibility of analysis achievable from cross-dimensional computations. Nonetheless, one important argument for the use of On-line Analytical Processing for data analysis does not result from the scalability, interactivity or intuitive data analysis. Instead, it stems from an importat aspect of multidimensional databases: a simple model of operation. To every OLAP operation, a data cube is the input—for every OLAP operation an output data cube is prepared.

The navigation of the data cube during analysis derives solely from five operators:

- **Drill-Down.**

  To view data at a finer granularity level, a *drill-down* operation is executed. It "zooms in" on the data.

- **Roll-Up.**

  The *roll-up* is the inverse of the drill-down. It moves from a fine granularity level to a coarser one and "zooms out" from the data.

- **Dice.**

  The *dice* operation selects a subset of the available locations along each axis and retrieves the corresponding "subcube".

- **Slice.**

  To reduce the dimensionality of the cube, a *slice* operation is executed. It cuts through the cube at given point along a given axis. It is a special case of the more general *dice* operation.

- **Pivot.**

  The actual presentation of the data retrieved is usually reduced to dimensionality suitable for the available viewing device. This is done using slice operations. The *pivot* operation changes the selection of dimensions retained. If enough information is available in the client program, this operation does not need to reevaluate the entire query. Instead, the necessary reduction in dimensionality can be performed in the client.

1.4. Categories of On-Line Analytical Processing Systems

On-line Analytical Processing systems can be categorized according to their approach used to the physical storage of the fact base or the frequency and algorithm used for the synchronization between the operational database and the data warehouse. Another feature examined when classifying those OLAP servers, which are based on relational database, is the type of database schema used. This section gives a brief overview of these frequently used criteria and their significance in real-world applications.

1.4.1. Physical fact storage

Two dominant approaches for the implementation of OLAP systems exist: *relational OLAP (ROLAP)* and *multidimensional OLAP (MOLAP)*. Both are named according to the way their facts are stored physically. The relational approach uses a value-based storage, while the multidimensional organization provides positional storage and retrieval. In other words, an ROLAP system runs on top of a relational database and generates SQL queries and a MOLAP system uses specialised storage structures for multidimensional data which support faster access when whole subcubes are read.

Relational On-line Analytical Processing stores its data in a separate relational database. Data access and data summarization are expressed in queries to the underlying database. These queries are generated by a query tool and executed by the database. The results are returned to the analysis tool for display. The navigation and presentation tool communications with the relational data repository and queries using a relational query language. For example, the OLAP engine could communicate with the relational database management system (RDBMS) through a standardized SQL interface.

Multidimensional On-line Analytical Processing uses multidimensional data structures to store tuples of data according to their position within the multidimensional data-space. The data structures used to achieve this range include grid-files [21], B*-trees [33], R*-trees [9], X-trees [10], HB-trees [37], UB-trees [5], GiST [27], arrays and set-based structures (e.g., a treaps based set-implementation [11, 46]). The choice of data structure is highly data dependent, as their performance and storage overhead is a variable of the database size, sparseness and data distribution. For example, UB-trees offer superior performance to multiple B*-trees, if a large number of dimensions exists and range-queries are frequently evaluated, which restrict the results to a small subset of the database. Grid-files offer superior performance for densely populated data-cubes, while wasting enormous amounts of space for very sparse structures.

Zhao *et al.* [52] contrasts the performance of ROLAP and MOLAP implementations. It introduces an algorithm for the computation of the data-cube in compressed sparse arrays, which appears to outperforms all value-based based approaches. It is suggested, that it may be faster to convert a table into a position based representation, cube the data and convert it back, than computing the data-cube from a value-based fact table directly. This implies, that a relational storage concept should never be chosen for performance reasons.

Sometimes a third type of On-line Analytical Processing system pops up in the literature: hybrid On-line Analytical Processing (HOLAP). However, this term

simply denotes a hybrid between a relational and multidimensional OLAP system. Any system that can not be classified as unambiguously belonging to one of the above categories, satisfies this definition.

1.4.2. Stars and snowflakes

Most operational databases, which act as sources to the data warehouse, are relational. This fact and the inability of the first databases which were based on the positional storage model to scale to large problems, created an environment, where ROLAP became dominant. This leads to question of how to model the data stored within the data warehouse. This data may be denormalized to allow faster access, which invalidates the conventional wisdom of how to design a database schema. Two distinct types of database schemata evolved which are used in data warehouses for On-line Analytical Processing: the *star schema* and the *snowflake schema*. Both schemata center around a central table, the so-called *fact-table*. This table contains all facts contained within the data warehouse. However, that's where the similarities end.

The star schema

The *star schema* is by far the most widely used and recommended design schema for databases used in decision support. In this schema, the database is denormalized as far as practical, so achieve a simple query syntax and fast response times. A vast, central table, which is surrounded by and linked to a few surrounding tables allows access to the database (see Figure 1.4). A number of smaller tables describe the dimensions of the data and are referred to as *dimension tables*. For example, in a simple business application, one might find a "product", "market" and "time" table. The native keys of entries in these dimension tables, are foreign keys to an entry in the fact table. Besides this multi-part key, the actual facts (i.e., measures) are also stored in these fact table entries as non-key fields. These measures are

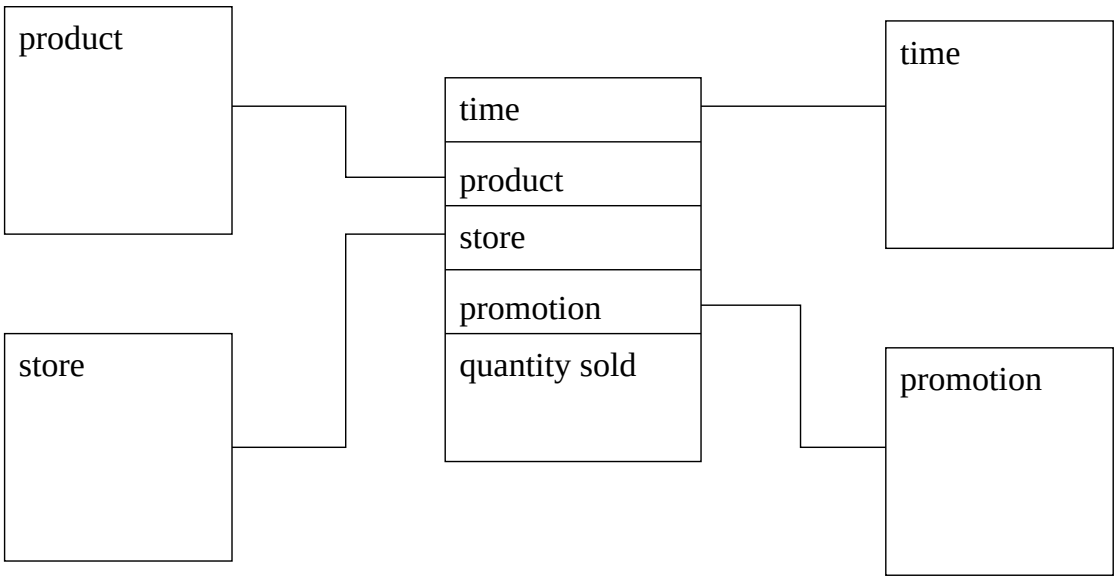

Figure 1.4.: Star schema

physically stored within the fact table. The dimension table are fully denormalized in this model to support fast browsing and optimize query performance.

Queries on the database are formulated using the data stored in the dimension tables. These provide the constraints to select subsets of the data. Using these constraints, it is easy to retrieve all facts satisfying these constraints from the fact table. The retrieval from the fact table can be performed without any join-operations, as all measures are physically available within this table, if a star schema is used. The dimension tables are many orders of magnitude smaller than the fact table, enabling graphical front-end applications to provide the user with interactive browsing of the dimensions to assemble queries.

The star schema denormalizes the database to a point, where only a fact table containing measures and references to dimension-values remains. The tables containing the dimension-values are completely denormalized, which increases the redundancy in the database and complicates the updating of data. While the re-

dundant storage of information would make the consistent updating of information impossible, this does not matter for OLAP which only performs data analysis and visualization and may in turn operate on a read-only data warehouse. Redundancy increases the overall size of the database, but also improves the intuitive understanding of the data. The overall structure of the fact table resembles written reports more closely than relational databases do. Kimball notes, that it even is beneficial to the readability in a user's point of a view, if some data in the dimension tables is repeated. Today, the star schema is the dominant approach to modeling data warehouses. However, every time an argument erupts whether it should be used or not, the overhead in storage space due to the denormalized storage of dimensions is quoted. One almost wonders how this discussion ever started, as the size of dimension tables is clearly negligible in multi-dimensional data warehouses. Usually the storage space used for facts will exceed the storage space used for dimensions by multiple orders of magnitude. According to Kimball [30], even a very large dimension will not occupy more than about 5 GB of space. Yet, for a data warehouse with such extraordinarily large dimensions, the fact table will reach multiple terabytes.

The star schema certainly provides the best browsing and query evaluation performance possible. It's design is simple and resembles the natural and intuitive view of an enterprise; this adds the benefit that users require no special training to fully understand the underlying data models and effectively use the database for decision support. However, this schema has its drawbacks: it is one extreme in the *data warehouse size* vs. *query performance* tradeoff and may cause an explosion in the size of the data warehouse. In addition, the synchronization process between the transactional database and the data warehouse is a non-trivial operation if this database scheme is used. Still, fast query execution makes this an ideal database schema for OLAP systems which require interactive responses to user input.

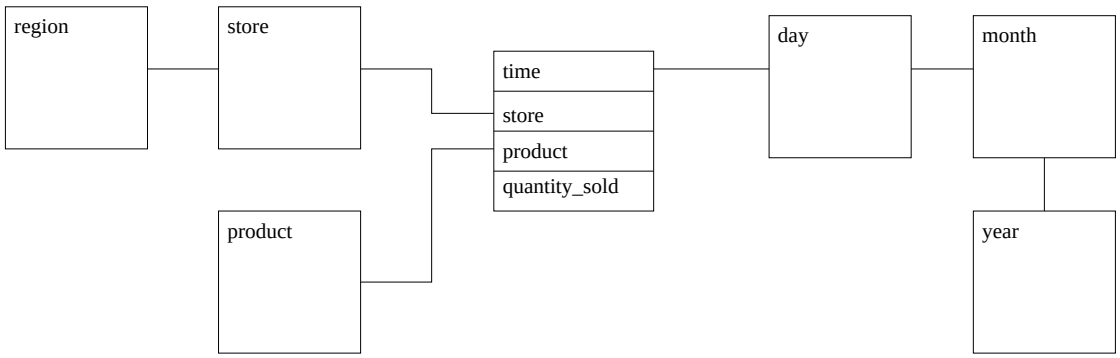

Figure 1.5.: Snowflake schema

The snowflake schema

A *snowflake schema* has its hierarchies decomposed and each one-to-many relationship mapped to a separate table (compare Figure 1.5). This inflate the number of tables used, while reducing the total storage size required for the fact base by eliminating unnecessary redundant entries in these dimension tables. Again, this schema consists of a central fact table and entries of the fact table are mapped to entries in the dimension tables using a foreign keys. The dimension tables may in turn refer to other tables. A time dimension may thus be nested through multiple tables, as multiple days belong to the same month and so forth. This saves space at the expense of a less intuitive browsing and a slower query evaluation. However, the dimension tables are multiple orders of magnitude smaller than the fact base. For this reason, the saved space can almost never be justified in face of the necessary processing overhead introduced by additional join-operations.

Since the browsing of the dimensional hierarchies slows to a crawl when using such a schema, it can seriously hampers the interactivity of OLAP systems. Kimball [30] cites another reason against the use of snowflake schemata. According to him, the average user–analyst would be intimidated by the complexity of the database schema, when confronted with it. Yet a snowflake model offers one decisive

advantage over star problems, whenever frequent and incremental synchronization with the operational database is necessary.

1.4.3. Feeding and maintaining the data warehouse

The data warehouses behind the OLAP systems have two modes of operations. During normal operation they are in read-only mode, processing queries. The systems then go off-line for the synchronization with the operational data base. While the query phase has highly variable requirements of resources, the synchronization process (also referred to as "feeding") has a predictable, constant and extremely intense workload.

First the data warehouse is cleared. Now data can be loaded into it, anew. The actual loading phase consists of the extraction of data from one or multiple transaction-oriented relational databases or legacy systems. From this data, dimensional keys are extracted. These dimensional keys help to categorize the data. Now it is possible to insert the fact into the new fact table. Afterwards, the newly created data warehouse is cleansed (i.e., duplicate entries and other inconsistencies are removed). In response to this update of the fact base, some (or all) of the pre-computed consolidated values will need to be recalculated.

Besides this naive approach to maintaining a data warehouse, newer and better methods exist as well:

- **incremental feeding.**

  Incremental feeding is the attempt to incrementally update the data warehouse, without clearing and re-calculating the entire cube. How this is done depends mostly on the dimensional structure and the consolidation functions used.

- **versioning.**

  Quass [42] introduces the *2VNC* algorithm, which allows the concurrent up-

date of the data warehouse. This has the benefit that the data warehouse and OLAP system are available around the clock.

1.5. On-line Analytical Processing as a foundation technology

On-line Analytical Processing proves to be an increasingly popular base technology to access and evaluate large data sets. However, it is not a panacea. Instead of viewing OLAP systems detached from their environment, current research focuses on integrating them with other tools. Three main directions for the development have emerged:

1. OLAP as a visual data exploration tool.

   Scientific experiments generate very large data sets in the multi-terabyte range. The COMPASS experiment, which is described in [18], will generate an object set of about 20 terabytes. The NA48 experiment at the CERN already records data at a sustained rate of 20MB/sec [41]. Some scientists [39] already warn, that data storage systems will soon become too slow to handle the enormous amounts of data generated. Although modern high performance computing techniques, cluster computing and parallel I/O are able adapting to process such enormous amounts of data, it remains a challenge to present this data in a useful and responsive fashion to human analysts. How, if we are not even able to move data quickly enough, can we view and analyze this data interactively?

   On-line Analytical Processing provides interactive access, data visualization and analytical features. Pre-aggregation enable the system to produce answers to frequently used queries instantaneously—independent of the size of the data warehouse. Recent research [49] even focuses on methods to approximate multidimensional aggregates to further extend the reach of OLAP

applications.

2. OLAP as a front-end to a database.

   The On-line Analytical Processing paradigm may serve as a flexible interface to a database. It provides database navigation, database queries and data analysis within a multi-dimensional model. This is of particular interest to all applications that require the detection of trends and exceptional events. Examples of such applications can be readily found in financial planning and portfolio management.

3. OLAP as a foundation for data mining technologies.

   Modern data mining techniques and knowledge discovery algorithms can provide superior performance and/or accuracy, when used in conjunction with an On-line Analytical Processing. The summary information provided and the different views at various granularities are a prerequisite to an "intelligent" behavior of the data mining algorithms. Observations on the summary information can be correlated to features of the raw data: this provides the infrastructure for dynamic data dependent optimizations.

1.6. The Limitations of On-line Analytical Processing

From this introduction, a few limitations and weaknesses of the current form of On-line Analytical Processing may have become apparent. This section addresses these open problems and outline possible solutions.

Although historical data is provided and evaluated, the integration of volatile data with data warehouses is not entirely understood at this time. Update may occur repeatedly, incrementally or even frequently. However, the update window (i.e., the time until a change in the operational data is reflected in the data warehouse)

in these data collections remains infinite compared to their operational counterparts. An algorithm for versioning data warehouses during updates exists [42]. This solves the consistency problem, but can not shrink the update window.

The performance and scalability promise of OLAP largely still remains to be fulfilled. Most implementations execute queries serialized on a dedicated server system. If modern load distribution techniques are used, then they remain limited to the underlying data warehouse. This is surprising, could be easily adapted to exploit both parallel and distributed execution environments:

- **Parallelization.**

  The materialization of a data cube is an inherently parallel process. While aggregating, the cube can be partitioned into disjunct patches, which may be processed in parallel. This works best on shared memory architecture, or other systems with high bandwidth, low latency communication. Sanjoy Goil [24] describes the effects of such optimizations on an IBM-SP2.

- **Distribution.**

  One of the distinguishing features of multi-dimensional databases is the independence of its dimensions (i.e., large parts of the query evaluation can be carried out independently and concurrently). As a consequence the evaluation of queries can be efficiently distributed across networks of workstations. No communication is necessary between these distributed processes. This approach is described in detail in chapter 3 and used in the system presented by Rauber and Tomsich [43].

To discover trends and exceptions in data interactively and to carry out analysis are arguments for the use of On-line Analytical Processing. However, the next generations of tools will aid the analyst by guiding the analytical process using knowledge discovery techniques built into the query tools and active database elements—i.e., rules and triggers—within the OLAP server.

Aggressive pre-aggregation improves the responsiveness, but leads to an explosion in the size of the data warehouse. Particularly with very large databases which contain a large number of dimensions or a large hierarchy of granularities, the adverse effect of pre-aggregation on scalability becomes apparent.

One of the most important advances in databases was the notion of data quality management. Instead of simply storing information, it is annoted with a quantification of it's trustworthiness. This measure of trustworthiness is referred to as the *data quality*. One of the problems of current OLAP systems is their ignorance of data quality. Neither the consolidation of quality-annoted facts is solved, nor the problem of requiring a query to provide a result of a certain quality or higher.

In a few years from now, current data warehouses will have accumulated a wealth of information—too much and too detailed information to reasonably evaluate. It is a well known fact, that data ages. Old data will not be required on the same fine granularity level as young data; old data is used to monitor trends. while young data also has to accurately reflect extreme data points to detect exceptional events. In order to provide high accuracy for recent data and providing for a long history, some data has to be purged from the system. Such a *forgetting* of data, would first aggregate the data to the next higher level and then purge the detailed data from the system. Still, the run-time behavior of On-line Analytical Processing is highly indeterministic. Future developments will have to overcome the worst-case complexity an ad-hoc query, in order to open new application domains to this data analysis technology.

Finally, security concepts for OLAP-based business information systems have to evolve. The integration of access control with On-line Analytical Processing remains an open question. The usability of executive information systems without adequate protection for the secrecy of strategic information is limited at best. The all-or-nothing approach from other types of data bases does not scale well: It necessitates a separate data cube per user and reduces the effectiveness of caching. A fine-grained security concept, which integrates well with multi-dimensional databases

is necessary.

All these problems are under currently under investigation and it is expected, that future systems will offer novel solutions to them. The requirements for successful solutions to these problems and outlines of possible approaches to them will are discussed in chapter 4.

1.7. Commercially available OLAP systems

According to "The OLAP Report" [1], an independent publication assessing the market for On-line Analytical Processing solutions and implementation services, the worldwide OLAP market grew from $1 billion in 1996 to over $2 billion in 1998. The growth is expected to continue at this rate for at least another two years and the market volume is projected to reach $4 billion in 2001. Competing estimates predict even higher growth rates and market sizes, reaching up to $8 billion in 2002. However, the growth rate will likely slide in the coming years as a certain degree of market saturation will be reached in the traditional market of business data analysis. Another slowing factor is the appearance of low-cost products, such as Microsoft's OLAP services which already captures market share from far more expensive alternatives. The amount of consulting required for current generation systems is also far lower than for earlier generation and may slow the growth of the market size.

With this background, it is not surprising, that every major provider of database solutions and data visualization tools offers at least one OLAP system. In total about 30 vendors compete in this high profile market. Data warehousing solutions and OLAP systems are available from a number of sources including IBM, Brio Technologies (which recently entered a distribution agreement with IBM), Red Brick Systems, Cognos, Microsoft, ORACLE, MicroStrategy, Hyperion, SGI, Platinium Technology, and Sequent. Table 1.1 lists the most important vendors and provides a breakdown of their respective market share (compare [1]).

Vendor	**1998**	**1997**	**1996**
Hyperion Solutions	28.7%	24.7%	n/a
Oracle	17.0%	20.7%	18.7%
Cognos	9.6%	10.3%	9.0%
MicroStrategy	6.4%	4.5%	3.4%
Comshare (incl. Essbase resales)	4.8%	7.5%	11.8%
Business Objects	4.4%	3.6%	1.0%
TM1, Inc.	4.1%	4.3%	2.2%
Seagate Software	3.5%	4.1%	4.2%
SAS Institute	3.6%	2.1%	1.5%
Information Advantage	2.9%	1.9%	1.4%
Applix	2.5%	2.5%	1.7%
Pilot Software	2.1%	3.9%	4.9%
IBM	1.9%	n/a	n/a
Gentia Software	1.5%	2.0%	2.5%
Informix	1.3%	1.2%	1.5%
Brio Technology	1.1%	0.8%	0.5%

Table 1.1.: Market share of the leading OLAP vendors.

1.8. Roadmap

The remainder of this thesis details the design and implementation of an extensible, modular, distributed and parallelizable On-line Analytical Processing research prototype. It is organized as follows:

- Chapter 2 reasons about the multi-dimensional data models used for On-line Analytical Processing systems. Three data models including those introduced by Agrawal *et al.* [2] and Bayer [5] are summarized, before a novel, set-based and efficiently parallelizable approach is presented and compared to the existing models. The algorithms necessary to implement basic OLAP functionality are derived for this model and analyzed.

- Chapter 3 provides a break-down of the system into functional components. The components are connected using a *software bus*, such as OMG's CORBA. Although, this section focuses on the coarse-grained parallelism introduced by the fact that the various dimensions of a multi-dimensional data warehouse are independent during most of the query processing, a discussion of fine-grained parallelism based on the model from Chapter 2 is included.

  In this chapter, we also present the inter-component interfaces. These are derived from the formal model which the goal of simplifying and minimizing the inter-component communication.

- Chapter 4 discusses the integration of fine-grained, per-entity security, data quality and active data base elements (i.e., rules and triggers). An overview of the challenges faced when implementing time-constrained On-line Analytical Processing concludes this chapter.

- Finally, a short summary and conclusion follows in chapter 5.

2. A formal data model

"Grace in all simplicity."

William Shakespeare, "The Phœnix and the Turtle"

2.1. The Necessity

Multidimensional data analysis uses a dimensional approach to modeling data. Although the implementors of On-line Analytical Processing systems are primarily concerned with choosing and composing appropriate data structures and algorithms, the implications of using a multidimensional data model, such as sparse data structures or the independence of dimensions affect both the structure as well as the implementation of the resulting systems.

No generally accepted formal model of multidimensional databases exists today, in contrast to relational databases which where formally introduced in [16]. The concept of OLAP was formulated based on "best practices" resulting from the experience with decision support systems in a business context. This worked to a certain extent; however, we are beginning to see the limits of such a practice: On-line Analytical Processing is used exclusively for the analysis of financial data. Although OLAP has the potential to offer superior performance for the analysis and navigation of a wide variety of data, the absence of a generic formal model for OLAP hinders its adoption rate for "mission critical applications." The benefits of

a formal model for OLAP are apparent: it would be far easier to verify the correctness, estimate the worst-case and typical system performance and test applications for the conformance with OLAP principles.

In order to make certain that the system described in the following chapters offers the expected features, provides correct answers and optimizes the frequently used operations, a formal data model needs to be defined and analyzed. In this section, existing models of relational databases are presented and discussed, before a set-based data model is formulated. Query evaluation algorithms are formulated based on this model. The set-based approach chosen offers two prominent advantages: sets are a very well understood mathematical concept which allows for easy correctness proofs and performance analysis, and they are a data structure which can be parallelized efficiently. A short analysis of their respective space and time complexity follows. Finally, a modification to the model is presented which provides "binary hypercubes" capable of classifying data quickly.

2.2. The Concepts

A data model which is to satisfy the needs of On-line Analytical Processing, has to support a number of fundamental features. User interface concerns require an intuitive data model, which can be easily understood by analysts from a variety of disciplines. The analytical approach chosen will necessitate a multi-dimensional representation of the stored data, in order to efficiently navigate and analyze it. This multi-dimensional representation will need to capture the natural, hierarchical structure and groupings identified in the source data. In interactive systems, speed is paramount. Besides just providing correct answers, an OLAP system has to provide answers within seconds. Providing response times in that order of magnitude for multi-gigabyte data sets, requires carefully designed algorithms and data structures chosen in accordance with them. Further, an easy mapping between the data model used and legacy systems, such as relational databases, is necessary.

The concepts detailed in the remainder of this section are prerequisites to providing efficient and intuitive functionality in any system. This section establishes the concepts of a data model used for OLAP, before formalizing it.

2.2.1. Multidimensional databases

The key-feature of On-line Analytical Processing, when compared to other analysis techniques, has always been its multidimensionality. OLAP systems derive their intuitive interface and data exploration capabilities from dimensional modeling. Almost all data collections we encounter display a clear differentiation between key attributes and data attributes. These data attributes are usually measures quantizing a real-world event. Still, such a measure is worthless without the key information relating it to the other contents of the database.

Visualizing such a database relating key information with measures would yield a large set of tuples in the relational database model (or, depending on the viewpoint, it could also be viewed as an associative set which maps values to keys). All the explicit knowledge about a system can be mirrored in such a flat database. Nonetheless, it would not faithfully reflect the actual structure of reality.

In searching for an alternative to the relational model, one inevitably comes across a model dealing with the natural structure of data: the multi-dimensional model. This approach to data modeling first identifies possible categories to order the measures by. These categories describe the semantics of the key attributes found in a relational database and coincide with partitions of these key attributes. For example, if data is collected over time, then the timestamp of each measure may consist of multiple attributes (a partition) describing the temporal location of the event documented. The same measure may also be categorized according to the location it occurred at, adding a spatial location. In this simple example, the database would be two-dimensional. Every fact would be (at least conceptually) stored at the location in a 2-dimensional space corresponding to the projection of its Cartesian-product of its spatial and temporal key. Obviously this can be extended

to any number of dimensions, possibly creating very sparse, high-dimensional data-spaces.

However, this does not justify the introduction of a new data model. Everything described so far could be emulated using relational modeling, without an increase of sparsity within the data structures. Multi-dimensional modeling provides two compelling advantages over relational modeling.

1. **Simplicity.**

   Relational database are designed for and used with normalized data models. While the multi-dimensional model provides a high-dimensional space, within which all data is positioned according to its key attributes, relational models require a spider-web of tables. It would be necessary to design a denormalized data model to achieve the simplicity of a multidimensional model. That would increase the storage overhead by orders of magnitude, complicate consistent update operations and contradict the principles of database design for relational databases.

   In contrast, multi-dimensional data models are easily understood, as they are equivalent to high-dimensional arrays. As such they also offer simple algorithms and intuitive operations. It is possible to select a cross-section of the cube (*slice* the cube) or select a sub-cube (*dice* the cube). The results of such operations will again be high-dimensional data cubes. Consistency is one of the major benefits of this model: even the reduction of dimensionality for output can be expressed in term of consecutive slice operations.

2. **Performance.**

   Calculating a cross-section of the virtual data-cube stored as a denormalized relational database would still involve testing a large number of records against the selection criteria. The overhead of performing such an operation for a multi-gigabyte data set certainly exceeds the response time expected from an interactive tool.

> Assuming an appropriate storage method, which clusters data accordingly, a multidimensional implementation may be able to retrieve a slice without "touching" any other records. In the multi-dimensional model, a cross-section would be literally sliced out of the cube.

Although the performance of multi-dimensional databases should prove superior to what may be expected from implementations of other models, it is essential to note that computers where never designed to efficiently process high-dimensional data-structures. For this reason, the actual performance of any such system will largely depend to the interactions between the chosen data structures and the model. In a sense, multi-dimensionality contradicts the architectures of modern computer systems: Computers are able to quickly process small data sets from cache memory or even data sets read linearly from memory. Apparently, multi-dimensional data spaces, which are inherently sparse, and the frequent cross-sections calculated during query evaluation require novel data structure and storage techniques.

2.2.2. Hierarchically structured data

To fully grasp the conceptual power of the multidimensional database models, it can be beneficial to start by comparing the dimensional model to its equivalent flat relational database. Both databases will contain the same information, with the exception of the *meta-data*. Example of meta-data found in multi-dimensional data warehouses would include information describing the hierarchical grouping of entries in a time dimension. We measure time at different granularities: in minutes, hours, days, months and years. A year contains 12 months, which in turn contain between 28 and 31 days, and so on... We will group the database entries according to those units of granularity.

On-line Analytical Processing systems attempt to mirror this intuitive simplification of data to manage complexity. When large amounts of fine-grained data are

available, the user would be unable to usefully analyze this data, as he would get lost in the sheer amount of information. It is important to provide the user with the facilities to easily change the granularity of the data, as it is seen fit.

As mentioned before, granularities form a hierarchy. This hierarchy has an identifiable, unique top and a unique bottom. The instance of the top element always represents the entire dimension, while instances of the bottom element represent atomic facts. Although one may be tempted to think that there will exist exactly one path between top and bottom, there can be an unlimited number of different paths between these two. This happens every time, when competing possibilities of decomposing a dimension exist. For example, the time dimension can be broken up according to the calendar year and the financial year. For every instance of one of these granularities, an ancestor and descendents can be determined. An *ancestor* is the instance which completely contains its descendants; a *descendent*is an element completely contained in its ancestor. This *contains* a relationship serves well to put down implicit data such as "*The year 1999 consists out of the months January, February, ...*"

The before-mentioned intuitive simplification of data has to be introduced into the data model easily. During that process, multiple fine-grained facts are summarized into a coarser-grained fact. This resulting fact provides summary information for its descendants. The process of generating this summary information is called *aggregation* or *consolidation*. The resulting *aggregation values* are then used to represent the data, when the cube is viewed at a coarser-level. In order to derive the aggregated value from a set of finer-grained values, a function has to be evaluated. This function is the *aggregation function* or *consolidation function*, a vital component of every OLAP system as it encodes the domain specific knowledge. Aggregation functions have a major impact on a system's performance; most important regarding the overall system performance are the algebraic properties, such as whether the aggregation function is distributive or not. For distributive aggregation functions, it is possible to reuse pre-computed intermediate results from

earlier queries, while non-distributive functions always require a re-aggregation of atomic facts.

2.2.3. Converting between relational and multidimensional data-models

As indicated before, relational and multidimensional databases share most of the information. Multi-dimensional databases are enriched with meta-data describing the structure of the data. In order to convert a relational database into a multidimensional one, it is necessary to add the required meta-data. Parts of this meta-data can be extracted automatically, parts have to be added by hand.

In order to convert a relational database to a multidimensional one, it is first necessary to identify the dimensions and measures present in the data. The measures can be found easily: all those attribute, that are used primarily in calculations are measures [2]. These are the attributes which can be aggregated into meaningful summary information. The remaining attributes should contain key information, which will not be affected by calculations or aggregation functions. This rule to distinguish between dimensions and measures is just an attempt to give a rule of thumb. Although it usually works, no general way to create a multi-dimensional model exists. In some case, dimension-values (i.e., categorical attributes) will be used to conduct calculations and in other cases measures will also double as categorical attributes.

The actual process of determining how many dimensions exist and which attributes belongs to which dimension is more tedious. In order to separate the different dimensions from each other the key attributes need to be divided into partitions, such that no key attribute belongs to more than one partition and that no key attribute from one set affects the interpretation of a key attribute from another set. Each of these sets represents one dimension. For example, a year-attribute affects the interpretation of a month-attribute, because as the value of

the year-attribute changes, a different month (a month of a different year) is denoted. This partitioning has the simple effect of lifting keys consisting of multiple sub-keys into their own partitions.

After the attributes of a relational database are mapped to dimensions, and information regarding the granularity hierarchy is supplied, it is possible to automatically extract meta-data from any database. To do so, the entire database has to be scanned once. During this scan, the values for the various granularities are extracted and a graph of categories built from them.

2.3. State of the Art

A few attempts have been made to formally describe a multidimensional database model. This section describes three of the most frequently used models which represent different approaches to and three evolutionary stages of multidimensional databases. Each of these was designed to address a specific problem and none proves optimal under the workloads of OLAP systems.

2.3.1. Modeling multidimensional databases

An abstract data model for multidimensional databases is reported by Agrawal *et al.* in [2]. The authors' stated goal is to unify the often divergent styles used in the various products providing multidimensional database functionality. The presented model is based on the hypercube-metaphor and defines some algebraic operations which implement the functionality of a multidimensional database. However, the model stays as close as possible to relational algebra. The operators used are designed to be translated into SQL and can be implemented either on top of relational databases or with specialized multidimensional database engines. This model provides a number of fundamental features which were either missing from or ill-supported in products at that time:

- **Symmetric treatment of all dimensions and measures.**

  Dimensions and measures are treated equally. That is, selections and aggregations are allowed both on dimensions and measures.

  Suppose a query might be required to find the total sales for each product of a given range for the sales price. In such a case, the sales fact is both a measure and also a grouping attribute. This query categorizes according to the measure. Asymmetric treatment of measures and dimensional attributes would make such a behavior very difficult to achieve. According to the authors, such queries are rather frequent during data analysis.

- **Support for multiple hierarchies along each dimension.**

  Multiple hierarchies for the type-categories of granularities may exist. For example, when navigating sales data the temporal dimension may have two different hierarchies for the same data. A financial analyst would operate on financial years, quarters, *etc.*, while a marketing specialist would need to correlate the data with vacations, holidays and similar events which requires calendar years, seasons, months and so on. Roll-ups and drill-downs may occur on any hierarchy in such a set-up. As such overlapping hierarchies are frequently encountered in the modeling of business processes, supporting them is a necessity.

- **Support for computing ad-hoc aggregates.**

  Aggregates other than those originally specified need to be computable. That includes both the cases, where an aggregation function different from the one pre-specified is to be used (e.g., the average sales are to be calculated in addition to the total sales) and also the case if an "unusual" grouping becomes necessary (e.g., the average sales over all Aprils is to be calculated).

- **Operators are closed and minimal.**

All operators are closed. That is, they are defined as a mapping from one cube to another. The benefits of such an approach are simplicity and composability. As all operators can be composed in any order, complex operators can be built from the builtin operators. To further simplify things and ease the algebraic optimization of queries, all operators were designed to be minimal.

- **Support for a query model.**

  While older models had one-at-a-time operational semantics, this model supported the query concept. That is, queries are composed from the operators and later materialized. That is a harsh contrast to the earlier practice of performing each operation separately and presenting the intermediate result to the user to choose the next operation. Aside from being more declarative and less operational, this approach provide the framework for query evaluation.

- **A separation of frontend and backend.**

  Front-end applications, such as query tools and analytical packages, are clearly separated from the back-end and communicate through the operators provided. These operators form the interface to the multidimensional database.

The data is organized in one or more hypercubes. There exists no formal way to decide which attributes of the data should be used as dimensions and which one are better made measures. Each hypercube consists of k dimensions, with a name D_i and a domain dom_i for each dimension. Elements are defined as a mapping $E(C)$ from $dom_1 \times \cdots \times dom_k$ to either a n-tuple, 0, or 1. $E(C)(d_1, \ldots, d_k)$ refers the element at position $d_1, \ldots, d_k$. Part of the meta-data is an n-tuple of names where each of the names describes one of the components of an element of the cube (remember, elements are n-tuples, too). If the cube contains no elements, that table will be empty. The model makes no distinction between measures and dimensions; for this reason, the logical model of a cube may contain more dimensions than the

physical representation of the same cube in a storage system. As the elements of the cube can be either 0, 1, or an n-tuple $< X_1, \ldots, X_n >$. If the element corresponding to $E(C)(d_1, \ldots, d_k)$ is 0, then that combination of dimension values is not contained in the database. A 1 indicates the existence of that particular combination. If a tuple is stored for a combination of dimension values, that indicates that additional information is available. Elements are either represented using 1s or tuples. However, these two representations can not be mixed in a single cube. A cube is considered *empty*, if either all its elements are 0s, or if the domain dom_i of one of its dimensions D_i has no values.

All operations on the data-cube are described in terms of and assembled from a few minimal operators. These operators are:

1. **Push.**

   The push operation converts dimensions into elements, which can be manipulated using an aggregation function. This operation is necessary to allow dimensions and measures to be treated uniformly.

2. **Pull.**

   The pull operation is the converse of the push operator. It creates a new dimension for a specified member of each element. This operator is useful for converting an element into a dimension, which can be used for joining. This operator is also needed for the symmetric treatment of dimensions and measures.

3. **Destroy Dimension.**

   The dimensionality of the cube is reduced. The model allows only single-valued dimensions to be removed. The presence of such a single-valued dimension implies, that for the remaining $k-1$ dimensions, there is a unique $k-1$ dimensional cube, which results from eliminating this dimension. Multi-valued dimensions can be destroyed by first applying a merge-operation to

transform them into single-valued dimensions.

4. **Restriction.**

   The restrict operator operates on a dimension of a cube and removes the cube values of the dimension that do not satisfy a stated condition. This operator realizes the *slicing* and *dicing* of the cube.

5. **Join.**

   The join operator relates information from two cubes. The result of joining an m-dimensional cube with an n-dimensional cube on k-dimensions, which are called the *joining dimensions*, is a cube with $m + n - k$ dimensions. The resulting dimensions will have values that are the union of the values of the source dimensions. The elements of the resulting cube are obtained by applying a combining function to the elements that get mapped to each other.

6. **Association.**

   The associate operator is a special case of the join operation. It is especially useful in OLAP applications for computations like *"express each months sale as a percentage of the quarterly sale"*. This operation is asymmetric and requires each dimension to be joined with a dimension of the other cube. During joining, the dimension-values can be mapped to arbitrary dimension-values of the other cube's dimension.

7. **Merge.**

   The merge operation implements an aggregation operation and its inverse. It is used whenever the level of detail changes. As a result multiple elements of the source cube are merged into one (i.e., changing to a higher level) or each element is replaced by multiple (i.e., moving to a finer granularity). Therefore, the merge either implements a *many to one* or a *one to many* mapping.

These operators are very close to relational algebra. This is both an advantage and a major drawback. These operators can be quickly and easily translated into SQL queries. This provides a migration path and allows the integration of existing systems with the multidimensional database. OLAP capabilities can be built on top of relational database systems. However, this faithful reliance on relational principles makes many of the operations unnecessarily slow when used in conjunction with multi-dimensional storage structures. In addition, the operators do not follow the OLAP principle well. Slice and dice degenerate to special cases of the restrict operator. Additionally, the symmetric treatment of dimensions and measures either incurs an overhead or disappears when mapped to a multi-dimensional implementation. Zhao *et al.* [52] show that that OLAP systems based on the relational paradigm will never achieve the performance of genuine multidimensional implementations. They even argue that a conversion from relational, value-oriented models to multidimensional, spatial structure for the query evaluation and an inverse conversion for the result would outperform the same query evaluated on the relational database.

2.3.2. A data-centric approach

R. Bayer *et al.* [5–7] provide a data-centric model of multi-dimensional databases in their publications on universal B-trees (UB-trees). It could be argued that this does not actually qualify as a formal model for multidimensional databases. However, the UB-tree is a data structure designed specifically to store multidimensional data organized according to the star schema. For this reason, it can be used unmodified at the core of a multidimensional data repository and its model constitutes the formal model of the associated multidimensional database.

The UB-tree access structure is a method to organize objects which populate an n-dimensional data-space, such that they can be stored, retrieved and manipulated efficiently. It was designed to improve upon the earlier data structures which either could not give performance guarantees or were unable to handle dynamic data

sets. The multidimensional space is mapped to a linear data-space in such a way that the multidimensional clustering is preserved. The performance guarantees are given for the access primitives and are logarithmic for the number of objects in the data-space.

The model underlying the UB-tree handles all dimensions symmetrical, but treats measures in a special way. This, in contrast to the abstract model described earlier, mirrors the realities of storing data: a difference exists between the data stored (the measures) and the position where the data is stored (expressed in terms of dimension-values). Basically, the UB-tree is a variant of the B-tree, in which the keys are *addresses* of *regions*. Formally, a *region* is defined as the set-difference between two *areas*. An area is the result of mapping the partitions of a cube, which was partitioned into 2^n sub-cubes where n is the number of dimensions of the original cube, to a structure which successively adds up sub-cubes. The addresses of the areas are ordered lexicographically and coincide with the addresses of the last sub-cubes included in them.

The algorithms (operators) defined are limited to insertion, deletion and querying. Regarding the query evaluation, point queries (exact match queries) and range queries are treated differently. The model treats the dimensions symmetrically during query evaluation through an appropriate key to address mapping. The Tetris algorithm [38] efficiently supports operations from the relational algebras, such as selection, sorting, grouping with aggregation and projection.

The benefits of the data-centric approach in modeling multidimensional databases is apparent: an associated data structure is available and can be used immediately to implement the database. The particular data structure, the UB-tree, is proven effective for insertion, point and range queries. With its symmetric handling of dimensions and the fact that the clustering of data is retained, it appears to offer superior performance for OLAP. It even performs better with the addition of dimensions. However, the concentration of providing efficient storage and retrieval neglected the user's needs. The UB-tree performs worse than even rela-

tional databases for hyper-plane queries (i.e., queries where only one attribute is restricted). Unfortunately, such queries are frequent in a decision support context. Another drawback of this particular model is its reliance on processing addresses encoding the information for all dimensions; this severely limits the possibilities for a parallelization of the query execution.

2.3.3. Conceptual modeling based on E/R schemes

To complete the overview of approaches to modeling the multidimensional databases, a conceptual model for multidimensional data warehouses which builds on E/R schemes needs to be discussed. Golfarelli, Maio and Rizzi [25] define multidimensional databases in dependence of entity–relationship schemes. This permits a reuse of the database schemata used for relational databases in the construction of multidimensional databases and the semi-automated construction of multidimensional data warehouses.

In this model, a multidimensional database consists of *fact schemes*. Fact schemes consist of *facts*, *dimensions* and *hierarchies*. A fact is a unit of business information (i.e., a single entity stored within the data warehouse); a dimension determines the granularity adopted for representing facts, and a hierarchy determines how fact instances may be aggregated and selected. A fact scheme is structured as a tree, whose root is a fact. Each vertex directly attached to the fact is a dimension and subtree rooted in dimensions are hierarchies. The dimension in which a hierarchy is rooted defines its finest aggregation granularity. The attributes in the vertices along each sub-path of the hierarchy starting from the dimension define progressively coarser granularities. A fact expresses a many-to-many relationship among the dimensions. Each combinations of values of the dimensions defines a *fact instance*, characterized by exactly one value for each *fact attribute* (i.e., measure). Again, measures and dimensions are treated differently in the model.

This model provides an easy path from relational databases, which were modeled using entity-relationship schemes, to a multidimensional data-warehouse. Un-

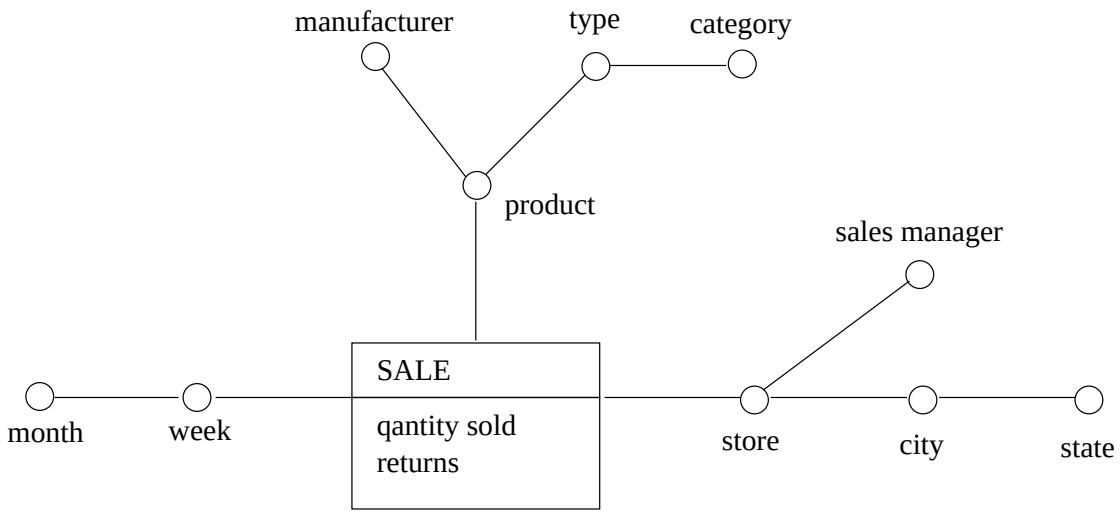

Figure 2.1.: A simple three-dimensional fact scheme generated from an E/R schema

luckily it remains too closely tied to relational databases to be used with multidimensional data structures.

2.4. The Formal Model

The models presented so far all have serious drawbacks when used for On-line Analytical Processing. Either they do not scale well, define operators which are less-than-optimal for OLAP or depend on a specific type of data storage. For this reason a novel, set-based data model is introduced here. It improves these deficiencies and works with any type of underlying data storage structure. Concurrency and scalability need to be considered during the design and become an integral part of the computational model.

It was designed in respect to the requirements of an easy and efficient implementation and leaves room for both parallel and distributed query processing. Furthermore, it relies solely on sets, nested sets and relations for its data struc-

tures and to define the operational semantics of its algorithms; it provides a well understood and easily proveable framework for a formal analysis. This model unifies elements of the models described above and satisfies a number of important design goals:

- **Symmetric treatment of all dimensions.**

  All dimensions are treated symmetrically. All dimensions have the same structure and priority. The order in which restrictions on the dimension-values are applied during query evaluation remains unspecified (i.e., even a concurrent application of these restriction is possible).

- **Ease of implementation.**

  This model makes use of sets extensively. They provide the hierarchical structuring of granularities and store intermediate results. As sets are a well-understood data type, their use eases the implementation. Even more important to this requirement is the simplicity of the algorithms for Roll-up and Drill-down.

- **A provable model.**

  The simple structure of the model and the pervasive use of sets permits proofs of the model and the verification of its implementations with respect to its correctness.

- **Support for ad-hoc queries.**

  Ad-hoc queries are supported. It is possible to specify an aggregation function and select and restrict its application to any subset of dimensional-values during a query. In order to provide efficient query execution, a restriction is introduced that limits the application of consolidation functions to measures only.

- **Scalability and support for parallel and distributed query processing.**

  The chosen approach to representing dimensions and dimension-values keeps them separate during query processing for as long as possible. This allows for a parallel execution of multiple instances of the query evaluation algorithm. In addition sets are a data type which can be parallelized efficiently [11, 13, 33]. Overall, this permits a superior scalability of databases building on this model.

- **Flexibility.**

  The model does not operate on any particular type of data. It can handle business data as well as scientific data. Any type of aggregation function can be specified, whether it is distributive or not.

- **No assumptions about the type and structure of the data warehouse.**

  Regarding the actual storage method used to realize the underlying data warehouse, the on-disk and the in-memory caches, no assumptions are made. The model operates solely on a multi-dimensional data cube; the positional vectors used can then be translated into the appropriate data access primitives.

- **Support for "virtual" data warehouses assembled from multiple databases.**

  An underlying data warehouse may be implemented using a single data repository or a number of separate storage nodes. The model does not assume a particular structure or access method to it. This feature permits the deployment of systems where the data is available in a distributed form only.

This section gives a short summary of the formal data model used for the system presented in this thesis. The description has been simplified as far as possible to

allow for easy reading while still mentioning the most important features of our approach.

2.4.1. Defining a multi-dimensional data-space

Relational databases provide a linear, uni-dimensional data-space. Facts are expressed as the instantiation of a relational schema. While the schema definition for a fact of j attributes contains a tuple of domain descriptions for each attribute of the fact

$$\text{Schema} := Dom(A_1) \times \cdots \times Dom(A_j),$$

an actual fact is a tuple of values chosen from those domains

$$\text{Fact} := a_1 \in Dom(A_1) \times \cdots \times a_j \in Dom(A_j).$$

However, such a definition does not capture the natural and logical structure of data. A different approach is necessary to reflect the hierarchical organization and multiple granularities observed in data. In order to express these facets of the data, multidimensional modeling is required.

A multidimensional data space can be defined as an extension to the relational data space by introducing additional semantical information. As this additional information consists of data about the structure of the data stored within the database, it is referred to as *structural meta-data*.

Multidimensional databases use a positional representation of data, as opposed to the value-based representation of relational databases. Instead of providing a repository for tuples, values (i.e., measures) are associated with positions (i.e., keys). To support this, the attributes ($A_i \in A$) are partitioned explicitly into two sets according to whether they contain a key or a measure. The resulting set of keys ($K \subset A$) and set of measure attributes ($V \subset A$) are disjunct ($K \cap V = \emptyset$) and add up to the original set of attributes ($K \cup V = A$). A naïve definition of positional databases in a relation ϕ which holds only if a tuple $t = k_1 \times \cdots \times k_{|K|} \times v_1 \times \cdots \times v_{|V|}$

is contained in the database

$$\phi : \; k\phi v \qquad \text{with } k \in Dom(K) \text{ and } v \in Dom(V).$$

That is, every instance of the database schema is broken into its key and value. The values are then associated with the keys and every value can be retrieved using its key. If ϕ holds, a vector k is the *position vector* of a fact v, since it represents the spatial position of the fact within the multidimensional data space. Yet, such a simple positional model does not offer any advantage over the relation model, alone.

In order to convey more information about the data and permit intuitive querying, it is important to recognize the fact that data is granular. That is, data does not only exist at one level of granularity, but can be viewed at a number of granularities. Even the data stored in conventional databases is granular, although the databases do not specifically support the management of such granular information. Examples for queries exploiting such granular data are *"What are the average sales per month?"* or *"What is the total revenue per year?"*.

Different levels of granularities materialize in the form of multiple associated attributes. For example, the different granularities of time may be represented using separate fields for the year, month and day. The term *semantically dependent* will be used for the relationship between such attributes. In general, *two attributes are semantically dependent if and only if, assigning a different value to one of them will affect the interpretation of the other one.* To illustrate this, assume the relationship between a month and a year attribute: if the value of the year field is changed, a month of a different year is referred to even though the month field remains unchanged. Using this definition, we can partition the set of keys K even further into n subsets $D_1, \ldots, D_n$ with the properties that $D_i \neq D_j$ with $i \neq j$, $D_i \cap D_j = \emptyset$ with $i \neq j$ and $D_1 \cup \cdots \cup D_n = K$. Additionally we require that for any selection of $a \in D_i$ and $b \in D_j$ with $i \neq j$, a and b have to be semantically independent. If these conditions are satisfied, we call those D_i *dimensions*. Every

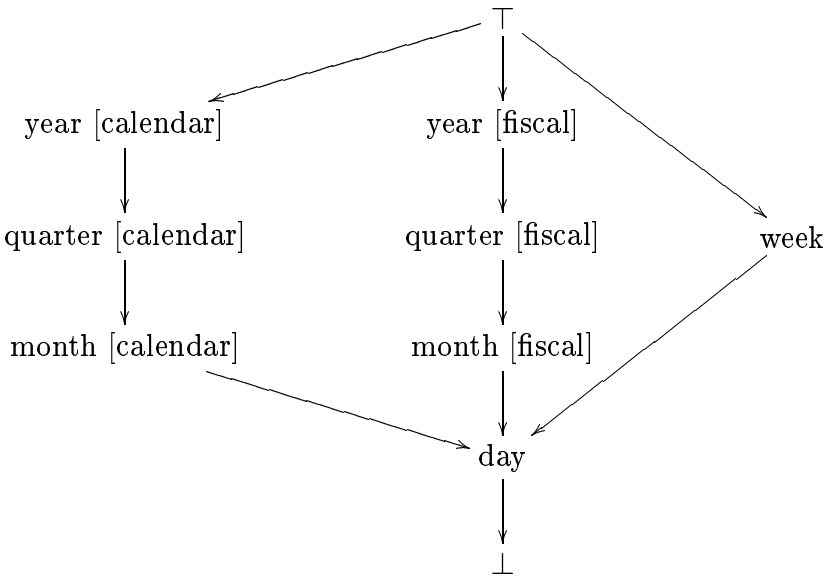

Figure 2.2.: Example of the granularity hierarchy present in a time dimension

dimension expresses a concept of the database. For example, a time-dimension will contain all time related fields (such as year, month, day, *etc.*), but no information related to a different concept such as a product classification.

Semantically dependent attributes usually imply a hierarchical ordering within their dimension which is equivalent to the hierarchy of granularities observed in the data. An example of such a hierarchy, as it can be found in a time dimension, is shown in Figure 2.2. The nodes of the depicted graph correspond to attributes of the dimension, while the vertices model a "contains" relationship. In order to guarantee that the resulting graph is a lattice, an artificial top node $\top$ and a bottom node $\bot$ are added. The top node represents the entire dimension, while the bottom node represents single facts. By adding the $\bot$ node, we require all facts to consist of the same attributes. Intuitively, a dimension-value consists of multiple sub-keys (i.e., elements from the domains of the K_i contained within the dimension).

More formally, given a dimension $D = \{K_1, \ldots, K_m\}$ of m different, semantically dependent attributes, a relation $\triangleright$ is introduced which models the hierarchical

ordering according to the meta data. $K \cup \{\top, \bot\}$ together with $\triangleright$ forms a directed acyclic graph, which we will call *hierarchy of granularities*; let $\triangleright^*$ be the reflexive and transitive hull of $\triangleright$. The following relations must hold:

$$\top \triangleright^* K_i \text{ for all } i \in \{1, \ldots, m\} \text{ and } K_i \triangleright^* \bot \text{ for all } i \in \{1, \ldots, m\}.$$

Every node must be a successor of the top node and a predecessor of the bottom node. Let $desc(K_i) = \{K_j | K_i \triangleright K_j\}$ and $asc(K_i) = \{K_j | K_j \triangleright K_i\}$. Furthermore, $desc(\top) = \{K_j | \top \triangleright K_j\}$ and $asc(\bot) = \{K_j | K_j \triangleright \bot\}$. A type K_i is said to be *uniquely decomposable*, if $|desc(K_i)| = 1$. In Figure 2.2, *week* is uniquely decomposable, while $\top$ is not.

Although this addition of type information in the form of a granularity lattice helps to understand the structure of the data, it does not include the mechanism to resolve queries in different granularities. This leads to the definition of the dimension values, which mirror the behavior of indices in relational databases and perform a logical grouping (and physical clustering) of data. For this purpose, a hierarchy of *dimension values* is maintained. Such a dimension value is a tuple

$$k = k_1 \in Dom(K_1) \times \cdots \times k_m \in Dom(K_m)$$

for a dimension D containing m attributes. If all these are merged into a single tree, all "contains" relationships present in the data are made explicit. Figure 2.3 shows an example of dimension values for a trivial granularity hierarchy.

Each node in this hierarchy is a tuple $key \times ancestors \times descendants$. The *ancestors*-set and *descendants*-set imply a part-of relationship on the keys: e.g., `March 1998` is a part of `1998`. Two nodes may contain the same value for the key field, but identify distinct spatial locations (e.g., `June1998` and `June1999` both have `June` as their key attribute). Apparently, the equivalence of nodes can be determined only from their location within the hierarchy of nodes. Let $\triangleright$ denote the *descendant* relation, such that $a \triangleright b$ holds, if and only if b is a *descendent* of a (e.g.,`1999` $\triangleright$ `June1999`). Now we can describe the position of any node n within the hierarchy

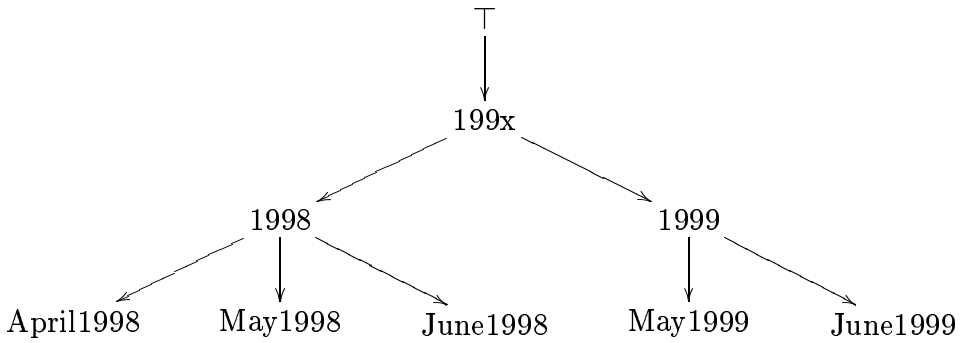

Figure 2.3.: Example of a time dimension: The arrows denote a "contains"-relationship.

as $p(n) = \{n\} \cup \{e | e \triangleright^* n\}$ where $\triangleright^*$ denotes the transitive closure over $\triangleright$. Two nodes can be tested for equivalence using their path: n_1 and n_2 are the same node, if and only if, $p(n_1) = p(n_2)$. For example, $p(\texttt{June1998}) = \{\top, \texttt{199x}, \texttt{1998}, \texttt{June1998}\}$ while $p(\texttt{June1999}) = \{\top, \texttt{199x}, \texttt{1999}, \texttt{June1999}\}$. Using the *descendant* relation, we can define the contents of the *descendants* and *ancestors* fields of a node as well: $n.descendants = \{e | n \triangleright e\}$ and $n.ancestors = \{e | e \triangleright n\}$. As a result, the node for `199x` is represented by the tuple $< \texttt{199x} \times \top \times \{1998, 1999\} >$. Since the nodes `1998` and `1999` contain a set of descendent nodes again, this leaves us with a structure of nested sets. This set-based notation was chosen over an equivalent relation-based notation for two reasons: it permits the query evaluation algorithm to be expressed as iterative term substitution. An early prototype, which was written in Scheme, actually implemented these structures with nested lists and expressed the entire query evaluation as a recursion over the levels of these lists.

An aggregation function $f_{aggregate}$ (often referred to as consolidation function) generates a fact containing summary information for a region of the data cube from a subset (i.e., a set of facts) of the data cube:

$$f_{aggregate} : \left\{ \bigtimes_{i=1}^{|D|} Dom(V_i), \cdots, \bigtimes_{i=1}^{|D|} Dom(V_i) \right\} \mapsto \bigtimes_{i=1}^{|D|} Dom(V_i).$$

To further develop the data model, the notion of a value aggregated to a higher granularity has to be introduced in such a way, that it can be stored within the original data cube. Such a value is derived from the set of values stored in the pertinent region of the data cube. Surprisingly this is very easy, by refining the naïve definition of the multidimensional database given before. Instead of mapping a vector $\vec{k}$ with components $k_{i\in\{1..|K|\}} \in Dom(K_i)$, we use a vector $\vec{n}$ of nodes to with arbitrary granularities. For this to be expressed in terms of the vector $\vec{k}$, an additional element e has to be added to the domain of each vector, which mean "unbound". Values are now associated through a relation τ with

$$\tau : \vec{n}\tau v \qquad \text{with } \vec{n}_i \in nodes(D_i) \text{ and } v \in Dom(V).$$

This allows for the storage of aggregated values (or arbitrary data) for any granularity within the same data cube containing the raw facts.

The model, as it was presented so far, does not support the aggregation of values on demand. That is, all aggregated data items have to be stored in advance or aggregated after the unavailability of a pre-aggregated value is detected. We regard this functionality to be an integral part of the OLAP system's query evaluator and not of the underlying multidimensional database, because it affects the caching logic and the load distribution of the query evaluator. For a detailed description of how this is implemented in the current prototype, please refer to Chapter 3.

2.4.2. Constraints and query evaluation

Queries to a multidimensional database consist of selecting a spatial region of the hypercube according to constraints on the key-attributes. We disallow testing value-attributes as it is commonly the case in OLAP systems, as this would make truly multidimensional implementation very difficult and make this model applicable to relational systems only.

In this model, queries are expressed as sets of constraints C and a granularity vector $\vec{g}$. The constraints indicate which subset of the hypercube contains relevant

information, while the granularvector specifies how to display the information and selects one node per dimension for the associated granularity hierarchy (e.g., in a database containing sales information over time per store and per product, a granularity vector might look like $\vec{g} =< \text{month}, \text{country}, \text{product category} >$).

The query algorithm proposed in this model is based on term substitution; for simplicity, we will only consider uniquely decomposable types at the moment. When starting a query, a set $S_i = \{\top\}$ containing an instance of the top level type is created. Until the desired granularity level is achieved, all nodes in S_i are replaced with their successors which satisfy the given constraints. The algorithm is outlined for the trivial case of uniquely decomposable granularity hierarchies in Figure 2.4. As soon as the result sets S_i are calculated, a Cartesian product of those will yield the position vectors of all data points satisfying the given constraints. However, the Cartesian product can expand to a very large number of position vectors and hide ranges of data points. For this reason it may be beneficial to retain the sets S_i and use these for fact retrieval. Another approach is the use of ranges as components to the position vector to retain a more concise representation. The interface to the physical data storage needs to translate the abstract positional representation into an access operation.

The algorithm given, works with uniquely decomposable granularity hierarchies only. In cases, when questions like *"What was the revenue for all days belonging to the calendar year 1999 and not to the fiscal year 1998?"* are to be answered, it fails. However, the extension for non-uniquely decomposable nodes is simple. Each possible path is traversed from the top type $\top$ to the desired granularity (which is a very complex task for large graphs) in the type hierarchy; these subgraphs are used as input to the query algorithm outlined above, yielding multiple sets S_i^j of nodes conforming to all constraints along one path. As soon as all these sets are available, their intersection is calculated yielding a set which contains only nodes satisfying all constraints in that dimension. For example, for the above query would first yield two distinct sets: one would contain all days in the calendar year

for each dimension D_i
  initialize a result set $S_i = \{\top \in nodes(D_i)\}$;
  while $desc(granularity(S_i)) \neq \emptyset$ and $granularity(S_i) \rhd^\star \vec{g}_i$
    determine constraints $C' \subset C$
      which are applicable to $desc(granularity(S_i))$;
    $S_i' = \bigcup_{x \in S_i} \left\{ y \;\middle|\; y \in x.descendants \wedge y \text{ satisfies all elements of } C' \right\}$;
    $S_i = S_i'$
  end while
end for

Figure 2.4.: The fact selection algorithm for hierarchies of uniquely decomposable granularities.

1999 and the other all days known to the data base that fall outside of the fiscal year 1998. The position vector would be generated from the intersection of these two sets. This is a very expensive proposition, unless a compact, intervall-based representation is adopted for these sets.

Still, one limitation remains to this algorithm: It fails whenever constraints are specified which refer to attributes which have a lower granularity level than the granularity specified in the granularity vector $\vec{g}$ (e.g., the desired granularity is "Month" and the "Days" are constrained). This occurs whenever an ad-hoc aggregate is desired. In the same spirit as before, when we excluded on-demand aggregation from the multidimensional data model, we disallow this in the fact selection process. If such functionality is desired, the application requiring the ad-hoc aggregate has to submit multiple queries and later aggregate the resulting facts. This guarantees a cleaner separation of mechanism and eases the implementation of distributed data warehouses based on multi-databases [20].

```
/* 1. execute the query for a single dimension */
printf("==== executing query for %s ====\n", dim);

long long           target_type_id = repository->get_type_id(dim, gran);
set< long long >    result_nodes;
set< long long >    active_nodes;
golap::Path         root_path(0); // an empty path

// initialize active set to { root };
active_nodes.insert(repository->instance_at(dim, root_path));

/* 2. calculate the result set */
while (active_nodes.size()) {  // as long there are active nodes
  /* 2.1. split between result and active nodes */
  for(set<long long>::iterator  i = active_nodes.begin();
      i != active_nodes.end();
      ++i) {

    if (repository->get_type_for_instance(*i) == target_type_id) { // compare the granularities

      /* 2.3.a this is a result */

      result_nodes.insert(*i);
      active_nodes.erase(*i);
    }
  }

  /* 2.2. calculate the expanded set */
  set< long long >    new_active_set;

  for(set<long long>::iterator  i = active_nodes.begin();
      i != active_nodes.end();
      ++i) {
    set<long long>  new_partial_set = instance_list_to_set(repository->expand(*i));
    new_active_set.insert(new_partial_set.begin(), new_partial_set.end());  // union
  }

  /* 2.3. apply constraints */
  apply_constraints(dim, new_active_set);

  /* 2.4. swap */
  active_nodes = new_active_set;  // swap
}
```

Figure 2.5.: A variant of the query evaluation algorithm, as it is used in the prototype implementation. It works correctly with non-uniquely decomposable nodes.

UPC	pkg. size	brand	sub-category	category	department	package type	diet type
90706287103	6 oz	Cold Gourmet	Frozen Foods	Food	Grocery	box	Diet
16005393282	6 oz	Cold Gourmet	Frozen Foods	Food	Grocery	box	Regular
57986858339	8 oz	Frozen Bird	Frozen Foods	Food	Grocery	box	Diet
67955177490	8 oz	Frozen Bird	Frozen Foods	Food	Grocery	box	Regular
46817560065	2 oz	Chewy Industries	Candy	Food	Grocery	can	Regular
84276830332	2 oz	Chewy Industries	Candy	Food	Grocery	can	Regular
51770124461	2 oz	Chewy Industries	Candy	Food	Grocery	bottle	Regular
33411763259	6 oz	Big Can	Soft Drinks	Drinks	Grocery	can	Diet
95946398896	6 oz	Big Can	Soft Drinks	Drinks	Grocery	can	Regular
88602993232	6 oz	Big Can	Soft Drinks	Drinks	Grocery	can	Diet

Table 2.1.: Example data from the product dimension

2.4.3. An Example

To illustrate how to apply this data model to a real-world problem, parts of the solution of the often cited grocery-store problem, which was created by Kimball [30] to illustrate the functionality of a data warehouse, is presented in this section.

The *product* dimension stores information about the products sold in the grocery stores. The merchandise hierarchy is an important group of attributes and contains information that uniquely identifies each product (the Universal Product Code/UPC), the brand, the product category and a sub-category and the department the product is sold in. Additionally the package type, package size and diet type are stored. From the textual description and the sample data in Table 2.1, the granularity hierarchy which is depicted in Figure 2.6 can be constructed immediately. Figure 2.7 shows a part of the of dimension-values for the product dimension, if the data from the sample table was entered into the data warehouse.

The *store* dimension describes the stores in the grocery chain. It is the geographic dimension of the grocery database. The store dimension contains a store identification (the store's name), the zip code of the store, the county, the state and the type of store plan. In contrast to the product hierarchy, this hierarchy does not only consist of uniquely decomposable types, as the floor plan is not related to the geographic features.

Suppose the data cube contains the sales numbers as a measure and aggregate

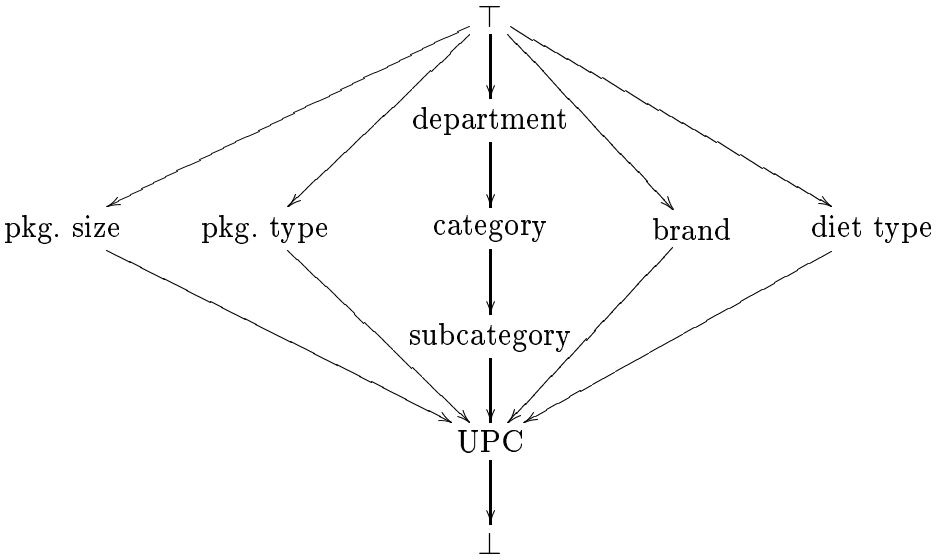

Figure 2.6.: Granularity hierarchy for the product dimension.

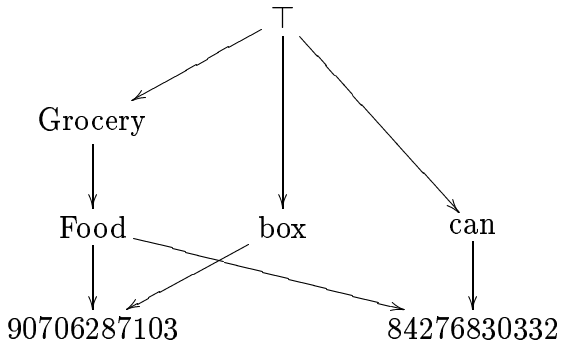

Figure 2.7.: Tree of dimension-values for the product dimension.

name	street address	city	state	region	floor plan
Store No. 1	999 Main Street	New York	NY	Eastern	Modern
Store No. 2	73 Main Street	Chicago	IL	Mid West	Original
Store No. 3	1 Main Street	Atlanta	GA	South East	Compact
Store No. 4	575 Main Street	Los Angeles	CA	Pacific	Modern
Store No. 5	123 Main Street	San Francisco	CA	Pacific	Original
Store No. 6	353 Main Street	Philadelphia	PA	Eastern	Compact
Store No. 7	839 Main Street	Pittsburgh	PA	Eastern	Modern
Store No. 8	651 Main Street	New Orleans	LA	South West	Original
Store No. 9	912 Main Street	Seattle	WA	Pacific	Compact
Store No. 10	752 Main Street	Dallas	TX	South West	Modern

Table 2.2.: Example data from the store dimension

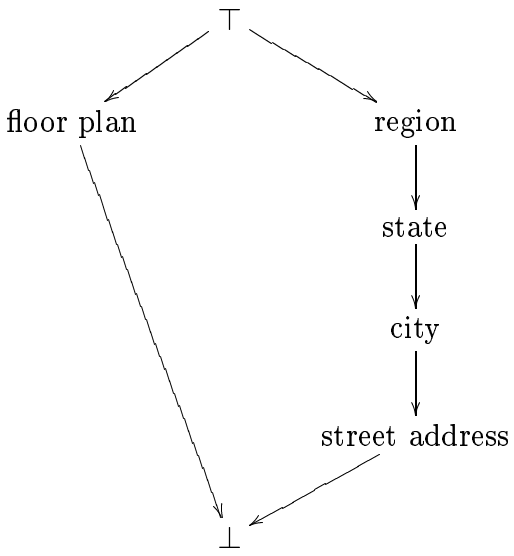

Figure 2.8.: Granularity hierarchy for the store dimension.

using summation. An analyst might ask *"What were the total sales of frozen foods for the stores in California?"* This question translates into a constraint set

$$C = \{\text{Products.Subcategory} = \text{“Frozen foods”}, \text{Stores.State} = \text{“CA”}\}$$

and the granularity vector

$$\vec{g} =< subcategory, store > .$$

Answering this query would traverse the trees of dimension-values and finally yield two sets, one for each dimension. The first set would include the UPCs of products belonging to the frozen food subcategory:

$$S_1 = \{90706287102, 16005393282, 57986858339, 67955177490\}$$

. The other set would include the stores

$$S_2 = \{\text{“}StoreNo.4\text{”}, \text{“}StoreNr.5\text{”}\}$$

. The position vectors relevant to the query would be the result of a Cartesian products

$$S_1 \times S_2$$

. The corresponding relational query is `SELECT * FROM table WHERE table.UPC in {90706287102, 16005393282, 57986858339, 67955177490} and table.storename in {``Store No. 4'', ``Store Nr. 5''}`. However, such a straightforward approach would prove extremly inefficient with large hierarchies of dimension-values. Therefore every serious implementation would stop at the subcetegory level and the store level, because no granularities are defined for more detailed levels. In that case the query would yield only one position vector, namely < “Frozen foods”, “CA” >. This would yield a much simple SQL statement `SELECT * FROM table WHERE table.subcategory=''Frozen foods'' AND table.state=''CA''`.

2.4.4. Supporting Classifier Systems

Classifier systems are consulted for questions like *"Did any sales of large quantities of Product A occur in Western Europe since last December?"*. Apparently they require a different data layout for optimal operation and constitute what Kimball [30] calls *fact-less data warehouses*. While some models support this implicitly [2], they usually impose a performance penalty for other queries. Aware of this problem, we support classifiers in our model using *binary hypercubes*. These result from a transformation of the "normal" data cube into a data cube containing no measures, but a boolean value. All attributes consist of keys and thus dimension-values. The aggregation function for the boolean values is defined as a `logical and` or a `logical or` depending on the application. This boolean measure can be stored implicitly: either a fact for a position vector is present or absent. As a result, a query either finds facts or fails. If facts are found, the question asked is answered "yes", otherwise "no" is returned.

2.4.5. An Assessment

Generally, this model should provide for a far higher degree of parallelism during query evaluation than the model discussed before. The trees containing the dimension-values are totally independent and can be traversed concurrently. It is thus possible to process the dimensions of a database in parallel when calculating the pertinent position vectors. Two distinct kinds of parallelism can be identified:

1. **external parallelism**

   The query evaluation algorithm can run concurrently for each dimension. No communication between these threads of execution is necessary up to the point where the actual fact retrieval is executed. From our results, a shared-nothing approach suffices for a reasonable speedup, assuming the meta-data is replicated. However, in a well designed star-model the same effect may be exploited.

2. **internal parallelism**

   Within each instance of the query evaluation algorithm, the expand–constrain iterations can be parallelized as well. At every expand operation, the active set can be split into subsets and the algorithm can be performed on those in parallel. This results in a tree-like branching of the algorithm. At the end, the union of these partial result sets has to be calculated to produce the result set. To exploit this type of parallelism a shared-nothing architecture suffices again.

   Furthermore, the constraints can be applied in parallel on a shared memory machine either by choosing an appropriate data structure or by creating multiple intermediate sets and intersecting those. Which approach will offer the best performance depends on the type of queries and the actual hardware paradigm used.

The direct translation of this data model and query evaluation algorithm should be possible as well. From the descriptions given above, only four distinct data structures prove necessary: **granularity vectors,** which specify the desired level of detail in *queries*, **sets of constraints,** which determine the subset of the data cube in a *query*, **position vectors,** which describe the spatial positions of facts, and **sets of tuples,** which encode the query results. Sets are usually reduced to (ordered) lists, whereas tuples and vectors map either to lists or arrays. Therefore, this actually requires only two data-types (i.e., lists and arrays).

We expect a very reasonable query evaluation performance, as the complexity of calculation the position vectors depends mainly on the number of nodes visited in the granularity hierarchies, because techniques for the efficient implementation for the storage of the dimension-values can be used, which simplify the evaluation of range queries and reduce the copying overhead. An even greater incentive to use this model is the fact, that the constraint application can be carried out in parallel for the different dimension to generate the sets of permissible indices. The fact, that

the structural dimension-values are present in a tree allow for the implementation of sophisticated user interfaces based on browsing this information. The same is true for the granularity hierarchy.

2.5. Summary

This chapter surveyed the different approaches to modeling multidimensional databases. As the existing models display major deficiencies either by limiting the parallelism or preferring relational implementations, a novel data model is briefly introduced which was designed to scale easily in parallel and distributed environments. We showed the viability of the model and its fact selection algorithm on some data from the grocery store example discussed by Kimball [30].

The main benefits of the presented model can be harnessed in applications with complex modeling needs. The uniform representation of attributes allows calculations to be performed on the categorical attributes, modifying the dimension tables. This flexibility may come in handy for deep and wide dimensions, and also in case dynamic dimensions are necessary. We are currently investigating a number of problems which may need these features, including a digital library application which could use dynamic dimensions.

Nonetheless, a high price is paid for these features. Both the space necessary to store the dimensions-values is very large as well as the amount of memory that may be used durin the set expansion process. Although we believe that this will not be a serious problem in practical applications, the worst case space and time complexity is frightening.

A direct translation of the fact selection algorithm into a program is possible and only simple data types, such as lists and arrays, are necessary. The implementation of *virtual* data cubes consisting of multiple, distributed storage nodes is possible.

3. A component-based architecture

"The biggest problem is that tools may not work together,
despite vendors' claims that their products are compatible."
Communications of the ACM, Sept. 1998

3.1. Overview

Today's data warehouse and OLAP architectures harbor two major shortcomings: First, they offer only limited parallelism, which severely hurts the scalability of these systems. Parallel implementations are based either on pipelining or on partitioning of queries [12]. Both types of parallelization stem from the underlying relational data base systems and largely ignore the particularities of multi-dimensional databases, such as independent dimensions. In effect, these parallel data warehouses are simply data-warehouses built on top of parallel RDMSs. The second shortcoming, which will have an even worse effect in the long term, is the fact that the present solutions are exclusively proprietary and closed systems, which can interface to a few different data-warehouses only. This may not prove an obstacle to the deployment and adoption of OLAP-based solutions in the short-term, as every major data base vendor offers at least one product for On-line Analytical Processing, but will become a serious reason problem in the future: With the growing dissemination of these products and the increasing amount of inter-company business

conducted electronically, the demand for a joint analysis of multiple, independently developed data warehouses will grow. Information technology providers are already faced with the challenges of a unifying multiple data warehouses whenever companies merge. Of course, the challenges faced in such situations exceed the mere availability of query and analysis tools and primarily involve the data description on the meta data level. However, the fact that different tools have different feature sets and may not work together complicates the task furthermore. Additionally, the absence of inter-operable components makes it difficult for implementors to customize the systems by mixing and matching the available components from different vendors.

While a large amount of publications exist on the various data structures and associated algorithms for OLAP, it remains a challenging task to implement and deploy an open system with clearly separated modules, as no *standardized architecture* exists, which describes the common components and functional units present. This section describes a component based architecture, which builds on the model described in Chapter 2 for the inter-component interfaces. It allows both for the development of distributed and parallel OLAP systems consisting of co-operating modules. The benefit of a well-defined architectural definition is self-evident: Clearly separated software modules simplify the development process considerably and accelerate the construction and deployment of applications. Different implementations of well-defined interfaces will remain inter-operable, so that changes are kept local. Above all, simple components with limited functionality are easy to design and implement. Complex systems can then be composed of such small components.

Component-based OLAP systems offer a number of benefits both to the user and the developer. The user can choose from different suppliers and, to give just one example, combine different query optimization strategies and query evaluation algorithms. Extensible OLAP systems offer a major benefit to developers, as well. Post-deployment modifications are simplified and testing of components can

improve the software quality. This chapter shows how to decompose an OLAP system into functional units, which can communicate using a "software bus" (e.g., CORBA).

A prototype system has been implemented as a proof-of-concept for this design. This prototype was written in C++ making extensive use of the ISO Standard C++ Library (formerly known as Standard Template Library), the freely available CORBA implementation MICO, the Gnu GUILE embeddable Scheme interpreter and the Gnu C++ compiler. It has been successfully compiled and tested on Alpha, Intel and PowerPC architectures with Linux and also on a MIPS-based SGI Power Challenge XL running IRIX 6.2[1].

This systems initial conception was motivated by the need to replace an older, C–based research prototype (see [34, 35] for a description) with a new, object-oriented, highly modular and parallelizable system to sustain future research at the Institute of Software Technology. While the current prototype can not yet act as a replacement of the current production system, work is being continued with the intention of providing a complete, extensible and distributed OLAP environment based on CORBA objects.

3.2. Architectural Considerations

The architecture [43] which is proposed here, fulfills several important, and often interrelated, goals:

- **Modularity.**

  All modules, which adhere to a predefined interface, can interact seamlessly. This allows for an individual development of these modules.

- **Inter-operability.**

[1] Using `egcs` instead of the default `MIPS Pro` compiler.

The system has to work with a wide array of databases and storage models. It is clearly unacceptable to force a rewriting of the query evaluator whenever a different data base system is used. As a consequence, the integration of multiple database systems based on wrapper modules needs to be supported. We use so called *storage managers* to achieve such transparent integration. The storage managers translate requests for facts from the internally used multidimensional model (i.e., location vectors) to the query language used (e.g., relational SQL).

- **Scalability.**

  OLAP requires consistent reporting performance, independent of the size of the underlying database or its dimensionality. Parts of each query can be materialized in parallel on separate machines. Multiple meta-data repositories replicate the meta-data across a network. Hierarchical and distributed cache architectures can maintain large tables of retained query results.

- **Extendability.**

  We want to add additional modules without rebuilding the system. Such modules may include both *query optimizers* and *storage managers*.

3.3. Components

This section describes the components necessary to create an OLAP system and interface it with a data warehouse. The data-flow between these components is visualized in Figure 3.1: a **query tool** injects queries into the system, which first pass through a **query optimizer**. These optimized queries are then transformed into an execution plan within a **query evaluation module**, which is executed using the **cache data** and the **data warehouse**. The data retrieved from these data sources is then reassembled, cached, and returned to the query tool. A **meta-**

data repository supplies information about the structure of the data, on how to access the data warehouse and regarding the contents of the cache.

3.3.1. Query Optimizer

Every query first passes through the *query optimization* module where a set of constraints is mapped to a set of optimized constraints. The only restriction on the optimization procedure is a trivial one: Both the unoptimized and the optimized query lead to the same subset of the data cube.

Actually the query optimizer is not a single component in our system, but an interface to multiple concurrently existing *optimization rules*. These are tiny modules transforming the constraints. The constraints are piped through all of them and they act as filters making it simple to evolve optimization strategies by chaining different of these modules together. For example, it is possible to start with a simple optimizer, which eliminates redundant constraints and detects range queries. When the data warehouse evolves and optimization becomes necessary, more sophisticated query rewriting can be added. Such sophisticated query rewriting strategies may exploit the parallelism found in the queries or use the knowledge gained from analyzing the query histories of users.

The current implementation of the query optimizer consists of a small, but efficient core for query optimization, mainly removing redundancies. If multiple constraints are detected for the same key-attribute, the intersection of the permissible intervals of values is retained. For example, if one constraint is `1990` $\leq$ `YEAR` $\leq$ `1998` and a second constraint demands `YEAR in {1998, 1999}`, then these two constraints will be replaced with a single constraint `YEAR equals 1998`. If the permissible intervals are all disjunct, query evaluation can abort immediately after query optimization, because no values can satisfy these constraints. This simple optimization strategy already results in significantly speed improvement. A second important optimization strategy already implemented is the detection of range queries. Because it is usually faster to evaluate a range query, such as `1995` $\leq$

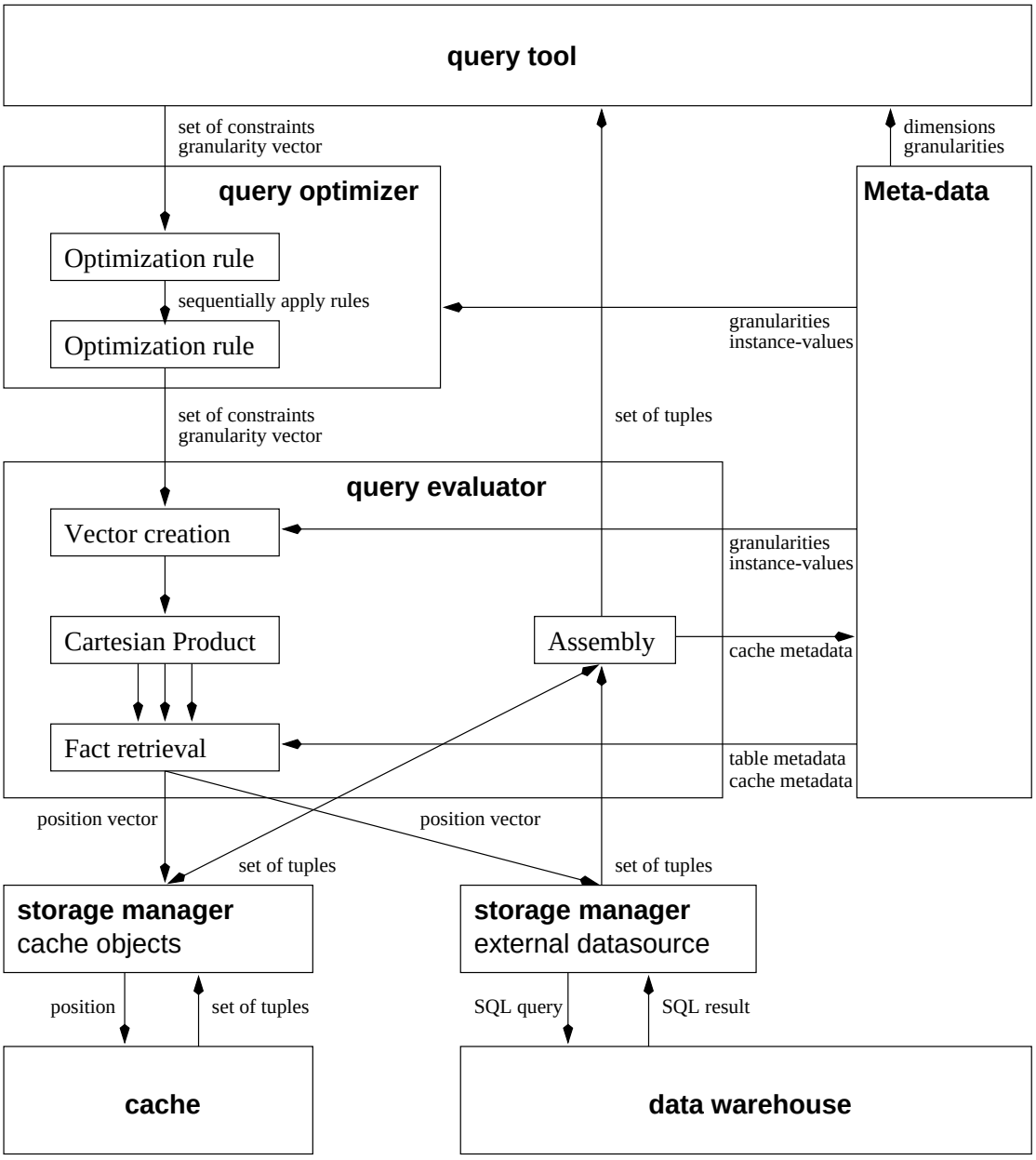

Figure 3.1.: The components of the OLAP system and their communication paths.

`YEAR` $\leq$ `1998` than one based on enumeration of valid choices, special care is taken to fold enumerations into ranges.

3.3.2. Query Evaluator

The *query evaluator* module is responsible for evaluating queries, i.e. for creating position vectors from the constraints and the meta-data, which can be used as input to the storage managers. For every query, a new instance of this module can be instantiated. These modules may run on different machines improving the overall scalability of the system as queries can be processed in parallel.

A number of algorithms offer themselves for query evaluation. We use an algorithm, which closely follows the model described in Chapter 2. A set of permissible instances is initialized with the $\top$-element. Then the contents of this set are replaced with the successors of the elements within the set which satisfy all given constraints. This expansion process is repeated, until only elements at the level of detail which was specified by the relevant component of the granularity vector remain. The Cartesian product over the permissible instances of all dimensions is the set of position vectors. The generated position vectors are passed on to the storage managers. Which storage manager is to be used, is determined using the meta-data repository. However, this set of position vectors does not only contain vectors which refer to unique spatial locations, but may also contain elements that denote entire rows or other subsets of the data space. This preserves the possibility of using efficient access methods for larger areas (e.g. range-queries of SQL-databases). The storage managers are called upon to retrieve the tuples from their physical storage. The results from this retrieval process are then assembles into a data cube again. For the purpose of returning and caching data cubes, a list-based representation is used. The assembly process produces such a list from the responses received from the individual storage managers. If the query result is cacheable, a cache object is created and a reference to it is stored in the meta-data repository.

For example: we can assume a query which requires a granularity of `MONTH` in

the time dimension and has a constraint "`YEAR equals 1999`". During expansion, we first replace the $\top$-element with the parts it consists of (which will have the type `DECADE`). In our case that is just `199x` (the value of `DECADE` is unconstrained). Now we iterate this replacement process, first yielding a set of `YEAR`s and finally the requested `MONTH`s. This set will contain `1999` and finally `May 1999` and `June 1999`, respectively. If a SQL-based data source were used, a query of the form `SELECT * WHERE year=1999 and (month=''May'' or month=''June'');` would be generated by the database wrapper.

The query evaluation is a critical component for an efficient OLAP system. It has to perform least-cost decomposition of queries: for example, if all the months of a year are cached and the data for the entire year is requested, the aggregated value for the year will not be retrieved from the data warehouse, if calculating it from the cached data incurs less overhead.

3.3.3. Fact retrieval and Cache access

The OLAP engine needs to access data from various data sources like relational data warehouses, multidimensional data warehouses and the local cache. We create wrapper modules to access all these different data sources in a uniform way by serving as *storage managers*. These provide multidimensional abstraction to the actual data source used. A request to retrieve data is always expressed as a position vector and the result is always encoded as a set of tuples. This position vector denotes the spatial location of the requested value within the data cube. This data cube is virtual structure and may be stored in a number of different physical storage structure. A storage manager transforms the position vector into a query which is meaningful to the actual storage method used (e.g., a SQL query). The mapping between regions of the data cube and storage managers is described within the meta-data repository. Additional information stored there may include communication bandwidths, access latencies and related data. From this information, the storage managers can be ordered with reagard to the relative cost of

accessing data stored in each. This allows it to prioritize the search for a data entry, if multiple (possible) storage locations exist. This meachnism is used to provide a tie-in for the in-memory cache of each computational node. This cache is always marked as the cheapest method of retrieving a data item. Only, if it is not found there, other storage managers are considered. The in-memory cache uses a least-recently-used replacement strategy and stores its entries in an associative list, which associates position vectors with lists of tuples. This list is physically realized through a B*-tree. At this time, the only other form of storage supported are SQL-databases. To access those, SQL-queries are generated from the position vectors and the meta-data.

Using a uniform interface to access both internal (i.e., cached) as well as external data simplifies the implementation of the query evaluator considerably and helps to isolate the system-specific code in these wrapper modules. Caching is an integral part of our architecture as we cache all query results to exploit the temporal locality of reference in OLAP applications during user-interaction. At the same time we reduce the overhead involved in converting position vectors to queries to external databases or re-aggregating cubes. We represent each cached cube by a separate *cache object*. Basically, such an object is simply another storage manager. It wraps around a previous query result and presents itself to the query evaluator according to the generic interface.

Currently, caches are local to every workstation. Different strategies to advertise the cached data globally across multiple machines to build hierarchies of cached data cubes from both local and remote cache entries are under consideration for future versions.

3.3.4. Meta-data repository

The *meta-data repository* is a catalog of information, describing the logical structure of the data warehouse and their interaction with the data base(s) storing it. For relational data warehouses, it basically contains a description of the scheme

of each relation and information on how to map the different data granularities to tables in the data warehouse. Apart from other meta-data, we keep information on the currently cached data cubes together with their location in the meta-data storage. Basically, the meta-data repository maintains a key-value table which provides read-only access for all modules and write access for the query evaluator only to register newly created cache entries. Almost all phases of the query evaluation—and consequently, all components of the OLAP system—depend on data from this repository (compare Figure 3.1).

For the implementation of distributed data warehouses, replicated meta-data repositories reduce the communication overhead—and particularly the communication latencies—considerably. However, the fact that a fair amount of cache-related meta-data is to be stored, complicates the replication strategy. While read-only replicas guarantee a high performance, caching is only possible local to each computational node, because the cache entries can not be advertised globally. A distributed and/or hierarchical cache architecture requires the presence of synchronized cache information.

3.3.5. Query tool

The query evaluator returns the query result as a set of tuples to the query tool which provides the user interface and constructs queries from user interaction. The user-interface is primarily concerned with the presentation of the data received from the OLAP engine. It performs the necessary reduction in dimensionality to represent the data-cube on a 2-dimensional viewing device. Navigation and data exploration need the support of an appropriate data presentation, which provides a point-and-click interface to perform the common OLAP operations (drill-down, roll-up and pivoting). The data displayed represents the current state of the OLAP system (i.e., the constraints and granularity) and, for that reason, forms the base for new queries, resulting in the (typically) high locality of data references in subsequent OLAP operations. In order to assemble ad-hoc queries, a communication

path to the meta-data repository is necessary. Information regarding the dimensions, granularities and hierarchies of dimension-values is necessary to construct meaningful queries.

The queries are transformed into a set of constraints and a granularity vector according to the following rules:

- **Drill-down.** A component of the granularity vector is replaced with one of its successors. A constraint may be added at the same time to reduce the visible domain.

  Assume, the user currently views data at the `DECADE` level and notices an anomaly within the aggregated data for the decade `199x`. To investigate further, a drill-down occurs to the next more detailed granularity (`YEAR`). To evaluate this query, the `time`-component of the granularity vector, which initially points to `DECADE`, is replaced with `YEAR`. Now imagine, the user sees that an anomaly is caused by some event in the year `1998`. To track the problem down, only the months of `1998` are required. The granularity will be adjusted to `MONTH` and an additional constraint `YEAR equals 1998` is introduced. A slight variation on this theme is the unconstrained drill-down: in that case, no additional constraints are added and only the granularity vector is modified. The user may choose to display the data for all `JUNE`s of a given `DECADE`. Such a query may start at the `DECADE` granularity, with a constraint on the value of the `DECADE` only. When this drill-down is issued, `JUNE` is added to the set of constraints and the relevant component of the granularity vector is set to `MONTH`. As a result of this operation, the values for `DECADE` and `MONTH` are constrained, while `YEAR` is unconstrained.

- **Roll-up.** All constraints on the current granularity in the dimension, where the roll-up occurs, are deleted. The component of the granularity vector corresponding to this dimension is replaced with one of its predecessor according to a traversal history or default traversal paths. A roll-up from `YEAR` to

`DECADE` clears all constraints set for `YEAR` and sets the `time`-component in the granularity vector to `DECADE`.

- **Pivot.** One dimension is replaced with another. Neither the granularity vector, nor the set of constraints are modified. The pivoting operation affects solely the way data is presented to the user requiring no reevaluation of the query.

So far, the presentation module has not been implemented in its final, graphical form. Instead, the current system prototype echoes the query results to the console and receives its queries from a command line interface.

3.4. Parallelism

The dimensions of a data cube are independent in OLAP. Since all data is indexed through these, they perform a similar function as indices on tables in relational databases. However, no primary index exists. Instead, all indices have an equal priority. As a consequence, they do not impose any additional constraints on the ordering of the query evaluation, but allow for a concurrent application of the query evaluation algorithm in each dimension. Surprisingly, this comes very close to the original promise of equal access in databases, that was made by relational databases, but never came true [30]. This type of parallelism takes place between multiple instances of the query evaluation algorithm that run concurrently. Therefore it is *external* to the query evaluation algorithm.

Additionally, the query evaluation algorithm can be parallelized as well. The model proposed in Chapter 2 is optimized for this: it uses sets as its main data type and iterates term substitution and the application of constraints to sets. Sets are a data type, that can be parallelized automatically and efficiently on shared memory systems [13]. If set partitioning is used, the term substitution process and applying the constraint can be sped up considerably, as well.

Apparently, during query optimization two disjunct types of intra-query parallelism can be identified and exploited, which each stem from distinct sources.

1. **external parallelism.** The query evaluation algorithm can run concurrently for each dimension. No communication between these threads of execution are necessary up to the point where the Cartesian product is calculated or another combination operator is applied. From our results, a shared-nothing approach suffices for a reasonable speedup, assuming the meta data is replicated.

2. **internal parallelism.** Within each instance of the query evaluation algorithm, parallelism may be exploited. In the case of the algorithm used in our implementation, the set expansion and constraint application can be parallelized. At every expand operation the active set can be split into subsets and the algorithm can be performed on those in parallel. This results in a tree-like branching of the algorithm. At the end, the union of these partial result sets has to be calculated to produce the result set. To exploit this type of parallelism a shared-nothing architecture suffices again, assuming that the communication architecture used provides sufficiently low latencies. The performance also depends largely on how much time is consumed for the encoding, transfer and decoding of the intermediate results. The constraints can also be applied in parallel on a shared memory machine either by choosing an appropriate data structure or by creating multiple intermediate sets and intersecting those.

Furthermore, multiple queries to the same data warehouse are also independent. For this reason they can be executed in parallel, adding inter-query parallelism. In sum, OLAP lends itself to parallelization very well. For this reason, a component-based approach—such as the one presented here—does not only improve the maintainability and modularity of the resulting system, but also allows for the transparent distributing of subtasks across a cluster of computational nodes.

This approach is a radical departure from the current methods of parallelizing OLAP queries. Although Gimbel *et al.* [23], use an object-oriented approach based on Java to parallelise OLAP queries, they do not provide a fully functional OLAP kernel. All queries are passed directly to multiple relational databases. Only pipelining and parallel joining is supported. The design thus follows closely what is known from parallel data base systems. The worst problem with the chosen approach is the absence of any cache to hold intermediate results and aggregated values. Sanjoy Goil describes the solution to a completely different problem of OLAP: he details the parallel integration of cubes and subcubes in [24]. The presented system uses partitioning to distributed the evaluation of an aggregation function over multiple processors.

3.5. Summary

This chapter briefly presented an architecture for modular OLAP systems with parallel and distributed query evaluation using CORBA objects. Multiple queries can be distributed across multiple processors/workstations by dynamically starting additional query evaluation modules. Within the query evaluator both the constraint-application and the fact retrieval support can be implemented to exploit hardware parallelism. The constraints may applied concurrently to each dimension (i.e., external parallelism) and the constraint application within a dimension can be parallelized and the fact retrieval may be also carried out in parallel (i.e., internal parallelism). As a result, the architecture presented is well suited for distributed and parallel query optimization. It reflects the formal model of evaluating OLAP queries outlined in Chapter 2. The fact that this model is based on sets, which allow for the use of efficient and parallel algorithms, increases the parallelism considerably.

The modules, that have been defined, all adhere to very simple interfaces. Only four simple data structures are used during inter-module communication:

1. sets of (position) vectors,
2. sets of constraints,
3. granularity vector,
4. sets of tuples.

The use of CORBA further simplifies the development of components. It provides a platform and language independent communication path between the various components.

The subject of modularizing OLAP systems and defining common inter-component interfaces is highly pertaining, because the definition of a *standardized architecture* would allow for the component-based development of open, modular and inter-operable systems. The next generation of OLAP systems will certainly be required to have a similar structure to what is proposed here, in order to support a wide-spread adoption of the technology and better integration with the highly-specialized data mining and knowledge discovery technologies available today.

4. Extending the scope of OLAP

"These are wild dreams."

Mary Shelley, "The Last Man"

4.1. Overview

Although the currently implemented system prototype offers only the basic functionality necessary in OLAP systems, plans for future developments exist already. In this chapter, some of the major directions considered for future inclusion are discussed. These planned improvements range from techniques to provide higher performance, improve the parallel query evaluation, security and data quality to self-healing databases and On-line Analytical Processing applications in control systems. Instead on just improving upon the performance or user-interface, the proposed additions, extensions and optimizations attempt to expand the scope and applicability of OLAP to a wide number of domains besides the analysis of traditional business-related information. Areas of particular interest are long running, time-critical decision taking systems for automated control and adaptive, financial planning and control applications offering prediction and adaptive, flexible response to rapidly changing global markets. These applications are outside of the current application domains of OLAP systems which are mostly used in business information systems to reason about strategic decisions and marketing. These tra-

ditional applications are neither time critical, nor automated. Understandly, these are two areas were OLAP engines lack the most. In addition, security concerns have been largely ignored during the last years of development. A simple all-or-nothing access policy was used, which interferes which the multi-level hierarchies present in modern organizations. The extensions described here hint to possible solutions for the problems. It is planned to implement and evaluate these in the future versions of the system described in the previous chapters.

4.2. High-Performance OLAP

Today, scientific experiments generate enormous amounts of data, often well in the multi-terabyte range. The COMPASS experiment [18] will generate an object set of about 20 terabytes. The NA48 experiment at the CERN already records data at a sustained rate of 20MB/sec [41]. Some scientists [39] already warn, that bus bandwidths are to become the next bottleneck in data storage systems. However, parallel I/O techniques, such as the parallel disk model [20, 49], are successful in moving the data in time. Nonetheless, unsolved problems remain for the management of large scientific data sets regarding the interoperability of the data repositories. No agreed specifications of interfaces to the data sets exist, the structure of the data is not stored independently and the meaningful analysis and exploration of simulation results is difficult.

The fact that information without appropriate data management and intuitive navigation is worthless has already been recognized in the business world. On-Line Analytical Processing (OLAP) [1, 14, 15, 17, 45, 50] and multi-dimensional data warehouses [2, 22, 28, 30, 31, 36, 51] have proven valuable technologies for the efficient processing of large sets of business data. On-line Analytical Processing operates within a multi-dimensional model of the source data. This model is granular, i.e., it allows to view the data at different granularities, where coarser views are synthetisized from finer data. Using such aggregates, it becomes possible to

efficiently detect unusual, interesting features in very large data sets. Data mining algorithms, for example, search for patterns in the coarse-grained data first and access details only where necessary. This mirrors the requirements of modern, scientific computing which already generates and analyses data which approaches the size limits for current hardware. An additional benefit of using OLAP for very large scientific data sets is the availability of meta data. Such meta data can not only structure the stored data, but also communicate background information on experiments to further increase the persistent value of a data cube.

On-line Analytical Processing offers **intuitive** data analysis with **interactive response times** in a **scalable** system. Furthermore the data is **inter-operable** between different analysis tools, because most structural information is stored separately in the **meta data**. Nonetheless, these techniques are not yet applicable to data outside of the business domain, and particularly grand-challenge type problems, for a number of reasons: today's implementations support only simple, distributive aggregation functions. The data cube is either pre-materialized in a central location or the query execution suffers from performance problems; data cubes are read-only and trigger-conditions are unsupported. Additionally, the multi-dimensional paradigm leads to problems regarding the efficient storage of the sparse data cubes and efficient processing of the data: Clustering and partitioning need to be employed. Still, the efficient implementation and processing of the multi-dimensional data space remains one of the most challenging research problems, as modern computer architecture is inherently contradicting such models.

High performance OLAP systems capable of handling the enormous data sets analyzed in scientific computing pose special requirements and require innovative solutions:

- Parallelization and distribution.

  Modern applications often operate on very large data sets and generate their output distributed across a cluster of computational and storage nodes. This trend was accelerated during the last few years because of the improved per-

formance of relatively cheap workstations [39, 44] and the increased availability of middle-ware [40] and software to build (inhomogeneous) clusters from commodity hardware [3, 4, 8, 26]. This changed situation requires a paradigm shift for On-Line Analytical Systems: Instead of materializing a data cube in a central location and/or replicating it on slave-servers, a *virtual* data cube, which may span a large number of storage nodes, needs to be constructed.

- Distributed, hierarchical caches.

  Distributed and hierarchical caches permit cache objects to be created local to the computational nodes where queries are evaluated, but advertise the cached data globally. If these cache objects are reused and built into a tree, whole hierarchies of cached data can be created. These hierarchies will reflect the decomposition of a data cube into consecutively smaller sub-cubes.

- Least cost decomposition for queries.

  Least cost decomposition of queries has to take multiple aspects of query evaluation into account. The data availability in an environment with local and remote caches as well as local and remote data storage facilities has pronounced effects on the latencies for data access. That is, in some cases an aggregated value can be recalculated faster from cached data than fetched from a remote data warehouse. At the same time, the cache contents and the structure of the data warehouse are important (e.g., if all the months of a year are cached and the data for the entire year is requested, the query evaluator should not retrieve this aggregated value from the data warehouse, if calculating it from the cached data would incur less overhead). Graph coloring and annotating the granularity hierarchy with information regarding the relative cost of different decompositions is necessary to provide this functionality.

4.3. Volatile data and writable data warehouses

A wide array of applications will not work with the write-once, read-many-times semantics of OLAP systems. In certain situations it is necessary to update old values either to correct a wrong entry or to update a fuzzy value with a better approximation. While the first case can occur in any application domain, the latter one will be usually found for problems requiring an iterative approximation of measures. Various optimization problems in economics belong into this category. Clearly, data that can be updated after it is initially stored in the data warehouse can be handled either by re-feeding the entire data warehouse or by treating the fact as volatile and modifying the algorithms for aggregate calculating, caching and query evaluation accordingly. Support for such volatile data is the most radical deviation from the usual read-only OLAP systems. Supporting a possibly changing fact base makes eager view materialization strategies infeasible. However, traditionally OLAP systems depended on massive pre-materialization to provide interactive query results. Even worse, concurrent updates and queries can lead to inconsistent query results. The reproducability of query results also suffers from this addition. To permit both the modification of the underlying database and the still guarantee reproducable query results, a versioning system can be used: By time-stamped every change, queries can be evaluated in any temporal context required.

Support for volatile data permits the monitoring and analysis of processes during their run-time. Three kinds of volatility, which offer different chances for optimizations, may occur in the data source:

- **Fully volatile.** Any data may change at any time and most optimizations are infeasible. Since caching and pre-aggregation are futile, a fully volatile data warehouse will can operate on top of any relational database system. All queries have to be re-computed from the database. The performance penalty of such a brute-force solution prohibts interactive querying of large databases.

- **Append-only.** New data can be appended at any time. However, the aggregated values to be updated can be pre-determined. That is, most of the cube is stable while the cube grows as new data is added continuously. Incremental feeding can be used in such cases. Data warehouses that use incremental feeding use this approach. A number of algorithms to realize the feeding and updateing of the aggregates exists [19].

- **Scattered.** Certain sections of the data space may change without notice. These sections can be non-continuous and scattered. However, it is known in advance which sections can change. In this case, most of the cube can be pre-materialized.

To negotiate the particular challenges of volatile data cubes—and particularly fully volatile cubes—a number of different strategies is available:

- **Lazy view materialization.** Views are materialized on demand. Interactive response times are achieved using parallelization and distribution. No upper limit for the response time can be given. Caching is used to exploit the temporal locality in accesses to atomic facts in consecutive OLAP queries.

- **Bottom-up update propagation.** This technique propagates updates to the fact base bottom-up; i.e., the materialized views are notified of a change in the underlying fact base. Whenever such an update happens, a view depending on the updated fact either needs to be updated or purged from the cache and recalculated when necessary. The most important benefit of such an incremental update procedure consists in the knowledge of the change that occurred. Often the aggregated value can be adjusted quickly using this additional knowledge. This is the case if *stateful aggregation functions* are used. A stateful aggregation function contains an internal state such that it can be updated when notified of a change. Examples of such stateful aggregation functions are summations, products and counts. The calculation of an

average is more complicated: In order to express it statefully, both the sum and the count need to be retained. When an update occurs these are updated accordingly and an average can then be calculated from these updated values.

Bottom-up propagation of updates is also a requirement for the introduction of active data base elements, such as rules and triggers. If a rule is evaluated on a particular view, this view has to remain current with all updates to the associated raw data. Bottom-up update propagation assures this behavior.

- **Approximation of aggregate values.** It remains to be evaluated, whether approximations of the aggregated value may be used to improve response times. The topic of approximating aggregated values has received some interest lately [49], but it is unclear whether it is useful in real-world applications.

Furthermore, an arbitration mechanism is required to maintain consistent queries during updates. Either a versioning system (in the spirit of the VNC algorithm, which was originally designed to retain an operational system while re-feeding the warehouse, introduced by Quass *et al.* in [42]) or a fact-based locking with multiple readers can be implemented. Either way, light-weight transactions are introduced into the OLAP-world.

4.4. Rules and Triggers

A natural extension of decision support systems consists in the ability to take decisions in accord with preset rules or goals. Whenever a condition specified in those rules is detected, an appropriate action is taken.

The Event–Condition–Action rule model is mostly used in active databases. Whenever a certain action (e.g., an insertion) is executed on the fact-base, a condition is evaluated. In relational database systems, such a condition usually contains a query and a test on the result of this query. If this condition evaluates

to true, the specified action is triggered. This action may in return cause events, condition tests and actions to be taken. OLAP systems differ slightly from this: At the higher levels (i.e., the levels containing aggregated data), only update operations remain—whenever the fact-base is changed, any modifications propagate up the aggregation hierarchy, where they become visible as update events to the aggregated data. The number of paths, which will be followed—assuming, that paths which contain no views with triggers are abandoned—, is less or equal to the number of active rules defined for the data cube. The maximum cost of following an aggregation path is a linear function of the depth of the aggregation hierarchy and the cost for updating the affected views. As a consequence, it is possible to determine the maximum amount of rules that may exist at one time and still obtain a given response time to all events. This makes it feasible to decide during the insertion of a rule, whether it can still be evaluated within a predefined latency or whether the rule should be rejected for violating a timeliness property.

Adding priorities to rules requires only a minor modification to the updating algorithm: During rule creation, the rule priorities have to be propagated downward through the view hierarchy. During rule evaluation, this information is now available at every branch (i.e., when a view is updated, which is part of multiple view is with attached rules) and the possible paths are followed in the order corresponding of their priorities. In this context, different priorities for rules are equivalent to different qualities of service. Higher priorities indicate a request for a better quality of service. In case of extremely high system loads, lower priority rules are facing a higher risk not to be evaluated after an update event.

To date, triggers are not directly supported in OLAP systems. Current solutions revert to defining rules and triggers for underlying relational systems in ROLAP. Although the functionality is equivalent for such a solution, it appears preferable to introduce triggers within the multi-dimensional model of the OLAP engine itself. The direct benefits are those mentioned above, i.e. the possibility to prioritize rules and use aggregated data directly in trigger conditions. This simplifies the

programming model for triggers and also provides a convenient way to implement an on-demand updating of certain sections of the data cube, if a semi-eager materialization strategy is chosen (i.e. a materialization strategy which keeps parts of the data cube materialized, while recalculating others).

No implementation of the suggested trigger mechanism exists so far. However, a related mechanism is used in a prototype developed at the Technical Research Center of Finland. Kiviniemi *et. al* [32] report that they use lazy aggregation technique to optimize OLAP for real-time applications. Their approach is based on the evaluation of a consistency criterion, which is reevaluated after each update event to decide whether certain aggregates should be recalculated.

4.5. OLAP for Control Systems

Interactivity and flexible, human-readable reporting are usually seen as defining components for on-line analytical processing. This requirement stems from the strong foundations of OLAP as a decision support tool. In contrast, most control applications do not offer and can not use a point-and-click interface to the data analysis tools during most of their operation. Still, because the point–and–click user interface known from OLAP can function both as a programming interface (to define rules and actions) and a diagnostic tool (to view and analyze the data base) for a control system, even the interactivity requirement of OLAP is satisfied.

Active OLAP systems result from the integration of analytical and decision support capabilities with active rule elements, which initiate actions whenever the fact-base satisfies pre-determined conditions (compare Section 4.4). In effect, active OLAP provides a very powerful mechanism for automated decision-making in control systems.

One particularly powerful aspect of a control system based on active OLAP, is the programming model. It separates the aspects of designing an automated control system into different tiers (which coincide with the task performed in traditional

control systems):

- **Data acquisition.** At the lowest level, data collected from sensors is mapped into values of a database system—the semantics of the sensor readings is defined and data conversions are performed.
- **Data analysis.** Next, an intermediate layer extracts analytical data and synthetises a system state from the available fact-base. This state is represented as values in one or multiple views of the data-cube. The design of this layer involves the usual tasks in OLAP design of selecting appropriate views and the providing consolidation functions.
- **Control.** At the highest level of the system, conditions in the database are mapped back to actions according to predetermined rules. These rules provide the control logic and form the foundation of the decision taking process.

The above aspects of a control system can also be found in traditional implementations of such systems. However, in contrast to the clear separation of responsibilities present in an active database system, the classic approach to the design of a control systems keeps those three system components interwoven. By separating these three aspects of designing and implementing automated control applications, both the maintainability and the re-usability of the resulting software system is improved. All communication between the components is carried out through the OLAP system, that chains them together and encapsulates the control logic in its meta-data and rule-set. Another major improvement of OLAP-based control systems over traditional systems is the historization concept inherent to OLAP: New information is appended to the data-base and data analysis is always done in a temporal context on a subset of the available (historic) data. Paired with the trigger mechanism and an updated fact-base, powerful adaptive control systems emerge. As a result, OLAP can become both the programming paradigm and the programming environment for data analysis and control systems. During programming, testing and diagnosis, the point-and-click interface for the data analysis and

decision support functions of an OLAP system can provide the user-interface to data-intensive control applications. It permits customization of real-time applications from a reusable foundation by using the meta-data to specify the data acquisition, data conversion, data analysis and the rule–action patterns for a given problem domain.

4.6. Security Considerations

Basically, information security in both relational and multi-dimensional data bases includes the same three aspects:

1. *Secrecy* is the protection of disclosure of information.
2. *Integrity* is the prevention and detection of improper modification of information.
3. *Availability* is the prevention of improper denial of services.

For OLAP applications, the secrecy aspect is most important and this section focuses on it. This can be easily justified: The integrity of the data can be assured easily using the same mechanisms, as for relational data bases. The same is true for availability. Secrecy is tightly coupled with the application environment of the data base and with the internal policies of the organization. As such, the organization structure of an organization needs to be modeled to provide a non-intrusive yet powerful security model.

The various combinations of access necessary for the proper functioning of a database system are often characterized as the roles known to the data base system. At the most basic level, this would differentiate between the administrator and users. In reality, usually far more roles are necessary, depending on the data to be stored, often even different positions within the same project group have different access permissions. Instead of granting the same permissions for the entire data base, access rights for distinct entities stored within the data base may differ.

The access to the basic facts stored within the data base is determined by the policy data stored with it. This policy data consists of permissions, which have been made explicit (either directly specified or indirectly). However, for OLAP applications, access control based solely on basic facts will not provide the necessary features. Suppose a multi-dimensional database system employed in an organization with multiple branch offices: In such an environment, employees of a branch office may have access to the data indicating their own performance (e.g., sales figures), but not to the data of other branches. At the same time, the aggregated data will be less sensitive (e.g., the average/total sales for a region). In contrast to most relational database systems security in data warehouses cannot depend on access information indicating whether access to a certain table may be granted, but rather closely follows the semantics of security in some object-oriented data base systems, which implement a per-object security property.

However, introducing security using a scheme different from *all-or-nothing* has one major drawback: the access permissions of the user executing the query become an integral part of the query's constraints. Formally, the query results for the same query q for a user i and a user j *will* differ $result(query, i) \neq result(query, j)$, if the permissions for these users differ $perm(i) \neq perm(j)$. Any model will need to take this anomaly into account and treat a user's permissions as an implicit constraint.

Apparently the integration of more sophisticated security policy than *all-or-nothing* will consume a considerable amount of storage space and introduce a performance penalty. Why would any sane data base designer consider this then? The answer is simple: Data on business processes is sensitive, but needs to be shared within an organization. Particularly, data in decision support systems, which allows the deduction of strategic decisions, has to be protected from spying eyes. On the other hand, locking this data down will not benefit any company, either. A number of analysts and managers will need continuous access to *some* of this data (more specifically, the data concerning their area of work), to determine their current performance and help them plan ahead. A sophisticated security policy would

increase the value of any commercially available OLAP solution and would be far close to the requirements of business users than currently available systems. A number of requirements, governing how to implement security for data warehouses in such a way that it provides a solution well-suited to the user's needs and is as non-intrusive as possible, can be established:

Easy mapping. To introduce an effective security policy, both users and data base administrators have to understand and accept it. To simplify this, it is paramount to provide an easy mapping from the organisatorical hierarchy to the internal model of security permissions. The organizational structure will usually be well understood and should provide a foundation for data base security. It has to be possible to express the organizational structures found in most organizations within the security model. For example, the following rule-set may apply:

1. *Upper management* has access to all data.
2. *Middle management* has access to the data pertinent to their work.
3. *Departments* are a logical grouping and decouple access patterns. Users from one department will usually not have access to all the data from other departments.
4. *Project groups* are assembled ad-hoc and have access to parts of the information from various departments.
5. *An employee* may be assigned to multiple project groups. In addition different employees within the same department may have different levels of access to the departments data.

Such organizational patterns make it necessary to provide a very fine-grained access control, which allows the grouping of arbitrary subsets of the subjects and objects. The resulting structures are similar to the hierarchies of granularities and dimension-values which are discussed in Chapter 2.

Fine-grained. The unit of protection must be the fact itself. Different entries at the same level of aggregation may be accessed by different user groups.

A single data-cube. A few security models have been proposed, which would require to materialize a separate cube for every set of access permissions. [29] This is problematic, if given a large amount of facts (requiring large cubes) or a large number number of different permission sets (requiring a large number of cubes). In typical OLAP systems, both of these cases will coincide, rendering this solution infeasible.

In addition to reducing the storage size, a single cube simplifies the maintenance and updating of the data warehouse. In addition it will have one large unified cache, which will accelerate the query performance considerably compared to the large number of separate caches for the other solution.

Compact storage. Security should not increase the required storage space unreasonably. The overhead added to the size of the facts is a critical factor, as a large number of facts will exist (many orders of magnitude more, than entries in the meta data repository).

Fast evaluation. The adoption of a security policy should not slow down the performance of the OLAP system unreasonably.

Economy of mechanisms. To guarantee simplicity, efficiency and maintainability, no new mechanisms should be introduced. Any new mechanism could limit the performance and add another possible bottleneck to the system. Instead, the core functionality of a multi-dimensional database system, which certainly will be well-optimized, should be extended to enforce the security policy. This will simplify other aspects of the design, such as caching, as well.

We propose a token based approach to the modeling of security policies, which manages those tokens in a dedicated dimension with some support from the meta

data repository. In this solution, tokens are used to determine whether a given is permitted to view a cetain entry. In order to realize that, all entries are distributed along an artificial security dimension. The domain of values in that dimension are the above-mentioned tokens. A user is given access to a fact, only if he is in possession of a matching token. As a result tokens act as a lock to each database entry and access is granted only if a user has a corresponding key in his keychain. The keychain in turn, is represented using a tree-based capability model, which encodes the organizational structure and security policies of an organization. In this model, each user is made up of a hierarchy of access tokens. It is thus possible to use the same model techniques as for dimension-values and granularities.

This combines the benefit of a fixed size token with the flexibility to model organizational structures, such as the one outlined above. Every fact within the data base will contain a field indicating the necessary token to access it. However, there will not be a one-to-one relationship between the tokens and the facts; in reality, facts with the same semantics will share the same access token. A typical case of facts with the shared permissions (and consequently same tokens) is the information accounting information gathered within the branch office.

The use of a dedicated dimension to store and evaluate the security information has a number of advantages: the security policy is enforced within the OLAP server, which improves the overall system security. Nonetheless, some client co-operation is necessary, as the client needs to pass an implicit constraint, which specifies the current user, to the server. In order to ensure security during this process, an authentication procedure needs to be used during the connection to the OLAP server. This authentication may set this implicit constraint for the opened connection, which would ensure safety even during the communication with untrusted clients. Such an approach should guarantee the secrecy of the stored data.

Another benefit of using a dimension to store the access permissions becomes evident immediately, when modeling the security policy of any hierarchical organization: While the finest granularity of the dimension will contain the access token

necessary to access the facts stored within the data base, the coarser granularities will model how the access rights associated with these tokens are mapped to the project groups, departments and employees.

To determine the access rights to an aggregated value, at least three distinct possibilities exist:

- **Minimum or Intersection.** Only users with access to all details are allowed to view the aggregate. This is a conservative approach, which is certain to evade all tracking attacks. However, it is a rather bad choice as a default since it severely restricts the information flow.

- **Maximum or Union.** Any user with access to one of the details is given full access to the aggregate. Although this constitutes an optimistic approach, it will mostly match the policies implemented in organizations.

- **Explicit.** The meta data contains an explicit access policy. This may be used to give any user full access to an aggregate. Whenever access policies outside of the ones described above are necessary—for example, if data from a certain aggregation level on should be public—, an explicit specification of the access rules becomes necessary.

4.7. Data Quality, Pruning and Data Reconstruction

One of the primary concerns in decision support is the quality of the data that decisions are based on. It can be best defined as the fitness for use [48], which already implies that the concept of data quality is relative. For this reason, the data quality appropriate for one use may be insufficient for another use. Since data warehouses increased the trend towards multiple uses of the same data, a portable method to describe data quality has to be specified. The representation of data quality is

straight-forward: Usually a percentage value describes the reliability of a fact. Such percentages can be mapped to fuzzy categories such as "credible", "improbable", *etc.* However, processing the quality-annotated data leads to a number of interesting questions when aggregating the data such as "Should only sufficiently reliable data be aggregated?" and "How does one derive a measure of reliability for an aggregated value from the atomic facts?" These questions have not been sufficiently answered, yet. However, as soon as these problems are overcome, then decision support will be lifted to an entirely new level: Analysts will be able to require the system to give responses that exhibit an arbitrary, user-defined probability. An appropriate representation of data quality permits the storage of predicted values with the normal data. Predicted facts can be annotated with a probability value.

Aside from storing information on the quality of data, another interesting application concerned with the quality and precision of data is the pruning of the fact base. For processes which continuously record new data at a high rate, such as intra-day stock prices, it may not prove practical to store all available facts at their finest granularity. Although recent events need to remain accessible in detail, older information may not be required in such detail. If this is true for a given application, detailed values need to be retained beyond a pre-set data lifespan. After that, only aggregated summary data remains stored within the data base and the detailed data is *archived*. If it should become necessary, aggregated data and neighboring facts may be used to approximate facts that where removed, if retrieving the raw data would take too long. This process involves the deduction of a plausible value from the remaining data base. A variety of different approaches to this reconstruction are available:

- **Interpolation.** The missing data may be interpolated from its neighbors.
- **Neutral elements/Null values.** An artificial fact can be generated which contains the neutral element regarding the applicable aggregation function (e.g., the number 0 regarding a summation). For these artificially generated

elements, a reliability (i.e., a data quality) of 0% is to be assumed.

- **Statistical methods.** Additional facts can be generated according to the statistical distribution expected for the data base. This approach should offer the best approximation of the original fact.

- **Approximation functions and compressed storage.** If an approximation function was generated before the facts where pruned from storage [49] (i.e. a lossy compression was applied), the stored function will directly yield an approximation.

Pruning and archiving is an effective method to reduce the size of a data warehouse for long-running data collection and analysis. Recent research [49] shows that data cubes can be approximated using wave-lets and related structures. This constitutes a kind of pruning, as not just the data volume but the precision is reduced as well.

5. Conclusion

"It was not even forgery.
It was merely the substitution of one piece of nonsense for another."
George Orwell, "1984"

On-line Analytical Processing (OLAP) is a relatively new database technology, developed to provide intuitive data analysis with interactive response times even for very large data bases. The main motivation behind its introduction was the fact, that modern transactional databases reach a peak trough-put in excess of one hundred thousand transactions per second while data analysis neither offered the required performance nor were the user-interfaces simple enough to be used effectively by someone unfamiliar with the data base system. In this environment, the point-and-click simplicity of OLAP coupled with a more natural—namely multidimensional—data model quickly established themselves as a vital segment in the data analysis market. Today, the market size for OLAP solutions doubles approximately every year.

However, due to the relative novelty of OLAP, both a widely agreed specification of the features required in such a system and a versatile multidimensional data model are still lacking. A number of different modeling approaches exist, but none of them fulfills the performance promise of OLAP. Particularly, the most widespread method to implement On-line Analytical Processing system, i.e., by

mapping the multidimensional operations to relational data base queries, offers suboptimal performance. In response to this unsatisfactory situation, a novel, set-based data model for OLAP was introduced in this thesis. It is based on sets and designed to provide both an easy translation into source code and simple verification of correctness. The most noteworthy feature of this model is its high amount of parallelism and the clear separation of functional components. In contrast to other models, it was designed with the intent of allowing for a highly parallel and distributed query evaluation.

The maturing of the technology and the recent developments in data warehousing and especially the ongoing attempts to standardize the meta data description languages, are welcome signs that solution vendors finally embrace open and inter-operable solutions. Still, the current systems provide at most a compatibility to multiple databases. While modern software engineering practice defines application specific protocols for inter-operable objects and components, no such project has begun for OLAP. Nonetheless, a component-based approach to building Online Analytical Processing systems offers a number of advantages, such as simpler development, faster deployment and the possibility to combine modules from various vendors as needed. A component-based solution can split the system along its functional boundaries into a query optimizer, query evaluator, data base access, cache modules and meta data repository. These modules are mostly independent and the inter-module communication is simple enough to be based on a common middle-ware technology such as CORBA.

Before OLAP can become such an ubiquitous technology as data base systems, a number of extensions to the basic functionality defined by Codd become necessary. On the data engineering front, data quality and security remain the next, big challenges. Technical aspects such as high-performance implementations, the addition of active data base elements (rules and triggers) and support for volatile data (as opposed to write-once, read-many-times) will remain the focus of research for the next years. Only when these basic limitations have been overcome, the use

of OLAP as a programming paradigm for data-intensive control systems can be explored.

OLAP is perceived in different ways by different user communities. All these views have one thing in common: they see it as one of the most important emerging technologies which promises to revolutionize data-intensive computing.

Bibliography

[1] *The OLAP Report.* Business Intelligence, 1999. `http://www.olapreport.com/`.

[2] R. Agrawal, A. Gupta, and S. Sarawagi. Modeling multidimensional databases. Technical report, IBM Almaden Research Center, 1996.

[3] A. Barak, S. Guday, and R. Wheeler. *The MOSIX Distributed Operating System, Load Balancing for UNIX*, volume 672 of *Lecture Notes in Computer Science.* Springer Verlag, 1993.

[4] A. Barak and O. La'adan. The MOSIX Multicomputer Operating System for High Performance Cluster Computing. *Journal of Future Generation Computer Systems*, 13(4–5):361–372, March 1998.

[5] R. Bayer. The Universal B-tree for multidimensional indexing. Technical report, Technische Universität München, November 1996.

[6] R. Bayer. UB-Trees and UB-Cache: A new processing paradigm for database systems. Technical report, Technische Universität München, March 1997.

[7] R. Bayer and V. Markl. The UB-Tree: Performance of multidimensional range queries. Technical report, FORWISS München.

[8] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, and C. V. Packer. Beowulf: A parallel workstation for scientific computation. In *Proceedings International Conference on Parallel Processing*, 1995.

[9] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The R* tree: An efficient and robust access method for points and rectangles. In *Proceedings SIGMOD'90*, pages 322–331, 1990.

[10] S. Berchtold, D. A. Kaim, and H.-P. Kriegel. The X-tree: An index structure for high-dimensional data. In *Proceedings 22th International Conference on Very Large Data Bases*, pages 28–39, 1998.

[11] G. E. Blelloch and M. Reid-Miller. Fast Set Operations Using Treaps. In *Proceedings Tenth Annual ACM Symposium on Parallel Algorithms and Architectures*, 1998.

[12] C. Bontempo and G. Zagelow. The IBM Data Warehouse Architecture. *Communications of the ACM*, 41(9):38–48, September 1998.

[13] L. Böszörmenyi and H. Kosch. High performance sets. In *Proceedings High Performance Computing 98*, pages 972–975, 1998.

[14] S. Chaudhuir and U. Dayal. Data Warehousing and OLAP for Decision Support. In *Proceedings SIGMOD'97*, pages 507–508, 1997.

[15] S. Chaudhuri and U. Dayal. An overview of data Warehousing and OLAP technology. In *SIGMOD Record*, March 1997.

[16] E. F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, June 1970.

[17] E. F. Codd, S. B. Codd, and C. T. Salley. Providing OLAP (On-Line Analytical Processing) to User-Analysts: An IT-Mandate. Technical report, E. F. Codd and Associates, 1993.

[18] Compass Collaboration. The compass proposal, addendum 1, May 1996.

[19] K. Czarnecki, U. W. Eisenecker, and P. Steyaert. Beyond objects: Generative programming. In *Proceedings of the European Conference on Object-Oriented Programming 1997*, 1997.

[20] A. Dogac, C. Dengi, and M. T. Öszu. Distributed object computing platforms. *Communications of the ACM*, 41(9):95–103, September 1998.

[21] M. Freeston. The BANG file: A new kind of grid file. In *Proceedings SIGMOD'87*, 1987.

[22] S. R. Gardner. Building the data warehouse. *Communicaitions of the ACM*, 41(9):52–60, September 1998.

[23] M. Gimbel, M. Philippsen, B. Haumacher, P. C. Lockemann, and W. F. Tichy. Java as a basis for parallel data mining in workstation clusters. In P. Sloot, M. Bubak,

A. Hoekstra, and B. Hertzberger, editors, *High Performance Computing and Networking Europe (HPCN Europe) 1999*, volume 1593 of *Lecture Notes in Computer Science*, pages 884–894, Amsterdam, April 1999. Springer Verlag.

[24] S. Goil and A. Choudhary. High performance OLAP and data mining on parallel computers. *Journal of Data Mining and Knowledge Discovery*, 1(4):391–417, 1997.

[25] M. Golfarelli, D. Maio, and S. Rizzi. Conceptual design of data warehouses from e/r schemes. In *Proceedings of the Hawaii International Conference On System Sciences*, January 1998.

[26] A. Grimshaw, A. Ferrari, G. Lindahl, and K. Holcomb. Metasystems. *Communications of the ACM*, 41(11):46–55, November 1998.

[27] J. Hellerstein, J. Naughton, and A. Pfeffer. Generalized search trees for database systems. In *Proceedings of the International Conference on Very Large Data Bases*, pages 562–573, 1995.

[28] W. Inmon. *Building the Data Warehouse*. Wiley, 1992.

[29] N. Katic, G. Quirchmayr, J. Schiefer, M. Stolba, and A. M. Tjoa. A prototype model for data warehouse security based on metadata. In *Proceedings of the 9th International Conference on Database and Exper Systems Applications (DEXA 1998)*, 1998.

[30] R. Kimball. *The Data Warehouse Toolkit: Practical Techniques for Building Dimensional Datawarehouses*. John Wiley & Sons, Inc., 1996.

[31] R. Kimball and K. Strehlo. Why decision support fails and how to fix it. *SIGMOD Record*, 24(3):92–97, September 1995.

[32] J. Kiviniemi, A. Wolski, A. Pesonen, and J. Arminen. Lazy aggregates for real-time olap. In M. Mohania and A. M. Tjoa, editors, *First Conference on Data Warehousing and Knowledge Discovery (DaWaK'99)*, volume 1676 of *Lecture Notes in Comupter Science*, pages 165–172. Springer Verlag, September 1999.

[33] D. E. Knuth. *Searching and Sorting*, volume 3 of *The Art of Computer Programming*. Addison Wesley, Reading, Massachusetts, 1973.

[34] A. Kurz and A. M. Tjoa. Integrating executive information systems and data warehouses. In *Proceedings International Conference On Business Information Systems*, 1996.

[35] A. Kurz and A. M. Tjoa. Data warehousing within intranet: Prototype of a web-based executive information system. In *Proceedings of the 8th International Workshop on Database and Expert Systems Application*, 1997.

[36] W. J. Labio, Y. Zhuge, J. L. Wiener, H. Gupta, H. Garcia-Molina, and J. Widom. The WHIPS prototype for data warehouse creation and maintenance. 1996.

[37] D. B. Lomet and B. Salzberg. The HB-tree: A multi-attribute indexing method with good guaranteed performance. In *Proceedings ACM Symposium on Transactions of Database Systems*, volume 15, pages 625–658, 1990.

[38] V. Markl and R. Mayer. The tetris-algorithm for sorted reading from ub-trees. Technical report, FORWISS München.

[39] P. Messina, D. Culler, W. Pfeiffer, W. Martin, J. T. Oden, and G. Smith. Architecture. *Communications of the ACM*, 41(11):36–44, November 1998.

[40] OMG, editor. *CORBA/IIOP 2.2 Specification*. June 1998.

[41] B. Panzer-Steindl. Central data recording for high data rate experiments at CERN. In *CHEP Computing in High Energy Physics*, 1998.

[42] D. Quass and J. Widom. On-line Warehouse View Maintenance. In *Proceedings SIGMOD'97*, 1997.

[43] A. Rauber and P. Tomsich. An architecture for modular On-Line Analytical Processing systems: Supporting distributed and parallel query processing using cooperating CORBA objects. In A. M. Tjoa, A. Commelli, and R. R. Wagner, editors, *10th International Workshop on Databases and Expert Systems Applications (DEXA 1999)*, September 1999.

[44] D. Ridge, D. Becker, P. Merkey, and T. Sterling. Beowulf: Harnessing the power of parallelism in a Pile-of-PCs. In *Proceedings IEEE Aerospace*, 1997.

[45] V. L. Sauter. Intuitive decision-making. *Communications of the ACM*, 42(6):109–115, June 1999.

[46] R. Seidel and C. R. Aragon. Self-adjusting binary trees. *Algorithmica*, 16, 1996.

[47] Sequent Computer Systems. Sequent delivers record single-system performance for TPC-C benchmark with ORACLE. http://www.sequent.com/news/releases/1998/nr-1292.html, October 1998.

[48] G. K. Tayi and D. P. Ballou. Examining data quality. *Communications of the ACM*, 41(2):54–57, February 1998.

[49] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proceedings SIGMOD'99*, 1999.

[50] H. J. Watson and B. J. Haley. Managerial considerations. *Communications of the ACM*, 41(9):32–37, November 1998.

[51] J. L. Wiener, H. Gupta, W. J. Labio, Y. Zhuge, H. Garcia-Molina, and J. Widom. A system prototype for warehouse view maintenance. In *Workshop on Materialized Views*, pages 26–33, 1996.

[52] Y. Zhao, P. M. Deshpande, and J. F. Naughton. An Array-Based Algorithm for Simultaneous Multimensional Aggregates. In *Proceedings SIGMOD'97*, pages 159–170, 1997.