THE CUBE-QUERY-LANGUAGE (CQL) FOR MULTIDIMENSIONAL STATISTICAL AND SCIENTIFIC DATABASE SYSTEMS

Andreas Bauer

Dept. of Database Systems
University Erlangen-Nuremberg
Germany
asbauer@immd6.informatik.uni-erlangen.de

Wolfgang Lehner

Dept. of Database Systems
University Erlangen-Nuremberg
Germany
lehner@informatik.uni-erlangen.de

Abstract

The "Cube-Query-Language" (CQL) is a new query language for flexible access to Multidimensional Database Systems. CQL provides a high level user interface for specifying queries in the context of multidimensional data analysis. The multidimensional view is widely accepted for typical decision support requirements like "Online Analytical Processing". CQL directly supports this view and circumvents the problems of formulating multiple groupby's and equijoins in a typical SQL-notation. Furthermore, in comparison to currently available OLAP-solutions, CQL provides a SQL like query interface, making it easy for an experienced SQL-programmer, to formulate decision support queries efficiently.

The paper details the two-step query processing found in the CQL approach: "data querying" and "data presentation". This description of the query language design is accompanied by a lot of examples, stemming from a joint research project with our industrial partner, thus showing the expressive power and flexibility of the query language design.

Keywords: Query Language, Multidimensional DBS

1 Introduction

During the last few years, the application area of "Online Analytical Processing" (OLAP) has reached a significant commercial market potential and an increasing interest within the database research community ([2]). Many commercial vendors offer OLAP-solutions with a more or less intuitive graphical user interface for querying and presentation ([1], [4], [9]). The "Cube-Query-Language " (CQL), which is introduced in this paper, provides a high level textual access to multidimensional databases. It is developed in the context of our CUBESTAR project. CUBESTAR is an experimental database system based on an extended multidimensional data model.

Proceedings of the Fifth International Conference on Database Systems for Advanced Applications, Melbourne, Australia, April 1-4, 1997.

Nowadays, in the case of "Relational OLAP", each query is transformed into complex SQL-statements ([4]). However, in the case of "Multidimensional OLAP", a query is mostly reformulated from a graphical user interface into a product specific query language ([9]). We believe that the missing of a (de-facto) standardized multidimensional query language must be eliminated by an intuitive textual query language for efficiently formulating complex decision support queries, which can be directly transformed into a query execution plan.

Therefore, the main advantage of CQL is to directly reflect the conceptual distinction between qualifying and quantifying data, which is natural to the multidimensional view ([11]). Furthermore, the intensive use of hierarchical structured dimensions is natively supported within the query language. In contrast to SQL, CQL supports aggregation operators as 1st class operators, which reflect the core of analytical processing. In comparison to other query languages, CQL supports client-server computing within the language design, e.g. the `select`-statement for computing values has to be executed on server side, whereas the `show`-statement is a primary candidate for execution on client side, handling all table presentation aspects. The key advantage of CQL and the data model behind the language is that the user doesn't have to take care about the origin of the data. Instead, the user specifies the context and the resulting figures for an analysis. Additionally, CQL implements constructs to use the feature extension of the CUBESTAR's data model (CROSS-DB) for selection and result presentation.

The presentation of the CQL syntax is accompanied by a lot of examples. These examples stem from a joint research project between our institute and a large European market research company. In their business, quantifying data like sales, stock, or pricing are collected and identified in a three dimensional context obtaining all observed articles (product dimension), all shops reporting some figures (shop dimension) and the month

in which the data is collected (time dimension). A possible (tabular) representation of such data is given in Figure 1.

product	shop	month	SALES	STOCK	PRICE
WAT813	WasherStore	07/96	13	2	1011.-
NOVA	WasherStore	07/96	2	8	999.-
DUETT	WasherStore	07/96	12	5	1234.-
WAT813	ElectroWorld	07/96	7	3	1111.-
DUETT	ElectroWorld	07/96	4	10	1199.-
WAT813	HyperWash	07/96	8	6	1099.-

Figure 1: A sample product tact table

These data serve as a dataset for different types of analysis like segmentations along a predefined aggregation hierarchy (e.g. product family - product group - product area) as well as some special analysis (e.g. cross assortment analysis). Additionally, product family or product group specific information based on time and location invariant features are used for giving further details of a given descriptor (e.g. the brand of a product).

product	brand	type	spin	water	energy
WAT813	ASKO	top	1300	65	18kw
NOVA	AEG	front	1000	68	19kw
DUETT	Miele	top	1400	63	18kw

Figure 2: A sample feature table for the 'washer' product family

The rest of this paper is organized as follows. In the next section, the architecture of the CUBESTAR and the CROSS-DB data model are sketched in order to provide a basic understanding of multidimensional analysis. The third section describes the CQL `select`-statement with the semantics of each single clause in full detail. The `show`-statement, e.g. the client side, is described in section 4. Section 5 introduces the CQL-supported session concept, followed by a comparision of CQL with other query language approaches. The paper closes with a conclusion, summarizing the current and ongoing work.

2 The CUBESTAR-Project

Before going into the detailed description of the "Cube Query Language" (CQL), this section sketches the principal architecture of the CUBESTAR and the main issues of the CROSS-DB data model, thus providing a basic understanding of the system and the underlying data model. For a detailed discussion of the model itself, we encourage the reader to refer to [7].

2.1 Client-Server Architecture of CUBESTAR

The CUBESTAR is an experimental multidimensional database system for performing efficient statistical analysis procedures on ultra large data sets. Based on the principles of modularity and scalability, the CUBESTAR is a typical client-server application which has great impact on the way of data analysis.

At the client, a user formulates a query within the notion of the `select`-statement and sends the request to the CUBESTAR Server (see (1) at Figure 3). The server receives the query, computes and temporarily keeps the query's result, and returns only a reference of the result to the client side. If the user is ready for presentation of the result, a `show`-statement 'activates' already computed results of former queries and returns the data to the client (see (2) at Figure 3). On the client side, these multidimensional data are flatened into a possible complex structured and well-ordered two-dimensional table. This two-step process is justified by the facts that on the one hand, the server should not be considered with presentation issues and, on the other hand, client hardware is able to efficiently process presentation aspects and is favoured by powerful implementation platforms (Java).

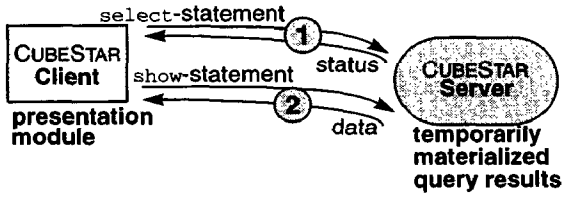

Figure 3: The two-phase query processing scheme of CUBESTAR

2.2 The CROSS-DB Data Model

As mentioned in the introduction, the CROSS-DB data model is a generic representative of already existing multidimensional data models capable of covering most of the existing multidimensional data modelling approaches and thus providing a superset of the different modelling techniques.

The cells of a multidimensional data cube hold single quantitative numeric values like sales of stock figures. In the sense of a composite primary key, these cells are uniquely identified by a set of qualifying information, e.g. the product 'WAT813' was sold in the shop 'ElectroWorld' in 'July96'. The identifying items like a product, shop, or period identifier belong to exactly one dimension (products, location, time). Based on the set of instances, classifications forming a balanced and partitioned tree of identifiers can be declared from the application's point of view. For example, single products are grouped by product families and these, in turn,

may be grouped into product groups and product areas. The time dimension gives a good example for grouping days by different classifications, i.e. by months - quarters - years or by weeks and weekly oriented time granules. Each level of a classification is called the *level of granularity*. The base level corresponds to the finest granule. The root of a classification tree reflects the coarsest granule. In analogy to factual instances, all identifiers within a classification hierarchy may serve as a coordinate for a specific cell in a multidimensional data cube. Therefore, a cell identified by the classification identifiers 'Washers', 'California' and '07/96' may hold the totals of sales for all washing machines sold in any shop in California in the given period.

To avoid extremely sparse cubes and high dimensionality, the CROSS-DB data model allows to characterize single classification nodes by further classification node specific information, called features. At the base granularity level, a feature details the description of a specific item, e.g. a washing machine may be characterized by its loading type ('top'- or 'front'-loader), its maximum spin speed, its water and energy usage and so on. Each higher level classification node covers the set of all features which are common to all subsumed base instances. For example, the node 'Washers' is described among others by a feature 'water usage' with a value range from 40 upto 105 liters, reflecting the minimum and maximum of water usage specifications for all washing machines. Furthermore, the node above the washers, 'home appliances' is characterized by a feature 'energy usage' with a value ranging from 0,5W upto 2233KW.

With this knowledge about the multidimensional data model in mind, the rest of this paper is dedicated to the detailed description of the "Cube Query Language" (CQL). The following section explains the computation part, i.e. the `select`-statement, whereas the fourth section explains the `show`-statement, primarily for execution on the client side.

3 CQL - Computation

This section covers the description of syntatic elements of the "Cube Query Language" (CQL) providing user access to the multidimensional data cube. In the following subsections, we first explain the construction process of building a query context. Then, the steps of selecting cube partitions (Slice and Dice), applying cell-oriented as well as aggregational operators to the specified cube, and handling of nested (sub-)queries are described in detail. At the end of the section, we introduce CROSS-DB's specific means of supporting horizontal analysis.

3.1 The Query Context

In this subsection the construction of a query's multidimensional context is presented. This process consists of the two steps: building the cube of discourse and selecting the special part of interest of the potentially very large data cube. The selection criteria may be based on qualifying as well as quantifying data.

The FROM-Clause:
Creating the universe of discourse

In the *FROM*-clause of a `select`-statement, the user specifies the dimensions of the cube, forming the analysis objective (cube of discourse) of the specified query. For example, the statement below constructs a three-dimensional context consisting of the product, location, and time dimension for the current query. The query returns all sales and stock figures of the current database.

```
SELECT SALES, STOCK
FROM Products P, Location L, Time T
```

The abbreviations following the dimension identifier address instances of the former dimension and are considered as aliases for the use in following clauses, e.g. the *WHERE*-clause. The instantiation process of a dimension may be seen as a role assignment.

The referencing of a specific cell must fulfill their dimension requirements. For example, the `SALES` variable could not be retrieved in a lower-dimensional context. On the other hand, it is possible to specify more dimensions as needed to reference a specific cell. For example, the two dimensional variable `CURRENCY` may reflect a conversion table and hold the time dependant exchange rate for a country specific currency into USD. When selecting this cell with the dimensions 'Location' and 'Time', the currency is supposed to be the same for all products.

```
SELECT CURRENCY
FROM Products P, Location L, Time T
```

In this case, the lower-dimensional data cube for this cell is artificially blown up, by logically copying the values into the undefined dimensions.

The WHERE-Clause: Slicing and Dicing

Slicing and Dicing within the query's context cube is done by specifying a selection predicate based on hierarchically structured qualifying information. The intensive use of classification hierarchies provides a powerful addressing scheme for quantifying data. The qualifying selection predicates are described within the *WHERE*-clause of a `select`-statement. For example,

```
... WHERE P.Group = 'Home Appliances'
```

limits the product dimension to all articles being subsumed by the product group 'Home Appliances'. The dimensions can be limited independently from one another. A single selection predicate consists of the dimension's alias and the granularity of the specification. Classification nodes can be positively addressed as well as excluded from the selection. To construct complex selection expressions, simple predicates can be combined to propositional logical predicates using parenthesis, *OR*, and *AND* constructs. The following statement demonstrates this for the product dimension.

```
SELECT SALES
FROM Products P, Location L, Time T
WHERE P.Group = 'home appliances' AND
      ((P.Family = 'washers' AND
        P.Article != 'DUETT') OR
       P.Family = 'dryers')
```

The selection process becomes multidimensional by listing complex selection expressions for each dimension in the *WHERE*-clause. The separator ',' between the predicates holds the semantics of the *AND*-operator in the multidimensional context. If a selection predicate is omitted for a single dimension D, the selection defaults to all elements of this dimension, e.g. `D.ALL = '*'`.

```
SELECT SALES
FROM Products P, Location L, Time T
WHERE P.Group = 'home appliances',
   L.Region = 'California',
   T.Year = '1996' AND T.Month != '01/96'
```

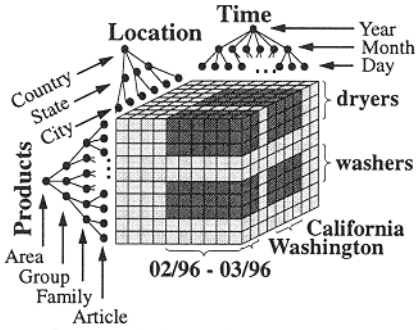

Figure 4: Selection by classification

Slicing and Dicing of the data cube may often not fulfill the user's selection requests. CQL refines the selection process allowing selection predicates not only based on classification nodes but additionally on feature values, characterizing the single items (see Figure 2). This additional selection capability results in a detailed selection mechanism which is illustrated in Figure 4. To distinguish between the two types of selection capabilities in the data cube, the use of a feature predicate is addressed by assigning the feature to the applicable dimension with an arrow.

```
... WHERE P->MaxSpin = '1400' AND
          P->WaterUsage < '65'
```

In addition to feature specification, this example further addresses the fact that feature values may have a numerical data type and therefore allow '=', '<', '<=', '>=', and '>' as well as their negations.

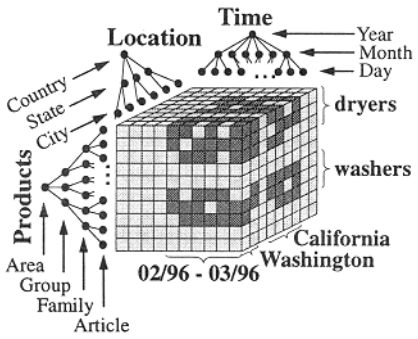

Figure 5: Selection by feature

The RESTRICT-Clause

As seen in the above subsection, limiting the data cube to the interesting partitions, is handled by the use of qualifying information in an appropriate manner. Nevertheless, CQL allows the user to further restrict the selected partitions of the multidimensional cube by predicates over quantifying data. The following example restricts the selected cells to only those sales figures which are outlier (less or equal five and greater or equal 20) for an analyist. To emphasize the two-step selection process, the predicates over quantifying information are specified in the *RESTRICT*-clause:

```
SELECT SALES
FROM Products P, Location L, Time T
WHERE P.Group = 'washers'
RESTRICT SALES >= 20 OR SALES <= 5
```

In the case of the *RESTRICT*-clause, only those cell variables can be used in a predicate, which are listed in the first line of the *select*-clause. Furthermore, the constant value 'NULL' can be used as an operand. The 'NULL'-value addresses cells, having no 'real' values, i.e. the sales of 'Nova' in the shop 'ElectroWorld' (Figure 1).

3.2 Operations based on the Multidimensional Data Cube

The data model distinguishes between two classes of operators on multidimensional data cubes: *cell-oriented* and *aggregational operators*. Cell-oriented operators are used to combine quantifying data from one or more cells into a new value, e.g. multiplying each sales figure with the corresponding price to get the turnover. Aggregating operators are used to build higher-level data over a selected area of cells of the same type, e.g. building the sum of all quarterly washer sales for single regions. Operators are specified in the *SELECT*-clause of a statement and cannot be mixed within a single

statement due to granularity consistency. The CQL representation and application of both classes are presented in the following two subsections.

Cell-Oriented Operators

Cell-orientied operators are used to join single cells of a selected data cube partition and derive new values for the operands. The set of cell-oriented operators consists of unary operators like the unary minus, the signum ('SGN()') and the absolute function making numeric values positive ('ABS()'). Binary cell-oriented operators are left associative and cover the set of basic arithmetic operations ('+', '-', '*', '/') and the minimum and maximum functions ('CMIN()', 'CMAX()') returning the smaller/greater value of the operands. Operands can be cells as well as constants.

The following CQL-statement for example, retrieves the turnover figures of the July96 sales values. The new cells, being delivered to the user, obtain the same dimensionality and granularity in each dimension as their source cells, SALES and PRICE.

```
SELECT SALES * PRICE
FROM Products P, Location L, Time T
WHERE T.Month = '07/96', P.Family = 'washer'
```

An implicit adjustment of dimensionality and granularity is performed by the operators, if the operands do not own the same granularity. Referring again to the data set of the introduction section, the computation of the articles' price in USD is specified with the following query:

```
SELECT PRICE * CURRENCY
FROM Products P, Location L, Time T
WHERE T.Month = '07/96', P.Family = 'washer'
```

As detailed in the subsection describing the *FROM*-clause, in a first step, the dimensionality of the CURRENCY-cube is adjusted to the dimensions, listed in the *FROM*-clause. In a second step, the granularities of the operands are adjusted to a common base. Thus, the sample query above results in values for each single article, each shop that sold the article, and each day, because every day the currency and therefore the price in USD changes.

Aggregation Operators and the UPTO-Clause

While cell-oriented operators on the one hand focus on the single cells and therefore result in the transformation of input data cubes to a target cube with the same or even finer granularity, aggregation operators on the other hand condense the input data cubes to ones with coarser granularity. When analyzing data in statistical environments, it is often necessary to compress an immense amount of detailed raw data into some few, but characteristic values. This often complex analysis process is handled by a series of applications of aggregating operators. To allow great modelling flexibility, CQL only supports the following basic aggregation operators:

```
SUM(), AVG(), COUNT(), CARD(), MIN(), MAX()
```

The following example illustrates the application of the SUM()-operator in the *SELECT*-clause by summing up SALES figures.

```
SELECT SUM(SALES)
FROM Products P, Location L, Time T
WHERE P.Group = 'Home Appliances',
     L.State = 'California', T.Year = '1996'
UPTO P.Family, L.Region, T.Month
```

Whereas cell-oriented operators implicitly decide the granularity of the target cube, aggregating operators need an explicit specification of the granularity of the target cube. This specification is done in the *UPTO*-clause. Referring again to the sample statement above, crossed-up figures are computed for each product family being subsumed by 'Home Appliances', each region in 'California' and the months '01/96' until '12/96'.

Thus, the application of an aggregation operation to a specified data partition always needs the definition of upto-values. These values fix the target granularity, e.g. the grouping level of the output cube. If the *UPTO*-clause is ommitted, the target cell defaults to one value for that dimension.

On the Semantics of Aggregation-Operators

While the semantics of the SUM()- and AVG()-operators may be clear, some remarks have to made wrt. the semantics of MIN()/MAX() and COUNT()/CARD().

The aggregation operators MIN() and MAX() obtain a single variable as operand and result in the lowest/highest value for each partition explicitly specified in the *UPTO*-clause. In contrast, the cell-oriented operators CMIN() and CMAX() are binary operators and take for each operand cell the lower, respectively the higher value.

Except for the COUNT() and CARD()-operator, 'NULL'-values do not have any impact on the application of aggregation operators, because they are treated as non existent. This semantics is also applicable wrt. the COUNT()-operator, returning the number of cells which are not 'NULL'-values. The CARD()-operator also counts the number of cells, but 'NULL'-values are not taken into account. Therefore, the application of the CARD()-operator results in the cardinality of the current cube.

3.3 Subqueries in the WITH-Clause

Up to now, only "flat" select-statements have been considered. To express more complex queries, subqueries can be included hierarchically within a single

select-statement. This mechanism allows to define variables in a subquery, which can be reused in the outer query. The following sample statement makes use of this possibility.

```
SELECT SALES / DENUM
FROM Products P, Location L, Time T
WHERE P.Group = 'Home Appliances',
   L.State = 'California', T.Month = '07/96'
WITH (SELECT SUM(SALES) AS NUM
      UPTO P.Article, L.Region, T.Month),
     (SELECT SUM(SALES) AS DENUM
      UPTO P.Family, L.Region, T.Month)
```

This query computes the sales ratio within the product family of appliances sold in a Californian store in July96 for every single item. For the sake of modularity of the query expression, the cubes holding the numerator and denumerator are each computed independently within a subquery. Therefore, the *WITH*-clause consists of a list of subqueries, defining new cubes which may be referenced in the outer query.

To stay consistent with the embracing query statement, the inner queries inherit the *FROM*-clause, e.g. the context, as well as the WHERE-clause, e.g. the current data partition. However, if the *FROM*-clause is specified in an inner query statement, the set of dimensions must be a subset of the outer query's *FROM*-clause.

3.4 Horizontal Analysis

As already illustrated in section 2, features describe the classification nodes more detailled. On the one hand, this enables a more detailed selection mechanism as shown in subsection 3.1, on the other hand, the use of features enables a more detailed illustration of aggregated values, called *feature split*. Extending the simple select-statement from section 3 by an application of a feature split, the result of a vertical analysis process, e.g. the SUM()-operator with *UPTO*-clause, is split into the different instances of the specified features:

```
SELECT SUM(SALES)
FROM Products P, Location L, Time T
WHERE P.Group = 'Home Appliances',
     L.State = 'California', T.Year = '1996'
UPTO P.Family, L.Region, T.Month
BY P->Energy, L->ShopType
```

This query reports monthly sales values for each product family per region. Each single sales figure is split up according to the features Energy and ShopType. For example, cells will be generated holding the sales of washers with low (or medium or high) energy usage sold in a cash&carry market (or retail store or hypermarket). Due to the classification node sensitive nature of the features, the *BY*-clause may only reference features, which are available at the highest common node, specified in the query, e.g. the *WHERE*-clause. This prevents feature splits which are not common to underlying data, i.e. splitting the washing machines by CPU-type and disk capacity (Figure 2), which would be applicable for 'computer'-sales figures.

To put it in a nutshell, the feature mechanism is a seamless extension of the multidimensional modelling approach with a hierarchical classification tree. In addition to a more detailed selection criterion, features enable the technique of horizontal analysis, e.g. splitting the result of a classification oriented aggregation operator (vertical analysis) according to the required features.

4 CQL - Presentation

As already sketched in the introduction, the design of CQL divides the analysis process into the steps of computation and presentation, to directly support the underlying client-server architecture. This distinction between computation and presentation was founded on experiences from a project with our industrial partner. Apparently different queries refer to the same data material and differ only in the presentation, e.g. with or w/o sorting, including of subtotals, etc.

Defining the Table Structure

The show-statement triggers the presentation process of the result of the last select-statement and produces a hierarchically structured table to illustrate the (flattened) multidimensional data cube. Optionally the keyword PERCENTAGE indicates that the table figures shall be shown as relative values. Furthermore, a cell identifier can be stated, to display the (temporarily materialized) results of former queries (see section 5 for more details).

In the *FROM*-clause of a show-statement, the dimensions of the queried data cube can be specified. These dimensions have to match with those in the query statement and are only used for defining aliases for use within the the current show-statement. The structure of the table is specified in the *STRUCTURE*-clause of the show-statement. Due to the two-dimensionial nature of a table, the *STRUCTURE*-clause is divided into the *HEADER*-subclause, specifying the horizontal appearance and the *STUB*-subclause, defining the vertical appearance of the table. Referring to the ongoing example, the show-statement

```
SHOW
FROM Products P, Location L, Time T
STRUCTURE
  HEADER P.Article
  STUB   T.Month (L.Region)
```

produces the nested table of Figure 6.

SALES		DUETT	NOVA	WAT813
01/96	north	31	79	66
	south	22	62	51
02/96	north	53	53	72
	south	24	98	33
03/96	north	42	23	44
	south	19	53	62

Figure 6: A sample table with a nested stub

As illustrated in the above Figure, the order of nesting the dimensions within the header or the stub is specified in the *STRUCTURE*-clause by nesting the items denoting the classification hierarchy in parenthesis. Therefore, articles are positioned on the horizontal axis. Within the stub of the table, the values are first grouped by month, and then nested by regions.

4.1 Including of Subtotals

The option for additionally displaying the column or row subtotals is specified by appending the keyword TOTALS after the classification items within the *HEADER*- or *STUB*-subclause.

```
STRUCTURE
  HEADER P.Article TOTALS
  STUB   T.Month (L.Region TOTALS)
```

This modified *STRUCTURE*-clause of the sample show-statement above leads to the following table (Figure 7).

SALES		DUETT	NOVA	WAT813	Σ
01/96	north	31	79	66	176
	south	22	62	51	135
	Σ	53	141	117	311
02/96	north	53	53	72	178
	south	24	98	33	155
	Σ	77	151	105	333
03/96	north	42	23	44	109
	south	19	53	62	134
	Σ	61	76	106	243

Figure 7: A sample table with nested subtotals

Again, the mechanism of including subtotals reflects the client-server idea behind the CQL-language approach. *Totals in the Large* are computed on the database server. *Totals in the Small* can be computed without any performance problems on modern clients like PC's, without concerning the DB-server with presentation aspects.

Sorting

Because of the set oriented character of the edges of a multidimensional cube, sorting is no data model supported operator. Nevertheless, sorting is an important tool for intelligent and intuitive data presentation. The show-statement of CQL supports two different sort modes: lexical sorting based on qualifiying information, e.g. on header and stub titles, and numerical sorting based on quantifying information, e.g. on the cell values.

The sort order for qualifying information, e.g. according to the items of a classification, is specified in the SORT-clause of the show-statement. The following example sorts the column arrangements by ascending article identifiers.

```
SHOW
FROM Products P, Location L
STRUCTURE
  HEADER P.Article
  STUB   L.Shop
SORTED BY P.Article
```

To sort the table based on quantifying information, the row or column, specifying the sequence, which defines the order of the complete table, has to be explicitly specified in the *SORT*-clause (e.g. `P.Article = 'Nova'`). The following sample table (Figure 8) illustrates this kind of table sorting.

SALES		WAT813	NOVA	DUETT
03/96	north	44	23	42
02/96	north	72	53	53
03/96	south	62	53	19
01/96	south	51	62	22
01/96	north	66	79	31
02/96	south	33	98	24

Figure 8: A sample table sorted by qualifying Information

The third way of specifying a sort criterion is to nest sortings by cumulated specifications for single sort criterions, based on qualifying data as well as on quantifying data.

Cumulating

Another characteristic of the show-statement is the mechanism of cumulation, e.g. summing up single values according to a predefined order. Figure 9 illustrates this perspective. The cumulation is triggered by adding the keyword CUM to the name of the classification item within the *STRUCTURE*-clause.

The following example is based on Figure 6 and cumulates the sales values along the article dimension.

```
SHOW
FROM Products P, Location L
STRUCTURE
  HEADER P.Article CUM
  STUB   L.Shop
SORTED BY P.Article
```

SALES		DUETT	NOVA	WAT813
01/96	north	31	110	176
	south	22	84	135
02/96	north	53	106	178
	south	24	122	155
03/96	north	42	65	109
	south	19	72	134

Figure 9: A sample table with cumulated rows

5 Supporting Session with CQL

In contrast to other query languages, CQL provides the possibility to name the resulting data cube and force a materialization of the query's result. For these new cells, the user is ensured that the data cube stays accessible for the current session or until it is explicitly dropped. In order to save processing time, the data partition may remain materialized during the session, to be implicitly used by the underlying optimization logic for efficiently answering queries of other users . With this mechanism in mind, often used queries can be explicitly pre-aggregated in order to get a considerable faster response to this family of queries ([8]).

Named cells can be displayed by use of the show-statement parameterized with the cell identfier. If select-statements are not explicitly named, each show-statement belongs to the last query. A new query or the end of the session unfixes/drops the temporarily materialized query's result.

```
CQL> SELECT SUM(SALES) AS VideoSales
CQL> FROM ...

CQL> SELECT ...

CQL> SHOW
CQL> STRUCTURE ...
     // refers to the last query

CQL> SHOW VideoSales
CQL> STRUCTURE ...
     // refers to the first query
```

6 Comparison to SQL and Others

Designing query languages for special purposes has a long tradition in the database research community. Many proposals for efficiently querying complex tables have been made especially in the area of statistical and scientific databases. [10] serves as a comprehensive survey of some approaches.

Within a project with our industrial partner, we modelled and programmed a set of typical queries taken from the aera of market retail research ([5], [6]) in different query languages. This section takes the *brand concentration analysis* as a representative query and compares the query sepcification in CQL, pure SQL ([12]), and Oracle Express ([9]). We intentionally omit a comparison with the CUBE-extension of SQL ([3]). As an extension to the common 'group by'-clause of SQL, the CUBE-operator computes all possible aggregation combinations based on a set of 'group by'-attributes. Thus, for N quantitative values the output's cardinality becomes 2N. First, this exponential growth of data volume is not capable in our application. Second, none of our observed analysis needs all (!) the aggregation combinations.

6.1 Brand Concentration

The brand concentration gives information, to what degree the sales of articles concentrates on certain brands. It is determined, upto which part single brands are responsible for the total sales. The sales values are sorted in a numerically descending order and are cumulated by their brands. The result is often presented as a so called "*Lorenz Curve*". A steep rising curve means, that with a few articles most of the sales is done, whereas a moderate rising means, that the demand for the different models is uniform.

```
SELECT SUM(SALES)
FROM Products P, Location L, Period T
WHERE P.Group = '...', L.Country = '...',
      T.Month = '...'
UPTO P.Group, L.Country, T.Month
BY P->Brand

SHOW
FROM Products P, Location L, Period T
STRUCTURE
  HEADER P->Brand CUM
  STUB   T.Month
SORTED BY T.Month = '...' DESC
```

Figure 10: Brand Concentration Analysis (CQL)

In the select-statement of the CQL-version, the sales figures are aggregated upto a single total sales value and then split up by the different brands, resulting in single sales figure for each brand. In the following show-statement, these numbers are cumulated in the direction of the brands. The ordering of the brands is handled by fixing the specified month (or country name) and by numerical sorting the figures accordingly to this month.

6.2 Brand Concentration (SQL Version)

As detailed in Figure 11, the SQL version of this query results in a typical star query with one relation holding the facts ('Panel'), e.g. sales figures for the single articles (Figure 1) and relations reflecting the classification hierarchy ('ProductClassification', 'ShopClassification'). Additionally, the relation 'ProductFeature' holds the charcteristics of the single articles (Figure 2). In a first step, aggregated sales figures are temporarily stored in the relation Temp. This relation serves in a second step for the cumulation process. Cumulation in

```
INSERT INTO Temp (Group, Country, Brand, SumSales)
VALUES (SELECT PC2.Group, PF.Brand, SC3.Country, SUM(P.Sales) AS SSales)
        FROM   Panel P, ProductFeature PF,
               ProductClassification1 PC1, ProductClassification2 PC2,
               ShopClassification1 SC1, ShopClassification2 SC2, ShopClassification3 SC3
        WHERE  PC1.Article = P.Article AND
               PC2.Family = PC1.Family AND
               PC2.Group = '...' AND
               SC1.Shop = P.Shop AND
               SC2.City = SC1.City AND
               SC3.Region = SC2.Region AND
               SC3.Country = '...' AND
               PF.Article = P.Article AND P.Month = '...'
        GROUP BY Group, Brand, Country)

SELECT SUM(T2.SumSales) AS CumSales
FROM   Temp T1, Temp T2
WHERE  T1.SumSales <= T2.SumSales
GROUP  BY T1.Group, T1.Brand, T1.Country
ORDER  BY CumSales

DROP TABLE Temp
```

Figure 11: Brand Concentration Analysis (SQL)

SQL is done by self joining the relation and a corresponding groupby expression. Furthermore, this rather complex query fails, if two brands obtain the same sales value. The only way is to avoid pure SQL and use dynamic SQL within an additional application program.

6.3 Brand Concentration (Express Version)

In comparison to the relational SQL, Express is a representative of a multidimensional query language ([9]). This multidimensionality can be seen in the four LIMIT-blocks of the sample query, illustrated in Figure 12. Each LIMIT-expression restricts a classification level (identified by a trailing '_D') either to explicitly specified items or implicitly to all subsumed items within the hierarchy. The hierarchy relationships are identified by a trailing '_R'. The feature brand must be modelled as an own dimension holding all possible brands. The dimension is restricted to only those brands which have sales greater than zero.

In analogy to the SQL solution, Express requires the definition of a new multidimensional cell (Temp), temporarily holding the total sales figures. The cumulation process is directly supported on the one hand by the CUMSUM-operator, but, on the other hand, requires a FOR-loop for every single brand.

6.4 Valuation

The comparison of the three language approaches wrt. the rather simple brand concentration analysis enables us to draw the following conclusions:

- SQL and the relational data model are not appropriate for handling OLAP requirements at the query specification level. The SQL-version is quite long and not correctly implementable in pure SQL. Perhaps SQL3 brings us a step forward. The multidimensional approach with the distinction of qualifying and quantifying information seems quite more intuitive than handling star queries.
- The Express query language enables a powerful limitation mechanism on the single dimensions and offers natural access to aggregation operations. The big disadvantage of Express consists of the very complex and unfamiliar query syntax.

Generally, the CQL query language approach inherits the advantages of both approaches.

- The query syntax of CQL is similar to SQL, making it easy to understand and use.
- The multidimensional data model with hierarchical classifications as background knowledge enables flexible analysis techniques.

7 Conclusion

This paper introduces the new query language CQL for Multidimensional Database Systems. The main advantages of this approach are: first, the distinction of qualifying and quantifying information is conceptually the heart of the language design. This results in advantages like handling aggregation operators as 1st class operators and using classification hierarchies as background knowledge at a conceptual level. Second, the distinction between the `select`- and the `show`-statement reflects the underlying client-server architecture, thus distributing specific tasks to the apropriate modules, e.g. computation to the server, presentation to the client. Last, the session concept makes the application of controlled materialization of query results easy and useful.

```
LIMIT Product_Group_D TO '...'
LIMIT Product_Family_D TO Product_Group_R
LIMIT Product_Article_D TO Product_Family_R

LIMIT Location_Country_D TO '...'
LIMIT Location_Region_D TO Location_Country_R
LIMIT Location_City_D TO Location_Region_R
LIMIT Location_Shop_D TO Location_City_R

LIMIT Time_Month_D TO '...'

LIMIT Product_Brand_D KEEP TOTAL(SALES Product_Group_D Location_Country_D Time_Month_D) GT 0

SORT Product_Brand_D TOTAL(SALES Product_Group_D Location_Country_D Time_Month_D)

DEFINE VARIABLE Temp DEC <Product_Brand_D Product_Group_D Location_Country_D Time_Month_D>

Temp = TOTAL(SALES Product_Group_D Location_Country_D Time_Month_D)

FOR Product_Brand_D DO
ROW Temp CUMSUM(Temp Product_Brand_D)
DOEND
```

Figure 12: Brand Concentration Analysis (Express)

At the implementation sector, we have a prototypical CUBESTAR operational, including a complete CQL-parser and a translation to SQL-statements for processing on Oracle 7.3. On the client side, we have implemented a rudimentary ASCII-based table generator, but are going to reimplement this client in Java.

Although, we live in a world of graphical user interfaces, we believe that a textual query language is essential for Multidimensional Database Systems. In addition to an extreme expressive power, queries can be processed in a batch-orientied manner (*"OFFline Analytical Processing"*).

8 References

[1] N.N.: *Essbase Analysis Server - Bringing Dynamic Data Access and Analysis to Workgroups Across the Enterprise*, Product Information, Arbor Software Corporation, 1995

[2] Codd, E.F.: Codd, S.B.; Salley, C.T.: *Providing OLAP (On-line Analytical Processing) to User Analysts: An IT Mandate*, White Paper, Arbor Software Corporation, 1993

[3] Gray, J.; Bosworth, A.; Layman, A.; Pirahesh, H.: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals, *12th International Conference on Data Engineering* (ICDE'96, New Orleans, LA, Feb. 26- Mar. 1, 1996), pp. 152-159

[4] N.N.: *The INFORMIX-MetaCube Approach*, Product Information, Informix Software, Inc., 1996

[5] Lehner, W.; Ruf, T.; Teschke, M.: Data Management in Scientific Computing: A Study in Market Research, in: *Proceedings of the International Conference on Applications of Databases* (ADB'95, Santa Clara, California, Dec. 13-15, 1995), pp. 31-35

[6] Lehner, W.; Ruf, T.; Teschke, M.: Improving Query Response Time in Scientific Databases using Data Aggregation - A Case Study, in: *Proceedings of 7th International Conference and Workshop on Database and Expert Systems Applications* (DEXA'96, Zürich, 9.-13. Sept. 1996), pp. 201-206

[7] Lehner, W.; Ruf, T.; Teschke, M.: CROSS-DB: A Feature-extended multi-dimensional Data Model for Statistical and Scientific Computers, in: *Proceedings of the 5th International Conference on Information and Knowledge Management*, (CIKM'96, Rockville, Maryland, Nov. 12-16, 1996), pp. 253-260

[8] Lehner, W.; Ruf, T.: An Redundancy-Based Optimization Approach for Aggregation Queries in Scientific and Statistical Databases, in: *Proceedings of the 5th International Conference on Database Systems For Advanced Applications* (DASFAA'97, Melbourne, Australia, April 1-4, 1997)

[9] N.N.: *Personal Express, Express Language Reference Manual, Volume I and Volume II, Version 5.0*, Oracle Cooperation, 1996

[10] Ozsoyoglu, G.; Ozsoyoglu, Z.M.: Statistical Database Query Languages, *IEEE Transactions on Software Engineering SE-11(1985)10*, pp. 1071-1081

[11] Shoshani, A.; Olken, F.; Wong, H.K.T.: Characteristics of Scientific Databases, *Proceedings of the 10th International Conference on Very Large Data Bases* (VLDB'84, Singapore, Aug. 27-31, 1984), pp. 147-160

[12] N.N.: *Database Language SQL*, ISO document ISO/IEC 9075, 1992