Efficient Processing of the Cube Operator

Martin Zirkel[+] *Volker Markl*[*] *Rudolf Bayer*[*+]

Institut für Informatik[+]
Technische Universität München
Orleansstr.34, 81667 München, Germany

Bayerisches Forschungszentrum [*]
für Wissensbasierte Systeme
Orleansstr.34, 81667 München, Germany

{zirkel,bayer}@in.tum.de volker.markl@forwiss.de
http://mistral.informatik.tu-muenchen.de

1 Introduction

This paper presents a part of the doctoral work with the theme: "*The impact of sorted reading from UB-trees on relational database systems*". Based on [Mar99] this doctoral work deals with the problems of the efficient implementation and analysis of sorted reading (UB-Cache [Bay97b], Tetris-Algorithm [MB98], [MZB99]) and its effect on query-processing and query optimization in relational databases and especially in the field of data warehousing (DW) and data-mining.

In our days, the B-Tree [BM72] is a de facto standard in relational database systems for one-dimensional data. Based on the B-Tree a new data structure for multidimensional data, the UB-Tree, was invented by [Bay96]. This data structure utilizes a space-filling curve to partition a multidimensional universe into disjoint regions while preserving multidimensional clustering. A space-filling curve (e.g., Z-curve) maps a d-dimensional universe to a one-dimensional universe. Therefore algorithms for point queries, insertion, and multidimensional range queries can be efficiently handled by a normal B*-tree, which is available in any commercial relational database system. In our prototype implementation we use the Z-curve. A *Z-address Z(x)* is the ordinal number of a tuple *x* on the Z-curve which is used a key for the B*-Tree [Mar99].

```
SELECT Years, Month4_Period,
       Productgroup, Country, SUM(Sales) AS Sales
FROM fact
WHERE Years > 1996 AND
       Years < 1999 AND
       Month4_period >= 1 AND
       Month4_period <=2 AND
       Productgroup  = 'video' AND
       Country = 'Germany'
 GROUP BY Years, Month4_Period, Productgroup,
Country WITH CUBE
```

Figure 1-1: Cube statement

As sorted reading (Tetris-Algorithm) is a query processing technique, which combines restrictions and sorting, it is extremely useful for an efficient implementation of the join operator as well as for grouping and cubing with aggregation functions. The Tetris-Algorithm uses a kind of *sweep line* technique to read data in the sort order of any attribute, while accessing each disk page of the table only once. The Tetris-Algorithm uses the partial sort order imposed by the UB-Tree in order to process a relation in some to sort order.

The contribution of this doctoral work is to use sorted reading for grouping and cubing queries with range restrictions and aggregations. In DW an enormous number of aggregates must be computed. Therefore an efficient implementation of the aggregate calculation is necessary. This creates the demand for materializing and storing temporary results. Due to the multidimensional nature of the data, UB-Trees to utilize a multidimensional clustering for intermediate result sets is very beneficial in order to accelerate grouping and aggregation calculation.

2 Usability in DW

DW applications normally use a multidimensional model to represent their data. The numeric data (measures) (e.g., sales, cost), which is the focus of the analysis, is organized along multiple dimensions. A dimension provides categorical data (e.g., container size of the product) that determines the context of the measures. Therefore a measure can be seen as a cell value of a data cube, which is identified by its coordinates. In the relational world this model is mapped to a *star schema* where the database consists of a single *fact* table and a single table for each dimension. Each tuple of the fact table contains a foreign key (that acts as a pointer) to each of the dimensions. Figure 2a shows the *snowflake schema* of GfK. A snowflake schema is a star schema where dimensions are normalized. GfK stands for "Gesellschaft für Konsumforschung", which loosely translated means "Consumer Product Market Research Institute".

[GCB+97] extends the SQL-query language by the CUBE-operator that now is available in commercial systems such as DB2 and SQL-Server (cf. Figure 1-1). The CUBE-operator computes all possible aggregations from the grouping attributes. Figure 1 shows an example of the CUBE-operator processed on the GfK-DW.

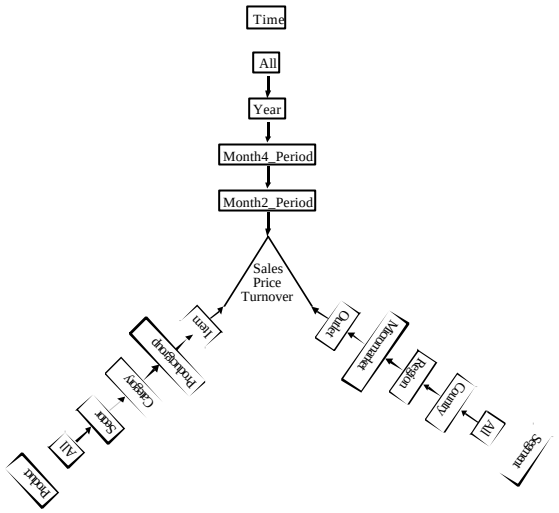

Figure 2-1: a) Star-schema of GfK;

Single table queries restrict, re-arrange or aggregate the tuples of one relation [Ull88]. A relation can be considered to be a d-dimensional space; therefore queries, that restrict an attribute A_i, return a subspace of the universe. A large set of these queries can be mapped to partial range queries. Usually a multidimensional range restriction on a cube or fact table is used to define a sub-cube, which is used for grouping.

For DW applications, a multidimensional *fact table* is the base relation on which the CUBE-operator is computed. Normally people are only interested in the sales of a special product group ('video') in specific period of time in a certain country (e.g., Germany); therefore a restriction is made on each dimension of the fact table by the WHERE-clause of the SQL-statement. Those restrictions form a multidimensional restriction on the fact table. Figure 1 is a typical example of a sub cube defined by the multidimensional restriction on the fact table.

In general a multidimensional index is used to utilize multi-attribute restrictions. To reduce the number of I/Os on the fact table the tuples should be clustered spatially on secondary storage. As grouping algorithms in conjunction with aggregation functions needs sorting for efficiency, a multidimensional index should provide an operator to read the relation in any total sort order. The Tetris-Algorithm on top of a UB-tree fulfils all of these requirements.

3 How to Handle the Cube Operator with UB-Trees

The research question is: *How can a multidimensional data-structure be used for efficient calculation of data-cubes with restrictions.*

3.1 Restrictions

To utilize a multidimensional restriction to compute a sub cube from a *fact table* it should be organizes as a UB-tree (e.g., *Year, Month4, Productgroup* and *Country*). Our earlier papers already have shown that multidimensional access methods, such as the UB-Tree, are extremely beneficial for storing and processing multidimensional fact tables because only the relevant data is retrieved from secondary storage [MZB99, MRB99].

3.2 Aggregation network

A lot of research has already been done in the field of efficient aggregation [HRU96]. [Leh98] provides a good overview of aggregation techniques. One important observation, is the fact that some super-aggregates do not have to be computed from the base table but can be derived from sub-aggregates (e.g., the total sales for a year can be derived from the sales of the four month periods). The derivability of aggregation results from sub-aggregates depends on the additivity of the aggregation function (see [Leh98] for details). An aggregation network, a directed graph, is a way to derive high-level aggregates from low-level aggregates. The lowest level, level 0, is the fact-table. An aggregation network for the query (cf. Figure 1) for the GfK-star-schema is shown in Figure 2-1. Each node of the directed graph represents one grouping, e.g. *Year and Country* in the second level of network represent the grouping according to year and country. For n grouping attributes the complete graph consists of 2^n groupings. From Figure 3-1 it can be observed that there are different paths to reach a node. The thick arrows represent one example to reach each node and therefore define the data flow to compute the cube.

3.2.1 Conventional Approach

In general aggregation functions like *count(), sum(), max(), min()* and *average()* for a relation e.g. *fact* (cf. Figure 1) are computed in two steps: First all tuples of the relation *fact* that have the same values of the grouping attributes *Years, Month4_Period, Productgroup* and *Country* are collected into the same group which can be interpreted as an equivalence class. Second, for each group the aggregation functions are applied to the aggregation-attributes, e.g. to compute the sum

of attribute *Sales*. The result set consists of one tuple for each grouping of *Years*, *Month4_Period*, *Productgroup*, *Country* and the accessory aggregation values, e.g. *sum*(*Sales*).

For aggregation, conventional algorithms first sort a relation according to the grouping attribute(s). Due to sorting, the aggregation of the equivalence classes can be computed very easily by simply reading the tuple stream in sort order.

Usually, this aggregation network is computed as follows: For each node the conventional aggregation algorithm is performed. The result set of a node of level i is the input for the node at level i+1. For our further considerations we assume that the size of the input at level 0 and 1 does not fit into the main memory. Thus an external sorting is necessary for these levels. For external sorting the whole query box defined by the multidimensional restriction must be read and written two-times each. To compute all nodes of level 0 and level 1 of our example four external sort operations are necessary.

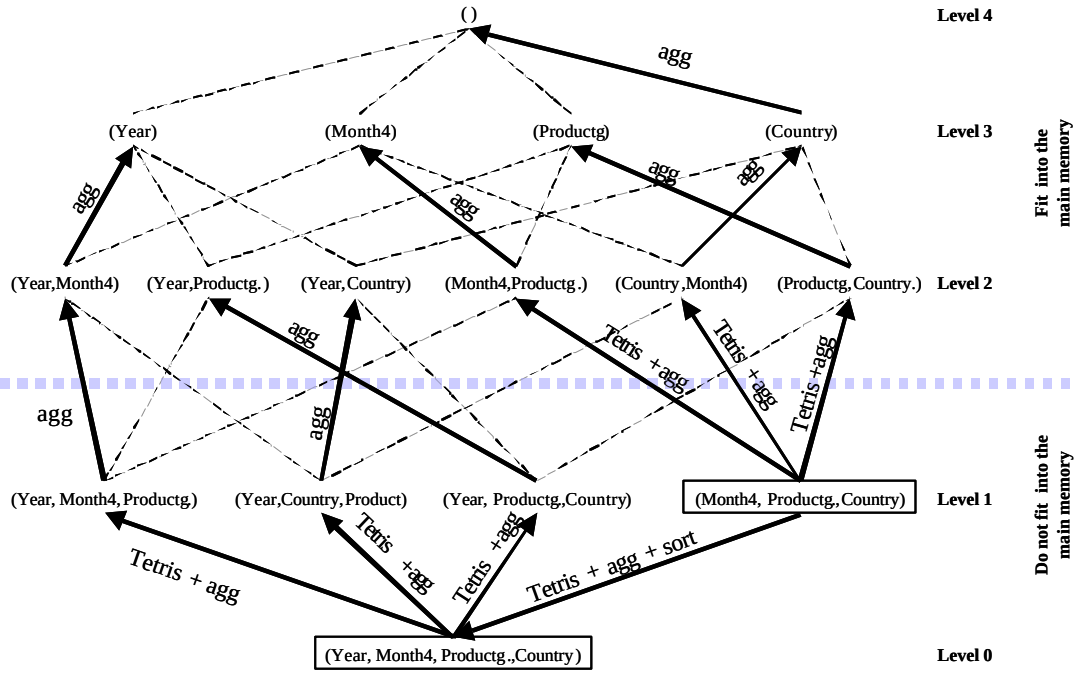

Figure 3-1: Aggregation Networks/ Aggregation Lattices

3.2.2 UB-Tree Approach

In our approach we organize the fact table as an UB-Tree (denoted by a box around the grouping attributes in Figure 3-1). To compute aggregates of level 0 and level 1 the *Tetris – Algorithm* can be used instead of external sorting. Although the relations denoted by the nodes of level 0 and level 1 in Figure 2b do not fit into main memory, they can be sorted in main memory with the Tetris–Algorithm. This is possible, since Tetris divides the query-box in sub queries-boxes of variable sizes, which fit in the main memory and are retrieved in sorted order [MZB99]. So the result set must only once be read and written from the external storage.

As the result set of level 1 does not fit in the main memory either, we will create a temporary UB-Tree for *Month4*, *Product* and *Country*. Efficient creation of a temporary UB-Tree requires the result set to be sorted according the Z-Value of the grouping attributes *Month4*, *Product* and *Country*. The Z-Value is easily computed by bit interleaving [Mar99].

To create the temporary UB-Tree the result set must be read and written to disk two times. Therefore to compute an aggregate from a grouping and store the result set in an UB-tree three steps have to be carried out. First, use the Tetris-Algorithm for sorting, second, compute the aggregation, and third, create a temporary UB-Tree by external sorting. The is denoted by *Tetris + agg + sort* (cf. Figure 3-1).

4 Summary

We have described an efficient algorithm for the computation of the cube operator. It is based on the Tetris-algorithm, which uses a *sweep line*

technique to read data in the sort order of any attribute in order to produce groupings on which aggregations function are performed. To derive a super-aggregate from a sub-aggregate we use the method of dynamic programming. We use an aggregation network where some sub-nodes are materialized as a UB-Trees. The contribution of the thesis to the problem "*How can a multidimensional data-structure be used for efficient calculation of data-cubes with restrictions"* is a new processing technique for the cube operator that should be faster than the conventional algorithms used in commercial database systems. Therefore the amount of pre-aggregates can be extremely reduced because the time to compute cubes on the fly with UB-Tree is fairly low.

Using the UB-tree for storing intermediate result sets in order to efficiently calculate super-aggregates in the next level is totally new.

References

[Bay96] R. Bayer. *The universal B-Tree for multidimensional Indexing*. Technical Report TUM-I9637, Institut für Informatik, TU München, 1996.

[Bay97b] R. Bayer. *UB-Trees and UB-Cache – A new Processing Paradigm for Database Systems*. Technical Report TUM-I9722, Institut für Informatik, TU München, 1997.

[BM72] R. Bayer and E. McCreight. *Organization and Maintainance of large ordered Indexes*. Acta Informatica 1, 1972, pp. 173 – 189.

[GCB+97] J. Gray, S. Chaudhuri, A. Bosworth, A. Layman, D. Reichart, M. Venkatrao, F. Pellow, and H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. Data Mining and Knowledge Discovery 1(1), 1997

[HRU96] V. Harinarayan, A. Rajaraman, and J. Ullman. Implementing Data Cubes Efficiently. Proc. of SIGMOD, 1996.

[Leh98] W. Lehner. Aggregatverarbeitung in Multidimensionalen Datenbanksystemen. Dissertation, Friedrich-Alexander Universität Erlangen, 1998

[Mar99] V. Markl. *MISTRAL: Processing Relational Queries using a Multidimensional Access Technique*. Ph.D. Thesis, Technische Universität München, 1999.

[MR98] V. Markl and R. Bayer. The Tetris-Algorithm for Sorted Reading from UB-Trees. In "Grundlagen von Datenbanken", 10th GI Workshop, Konstanz, 1998.

[MRB99] V. Markl, F. Ramsak, and R. Bayer. *Accelerating OLAP Queries by Multidimensional Hierarchical Clustering*. Proc. of IDEAS, Montreal, Canada, 1999.

[MZB99] V. Markl, M. Zirkel, and R. Bayer. *Processing Operations with Restrictions in Relational Database Management Systems without external Sorting*. Proc. of ICDE, Sydney, Australia, 1999.

[Ull88] J.D. Ullman. *Database and Knowledge Based Systems Volume I.* Computer Science Press, Rockville, MD, 1988.