# Improving OLAP Performance by Multidimensional Hierarchical Clustering

Volker Markl Frank Ramsak Rudolf Bayer
Bayerisches Forschungszentrum
für Wissensbasierte Systeme
Orleansstraße 34
81667 München
Germany

{volker.markl, frank.ramsak}@forwiss.de, bayer@in.tum.de

## Abstract

*Data-warehousing applications cope with enormous data sets in the range of Gigabytes and Terabytes. Queries usually either select a very small set of this data or perform aggregations on a fairly large data set. Materialized views storing pre-computed aggregates are used to efficiently process queries with aggregations. This approach increases resource requirements in disk space and slows down updates because of the view maintenance problem. Multidimensional hierarchical clustering (MHC) of OLAP data overcomes these problems while offering more flexibility for aggregation paths. Clustering is introduced as a way to speed up aggregation queries without additional storage cost for materialization. Performance and storage cost of our access method are investigated and compared to current query processing scenarios. In addition performance measurements on real world data for a typical star schema are presented.*

## 1. Introduction

Data processing in data warehousing (DW) applications uses drill-down operations as well as slicing and dicing according to several dimensions. For this reason, multidimensional data models, multidimensional query languages and even multidimensional DBMS (MDBMS) have been developed by the research community and implemented as commercial products. To a large extent, relational DBMS are used for decision support applications, since these systems are well researched and are reported to provide more efficiency for huge databases than MDBMS. Regardless, whether a multidimensional or relational paradigm is used to model and query OLAP data, queries result in multidimensional range restrictions in combination with sort operations and aggregations. Therefore any DBMS storing OLAP data must efficiently handle this typical query pattern.

Pre-computation, clustering and indexing are common techniques to speed up query processing. Pre-computation results in the best query response time at the expense of load performance and secondary storage space. For DW applications, pre-computation is mostly discussed for aggregation operations [CD97]. However, one requirement of DW is to efficiently deal with ad-hoc queries. Then, deciding which queries to pre-compute becomes extremely difficult. Pre-computation also leads to a view maintenance problem.

Indexing is used to efficiently process a query if the result set defined by the query restrictions is fairly small. Most OLTP applications use B-Trees as their standard indexing scheme. Favoring retrieval response time over update response time allows one to build several indexes on one table or data cube of a DW. Bitmap indexes are widely discussed as an improvement over B-Trees for DW applications, since they efficiently evaluate queries with multi-attribute restrictions. However, the overall result set still must be relatively small. This is a major drawback of bitmap indexes, since usually a relatively large part of a cube has to be accessed in order to calculate aggregated measures.

Clustering places data that is likely to be accessed together physically close to each other. The goal of clustering is to limit the number of disk accesses required to process a query.

The contribution of our paper is a clustering scheme for the fact table of a data warehouse according to multiple hierarchical dimensions, so that star joins result in multidimensional range queries on the fact table. Using a multidimensional organization of the fact table based on

Z-ordering, query processing can exploit clustering in order to reduce random disk accesses. Instead of an artificial schema and data like the TPC-D benchmark used in our earlier paper [MZB99], we had the chance to do performance measurements on real world data provided by our project partners, namely a relational DW for a fruit juice company using a star schema with a fact table of 26 million records (an overall size of 7 GB). We compare our multidimensional approach to traditional bitmap indexing. On this real-world data we experienced a performance increase up to a factor of ten.

The rest of the paper is organized as follows: Section 2 surveys related work. In Section 3 we describe our terminology and identify a standard query pattern for OLAP queries. Section 4 introduces *multidimensional hierarchical clustering (MHC),* and shows how MHC and the *Tetris algorithm* are used to efficiently process star joins. Section 5 gives simulation results, whereas Section 6 presents performance measurements on real world data. Section 7 draws conclusions and gives an outlook on future work.

## 2. Related Work

The new requirements and research issues set by OLAP applications are summarized in [Wid95, WB97]. Besides the questions of data management (e.g., data cleansing, data maintenance) there are two issues of great importance. First, the question of providing a 'good' data warehouse architecture combining a conceptual, a logical, and a physical data model. An overview of the most popular models can be found in [BSH+98]. All these approaches have in common that they are based on a multidimensional data model. On the logical and physical level two main streams have been established – ROLAP that is based on the relational model and MOLAP that uses MDBMS.

The second important issue is the question of performance optimization. Due to the completely different query characteristics of OLAP applications in comparison to OLTP new questions have to be addressed here. The performance problem is heavily linked to the physical data model.

The index selection problem for ROLAP applications is widely discussed in the research community [GHR+97, Sar97]. Especially bitmap indexes provide good performance because of their compactness and support of star joins [CI98]. A common way of performance improvement is the usage of materialized views - often in combination with indexing methods [Moe98, WB98]. Due to the large number of possible views a selection problem has to be solved besides the maintenance issue [Gup97, SDN+96, SDN+98]. Clustering of OLAP data plays a key role in providing good performance. Clustering has been well researched in the field of access methods. B-Trees [BM72], for instance, provide one-dimensional clustering. Multidimensional clustering has been discussed in the field of multidimensional access methods (e.g., [GG97] and [Sam90]). [ZSL98] addresses the issue of hierarchical clustering for the one-dimensional case.

## 3. Processing OLAP Queries

On the conceptual level a multidimensional (MD) view on the data models has been established by academia and the industry for OLAP applications [CD97]. In the MD model the numeric (quantitative) data (*measures*) (e.g., sales, cost) which is the focus of the analysis is organized along multiple *dimensions*. The dimensions provide categorical (qualitative) data (e.g., container size of a product), which determines the context of the measures. Therefore the measures can be seen as a value in a MD space – one often refers to this model as a MD *cube*. An important concept of OLAP data models is the notion of *dimension hierarchies*. Hierarchies are used to provide structure to the otherwise flat dimensions. Often the data in the dimensions can be categorized according to some additional characteristics (e.g., shops could be classified according to their location). Usually OLAP users are not interested in single measures but in some form of summarized data (e.g., sales in a certain area). Hierarchies provide an appropriate method of describing the level of aggregation for a dimension.

Typical OLAP operations are drill-down, roll-up and slice-and-dice [Kim96] and usually multiple dimensions are restricted at the same time. In general one can state that these operations in a MD model lead to range restrictions on the lowest hierarchy level of each dimension [Sar97].

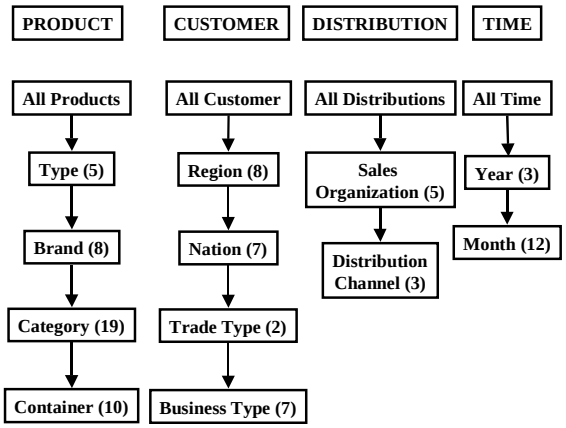
### 3.1 The Physical Data Model

In the following we are concentrating on ROLAP where the conceptual MD model is mapped onto a relational database schema. The most established relational data models for OLAP applications are the *star schema* and the *snowflake schema*. In both approaches a central fact table contains the measures and the dimension tables are situated around it. The connection between a fact tuple and the corresponding dimension members is realized via foreign key relationships. In ROLAP hierarchies on dimensions are usually modeled implicitly by a set of attributes $A_1$, ..., $A_n$ where $A_i$ corresponds to hierarchy level *i*. We call such a sequence of attributes *hierarchically dependent.*

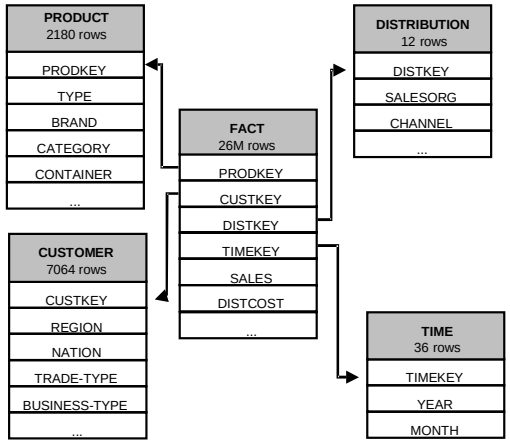
### 3.2 Running Example: The 'Juice & More' Schema

In this paper the following schema of the beverages supplier 'Juice & More', a real customer of one of our

project partners (the company and the data presented here have been made anonymous) will serve as running example. In 'Juice & More' data is organized along the following four dimensions: CUSTOMER, PRODUCT, DISTRIBUTION and TIME. Figure 3-1a shows the hierarchies over the dimensions (the number in parentheses specifies the maximal number of level members).




(a)




(b)

**Figure 3-1 Hierarchies in the 'Juice & More' schema and the corresponding star schema**

The ROLAP data model for the 'Juice & More' schema (Figure 3-1b) is a typical star schema with one fact table FACT and a table for each of the 4 dimensions. Let 'SALES' and 'DISTCOST' be some of the measures in the fact table.

### 3.3 Star Joins

Queries usually contain restrictions on multiple dimension tables (e.g., sales for a specific customer group and for a specific time period are asked) that are then used as restrictions on the usually very large fact table. This so-called *star join* is typical for ROLAP. Figure 3-2a shows the template for star joins on 'Juice & More'. We are focusing on hierarchical restrictions on the dimension tables (e.g., Product.Type='Juice' AND Product.Brand='XYZ'). In Section 4.2 we present a technique which maps a star join to a multidimensional range query (see Figure 3-2b) on the fact table.

```
Select  <MEASURE AGGREGATION>
From    Fact F, Customer C, DISTRIBUTION D,
        Product P, Time T
Where   F.ProdKey = P.ProdKey AND
        F.CustKey = C.CustKey AND
        F.TIMEKEY = T.TIMEKEY AND
        F.DISTKEY = D.DISTKEY AND
        <CUSTOMER RESTRICTION> AND
        <DISTRIBUTION RESTRICTION> AND
        <PRODUCT RESTRICTION> AND
        <TIME RESTRICTION>
```

(a)

```
Select  <MEASURE AGGREGATION>
From    Fact F
Where   F.ProdKey BETWEEN Pkey1 AND Pkey2 AND
        F.DistKey BETWEEN Dkey1 AND Dkey2 AND
        F.CustKey BETWEEN Ckey1 AND Ckey2 AND
        F.TimeKey BETWEEN Tkey1 AND Tkey2
```

(b)

**Figure 3-2 Star-Join Example and Multidimensional Range Query**

## 4. Clustering OLAP data

Since symmetrical clustering with respect to several dimensions is hard to achieve, most physical OLAP storage models either use non-clustering indexes like secondary B-Trees or cluster data with composite B-Trees. The most prevalent OLAP data structure are bitmap indexes (e.g., [OQ97]). Bitmap indexes are useful, if multiple restrictions in low cardinality attributes like REGION or BRAND result in a very small selectivity (i.e., ratio of result set size and table size) for the conjunctive restriction. However, bitmap indexes are non-clustering secondary indexes which for small result sets may require a random access for every tuple. For large result sets (i.e., when for a page size of $C$ tuples the selectivity of a query exceeds $1/C$) they may require an access to every page of the table in worst case.

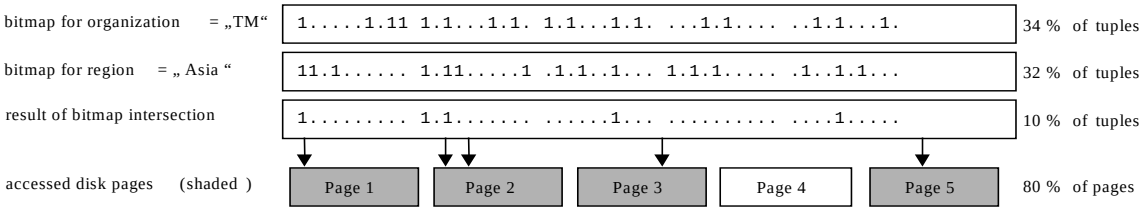



**Figure 4-1 Bitmap Index Intersection**

Figure 4 -1 shows how bitmap indexes process a query that calculates the total sales of customers in Asia for distribution organization “TM” (zero bits in each bitmap are indicated by a dot in the figure).

For each restriction, the bitmap is retrieved from the corresponding bitmap index. After intersecting these two bitmaps by a bitwise AND-operation the tuples corresponding to 1-bits are retrieved. In the figure we assume $C = 10$ tuples to fit on one page, thus ten consecutive bits correspond to the tuples on one disk page. The selectivity for the dimensions is 32% respectively 34%, resulting in an overall selectivity of 10%. Since the data is not clustered on the pages, the query needs to retrieve 80% of the fact table to retrieve 10% of the tuples.

In practice this ratio is even worse: Actual values for $C$ range between 20 and 400 for 8kB pages. For the ‘Juice & More’ data warehouse the actual value is $C = 30$. Therefore bitmap index intersection might result in a full table scan already, when the conjunctive selectivity of a query exceeds 3.33%.

## 4.1 A Formal Description of Hierarchies on Dimensions

For our definition of MHC we use a set concept to formally define hierarchies: A dimension $\mathbb{D}$ consists of a base type having a set of values $\mathbb{V}=\{v_1,...,v_n\}$. A *hierarchy* of *depth h* over $\mathbb{D}$ is an ordered set of $h+1$ levels, i.e., $H=\{\mathbb{L}_0,...,\mathbb{L}_h\}$ (see Figure 4–2). Each *hierarchy level i* of $H$ over $\mathbb{D}$ is a set of sets $\mathbb{L}_i = \{ m_1^i, ..., m_j^i \}$ with $m_k^i \subseteq \mathbb{V}$ for $k=1,..,j$. Each $m \in \mathbb{L}_i$ is a member set (or member) of the hierarchy of level $i$ containing all members of a category. Usually a member $m$ is assigned a name *label*($m$) (e.g., ‘Orange Juice’ for $m_1^1$) instead of enumerating all values $v_k \in m$. The subset relationship $\subseteq$ between the members of two neighboring levels $\mathbb{L}_i$ and $\mathbb{L}_{i+1}$ defines a hierarchical relation (i.e., partial ordering) between the levels (e.g., the product ‘OJ0.7L’ is in the product category ‘Orange Juice’). Increasing the level of a hierarchy increases the *granularity* of the categorization, i.e., the data is classified according to finer categories.

With the base set $\mathbb{V}$ as the only member of level $\mathbb{L}_0$ (i.e., $\mathbb{L}_0 = \{\mathbb{V}\}$) a hierarchy $H$ builds a hierarchy tree (we will explain how to deal with complex hierarchies in Section 4.2.2.) with the root level $\mathbb{L}_0$. The nodes of $H$ are the hierarchy members (or member labels) connected by edges which are defined by the subset relationship between members of neighboring levels. The *children* of a member $m_k^i$ of level $i$ are all members $m_l^{i+1}$ of the lower level $i+1$ that are subsets of $m_k^i$, i.e.,

children($m_k^i$)=$\{ m_l^{i+1} \in \mathbb{L}_{i+1} \mid m_l^{i+1} \subseteq m_k^i \}$ (e.g., the set {{‘Apple Juice 0.5L’},{ ‘Apple Juice 1L’}} is the children set of ‘Apple Juice’). The *parent* of a member $m_k^i$ of level $i$ then is the member $m_l^{i-1}$ of the upper level $i$-1 that is a superset of $m_k^i$, i.e.,

*parent*($m_k^i$)=$\{ m_l^{i-1} \in \mathbb{L}_{i-1} \mid m_l^{i-1} \supseteq m_k^i \}$ (e.g. ‘Orange Juice’ is the parent of ‘OJ 0.7L’).

The bijective function $ord_m$ (see Figure 4–2 for an example) defines a numbering scheme for the children of

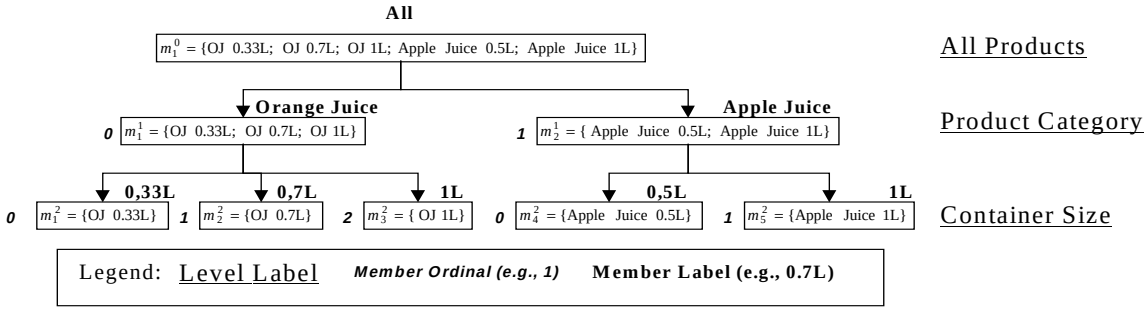



**Figure 4-2 Example Hierarchy in Member Set Representation**

a member $m$ of $H$. $Ord_m$ assigns each subset (child) of $m$ a number between 0 and the total number of children of $m$ i.e.,

$$ord_m : children(m) \rightarrow \{0,...,|children(m)|-1\}.$$

Hierarchies should never relate members of different dimensions, since dimensions are independent and thus such a hierarchy could be split up in two separate hierarchies (see Section 4.2.2).

## 4.2 Multidimensional Hierarchical Clustering

Clustering of one-dimensional objects and single object hierarchies has been discussed to a large extent (e.g., [ZSL98], [BK89], [Sal88]). However, OLAP queries often impose restrictions with respect to hierarchies over multiple dimensions. The result set satisfying these restrictions is usually quite large; for presentation it is grouped and aggregated or ranked. Clustering data with respect to multiple hierarchies can substantially speed up these operations.

If the order of dimensions during drill down is known in advance, clustering the fact table in this dimension order will result in a good query performance. In principle, a concatenated clustering index (i.e., $B^*$-Tree) on the hierarchy levels of all dimensions in *one* lexicographic order is maintained. Even when creating multiple concatenated B-Trees there is a high probability that the pre-defined clustering order will not be very useful for a particular query.

MDBMS use multidimensional arrays to physically cluster data. However, for non-aggregated data this often leads to sparsity problems, which are discussed in more detail in Section 4.3. Multidimensional access methods as commonly used in spatial DBMS provide multidimensional clustering in order to efficiently answer multidimensional range queries. In combination with a suitable hierarchy encoding these methods can be used to significantly speed up OLAP queries.

**4.2.1. Order Preserving Encoding of Hierarchies by Surrogates.** Many attributes in relational DBMS in general and in data warehouses in particular have an actual domain of a very small set of values. A typical example is the attribute REGION of the dimension table CUSTOMER of 'Juice & More', which has an actual domain of 8 values. However, a much longer character string is used to store the regions. We call the data type of an attribute to be an *enumeration type*, if its actual domain $\mathbb{A}$ consists of a relatively small finite set of values.

In order to maximize the entropy of an enumeration type $\mathbb{A}$ we define an order preserving one-to-one mapping $f$ and its inverse function $f^{-1}$:

$$f: \mathbb{A} \rightarrow \{0,\ldots,|\mathbb{A}|-1\} \text{ such that for } a, b \in \mathbb{A}: f(a) < f(b) \Leftrightarrow a <_{\mathbb{A}} b$$

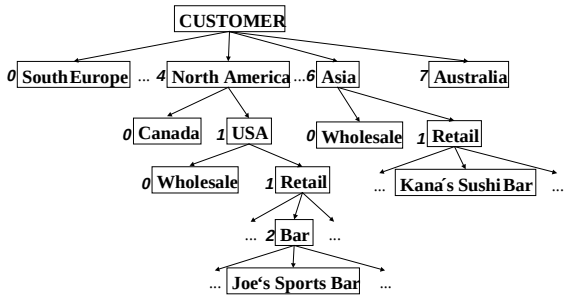
If there is no reasonable ordering on an enumeration type (e.g., it does not make sense to ask for REGION < "Middle Europe"), we drop the requirement on $f$ to be order preserving:

$$f: \mathbb{A} \rightarrow \{0,\ldots,|\mathbb{A}|-1\}, f \text{ injective}$$

We call $f$ a *surrogate function* for an enumeration type. For each value $a \in \mathbb{A}$ we call $f(a)$ the *surrogate* of $a$. For a very compact representation we number surrogates in sequential order. Figure 4–3a lists the values of the enumeration type REGION and the corresponding surrogates.

| REGION | f(REGION) |
|---|---|
| South Europe | 0 |
| Middle Europe | 1 |
| Northern Europe | 2 |
| Western Europe | 3 |
| North America | 4 |
| Latin America | 5 |
| Asia | 6 |
| Australia | 7 |

(a)




(b)

**Figure 4-3 Surrogates for REGION and the entire Customer Hierarchy**

To efficiently encode hierarchies, we introduce the concept of *compound surrogates* for hierarchies [Mar99]. Since we require hierarchies to form a disjoint partitioning, a uniquely identifying compound surrogate for each child node of a hierarchy member exists and can be recursively calculated by concatenating (○) the compound surrogate of the member with the running number of the child node as calculated by the surrogate function *ord* from Section 4.1. Thus, for a member $m^i$ of hierarchy level $i$ of hierarchy $H$ we define its compound surrogate:

$$cs(H, m^i) = \begin{cases} ord_{father(m^i)}(m^i) & \text{, if } i = 1 \\ cs\left(H, \text{father}(m^i)\right) \circ ord_{father(m^i)}(m^i) & \text{, otherwise} \end{cases}$$

The path North America → USA → Retail → Bar (Figure 4–3b) has the compound surrogate:

$ord_{\text{Customer}}$ (North America) ∘ $ord_{\text{North America}}$ (USA) ∘ $ord_{\text{USA}}$(Retail) ∘ $ord_{\text{Retail}}$(Bar) = 4∘1∘1∘2.

The upper limit of the domain for surrogates of level *i* is calculated as the maximum fan-out (number of children) of all members of level *i*–1 of a hierarchy *H*, i.e.,

$$\text{surrogates}(H, i) = \max \{\text{cardinality}(\text{children}(H, m)) \text{ where } m \in \text{level}(H, i - 1)\}$$

A path Φ through a hierarchy of depth *h* is specified by a list of members $m^1$, ..., $m^h$, where $m^i$ is a member of hierarchy level *i*. With $l_i = \lceil \log_2 \text{surrogates}(H, i) \rceil$ a *fixed length compound surrogate* can be stored in a very compact way by binary encoding.[1]

$$cs(H, \Phi) = \text{cs}(H, m^h) = ord_{father(m^1)}(m^1) + ord_{father(m^2)}(m^2) \cdot 2^{l_1} + \ldots + ord_{father(m^h)}(m^h) \cdot 2^{l_1 + l_2 + \ldots + l_{h-1}}$$

With $l_1$=3, $l_2$=3, $l_3$=1, and $l_4$=3 for the CUSTOMER dimension (cf. Figure 3-1a) this formula leads to the compound surrogate:

$$\text{cs(H, Bar)} = \underbrace{100}_{l_1=3}\ \underbrace{001}_{l_2=3}\ \underbrace{1}_{l_3=1}\ \underbrace{010}_{l_4=3}{}_2 = 538$$

Usually growth expectations for a hierarchy are known well in advance. Often hierarchy trees are even static. Therefore it is possible to determine a reasonable number of bits for storing each surrogate of the compound surrogate of a hierarchy. Since hierarchy trees grow exponentially, the overall number of bits necessary to store a compound surrogate is relatively small. For instance, a hierarchy tree with four branches on 8 levels already represents $4^8$ = 65536 partitions and is stored by 16 bits.

The maximum length $l_{\max}$ of a compound surrogate for 'Juice & More' can be computed from the maximum fan-out of the hierarchy levels given in Figure 3-1a. For any of the 4 hierarchies $l_{\max}$ does not exceed 15 bits and thus can be stored in a single integer value.

The lexicographic order on the hierarchy levels is preserved by this very compact fixed length encoding to integers. Point restrictions on upper hierarchy levels result in range restrictions (intervals) on the finest granularity of a hierarchy. For instance, the point restriction NATION = "USA" on the second level of the CUSTOMER hierarchy with *f*("North America")=4=$100_2$ and *f*("USA")=1=$001_2$ maps to the range restriction $cs_{\text{customer}}$ between 528=100 001 0 $000_2$ and 543 =100 001 1 $111_2$ (i.e., to the interval [528,543]). Thus, a star join with this surrogate encoding for the foreign keys of a fact table results in a range restriction on each compound surrogate, if some hierarchy level of each dimension is restricted to a point. In the same way intervals on the children of one hierarchy level result in a range on the corresponding compound surrogates (e.g., year = 1998 and month between April and June). A star join on a schema with *d* dimensions creates a *d*-dimensional interval restriction on the fact table.

**4.2.2. Dealing with Complex Hierarchy Graphs.** If two levels of a hierarchy graph are linked by several paths, there are several possibilities to define a hierarchy tree and therefore several ways to calculate the compound surrogates for physical clustering:

If the order on the lowest level of granularity is identical for two hierarchy paths, then one path can be derived from the other path by an order preserving function on the lowest level of granularity. Then the clustering order for both hierarchy paths is identical. Thus, the clustering order for WEEK and MONTH in Figure 4–4a is identical. Both can be computed by an order preserving function from DAY, the lowest granularity level of the TIME hierarchy.

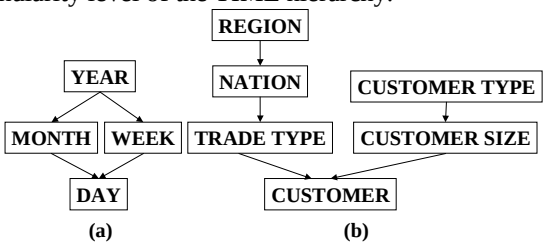



**Figure 4-4 Complex Hierarchy Graphs**

If the query profile is known, the most useful path of the hierarchy graph used for restrictions, sort operations or grouping should be chosen. Thus, if in Figure 4–4b queries on CUSTOMER usually restrict REGION and NATION, this path should be chosen for clustering.

If the query profile is not known, all paths of a hierarchy graph may be used for clustering, since hierarchies may be used for restrictions independently during drill-down. For clustering the different paths then can be considered to be independent dimensions. In the hierarchy graph of Figure 4–4b both the REGION and the CUSTOMER hierarchy might be used for clustering. However, this approach increases the clustering dimensionality and thus should be used with care.

Other issues in the context of complex hierarchies are unbalanced hierarchies, slowly changing dimensions and multiple inheritance. Unbalanced hierarchies occur, if some hierarchy members have more child levels than others. This means, that the compound surrogates of Joe's Sports Bar and Kana's Sushi Bar in Figure 4–3b have different lengths. Using variable length compound surrogates or padding the shorter compound surrogate

[1] In general we use *variable length compound surrogates* that need $l_i(m) = \log_2 |\text{children}(m)|$ bits to store the surrogate for any child of *m*. However, since the hierarchy of 'Juice & More' is quite balanced (i.e., most hierarchy members have the same number of children), we chose *fixed length compound surrogates* for clustering the 'Juice & More' fact table.

with zero bits solves this problem without any impact on clustering.

Slowly changing dimensions can be addressed by marking each node of a hierarchy tree with a validity time interval. An object is physically clustered and retrieved with respect to its validity time. Reorganization of the physical clustering is not necessary: Even with a new classification upon a certain point of time the existing clustering should be correct from a historic perspective. If the business type of Joe's Sports Bar changes from bar to restaurant in 1998 (cf. Figure 4–5), all previously clustered data still is correct.

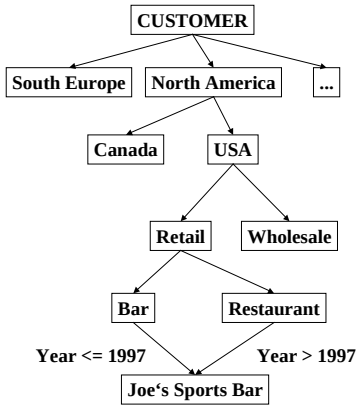



**Figure 4-5 Change of a hierarchy over the time**

The total sales over all bars in 1997 must include Joe's Sports Bar, whereas it is included in restaurants for 1998. However, each object of a hierarchy needs information about re-classification in order to correctly calculate the total sales to Joe's Sports Bar over the last years.

Multiple inheritance (e.g., Joe's Sports Bar is considered to be both a bar and a restaurant at the same time) is solved similarly to slowly changing dimensions: One of the several possible paths to a hierarchy node is chosen for clustering. The other paths of a hierarchy graph to that object then merely store a pointer to the sub-tree that actually stores the object. If multiple aggregation paths are possible, precautions must be taken that only one of these paths is used for aggregation.

## 4.3 Choosing a suitable Data Structure: The UB-Tree

In principle, MHC may be implemented by any multidimensional access structure in combination with the surrogate calculating function of Section 4.2.1. However, using R-Trees [Gut84] or $R^*$-Trees [BKS+90] may result in a sub-optimal performance, since these structures may subdivide the universe into overlapping tiles, which may result in multiple accesses to one disk page. Therefore the most interesting candidates are Grid-Files [NHS84], hB-Trees [LS90], $R^+$-Trees [SRF87] or space filling curves in combination with one-dimensional access methods [OM84, Jag90]. All of these methods provide a disjoint partitioning of multidimensional space. Because of its inherent hierarchical data space organization and its easy implementation, we use the UB-Tree for the 'Juice & More' data warehouse. However, the principle benefits of our technique also apply to the other access methods.

The UB-Tree [Bay96, Bay97a] is an access method for multidimensional point data and thus copes with sparsity without any additional overhead. It utilizes a space filling curve to create a hierarchical disjoint partitioning of a multidimensional universe while preserving multidimensional clustering. Using the Z-curve it is a variant of the zkd-B-Tree [OM84] partitioning the multidimensional space into *Z-regions* (i.e., a subspace of the multidimensional space defined by an interval on the Z-curve). Each Z-region corresponds to one page on secondary storage. The UB-Tree requires logarithmic time (in the number of actual values in the data cube) for the basic operations of insertion, point retrieval and deletion, and storage requirements are also linear.

By the virtue of compound surrogates from Section 4.2.1 for each dimension, the UB-Tree creates a MHC. This clustering is efficiently exploited by the UB-Tree range query algorithm [Bay96, Mar99] to answer queries with point or range restrictions in multiple hierarchies. Range queries are processed by retrieving all Z-regions that intersect the query box and thus linearly depend on the result set size.

The problem of the zkd-B-Tree when handling tuples with identical leading bits in some attributes [LS90] does not occur for MHC with UB-Trees: The leading bits of each dimension belong to the top level hierarchies and therefore partition the data space with respect to that dimension. Since the interleaving order of bit-interleaving hierarchically organizes the data space, the boundary of each Z-region exactly reflects the hierarchy over each dimension. The first hierarchical split levels correspond to the upper nodes of the hierarchy tree. Therefore, a query box defined by hierarchical restrictions over a multidimensional hierarchically clustered UB-Tree will contain most Z-regions completely. The retrieval overhead is minimal; almost all data being retrieved is part of the result set.

## 4.4 Addressing Sparsity

*Sparsity* is defined as the percentage of a domain that is not existent in the actual domain. For a multi-dimensional data cube sparsity is the ratio between the number of cells not containing any data and the overall number of cells of a data cube. Some OLAP tools allow one to mark dimensions to be sparsely populated and then specially handle them. However, a data cube is formed as the cross product over the domains of all dimensions. Therefore, even for non-sparse dimensions the sparsity of the entire cube becomes extremely high soon. The 'Juice

& More' schema, for instance, is a star schema with four independent dimensions with a sparsity of 99,8% (cardinalities taken from Figure 3-1b):

$$\text{sparsity(Juice \& More)} = 1 - \frac{26\ \text{Mio}}{7030 \cdot 5600 \cdot 36 \cdot 12} =$$

$$= 0{,}9984 \text{ with}$$

$$\text{sparsity(star schema)} = 1 - \frac{|\text{Fact Table}|}{\prod_{i=1}^{d} |\text{Dim Table } i|}$$

To our knowledge sparsities of more than 99% are typical for data warehousing applications (e.g., the TPC-D benchmark). Thus, in practice sparsity forbids to materialize an entire data cube of raw data. Physical data organization in a multidimensional array is only feasible for highly aggregated data. However, serious decision support applications require a deep drill down into interesting areas of a data cube. Therefore it is necessary to have a physical representation of a sparsely populated data cube that offers efficient access to parts of that cube. With MHC drill down defines a subspace of a data cube by range restrictions in several dimensions. Thus, MHC is a method to cluster sparse data with respect to several dimensions in combination with an efficient range query and sort algorithm (cf. Section 4.5) for efficient handling of drill down queries.

## 4.5 Processing OLAP Queries on Multidimensionally Clustered Data - The Tetris-Algorithm

With MHC a star join in data warehousing applications basically maps to group/aggregate operations in combination with multi-attribute range restrictions as shown in Section 4.2. This typical query pattern is efficiently handled by the Tetris algorithm [MB98], a generalization of a multidimensional range query algorithm (see [MZB99] for a detailed description and analysis): Basically, the partial sort order imposed by a multidimensional partitioning is used to process a table in some total sort order. Disk accesses are only necessary for the query space defined by the multidimensional hierarchical restrictions. With sufficient, but modest, cache memory each disk page is accessed only once (see Section 5).

Figure 4-6 illustrates how Tetris processes a hierarchically clustered relation to aggregate the sales for each different fruit juice for all customers in Asia. The restriction of REGION='Asia' results in an interval in the CUSTOMER dimension. The same holds for the restriction TYPE='Juice' for PRODUCT. The boundaries of each query interval correspond to Z-region boundaries and thus minimize the number of Z-regions only partly contained in the query box.

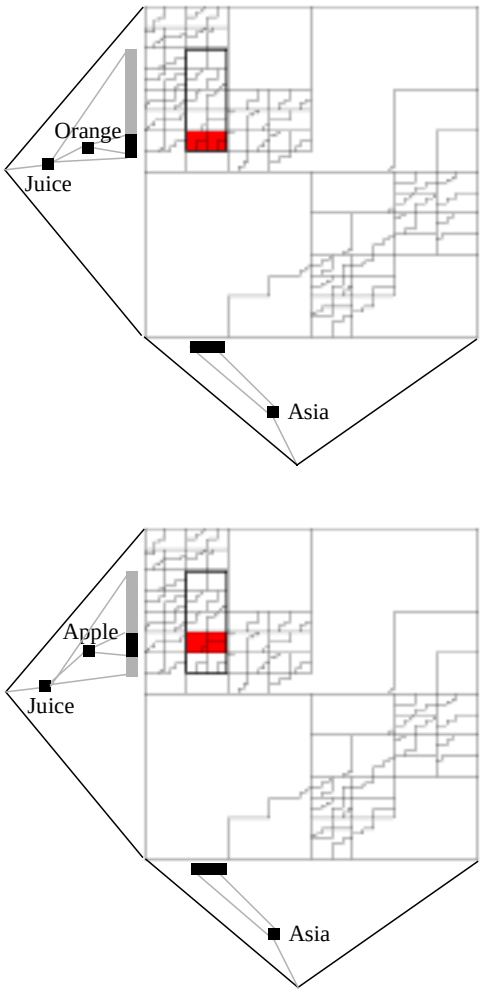



**Figure 4-6: Processing a query box in sort order with the Tetris algorithm**

The query box is read in sort order from bottom to top; the aggregates for each juice type are calculated on the fly. The part of each Z-region from which tuples are cached is shaded. When all Z-regions intersecting the 'Orange Juice' slice have been read, this slice is sorted and aggregated. In the same way the next slices ('Apple Juice', 'Cherry Juice', etc.) are processed. This continues until the entire product interval defined by the restriction to 'Juice' has been handled.

Tetris usually avoids external sorting, since only slices of the result set need to be sorted at a time. Thus only sublinear temporary storage is required with respect to the result set size. I/O-time is linear with respect to the result set size. In contrast to a standard merge-sort algorithm sorting is no longer a blocking operation. Aggregations can be calculated on-the-fly and allow for better interactive response times. In addition, the Tetris algorithm efficiently processes iceberg queries for ranking [FSG+98], if the desired measure is used as a further

dimension of the UB-Tree. The Tetris algorithm then does not read the entire query box in the sorting dimension, but terminates after processing the first slices.

### 4.6 Materializing Aggregates

MHC is not only applicable to the fact table itself, but can also be used to organize views with materialized aggregates. Higher aggregation levels result in a UB-Tree with shorter compound surrogates or reduced dimensionality. It makes sense to store pre-computed aggregates for the highest aggregation levels with restrictions in only one dimension, e.g., the total sales on a yearly basis. Since not all possible aggregations can be stored in general, MHC allows one to derive many further aggregates efficiently from the raw data. This avoids materialization of many aggregation levels and thereby reduces the view maintenance problem to a large extent. The 'space–time' tradeoff between preaggregation and query response time still exists, but MHC reduces it significantly.

## 5. Performance Analysis

The cost functions used for our analysis take clustering, prefetching as well as CPU-time and I/O-time into account and were derived in [Mar99] and [MZB99] based on [HR96]. For retrieving or grouping and aggregating (i.e., sorting) a relation in combination with multidimensional hierarchical restrictions we simulated response times and intermediate temporary storage. We consider several organizations of the fact table of a star schema: MHC, a composite secondary index (CSI, clustering B*-Tree) over all attributes (foreign keys of each dimension and measure attributes), a single secondary index (SSI, non-clustering B*-Tree) on the attribute with the least selectivity, a full table scan (FTS), and bitmap index intersection (BII), which combines the bitmaps of each restricted attribute to determine the result set of the query. In the following we assume that the fact table is stored on $P$ disk pages.

### 5.1 Retrieval with Multi-Attribute Restrictions

An FTS to answer multidimensional range queries with selectivity $s_j$ in dimension $j$ can exploit prefetching techniques to reduce the number of random page accesses at the expense of having to read the entire table. Using a CSI with a composite B*-Tree in lexicographic order $A_1$, ..., $A_d$, the index for the restriction in $A_1$ may be used at the expense of having a random access for each page. A SSI on $A_j$ requires a random page access for each tuple satisfying the restriction in $A_j$, since no clustering of the tuples is available. The number of random accesses of a SSI is limited to $P$, if the row identifiers of the SSI are sorted and then processed in physical page order for data page retrieval. For point restrictions on the index attribute (e.g., REGION = 'Asia'), sorting of row identifiers may even be avoided: index pages for tuples with identical index attributes may be organized in the physical order of the row identifiers. Then point restrictions will get a list of row identifiers sorted according to the physical location of the tuple. This makes a SSI not to degenerate and behave similarly to an FTS in worst case.

BII requires that the corresponding part of each bitmap index has to be retrieved. In addition, BII requires a random access for each tuple satisfying the restrictions in all attributes. The result of BII is a bitmap, which is used to access data pages in physical order. Thus multiple random accesses to one data page will not occur.

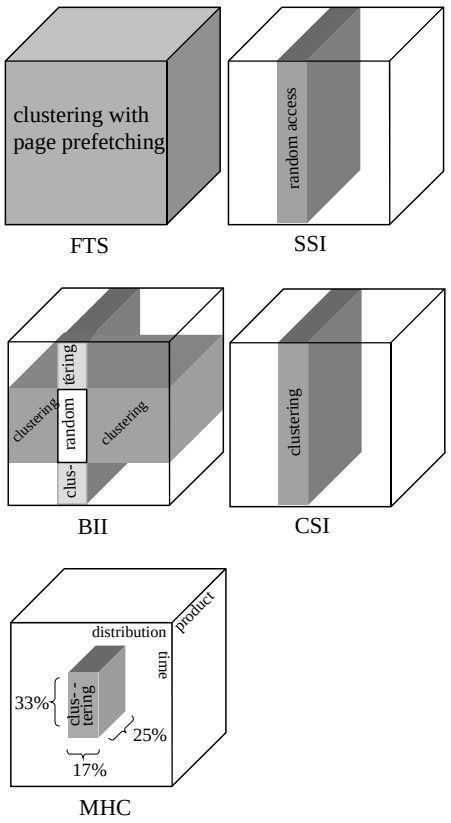



**Figure 5-1: Access Methods and Clustering**

The shaded part of each cube in Figure 5-1 shows the part of a three dimensional database which is retrieved by the corresponding access method to answer a query to compute the sales for one year ($s_{TIME}$ = 33%) for all fruit juices ($s_{PRODUCT}$ = 25%) sold by direct marketing ($s_{DISTRIBUTION}$ = 17%): an FTS retrieves the entire database exploiting clustering and prefetching. In contrast to that a SSI will rarely utilize any clustering benefits for small result sets. BII retrieves each bitmap by clustered access, whereas the data itself will often be spread over many data pages and then must be retrieved by random access to one page for nearly every tuple. However, for larger result sets the probability rises that prefetching might be applicable for bitmap indexes. This means, that BII will

not be much less efficient than an FTS in worst case. A CSI with the DISTRIBUTION dimension as first attribute in concatenation order utilizes clustering but only exploits the 17% restriction on DISTRIBUTION. In contrast to that, MHC utilizes the restrictions of all dimensions and retrieves the data in a clustered way. Both effects contribute substantially to the performance advantage of MHC.

## 5.2 Simulation of Response Times for Queries with Multi-Attribute Restrictions

Figure 5–2a shows a simulation of the sales query ($s_{TIME}$ = 33%, $s_{DISTRIBUTION}$ = 17%, $s_{PRODUCT}$ = 25%, $s_{CUSTOMER}$ = 100%, see Section 5.1) for 4-dimensional hierarchical clustering compared to other access techniques for table sizes up to one million pages.

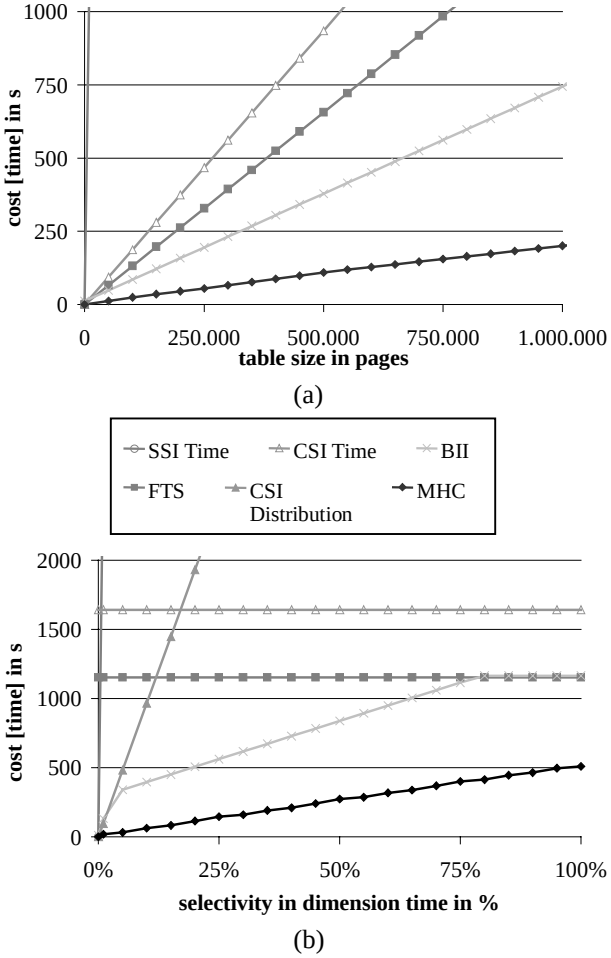



**Figure 5-2: Response times for queries with multi-attribute restrictions on 'Juice & More'**

Varying the selectivity of the restriction in TIME for a table size of $P$ = 878k pages (about 7GB for a page size of 8kB) shows that MHC is superior to both a SSI and a CSI on the TIME dimension, since these access methods cannot exploit any restriction but the one on TIME. MHC exploits the restrictions on DISTRIBUTION and PRODUCT in addition to the restriction on TIME. Thus MHC is also superior to an FTS and to BII using bitmap indexes for all four dimensions (Figure 5–2b). For a selectivity of 75% on TIME (i.e., an overall selectivity of 3,1875 %) an FTS is already preferable to BII. Since bitmap indexes do not cluster the data, the result set defined by the restrictions in all dimensions must be sufficiently small for BII to be competitive.

## 5.3 Simulation of Response Times for Queries with Multi-Attribute Restrictions and Sort Operations

Using the same parameters as in Section 5.2 and additionally using a main memory cache of 32 MB for the merge sort algorithm Figure 5–3 shows the cost [in s] for sorting the result set of a fact table defined by restrictions in multiple hierarchical dimensions.

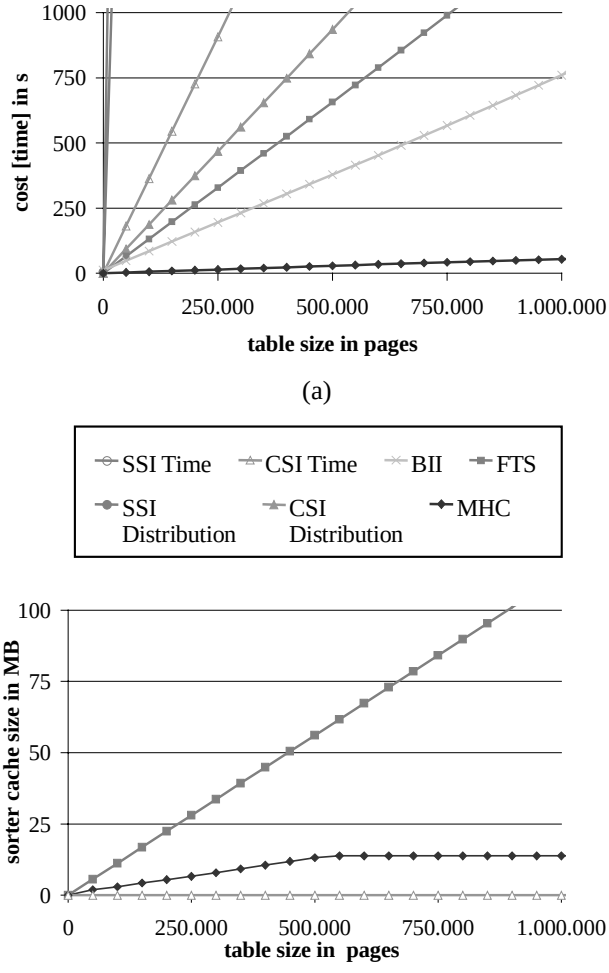



**Figure 5-3: Sorting the restricted 'Juice & More' fact table according to the TIME dimension**

Figure 5–3a shows the cost [in s], if the sales query of Section 5.1 is modified such that the sales are not calculated for the entire year, but aggregated by month. This query requires sorting the 4-dimensional query box by month to calculate the monthly sales. Again the table size is varied from one page to one million pages. The speed up of the MHC and Tetris grows superlinearly with increasing table size, since due to the result set size all other access methods require an external merge sort to calculate the monthly groups.

Figure 5–3b shows that the temporary storage for the merge sort algorithm used by FTS, BII, CSI distribution and SSI distribution soon exceeds the main memory sorter cache of $M$ = 32 MB. In contrast to caching the entire result set, sorting with Tetris only requires to cache one Tetris slice and never requires more than 14 MB of cache. Thus sorting with Tetris can take place in main memory.

A CSI or SSI on TIME does not require any sorter cache. The tradeoff of these two access methods is the inability to use restrictions in multiple dimensions. Overall, MHC and Tetris outperform any access method either with respect to response time or with respect to both response time and temporary storage requirements.

### 5.4 Further Analysis

Using our cost functions we found out that MHC and the Tetris algorithm are superior to one-dimensional access methods, unless a strongly preferred sort order exists or the restrictions are not selective enough to make up the tenfold speed of an FTS. A limitation of our technique is the number of dimensions: Increasing dimensionality exponentially reduces the potential of multidimensional space partitioning to create a total sort order in one dimension. Our theoretical and practical analysis [Mar99] shows that multidimensional indexes of up to 6 dimensions are handled very well with table sizes larger than 1 GB. These dimensionalities are typical for data warehousing applications. With larger table sizes even further attributes could be added to the MHC in order to speed up queries with restrictions or sorted processing in these attributes.

## 6. Performance Measurements

In this section we present measurements performed on the ‘Juice & More’ schema with our prototype implementation of MHC with UB-Trees on top of the commercial Oracle8 Server. For comparison reasons we also conducted measurements with native Oracle access methods: full table scan (FTS) and bitmap indexes (BII). The bitmap indexes were created on each hierarchy level. Secondary indexes are not included in our comparison, because earlier experiments with several queries of the TPC-D benchmark showed that they are neither competitive to UB-Trees nor to FTS or BII [MZB99].

### 6.1 Measurement Environment

The measurements were performed on a SUN Enterprise with four 300 MHz UltraSPARC processors and 2 GB RAM under Solaris 2.6. As secondary storage a partition on a SPARCstorage Array with Raid-Level 0 (6 disks striping, 5-6 MB/s transfer rate per disk) was used.

It is important to note that our implementation still causes significant overhead due to the fact that we have implemented the UB-Tree on top of a DBMS and not in the kernel itself. First, the number of SQL statements that have to be processed (UB: 1 statement for each page in the result set, Oracle methods: 1 statement in total) leads to extensive inter-process communication (about 30% of the total processing time) and DBMS overhead (e.g., parsing of statements). Second, our table is larger than the one for the FTS and the bitmap indexes due to unimplemented compressing techniques in the UB-Tree (for 8 KB pages: UB: 878362 pages, FTS: 723539 pages, BII: FTS+31134 pages).

### 6.2 Results

We present the results of 8 star joins on the ‘Juice & More’ schema which are classified according to the number of hierarchy levels restricted per dimension (see Table 6-1). For each query type we measured 2 instances which due to the non-uniform data distribution in the fact table result in different overall selectivities of the query.

**Table 6-1 Query Types and Selectivity of Measured Instances**

| | Customer | Product | Distribution | Time | Selectivity Instance 1 | Selectivity Instance 2 |
|---|---|---|---|---|---|---|
| Q1 | 2 | 2 | 0 | 1 | 0,0414% | 0,3176% |
| Q2 | 1 | 1 | 1 | 1 | 0,029% | 3,4383% |
| Q3 | 1 | 1 | 1 | 0 | 0,0182% | 2,0981% |
| Q4 | 0 | 1 | 0 | 1 | 6,0000% | 34,0000% |

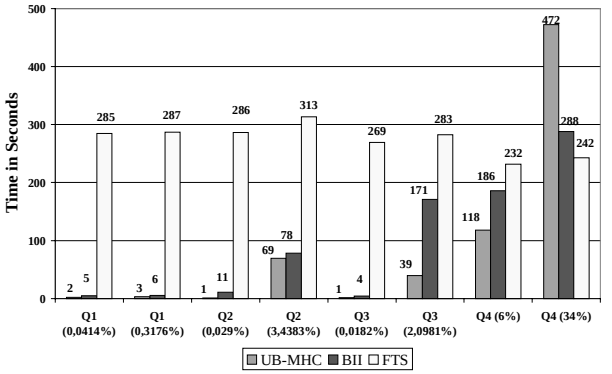



**Figure 6-1 Performance Results**

Figure 6-1 shows that multidimensional hierarchical clustering provides fast OLAP query processing in comparison to traditional techniques. In 7 of 8 queries our

prototype implementation of MHC was significantly faster (up to a factor of 10 for Q2(0,029%)). Already a restriction of 2 out of 4 dimensions suffices if the overall selectivity is low enough (i.e., 6% for Q4). However, for an overall selectivity of more than 10% an FTS in general is preferable to any index method - this observation also holds for MHC (see Q4(34%)). Further measurements which are not reported here underline following observation: MHC results in faster response time than the native methods if restrictions in multiple dimensions can be utilized and the overall selectivity is below 10%. In addition the resource requirements of MHC were significantly less than these of native access methods which makes it especially valuable in multi-user environments.

## 7. Conclusions and Future Work

We have defined an encoding scheme for hierarchical dimensions that enables clustering of data with respect to multiple hierarchical dimensions. MHC can be implemented with any suitable multidimensional access method. MHC reduces the number of random accesses to the fact table for star joins and other queries with restrictions in multiple hierarchies by a factor of about *C*, where *C* is the page size in tuples. In addition, sort operations as necessary for grouping and aggregation are performed on the fly without additional I/O. For dimensionalities typical for data warehousing only I/O-time linear in size of the result set prior to aggregation and sublinear temporary storage are necessary to aggregate parts of a data cube. Thus secondary storage space and pre-computation time for many aggregates and bitmap indexes can be avoided. In addition the widely discussed view maintenance problem is minimized. In our prototype implementation of MHC we use a UB-Tree with Z-ordering as the underlying multidimensional access method. The benchmark results for typical queries of a 7 GB real world retail data warehouse confirmed our analytical expectations and showed significant speedups up to factor 10 in response time. Depending on the query, temporary storage requirements for sorting are reduced by several orders of magnitude. Our clustering approach also holds not only for ROLAP but also for MOLAP implementations of a DW since both ROLAP fact tables and MOLAP data cubes can be clustered in this way.

In our future work we are particularly interested in doing tests in multi-user environments where we expect even more significant speedups. In addition we are investigating a methodology for query optimization with multidimensional indexes, both for heuristics-based and cost-based query optimizers.

## Acknowledgments

We thank our project partners SAP, Teijin Systems Technology, NEC, Hitachi and the European Commission for funding this research work. In particular we thank our master student Roland Pieringer for his effort in doing the performance measurements reported in this paper and SAP for providing the real data and the test environment.

## References

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

[Bay97a] R. Bayer. *The universal B-Tree for multidimensional Indexing: General Concepts*. World-Wide Computing and Its Applications '97 (WWCA '97). Tsukuba, Japan, 10-11, Lecture Notes on Computer Science, Springer Verlag, March, 1997.

[BK89] E. Bertino and W. Kim. *Indexing Technique for Queries on Nested Objects*. IEEE Transactions on Knowledge and Data Engineering, 1989, pp. 196-214.

[BKS+90] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. *The R*-Tree. An efficient and robust Access Method for Points and Rectangles*. Proc. of ACM SIGMOD Conf., 1990, pp. 322-331.

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

[BSH+98] M. Blaschka, C. Sapia, G. Höfling, and B. Dinter. *Finding Your Way through Multidimensional Data Models*. Proc. Intl. Workshop on Data Warehouse Design and OLAP Technology, Vienna, August 1998.

[CD97] S. Chaudhuri and U. Dayal. *An Overview of Data Warehousing and OLAP Technologies*. ACM SIGMOD Record 26(1), Marc 1997.

[CI98] C. Chan and Y. Ioannidis. *Bitmap Index Design and Evaluation*. Proc. of ACM SIGMOD Conf., 1998.

[FSG+98] M. Fang, N. Shivakumar, H. Garcia-Molina, R. Motwani, and J. Ullman. *Computing Iceberg Queries Efficiently*. Proc. of VLDB Conf., 1998, pp. 299-310.

[GG97] V. Gaede and O. Günther. *Multidimensional Access Methods*. ACM Computing Surveys 30(2), 1997.

[GHR+97] H. Gupta, V. Harinarayan, A. Rajaraman, and D. Ullman. *Index Selection for OLAP*. Proc. of ICDE, 1997.

[Gup97] H. Gupta. *Selection of Views to Materialize in a Data Warehouse*. Proc. of the Intl. Conference on Database Theory, Athens, Greece, January 1997.

[Gut84] A. Guttman. *R-Trees: A dynamic Index Structure for spatial Searching*. Proc. of ACM SIGMOD Conf., 1984, pp. 47-57.

[HR96] E.P. Harris and K. Ramamohanarao. *Join algorithm costs revisited*. VLDB Journal, 5, 1996.

[Jag90] H.V. Jagadish. *Linear Clustering of Objects with multiple Attributes*. Proc. of ACM SIGMOD Conf., 1990, pp. 332 – 342.

[Kim96] R. Kimball. *The Data Warehouse Toolkit*. John Wiley & Sons, New York. 1996.

[LS90] D. Lomet and B. Salzberg. *The hB-Tree: A Multiattribute Indexing Method with good*

*guaranteed Performance*. ACM TODS, 15(4), 1990, pp. 625 – 658.

[Mar99] V. Markl. *MISTRAL: Processing Queries with a Multidimensional Access Technique*. Ph.D. Thesis, TUM 1999.

[MB98] V. Markl and R. Bayer. *The Tetris-Algorithm for Sorted Reading from UB-Trees*. In "Grundlagen von Datenbanken", 10th GI Workshop, Konstanz, 1998.

[Moe98] G. Moerkotte. *Small Materialized Aggregates: A Light Weight Index Structure for Data Warehousing*. Proc. of 24th VLDB Conf., New York, USA, 1998.

[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.

[NHS84] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. *The Grid-File*. ACM TODS, 9(1), March 1984, pp. 38-71.

[OM84] J. A. Orenstein and T.H. Merret. *A Class of Data Structures for Associate Searching*. Proc. of ACM SIGMOD-PODS Conf., Portland, Oregon, 1984, pp. 294-305.

[OQ97] P. O´Neill and D. Quass. *Improved Query Performance with Variant Indexes*. Proc. of ACM SIGMOD Conf., Tucson, Arizona,1997, pp. 38-49.

[Sal88] B. Salzberg. *File Structures: An Analytic Approach*. Prentice Hall, 1988.

[Sam90] H. Samet. *The Design and Analysis of Spatial Data Structures*. Addison Wesley, 1990

[Sar97] S. Sarawagi. *Indexing OLAP data*. Data Engineering Bulletin 20 (1), 1997, pp. 36-43.

[SDN+96] A. Shukla, P. Deshpande, J. Naughton, and K. Ramasamy. *Storage Estimation for Multidimensional Aggregates in the Presence of Hierarchies*. Proc. of 22nd VLDB Conf., Mumbai (Bombay), India, 1996.

[SDN+98] A. Shukla, P. Deshpande, and J. Naughton. *Materialized View Selection for Multidimensional Datasets*. Proc. of ACM SIGMOD Conf., 1998.

[SRF87] T. Sellis, N. Roussopoulos, and C. Faloutsos. *The R+-Tree: A Dynamic Index for Multi-Dimensional Objects*. Proc. of 13th VLDB Conf., Brighton, England, 1987, pp. 507-518.

[WB97] M.C. Wu and A.P. Buchmann. *Research Issues in Data Warehousing*. BTW'97. 1997.

[WB98] M.C. Wu and A.P. Buchmann. *Encoded Bitmap Indexing for Data Warehouses*. Proc. of ICDE, Orlando, 1998.

[Wid95] J. Widom. *Research Problems in Data Warehousing*. Proc. of 4th CIKM, November 1995.

[ZSL98] C. Zou, B. Salzberg, and R. Ladin. *Back to the Future: Dynamic Hierarchical Clustering*. Proc. of ICDE, 1998, pp. 578-587.