Transbase®: A leading-edge ROLAP Engine supporting multidimensional Indexing and Hierarchy Clustering[∇]

R. Pieringer[1], K. Elhardt[1], F. Ramsak[2], V. Markl[3], R. Fenk[2], R. Bayer[2]

[1] Transaction Software GmbH
Thomas-Dehler-Str. 18
D-81737 München
{pieringer,elhardt}@transaction.de

[2] Bayerisches Forschungszentrum für Wissensbasierte Systeme
Boltzmannstr. 3
D-85747 München
{ramsak,fenk}@forwiss.de, bayer@in.tum.de

[3]IBM Almaden Research Center, K55/B1,
650 Harry Road, San Jose, CA 95120-6099
marklv@us.ibm.com

Abstract: Analysis-oriented database applications, such as data warehousing or customer relationship management, play a crucial role in the database area. In general, the multidimensional data model is used in these applications, realized as star or snow-flake schemata in the relational world. The so-called star queries are the prevalent type of queries on such schemata. All database vendors have extended their products to support star queries efficiently. However, mostly reporting queries benefit from the optimizations, like pre-aggregation, while ad-hoc queries usually lack efficient support. We present the DBMS Transbase® in this paper, which provides a new physical organization of the data based on hierarchical clustering and multidimensional clustering combined with multidimensional indexing. In combination with new query optimizations (e.g., hierarchical pre-grouping) significant performance improvements are achieved. The paper describes how the new technology is implemented in the Transbase® product and how it is made available to the user as transparently as possible. The benefits are illustrated with a real-world data warehousing scenario.

1 Introduction

Data warehousing (DW), online analytical processing (OLAP), and customer relationship management (CRM), have become a major market in the database area through the last decade. The multidimensional paradigm seems to be the undisputed winner as a design choice for such databases. The conceptual model adopted is a data warehouse consisting of facts (or measures) organized into a set of dimensions, which in turn are organized into levels of different aggregation granularity (i.e., detail) that comprise one or more hierarchies. Even though proprietary multidimensional database management systems (DBMSs) exist, the vast majority of systems use relational DBMSs as the underlying storage system.

For relational databases, the multidimensional data warehouse consists of one or more *star schemata* [CD97a], featuring a central fact table

[∇] This work is funded by the European Union under the IST-Project EDITH (IST-1999-20722)

surrounded by so-called dimension tables. The most prevalent kind of queries submitted to such a system is the *star query*. Star queries impose restrictions on the dimension values that are used for selecting specific facts; these facts are further grouped and aggregated according to the user demands. The join of the central (and usually very large) fact table with the surrounding dimension tables (also known as a *star join*) has been identified as frequent, major bottleneck in evaluating such queries. Various solutions have been proposed over the years to cope with these problems. Indexing schemes [NG95, NQ97, Sar97, CI98, WB98, Wu99, WOS01] and precomputation of aggregation results [GM95, Rou98, Sri96] have been studied extensively in the research community and are also, to some extent, used in commercial systems [ACN01, Ora01, Zah00].

While these solutions work well in reporting scenarios, they do not support acceptable performance for *ad hoc* star queries, i.e., queries that are not known in advance, which become more and more important in online applications. For this kind of queries the usage of precomputed aggregation results is extremely limited or even impossible in some cases. Even when elaborate indexes are used, due to the arbitrary ordering of the fact table tuples, there might be as many I/Os as there are tuples in the result set.

To overcome these deficiencies, new alternatives of the physical organization of data have emerged [Des98, MRB99, KS01]. The idea is to incorporate the two fundamental properties of the conceptual data model into the data storage: the multidimensionality and hierarchy semantics. These organizations exploit a special kind of key that is based on the hierarchy paths of the dimensions, in order to achieve hierarchical clustering of the facts. This physical clustering results in a reduced I/O cost for the majority of star queries, which are based on the dimension hierarchies. Moreover, [MRB99] and [KS01] exploit a clustering, multidimensional index for storing the tuples. A typical star join then is transformed into a multidimensional range query, which is very efficiently computed using the underlying multidimensional data structures.

In this paper we present the Transbase® DBMS from Transaction Software GmbH that incorporates state of the art techniques for analysis-centric applications. More precisely, it supports the UB-Tree as native multidimensional index and allows for clustering of data according to hierarchy semantics. Taking advantage from the knowledge of hierarchies, not only the physical storage of the data can be optimized but also query processing can largely be improved. We will discuss the basic concepts of the underlying technology as well as how it is implemented in the DBMS kernel. The evaluation in a real-world data warehousing scenario shows significant performance improvements over traditional techniques.

The rest of the paper is organized as follows. Section 2 covers related work and in Section 3 we introduce the basic technology used in Transbase®. Section 4 shows the user's view with an example before we address the implementation issues in Section 5. Section 6 covers the automatic maintenance of hierarchy clustering and Section 7 is dedicated to the query processing for multidimensional hierarchical clustering. In Section 7 we provide an evaluation of the techniques in a real-world scenario; Section 9 summarizes our contribution.

2 Related Work

Due to the large body of work in the area of data warehousing and optimizing DBMSs for these applications we can only give an incomplete account of related work in this field. We cover general approaches first and then address commercial solutions.

2.1 Star query optimization

One of the most important parts of a star query is the processing of the star join. Star join processing has been studied extensively and specific solutions have been also implemented in commercial products. See [CD97b] for an overview.

To compute the star join, most systems avoid building the Cartesian product of the fact table with the dimension tables as the resulting cardinality leads to a non-tolerable overhead. Thus, one tries to apply dimension restrictions also to the fact table in order to reduce the join size. Bitmap indices are often used to speed up the access to the fact table. The bitmaps corresponding to the different dimension values are ANDed or ORed depending on the selection condition. The resulting bitmap is used to extract tuples from the fact table [NG95, NQ97]. When the query selectivity is high, only a few bits in the result bitmap are set. If there is no particular order (clustering) among the fact table tuples, we can expect each bit to access a tuple in a different page.

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. The importance of good physical clustering in OLAP has been shown in [KR98], where packed R-trees are exploited for storing the results of the data cube operator ([Gra96]). In [Des98], the benefits of hierarchical clustering for star queries was observed as a side effect of using a chunked file organization for enabling caching with chunk as the caching unit.

Among others, in [MRB99] the UB-tree (see Section 3.1) is used as a primary organization of the fact table. Surrogate keys based on the dimension hierarchies are exploited and hierarchical clustering of the fact table is achieved. Consequently star joins are transformed to multidimensional range queries. The combination of the two mechanisms results in a greatly reduced I/O cost for star joins.

In [KS01] a physical organization based on a hierarchical chunking of the fact table is presented. Fact data are clustered physically according to the dimensional hierarchies. To achieve this clustering, special path-based dimension keys are exploited. In particular, these keys guide the clustering (called *chunking*) process. Star joins are transformed to range queries in the multidimensional and multi-level data space of a cube. The adopted multidimensional structure is a variant of the Grid File [NHS84].

Several aspects of processing and optimizing star join queries on hierarchically clustered fact tables are also presented in [TT01]. The paper considers a star schema with UB-Tree organized fact tables and dimension tables stored sorted on a composite surrogate key. For a particular class of star join queries, the authors investigate the usage of sort-merge joins and a set of other heuristic optimizations.

A general query processing framework that addresses all issues involved in star query processing over hierarchically clustered fact tables has been presented in [Kar02] and [Pie03]. In this paper, we describe in more detail, how this framework is implemented in a real DBMS. We extend the concepts of pre-grouping, first introduced in [CS94], [YL94] and [YL95] by including hierarchical information.

2.2 Hierarchies in Oracle

Hierarchies in Oracle are not used, in order to cluster data in the fact table ([Ora01]). Hierarchies are additional information (meta data) for special features for query processing. For example, query rewriting needs hierarchical information to use materialized pre-aggregated views in the query formulation. Thus, the definition of dimensions and hierarchies does not influence the physical clustering of the data.

A dimension, created by a `CREATE DIMENSION` statement contains one or more hierarchies. The hierarchy levels can be placed in any table. For each hierarchy level, one or more feature attributes can be assigned (via the `DETERMINES` clause). Hierarchies can be modified by adding or dropping hierarchy levels.

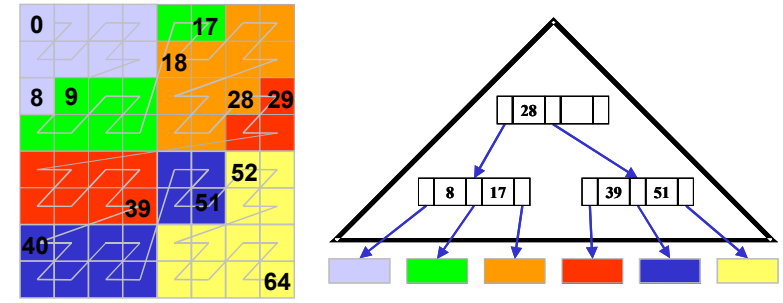
3 Basic Concepts

In this section we briefly introduce the fundamental concepts that are used in the Transbase® DBMS to support OLAP applications. On the one hand, we introduce the multidimensional index available in Transbase® and on the other hand explain the basic idea of hierarchy clustering.

3.1 The UB-Tree

We just give a short introduction to UB-Trees here, details can be found in [Bay97, Ram00]. The basic idea of the UB-Tree is to use a space-filling curve to map a multidimensional universe to one-dimensional space. Using the Z-curve for preserving multidimensional clustering it is a variant of the zkd-B-Tree [OM84]. A Z-address $\alpha = Z(x)$ is the ordinal number of the key attributes of a tuple x on the Z-curve, which can be efficiently computed by bit-interleaving. A standard B-Tree is used to index the tuples taking the Z-addresses of the tuples as keys. The pagination of the B-Tree creates a disjunctive partitioning of the multidimensional space into so-called Z-regions. This allows for very efficient processing of multidimensional range queries.

Figure 1 shows a Z-region partitioning for a two-dimensional universe and the corresponding B-Tree. The interval limits of the Z-regions are also depicted.



Z-region partitioning of 2-D space **B+-Tree with Z-values**

Figure 1 UB-Tree: Z-region partitioning and underlying B-Tree

The processing of basic operations, i.e., insertion, deletion, update, and point query, of the UB-Tree are analogous to the basic operations of the B-Tree. For each tuple the corresponding Z-address is computed, and with the resulting value the underlying B-Tree is accessed. Thus, all basic operations require only cost proportional to the height of the tree. The only recommendable modification to the standard B-Tree algorithms is an adaptation of the split algorithm to achieve a "good" (as rectangular as possible) Z-region partitioning.

A UB-Tree is especially good in processing multidimensional range queries, as it only retrieves all Z-regions that properly intersect the query box. Consequently, it usually shows the nice property that the response time of the range query processing is proportional to the result set size.

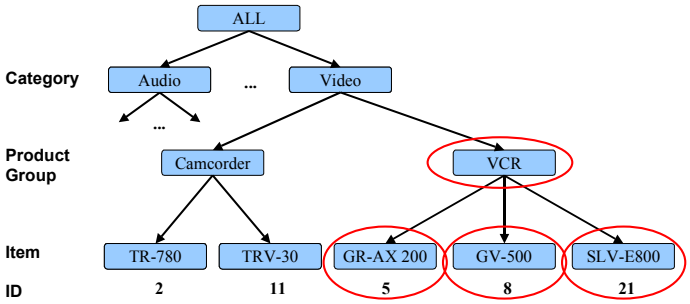
3.2 Clustering of Hierarchies

Hierarchies play an important role in various application domains. They are used to provide a semantic structure to data, e.g., a geographical classification of customers in a data warehouse. As the hierarchies cover the application semantics they are used frequently by users to specify the restrictions on the data as well as the level of aggregation. The restrictions on the hierarchies usually result in point or range restrictions[1] on some hierarchy levels [Sar97]. The problem that arises is that these restrictions on upper hierarchy levels lead to a large set of point restrictions on the lowest level, i.e., the level with the most detailed information. This situation is depicted in Figure 2 (a): restricting the level 'Product Group' to the value 'VCR' leads to the set of ids {5,8,21} and not to the interval [5,21], as the item with id 11 does not belong to the specified product group.

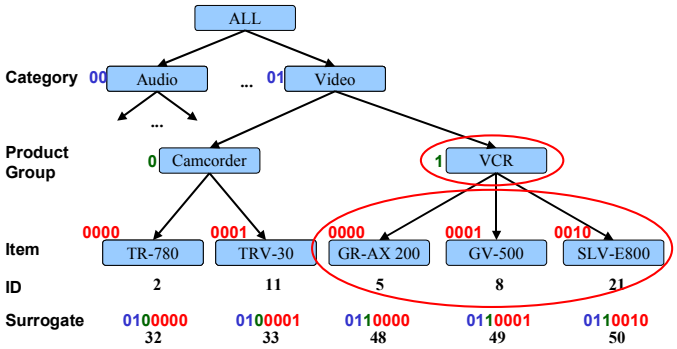
For most access methods it would be more efficient to process one range restriction instead of a set of point restrictions. The resulting question is how to map a point/range restriction on a higher hierarchy level to a range restriction on the lowest level? To this end, Transbase® applies the clustering scheme for hierarchies as proposed in [MRB99]. A special kind of keys is used for the elements of the lowest level which reflect the hierarchy semantics, i.e., keys which adhere to the partial order defined by the hierarchy levels. These so-called *(compound) surrogates* guarantee

[1] Range restrictions on hierarchy levels are only meaningful if an order on the elements of the hierarchy level is defined.

that the keys of all elements in a sub-tree of the hierarchy lie within a closed interval (Figure 2 (b)) such that a key of an element not lying in the subtree is not within the interval. In our example, the restriction to the product group 'VCR' now leads to the interval [48,50]; the item with id 11 is mapped to the surrogate 33 that does not violate the interval.



(a)



(b)

Figure 2 Example of hierarchy clustering: (a) non-clustered vs. (b) clustered hierarchy

We refer to this technique as *hierarchy clustering (HC)* from now on. If we combine HC and multidimensional indexing on multiple hierarchy encoding as it is done in Transbase®, then we speak of *multidimensional hierarchical clustering (MHC)*.

4 Example: MHC in Transbase®

In this section, we will briefly present the user's perspective when using the OLAP functionality in Transbase® ([Tra02]).

For our discussions we use a conventional star schema [CD97a] with a fact table consisting of *dimension* (qualitative) and *measure* (quantitative) attributes [Kim96]. For the dimensions typically one or more hierarchical classifications based on the dimension attributes (often referred to as *features*) exist. The primary key of the dimension represents the most detailed level of the dimension hierarchies.

In this paper, we focus on star schemata for the ease of description. However, the algorithms also are implemented for general snowflake schemata.

4.1 Sample Schema

As running example throughout this paper we use the schema depicted in Figure 3. This data warehouse stores sales transactions recorded per item, store, customer, and date. It contains one fact table *FACT*, which is defined over the dimensions: *PRODUCT*, *CUSTOMER*, *DATE,* and *LOCATION* with the obvious meanings. The measures of *FACT* are *price, quantity* and *sales* representing the values for an item bought by a customer at a store at a specific day. The schema of the fact and dimension tables is shown in Figure 3 and the dimension hierarchies are depicted in Figure 4.

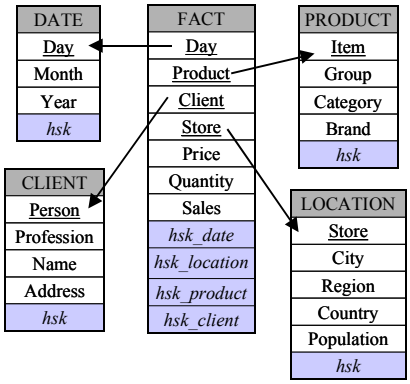


Figure 3: Sample Schema: standard star schema and HC extension (*hsk**)

The dimension *DATE* is organized in three levels: day – month – year. The dimension *CUSTOMER* is organized in two levels: customer – profession. For each customer the dimension table contains an ID, a name, an address, and a profession. The dimension has two hierarchical attributes (*person_id, profession*) and two feature attributes (*name, address*). The *LOCATION* dimension is organized by four levels: store – city – region – country. Stores are grouped into cities, these are grouped into regions and the regions finally are grouped into countries. For each city, the population is stored as feature attribute. The dimension has four hierarchical attributes (*store, city, region, country*) and one feature attribute (*population*) that is assigned to the city level.

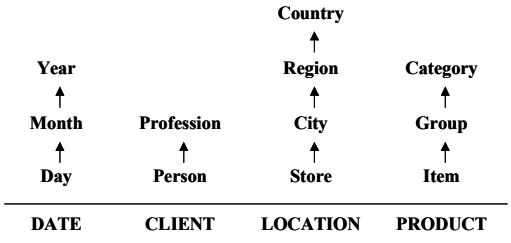


Figure 4: The dimension hierarchies of the example

Finally, the *PRODUCT* dimension is organized into three levels: item – group – category. Items are grouped into product groups and those are further grouped into categories (e.g., "air condition"). The attribute *brand* characterizing each item is a feature attribute.

Star queries are written in standard SQL, i.e., the join attributes between the fact and dimension tables are the dimension key attributes:

```
SELECT SUM(sales)
FROM FACT F, DATE D, PRODUCT P , LOCATION L
WHERE D.day=F.day AND P.item=F.product AND
L.store=F.store AND D.year=2002 AND P.category = 'Air
Condition' AND L.population > 1000000
GROUP BY D.year, D.month
```

This query returns the sum of sales for the year 2002 for air conditions in cities with a population larger than one million.

4.2 The User's View: Hierarchy Specification by extended DDL

To allow the user for defining such a schema, the DDL has been extended to express hierarchies and the desired physical organization of the fact table according to the dimension hierarchies. This is achieved by specifying an additional field per dimension table per hierarchy that basically represents the compound surrogate derived by HC as described earlier.

The keyword `SURROGATE` is used to mark the definition of a surrogate field as opposed to the definition of a standard user visible field. In the dimension table, we denote a compound surrogate by the keyword `COMPOUND`. The hierarchy levels are specified after this keyword by enumerating the hierarchy levels from top to bottom. The maximum fan-out of each hierarchy level is denoted by the keyword `SIBLINGS` (to specify the number of bits reserved for the hierarchy level) after the corresponding hierarchy level.

Figure 5 shows the DDL for the Location dimension: the surrogate specification defines the hierarchy depicted in the Figure 4. The `SIBLINGS` information specify that in this hierarchy there are at most 10 countries, at most 50 regions per country, at most 50 cities per region, and at most 1000 stores per city. Thus, the Location dimension may at most contain 10*50*50*1000=25.000.000 members.

```
create table Location (
       country char(*) NOT NULL,
       region char(*) NOT NULL,
       city char(*) NOT NULL,
       store char(*) NOT NULL,
       SURROGATE cs_location COMPOUND (
       country SIBLINGS 10, region SIBLINGS 50,
       city SIBLINGS 50, store SIBLINGS 1000),
       PRIMARY KEY (store)
);
```

Figure 5 Extended Create-Statement for dimension tables

For the fact table specification (see Figure 6), we now have to specify the physical organization, besides the standard relationships to the dimension tables. As we want to cluster the data according to the hierarchies of the

dimension, we use the surrogates instead of the "logical" keys. To this end, we introduce the concept of reference surrogates. Technically, a reference surrogate is an additional system maintained field in the fact table. Because of the necessary foreign key constraints in the fact table, it is possible to decide to which dimension the reference surrogate belongs.

We use again the keyword `SURROGATE` to denote a surrogate. Then we use the keyword `FOR`, in order to assign the reference surrogate to a dimension:

```
create table fact (
   product char(*) NOT NULL references product (item),
   store char(*)NOT NULL references location(store)
   time integer NOT NULL references date(day),
   sales numeric(10,3),
   price numeric(10,3),
   quantity numeric(10,3),
   SURROGATE cs_prod FOR product,
   SURROGATE cs_store FOR store,
   SURROGATE cs_time FOR time,
   PRIMARY HCKEY (cs_prod, cs_store, cs_time))
```

Figure 6 Extended Create-Statement for the fact table

A different keyword for the key specification the UB-Tree (`PRIMARY HCKEY`) is used to specify the index access method, namely a UB-Tree (HC stands for HyperCube as the UB-Tree is called in Transbase®).

All further statements (especially `INSERT` and `UPDATE`) may (and must) ignore the additional fields. This is comparable to the creation of a secondary index which is made up by the user but then becomes a system maintained part of the database.

It is important to note that all fields created by the `SURROGATE` specification are system maintained and are not visible to the user. Even though the fields are really stored in the tables, the user only works on a view of the table which projects the surrogate fields out.

As the example shows, the user can very naturally specify the physical organization of the data according to the schema semantics. In the following sections we will discuss the internals of the system transparent to the user.

5 Implementing HC in the DBMS Kernel

For the implementation of HC in the Transbase® kernel various issues have to be solved. The most important one is the internal representation and management of the surrogates as well as schema extensions, necessary for the automatic processing.

Before we continue, we want to introduce some terminology that is frequently used in the remainder of the paper. We refer to the fact table as *FT* and to the dimension tables as D_i. We use *FT.m* to denote a measure attribute of the fact table, $D_i.h_j$ to denote a hierarchical attribute (h_1 denotes the leave level of the hierarchy, i.e., it is the key of the dimension table), and $D_i.f_k$ denotes a feature attribute of the dimension. *PRED* and *AGGR* are placeholders for any predicate resp. aggregation function on the specified attribute.

5.1 Internal Representation of Compound Surrogates

As already indicated in Figure 3, the `SURROGATE` specification in the "create table" statement leads to the creation of an extra, non-visible, field in the dimension or fact table containing the encoding (*surrogate key, "hsk"*) of the corresponding hierarchy. It is unique for each dimension tuple as each leaf member of the hierarchy is assigned a unique value by HC.

The *hsks* are also contained in the fact table, as there they are required for the physical clustering of the data. As the logical dimension keys and the *hsks* are substitutes to each other, one may save a lot of space in the fact table if one would suppress the logical keys and just keep the *hsks*, if available for a dimension. However, this may lead to drastic performance decrease in cases where the original keys of the fact table tuples are accessed in the query. This would lead to expensive joins with the dimension tables. The suppression of logical keys is not implemented in Transbase®.

Internally, compound surrogates are fixed length bit strings. The length corresponds to the siblings specification of all surrogate components.

5.2 System Catalog Extension

In order to implement hierarchical clustering the system needs to have knowledge about the defined surrogates. The system catalog extension is designed to even allow several compound surrogates (and thus hierarchies) within one table.

5.3 Automatic Index Creation

For efficient query retrieval and surrogate maintenance, we need two special secondary indexes on the dimension table. Let *cs* be the field name of the compound surrogate and $h_t,..,h_1$ be the list of field names of the levels for the compound surrogate definition. Transbase® automatically creates the following indexes (here described by the following virtual SQL `CREATE` statements):

```
CREATE INDEX "@@sys_surrCSX_<surrid>" ON <dim_table>
(cs);
CREATE INDEX "@@sys_surrHX_<surrid>" ON <dim_table>
(h_t,..,h_1,cs);
```

Thus, the index names consist of two components, a prefix that marks the index as system index, and a generated suffix of the kind of the index (*CSX* or *HX*) and the surrogate id of the corresponding surrogate. The indexes are needed for the computation of compound surrogates, for the lookup of reference surrogates (see section 6.2) and for query processing. Of course, these indexes cannot be dropped by the user.

5.4 Multiple Hierarchies

A dimension of a data warehouse may include several independent hierarchies. Because of the representation of hierarchies by compound surrogates, we have to deal with several compound surrogates, one for every possible hierarchy.

An example of several hierarchies is shown in Figure 7. The customer dimension has a geographical hierarchy with country – region – town – customer and an organizational hierarchy with profession – customer.

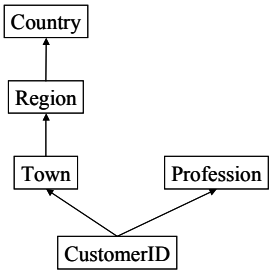


Figure 7: Customer Dimension with two Hierarchies

We have to distinguish between two levels of hierarchies. The conceptual level defines hierarchies on the conceptual data warehouse schema. Depending on the data warehouse application, multiple hierarchies may be defined on all dimensions representing the application data model. Usually hierarchies represent drill and aggregation paths for user queries.

The physical level of hierarchies is responsible for the clustering properties of the hierarchies, in combination with hierarchies of other dimensions, i.e., the complete MHC schema. The number of clustering hierarchies is restricted due to the properties of the clustering multidimensional index. It usually makes sense to use only one hierarchy per dimension for clustering, because in most times, hierarchies are dependent or one hierarchy is more important for user queries. However, in some cases, two or more hierarchies may be required for clustering (e.g., most user queries restrict both of these hierarchies). In addition, one dimension table may be used for several fact tables, that use different hierarchies for clustering. Thus, we have to provide multiple hierarchies for one dimension.

The internal structures allow to establish an arbitrary number of hierarchies, represented by compound surrogates. However, we require that the leaf level of all hierarchies is the same (a so called shared leaf level), usually the primary key of the dimension table. This hierarchy property is checked when creating the table and the compound surrogates in the DDL statement. With multiple hierarchies allowed on one dimension table, we can use the dimension table for several fact tables that can be clustered w.r.t. different hierarchies. So we avoid redundancy problems for replicated dimension tables.

Every compound surrogate is assigned a unique id. This so called *surrid* is referred to by the reference surrogates. One fact table includes an arbitrary number of reference surrogates specified by the *surrid* of the corresponding compound surrogate of the dimension tables. Thus, we can use reference surrogates of several hierarchies of one dimension table within one fact table. These reference surrogates may be used as index key attributes and thus for clustering the fact table according to several hierarchies.

6 Maintenance of MHC

After discussing the internal representation of the surrogates, we now turn to the automatic maintenance of the hierarchical clustering. We start with the insertion of new dimension members and continue with the insertion of fact table tuples. Finally, we will address the issue of major reorganization of dimension hierarchies.

6.1 Computation of Compound Surrogates

Computation of a compound surrogate *hsk* occurs when a tuple is inserted into the dimension table. Updates of hierarchy fields also may lead to a re-computation of *hsk*.

In the following picture, the insertion algorithm is depicted for our example product hierarchy. Figure 8 schematically shows the insertion of a new tuple. We assume the values v_i for each hierarchical field h_i. Considering the hierarchy, the insertion of a tuple means the creation of a new path $(v_k,\ldots,v_1)$ in the dimension D.

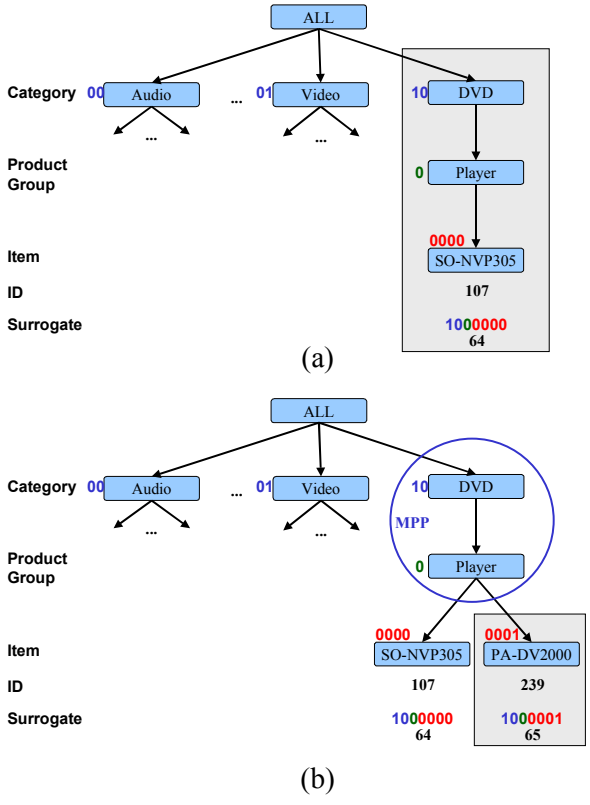


Figure 8: Insertion of new tuples into a dimension

For the computation of the compound surrogate *hsk* we have to check if there exists already a prefix of the new path in *D*. For a tuple to be inserted, we call the already existing part of the new path the *matching prefix path* (MPP). The MPP may be empty as in Figure 8 (a) – in this case a new root element, here "DVD", has to be created and the forest grows by one tree.

A non-empty MPP (see Figure 8 (b)) comprises levels (h_1,...,h_k) for some k with $k > 1$ and $k \leq t$. The number k then is called the *match level* of the new tuple's path (2 in our example). At the next lower level, i.e., the first non-matching level, the surrogate for the new value is determined. Usually, the maximum surrogate is incremented by one, but one may also use different schemes to compute the surrogate, for example if one wants to reuse surrogates from deleted elements. According to the `SURROGATE` definition the single surrogate values are concatenated to build the compound surrogate *hsk*.

6.2 Insertion into the Fact Table: Lookup of Reference Surrogates

An insertion of a tuple into the fact table specifies the key dimension attributes (the dimension attributes of the leaf hierarchy level). For each dimension, however, the corresponding compound surrogate must be found.

This so called lookup is to be performed before the insertion of the tuple into the fact table because the tuple has to be extended by the corresponding compound surrogates for the dimensions (reference surrogates). For every dimension d_i, a direct access to the dimension table is performed to retrieve the corresponding *cs*.

Depending on the number of dimensions, a number of B*-Tree accesses is necessary and thus will decrease the insert performance. Especially for bulk loading, such a procedure may cause long insertion times, because the caching of the dimension tables may be not optimal. Thus, an alternative lookup concept has been implemented in Transbase® by loading a hash table with the dimension keys and corresponding compound surrogate for every dimension before inserting the fact table tuples. Then the lookup is very efficient, because only main memory accesses will be necessary in the hash tables.

6.3 Hierarchy Reorganization

Usually, dimension hierarchies are very stable. However, there are various application domains, where frequent hierarchy reorganizations, e.g., moving of sub-trees, occur and should be reflected in the data organization. Such operations require additional support by the DBMS as local operations on dimension tables now have immediate influence on the organization of the fact table. These operations are usually much more expensive. For example, reclassifying one product group to a different category will cause changes in the *hsks*. In order to have a correct physical clustering of the fact table all fact tuples corresponding to the changed product group have to be updated. Of course, arbitrary hierarchy reorganizations by data updates are supported, but it has to be kept in mind that the necessary fact table updates may be very time critical.

We have implemented similar methods as for star query processing, in order to support hierarchy reorganization and the corresponding fact table reclustering as good as possible. For example, we optimized multi-query box algorihtm methods that are necessary, if a number of hierarchy paths change and require corresponding reorganization of the fact table records. An alternative method is to delay the reorganization of the fact table until a background process uses idle time of the warehouse application for the reclustering. In this case, we introduce two h-surrogates, where one

contains the previous (hsk_1) and the other (hsk_2) contains the new value. hsk_1 is updated, when the corresponding fact table records are reorganized. If hsk_1 and hsk_2 both have the same value, no reclustering is necessary.

7 Query Processing with MHC & UB-Trees

In this section we discuss the basic star query processing for DW schemata with MHC and multidimensional index organization and its advantages. For the discussion in this paper, we restrict ourselves to standard "star queries".

7.1 Query Template

Figure 9 shows the SQL query template for simple ad hoc star-queries. The notation {X} represents a set of X objects. The template defines typical star queries and uses abstract terms that act as placeholders. Note that the queries that conform to this template generally have a structure that is a subset of the above template and they instantiate all abstract terms.

Our template is applied on a schema similar to the one in Figure 3, which is a typical star schema. It specifies the restrictions on the various dimensions, the star-join between the fact table, and the required dimension tables and the subsequent grouping and aggregation. In general any attribute (hierarchical, feature, or measure) can appear in a `GROUP BY` clause. However, most queries group the result by a number of hierarchical and/or feature attributes. Finally, there is an `ORDER BY` to control the order of the presented results.

```
SELECT      {D.h},{D.f},{FT.m},
            {AGGR(FT.m) AS AM}, {AGGR(D.h) AS AH},
            AGGR(D.f) AS AF}
FROM        FT,{D1},…,{Dn}
WHERE       FT.d1 = D1.h1  AND
            FT.d2 = D2.h1  AND      } Star-Join condition
            ...
            FTdn = Dn.h1   AND

            PRED(D1)   AND          } „local“ restrictions on the
            PRED(D2)   AND          } dimension tables
            ...                     }
            PRED(Dn)   AND          }
                                    } restrictions to measures
            PRED({FT.m})            }
GROUP BY    {D.h},{D.f},{FT.m}
HAVING      PRED({AM},{AH},{AF})
ORDER BY    <ordering fields>
```

Figure 9: Star Query Template

7.2 Star Query Processing with MHC

Processing of star queries as described above can roughly be divided into three major steps:

(1) Evaluation of dimension predicates

(2) Fetching result tuples from the fact table

(3) Residual joins, grouping and aggregation, sorting

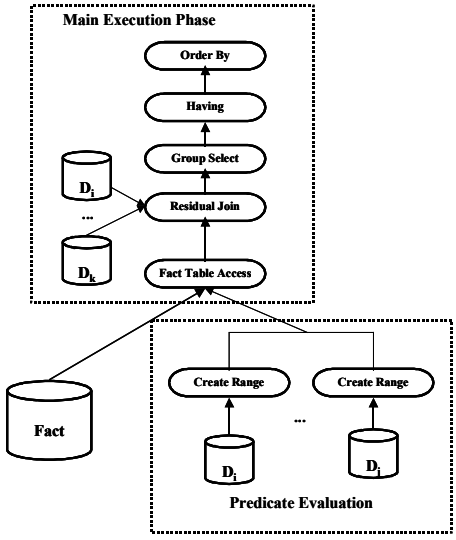


Figure 10: Standard Abstract Execution Plan

The first step evaluates the predicates on the dimension tables. The second and third step are often considered together as not all restrictions can be only evaluated on the fact table. In the following, we speak of two processing phases:

- Predicate Evaluation
- Main Execution Phase

In the presence of MHC and a multidimensional index for the organization of the fact table new optimizations can be applied to these phases. This is reflected already in the abstract execution plan (AEP) [Kar02] of Figure 10.

7.3 Predicate Evaluation and Fact Table Access

In the predicate evaluation phase the first benefit of HC can be observed. Instead of generating a large set of point restrictions, the local predicates on the dimensions usually result in a (small) set of range restrictions. This is especially true for predicates with hierarchical restrictions (cf. Figure 2 (b)) but also many feature restrictions will lead to ranges if there is some dependency between the defined hierarchy and the feature values. The details of the interval generation are discussed in [Kar02].

After the generation of the intervals per dimension, the combination of intervals is transformed into a number of query boxes that are executed by the *Fact Table Access*. At that point, the multidimensional organization of the fact table yields a major cost saving: relying on the clustering and the

efficient range query algorithm of the UB-Tree the number of I/O to fetch the required data is drastically reduced.

7.4 Main Execution Phase: Grouping and Aggregation

The Main Execution Phase joins the tuples that are fetched from the fact table with all necessary dimension tables (*Residual Join* operator). After the residual join the tuples are grouped (*Group-Select* operator), filtered again (*Having* operator) and sorted (*OrderBy* operator). This phase strongly benefits from the additional information which is encoded in the fact table tuples via the reference surrogates. The reference surrogates represent an encoding of the hierarchy path of each level member – therefore important optimizations can be realized for GROUPing on dimension fields. Especially, in many cases, a great deal of the grouping work can be done locally on the fact table tuples using prefixes of the reference surrogates.

The important point is that this pre-grouping step is done before the fact table tuples are joined with dimension tables. As the grouping operation typically greatly reduces tuple cardinality, the cost for the successive join operation is also greatly reduced. This technique is called *Hierarchical Pre-grouping*. In detail, if the GROUPing field belongs to hierarchy level h_k or is functionally dependent on it then the h-surrogate prefix $h_t/h_{t-1}/.../h_k$ is dynamically computed and the GROUP operation is done on that value.

A simple example illustrates the effects of the hierarchical pre-grouping method: Assume we have a DW with a time dimension (besides other dimensions) categorized by year – month – day and a well populated fact table. A query restricting the result to year 2001 (besides other restrictions) qualifies 100.000 fact table records. If the result has to be grouped w.r.t. month, we have to join 100.000 records with the time dimension table. When applying hierarchical pre-grouping, the number of join operations is reduced by a factor of 30, because all days of one month are aggregated to the month.

Our measurements with a five dimensional real world DW show an average reduction of the join cardinality by more than a factor of 100. This leads to an overall speedup of a factor of 4 to 7 for the grouping and residual join execution phase.

Figure 11 shows the final abstract execution plan used in Transbase®. Due to the effective pre-grouping steps, the costly overall residual join step is avoided if possible.

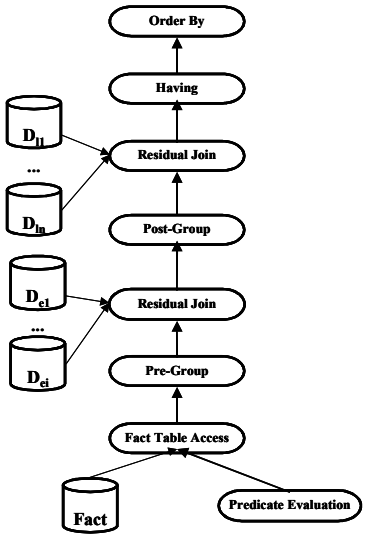


Figure 11: Abstract Execution Plan with Pre-Grouping

We do not enlarge on this optimization in this paper. For detailed information about hierarchical grouping please refer to [Pie03].

8 Evaluation of MHC

In this section we briefly want to illustrate the benefits of MHC and multidimensional indexing in the context of a real-world data warehouse application.

The measurements are performed on a two processor PC Pentium III, 750 MHz, with 256 MB RAM and 30 GB IDE hard disk. We used Windows 2000 as operating system and Transbase® with the MHC implementation as DBMS.

The DW schema of a large electronic retailer consists of a fact table with three dimensions *CUSTOMER*, *PRODUCT* and *DATE* and 3 measures: *quantity*, *value,* and *unit_price*. The *CUSTOMER* dimension contains 1,4 million records, *PRODUCT* consists of 27.000 products and the *DATE* dimension covers 7 years on day granularity. 15.543.380 records are stored in the snapshot of the fact table, amounting to 1,5 GB.

The query workload consisted of 220 ad hoc star queries from a real-world application. We classified the queries into three groups according to their selectivity on the fact table (i.e., number of tuples retrieved from the fact table):

- [0.0-0.1]: 0% to 0.1% of fact table, i.e., 0 to about 15K records
- [0.1-1.0]: 0.1% to 1% of fact table, i.e., 15K to 160K records
- [1.0-5.0]: 1.0% to 5.0% of fact table, i.e., 160K to 780K records

The goal of the performance evaluation was to measure three alternative query processing techniques:

- *STAR*: conventional star join processing without MHC and without multidimensional indexing; STAR uses secondary indexes that are created on the dimension keys of the fact table. The restrictions on the dimension tables are evaluated and the resulting dimension keys are used for index intersection on the

fact table. The resulting records are joined with the dimension tables, in order to perform grouping and get the final result. This is the typical processing of star queries in commercial DBMSs (e.g., *star transformation* in Oracle [Ora01]). This processing has two major steps: the index intersection and the tuple materialization. While the index intersection has largely been optimized (e.g., with bitmap indexes [NQ97]) the materialization of results is still the bottleneck of non-clustering indexes. Consequently, we neglect the index intersection time for STAR and just measure the time for fact record materialization, residual joins and grouping. Thus, the times for STAR have to be considered as lower bounds of the real processing time.

- *MHC*: applying MHC and multidimensional indexing to the fact table
- *OPT*: MHC with pre-grouping optimizations

For MHC and OPT the complete processing including index access is measured.

Table 1: Response time (in sec) for the three techniques for the three query classes

FT Sel. %	**[0.0-0.1]**			**[0.1-1.0]**			**[1.0-5.0]**		
	STAR	**MHC**	**OPT**	**STAR**	**MHC**	**OPT**	**STAR**	**MHC**	**OPT**
MIN	0	0	0	65	2	2	274	11	6
MAX	30	6	3	290	9	6	1219	47	27
MEDIAN	1	1	1	182	8	5	477	23	13
STD-DEV	4.9	1.2	0.5	75.6	3.1	1.6	346.0	14.1	7.9

Table 1 shows the response time analysis (in seconds) for the three alternative processing plans. As the three classes contain queries with different result set size and thus different response times we use the maximum, minimum, median time and the standard deviation to analyze the performance.

Our results show that the standard STAR processing is outperformed by our approaches. However, for small queries, i.e., the class [0.0-0.1], the speedup is below an order of magnitude. In general, for small result sets, the advantage of clustering over non-clustering is not that large. The picture changes drastically, when we consider larger queries (classes [0.1-1.0] and [1.0-5.0]), which are more typical for OLAP applications. The hierarchical clustering of MHC leads to an average speedup compared to STAR of 24 and with the additional optimization of pre-grouping an additional factor of about two is gained. Note also that STAR has a very high deviation in the response times for queries within one class. This is mainly for two reasons: (a) STAR performance deteriorates very fast as the fact table selectivity is increased and (b) since the fact table is not stored clustered the number of performed I/Os may differ significantly from one query to another. On the other hand, the deviation for MHC and OPT remains low, showing a much more stable behavior.

9 Summary

In this paper we presented the Transbase® DMBS supporting multidimensional hierarchical clustering (MHC). Multidimensional indexing and hierarchical clustering is combined for the primary organization of the fact table of a typical star schema. Transbase® is running on UNIX systems as HP-UX, SUN Solaris, AIX and Linux and on Windows platforms.

The integration is made almost transparent to the user: only when creating the dimension tables and the fact table the hierarchy and the clustering specification has to be provided. From then on, the system is automatically maintaining the hierarchical clustering and uses it for query processing.

We presented some of the implementation issues of MHC and advanced query optimization features enabled by MHC. The measurements on a real-world data warehouse illustrated that MHC can dramatically improve star query processing on such organized schemata.

The improvements of the query processing are not restricted to star schemata, also snowflake schemata are supported. Therefore, our approach is usable in many real data warehouse applications, even complex ones.

In the future we are implementing some advanced algorithms, in order to support complex expressions in aggregation functions, improve queries with non-clustering dimensions and support joins over multiple fact tables.

Bibliography

[ACN01] S. Agrawal, S. Chaudhuri, V. R. Narasayya: Materialized View and Index Selection Tool for Microsoft SQL Server 2000. SIGMOD Conference 2001

[Bay97] R. Bayer. The universal B-Tree for multi-dimensional Indexing: General Concepts. WWCA '97. Tsukuba, Japan, LNCS, Springer Verlag, March, 1997.

[CD97a] S. Chaudhuri, U. Dayal: An Overview of Data Warehousing and OLAP Technology. SIGMOD Record 26(1): 65-74 (1997)

[CD97b] S. Chaudhuri, U. Dayal: Data Warehousing and OLAP for Decision Support (Tutorial). SIGMOD Conference 1997: 507-508

[CI98] C. Y. Chan, Y. E. Ioannidis: Bitmap Index Design and Evaluation. SIGMOD Conference 1998: 355-366

[CS94] S. Chaudhuri, K. Shim: Including Group-By in Query Optimization. VLDB 1994: 354-366

[Des98] P. Deshpande, K. Ramasamy, A. Shukla, J. F. Naughton: Caching Multidimensional Queries Using Chunks. SIGMOD Conference 1998: 259-270

[Gra96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh: Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Total. ICDE 1996: 152-159

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

[GM95] A. Gupta, I. S. Mumick: Maintenance of Materialized Views: Problems, Techniques, and Applications. Data Engineering Bulletin 18(2): 3-18 (1995)

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

[KR98] Y. Kotidis, N. Roussopoulos: An Alternative Storage Organization for ROLAP Aggregate Views Based on Cubetrees. SIGMOD Conference 1998: 249-258

[KS01] N. Karayannidis, and T. Sellis: SISYPHUS: A Chunk-Based Storage Manager for OLAP Cubes. DMDW 2001

[Kar02] N. Karayannidis, A. Tsois, T. Sellis, R. Pieringer, V. Markl, F. Ramsak, R. Fenk, K. Elhardt, R. Bayer: Processing Star Queries on Hierarchically-Clustered Fact Tables. VLDB 2002

[MRB99] V. Markl, F. Ramsak, R. Bayern: Improving OLAP Performance by Multidimensional Hierarchical Clustering. Proc. of the Intl. Database Engineering and Applications Symposium, pp. 165-177, 1999.

[NG95] P. E. O'Neil, G. Graefe: Multi-Table Joins Through Bitmapped Join Indices. SIGMOD Record 24(3): 8-11 (1995)

[NHS84] J. Nievergelt, H. Hinterberger, K. C. Sevcik: The Grid File: An Adaptable, Symmetric Multikey File Structure. TODS 9(1): 38-71 (1984)

[NQ97] P. E. O'Neil, D. Quass: Improved Query Performance with Variant Indexes. SIGMOD Conference 1997: 38-49

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

[Ora01] Oracle 8i Documentation, 2001.

[Pie03] R. Pieringer, K. Elhardt, F. Ramsak, V. Markl, R. Fenk, R. Bayer, N. Karayannidis, A. Tsois, T. Sellis: Combining Hierarchy Encoding and Pre-Grouping: Intelligent Grouping in Star Queries. ICDE 2003

[Ram00] F. Ramsak, V. Markl, R. Fenk, M. Zirkel,K. Elhardt, R. Bayer.Integrating the UB-Tree intoa Database System Kernel. In VLDB2000,Proceedings of International Conference on Very Large Data Bases, 2000, Cairo, Egypt, 2000.

[Rou98] N. Roussopoulos: Materialized Views and Data Warehouses. SIGMOD Record 27(1): 21-26 (1998)

[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): 36-43 (1997)

[SDJL96] D. Srivastava, S. Dar, H. V. Jagadish, A. Y. Levy: Answering Queries with Aggregation Using Views. VLDB Conference 1996: 318-329

[Tra02] The TransBase HyperCube relational database system, available at: http://www.transaction.de/

[TT01] D. Theodoratos, A. Tsois: Heuristic Optimization of OLAP Queries in Multidimensionally Hierarchically Clustered Databases. DOLAP 2001.

[WB98] M. C. Wu, A. P. Buchmann: Encoded Bitmap Indexing for Data Warehouses. ICDE 1998: 220-230

[WOS01] K. Wu, E. J. Otoo, A. Shoshani: A Performance Comparison of bitmap indexes. CIKM 2001: 559-561

[Wu99] Ming-Chuan Wu: Query Optimization for Selections Using Bitmaps. SIGMOD Conference 1999: 227-238

[YL94] W. P. Yan, P-Å. Larson: Performing Group-By before Join. ICDE 1994: 89-100

[YL95] W. P. Yan, P.-Å. Larson: Eager Aggregation and Lazy Aggregation. VLDB Conference 1995

[Zah00] Markos Zaharioudakis, Roberta Cochrane, George Lapis, Hamid Pirahesh, Monica Urata: Answering Complex SQL Queries Using Automatic Summary Tables. SIGMOD Conference 2000:

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