Serdica J. Computing **5** (2011), 309–322

Serdica
Journal of Computing

Bulgarian Academy of Sciences
Institute of Mathematics and Informatics

A SUPER-DIMENSION APPROACH IN ROLAP ENVIRONMENTS

Ina Naydenova

Abstract. Often the designer of ROLAP applications follows up with the question "can I create a little joiner table with just the two dimension keys and then connect that table to the fact table?" In a classic dimensional model there are two options—(a) both dimensions are modeled independently or (b) two dimensions are combined into a super-dimension with a single key. The second approach is not widely used in ROLAP environments but it is an important sparsity handling method in MOLAP systems. In ROLAP this design technique can also bring storage and performance benefits, although the model becomes more complicated. The dependency between dimensions is a key factor that the designers have to consider when choosing between the two options. In this paper we present the results of our storage and performance experiments over a real life data cubes in reference to these design approaches. Some conclusions are drawn.

1. Introduction. The question of how to represent the dependence between two or more dimensions without going through the fact table leads the designer of ROLAP environments to the idea of composite dimension (see [3],

ACM Computing Classification System (1998): H.2.1, E.5.
Key words: combine, composite, dimension, concatenation, OLAP.

[4], [6]). Let us imagine that the two dimensions are Product and Market in a retail setting. Suppose that we have the fact table that records actual sales of Products in the various Markets over Time. The desire to represent the dependence between the Product and Market dimensions is based on the suspicion that "Products are highly correlated with Markets in our business". So when does the designer choose separate dimensions and when does the designer combine the dimensions? The theoretical expectations are that the effective composite is a small one, i.e. with less meaningful dimension value combinations. According to Ralph Kimball [4], if Products are extremely correlated with Markets, combining the two dimensions makes eminent sense, but rarely do Product and Market have such a nice relationship.

Combining dimensions into a single super-dimension is an important sparsity-handling method in MOLAP systems, but it is not widely used in ROLAP environments. It is almost never mentioned as an alternative design technique in books or papers on data warehousing design. Why is this approach so unpopular in ROLAP? We tried to find studies dedicated to this technique, experimental results or comparisons of the two approaches. But we could not find any. There are publications about decomposition of relationships based on functional dependences, multivalued dependences or other logical constraints. The approach we are talking about refers to dependences in the widest sense of statistical relationships (the relationship between Market and Product dimensions is really many-to-many; when most Products are sold in most Markets, it becomes obvious that we need two dimensions because otherwise our combined dimension looks like a Cartesian product of the original dimensions). Dependences that can be expressed by normalization of the data model are not of interest to us.

In this paper we present the results of our storage and performance tests over real-life data cubes in reference to these design approaches. We are aware that our experiments do not provide scientific evidence on the advantages and disadvantages of the two approaches, nor are they a statistically valid survey. The aim here is to share the measurements and the conclusions that we have made since we have not found other similar studies.

Our tests are based on typical OLAP queries executed repeatedly on several design schemes: classical scheme in which the dimensions are modeled independently plus a few schemes that combine 2 or 3 dimensions in a super-dimension. Because the presence of indices on tables can significantly affect the execution plans and query performance, tests were made on two widely used index schemes in ROLAP—indexing of the dimensions by B-tree or bitmap indices.

The paper is organized as follows. Section 2 outlines a multidimensional

view of data and different OLAP architectures. Section 3 describes our test cases—tested design and indexing schemes, platform and queries description. Section 4 presents the tests results and observations. Finally we make some conclusions in Section 5.

2. OLAP systems and multidimensional view of data. Multidimensional models lie at the core of OnLine Analytical Processing (OLAP) systems. Such systems provide fast answers for queries that aggregate large amounts of detail data to find overall trends, and they present the results in a multidimensional fashion, which renders a multidimensional data organization ideal for OLAP. Sometimes OLAP and data warehousing are used as synonymous terms. Thomsen [7] considers them complementary in that data warehousing makes raw data available to end users and ensures its accuracy and consistency whereas OLAP focuses on the end user's analytical requirements [13]. Actually a data warehouse is a large repository of data integrated from several sources in an enterprise for the specific purpose of data analysis and decision support [5]. In a multidimensional data models, there is a set of numeric measures (facts) that are the objects of analysis. Each of the numeric measures depends on a set of dimensions, which provide the context for the measure. For example, the dimensions associated with a sale amount can be the store, the product, and the date when the sale was made. The dimensions together are assumed to uniquely determine the measure. Often the dimensions are hierarchical; the time of sale may be organized as a day-month-quarter-year hierarchy, the product as a product-category-industry hierarchy [5].

There are two types of OLAP storage options: Relational OLAP (ROLAP) and Multidimensional OLAP (MOLAP). In ROLAP, the data themselves are stored in a relational database, whereas with MOLAP, a large multidimensional array is built with the data [2]. One difficulty with MOLAP systems is that the array is often sparse. These typically include provisions for handling sparse arrays, and they apply advanced indexing and hashing to locate the data when performing queries. ROLAP systems also employ specialized index structures, such as bit-mapped indices, to achieve good query performance [9].

In ROLAP it is beneficial to view data in terms of a dimensional model which is composed of a central fact table and a set of surrounding dimension tables, each corresponding to one of the components or dimensions of the fact table. Conceptually this leads to a star-like data structure, which is called a star scheme (see Fig.1 below). Star schemes do not explicitly provide support for attribute hierarchies. They can be refined into snowflake schemes providing support for attribute hierarchies by allowing the dimension tables to have sub-

dimension tables. There is a debate on the benefits of having such sub-dimension tables, since it will, in general, slow down query processing, but in some cases it provides a necessary logical separation of data [14]. In snowflake schemes the dimensional hierarchy is explicitly represented by normalizing the dimension tables. This leads to advantages in maintaining the dimension tables, but the de-normalized structure of the dimensional tables in star schemes may be more appropriate for browsing the dimensions [5]. To avoid unnecessary complications in our tests we used the star scheme design.

3. Test cases. We are going to execute typical OLAP queries against 7 design schemes and two indexing schemes. We will measure the query timing and the allocated disk storage for every scheme. The RDBMS used for these experiments is Oracle Database (Release 11 g Enterprise Edition Release 11.2.0.1.0 64 bit) as one of the most popular RDBMS (48.1% of the market share according to Gartner 2010) .

Our base test cube is an extract from an enterprise data warehouse of an international holding company. It has 46 593 527 nonempty cells and the following structure:

Cube: LAB_TEST

Dimensions	***Facts***
Time (26 members)	Number of clients
Products (437 members)	Number of contracts
Sales Channels (12 members)	
Organization units (349 members)	
16 more dimensions that are out of the test scope	

Dimension hierarchies

Products	***Sales Channels***	***Organization units***	***Time***
– Industry	– Channel Groups	– Parent unit	– Month
– Business Lines	– Base Channels	– Base units	– Quarter
– Product Lines			– Year
– Products			

As was mentioned above, the approach of combining dimensions into a single superdimension is a well-known technique in MOLAP systems. To determine which dimensions are suitable for concatenation, MOLAP designers use specialized software or manually calculate the sparsity of data. The sparsity of data is

the ratio of empty cells in a cube to the total number of cells in a cube. Data is said to be 5% dense (or 95% sparse) if only 5% of the possible combinations of the cells in a multidimensional measure actually contain data. We will apply the same approach to estimate the sparsity of the composite of dimensions. Let's assume that a super-dimension "Products – Channels" is formed on the base of the dimension "Products" and the dimension "Channels". The super-dimension contains only combinations of products and channels that have a business meaning. For example if product "a" is offered by channel "b", the combination between "a" and "b" is part of the dimension "Products – Channels". If a channel "b" is not relevant for a product "c" then the combination between "c" and "b" is not an element of the super dimension. We estimate the sparsity % of dimension "Products – Channels" as a ratio between the number of elements of the super-dimension "Products – Channels" and the number of elements of the Cartesian product of dimensions "Products" and "Channels".

A super-dimension with high sparsity percentage is a better candidate than others, because it will be smaller than others, its processing will be faster and more storage will be saved. Note that we expect to save storage not only because the dimension itself takes less space, but mainly because a superdimension scheme will reduce the size of the fact's foreign key columns. The size of the foreign key columns depends linearly on the number of dimensions and logarithmically on the number of dimensions' elements involved in the concatenation.

Design schemes

The design schemes involved in our tests are the following:

- ***base scheme***—a classical design star scheme:

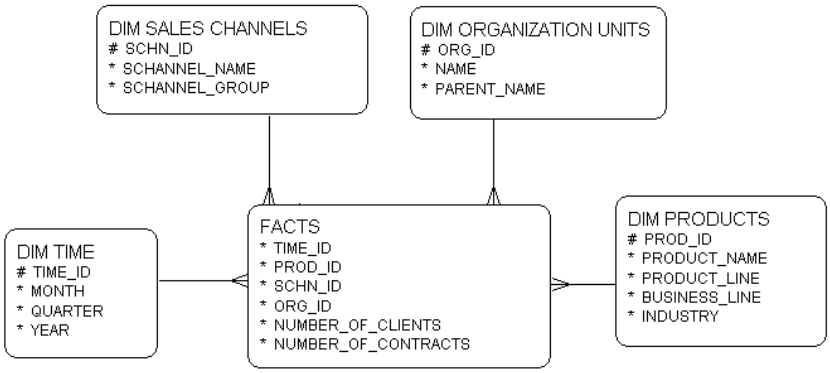

Fig. 1. Star scheme in ROLAP systems

This scheme must participate in our experiment as a representative of the

standard approach of ROLAP data modeling.
In the test cases we have included the best and worst combinations (in terms of sparsity %) among the possible combinations of two dimensions.

- ***p-schn scheme***—this is a scheme with a superdimension constructed on the basis of a concatenation of dimensions of products and sales channels:

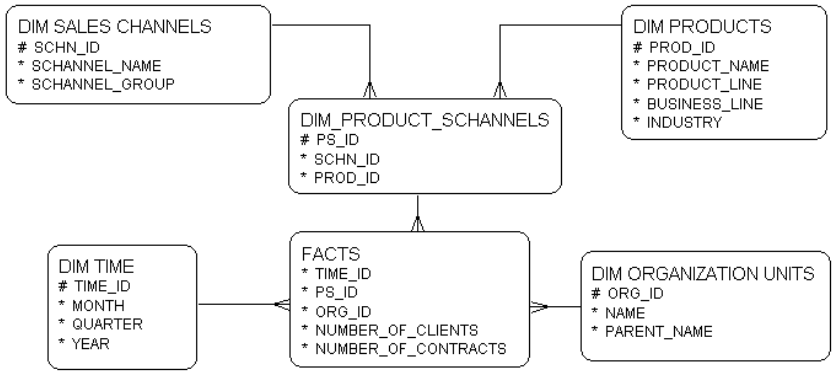

Fig. 2. Normalized scheme with concatenated dimensions of products and sales channels

All possible combinations are 5 244, the actual ones are 950. The sparsity is approximately 82 %. The new dimension dim_product_schannels (see Fig. 2) references only primary key columns of the dimensions of products and sale channels;

- ***p-schn denormalized scheme***—this is a modification of the p-schn scheme. The new dimension includes all product and sale channels columns and completely replaces the original dimensions (see Fig. 3);
Our theoretical expectation is that the allocated storage in the case of the classic design scheme will be greater than in the case of super-dimension scheme. What is more interesting for us is whether the performance will be better.
We include this denormalized scheme in out tests, because the de-normalization of superdimensional hierarchies will lead to better performance without significantly increasing the allocated storage.
- ***p-org scheme***—a scheme with superdimension constructed on the basis of a concatenation of dimensions of products and organization units. All possible combinations are 152 513, the actual ones are 15 731, the sparsity is approximately 90%. The new dimension references only primary key columns of the dimensions of products and organization units;

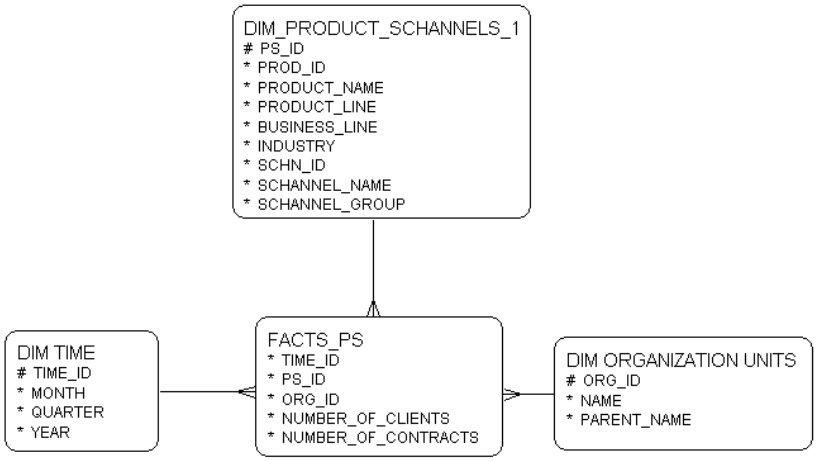

Fig. 3. Denormalized scheme with concatenated dimensions of product and sales channels

- ***p-org denormalized scheme***—this is a modification of the p-org scheme. The new dimension includes all columns with products and organization units and in such way completely replaces the original dimensions;
- ***p-org-schn scheme***—this is a scheme that concatenates 3 dimensions —products, organization units and sale channels. The sparsity here is approximately 99%. All possible combinations are 1 830 156, the actual ones are 22 728;
- ***p-org-schn denormalized scheme***—this is a modification of the p-org-schn scheme. The new dimension includes all columns with products, organization units and sale channels and completely replaces the original dimensions.

Indexing schemes

Database indices provided today by most relational database systems use B^+-tree indices to retrieve rows of a table with specified values involving one or more columns. The leaf level of the B-tree index consists of a sequence of entries for index key values. Each key value reflects the value of the indexed column in one or more rows in the table, and each key value entry references the set of rows with that value [10]. Another commonly used index type in RDBMS is the bitmap index. Bitmap indices are known as the most effective indexing methods for range queries on append-only data, and many different bitmap indices have been proposed in the research literature (see [8], [10], [11], [12]).

The queries in our test will be executed against the described design

schemes in two indexing versions—the first one uses B-tree indices on the dimension table, while the second one adds a bitmap indices on the fact tables. For simplicity we will refer to these schemes below as B-tree and Bitmap index scheme.

Test queries

Typical OLAP operations include *rollup* (increasing the level of aggregation) and *drill-down* (decreasing the level of aggregation or increasing detail) along one or more dimension hierarchies, *slice and dice* (selection and projection), and *pivot* (re-orienting the multidimensional view of data) [5].

Our tests include 5 queries, which provide the typical OLAP operations. Of course, if we include more types of queries we will be closer to the real OLAP application working process. But we aim to test simple queries, because we want to reduce the influence of side factors as heuristics implemented in the database optimizer, system parameters and others, that would prevent us from assessing the impact of scheme design on system productivity and therefore do not interest us.

Slice and Dice operation

Slicing takes a d-dim cube and returns a $(d-k)$-dim cube. We specify one fixed value for each of k dimensions. According to some definitions dicing refers to range selection in multiple dimensions (the dimensionality is not reduced) [1]. People usually talk about "slice-and-dice" together and do not make such a distinction.

The following query gives us a slice for a specific month and product industry:

```
select
     products.product_name,
     organization_units.name,
     sales_channels.name,
     sum(cnt_contracts) cnt_contracts,
     sum(cnt_clients) cnt_clients
from lab_test
join time
  on time.time_id = lab_test.time_id
join products
  on products.prod_id = lab_test.prod_id
join organization_units
  on organization_units.unit_id = lab_test.unit_id
join sales_channels
  on sales_channels.schn_id = lab_test.schn_id
```

```
where time.year_no = 2010 and
      time.month_no = 05 and
      products.industry = 'INSUARANCE'
group by products.product_name,
         organization_units.name,
         sales_channels.name;
```

The query output contains information about dimensions of products, organization units and sales channels and cube measures.

Drill Down and Rollup operation

The two basic hierarchical operations when displaying data at multiple levels of aggregations are the "drill-down" and "roll-up" operations. Drill-down refers to the process of viewing data at a level of increased detail, while roll-up refers to the process of viewing data with decreasing detail [1]. Many RDBMS now support a ROLLUP command that computes a single result table for a collection of related GROUP-Bys:

```
select
  products.industry,
  products.busline,
  products.prodline,
  products.product_name,
  sum(cnt_contracts) cnt_contracts,
  sum(cnt_clients) cnt_clients
from lab_test
join time
  on time.time_id = lab_test.time_id
join products
  on products.prod_id = lab_test.prod_id
where time.year_no = 2010 and
      time.month_no = 06
group by rollup (products.industry,
                 products.busline,
                 products.prodline,
                 products.product_name);
```

A rollup list should contain the different levels of one dimension, ordered by decreasing granularity, for example industry, business line, product line and product levels for Products dimension. The above rollup operation will lead to 5 group-bys: (industry, business line, product line, product), (industry, business line, product line), (industry, business line), (industry), (nothing = grant total).

In such a way all levels of aggregations needed for roll-up and drill-down shifting are returned by a single SQL query.

• **Pivot operation**

Pivoting is concerned with information display. Pivot is an option to choose which dimensions to show in a (usually) 2-d rendering: choose some dimensions X1, ..., Xi to appear on the x-axis and some dimensions Y1, ..., Yj to appear on the y-axis. Pivot deals with presentation of data and although there are pivot/unpivot operators in sql, OLAP tools generate ordinary select queries (GROUP BY X1, ..., Xi, Y1, ..., Yj) and display the values in a different grid representation. This is why a query using the pivot operator will not be included in our test.

• **Description of queries**

Let us summarize what the 5 testing queries actually do:

— SQL1 return data for a specific month and industry (see the above slice and dice example);

— SQL2 rollup the product dimension for a specific month (see the Drill Down/Roll up example above);

SQL3, 4 and 5 represent the dice operation over different dimensions. They all return information about a month, year, product name, sales channel name, organization unit name and fact number of clients and number of contracts. Also:

— SQL 3 has a range filter over products and sales channels;

— SQL 4 has a range filter over products and organization units;

— SQL 5 has a range filter over sales channels and organization units.

Test procedure

For every design and index scheme we run the queries in series of 6 consecutive executions. Also before every series (the series is tied to a specific type of query—for example SQL1) we clean up the database instance memory structures related with the query execution (as database buffer cache and shared pool).

Table 1. The average time of the query series with a B-tree index scheme

	SQL 1 (slice)	SQL 2 (roll up)	SQL 3 (dice)	SQL 4 (dice)	SQL 5 (dice)	Total time	Average time
base scheme	39.07	248.03	36.37	30.05	36.71	390.22	78
p-org scheme	40.45	245.71	36.22	27.07	28.12	377.57	76
p-org denormalized	35.86	244.75	33.83	26.71	28.16	369.31	74
p-sch scheme	35.58	249.22	32.51	27.26	27.46	372.03	74
p-schn denormalized	34.43	245.67	33.15	26.53	26.71	366.49	73
p-org-schn	30.66	223.07	28.85	23.39	23.18	329.15	66
p-org-schn denormalized	30.25	225.83	29.23	23.34	23.48	332.12	66

Then we remove a first execution from our test and estimate an average time for the series.

4. Results. In Table 1 and Figure 4 the average times of the 5 series with a B-tree index scheme are presented. The last 2 columns show the total and average time for all queries.

In Table 2 and Figure 5 the average times of the 5 series with a Bitmap index scheme are presented.

In Table 3 the allocated storage for every scheme is presented.

On the basis of the above results we can make the following observations:

- The bitmap indexing scheme is more efficient than the B-tree;
- The sparsity does not impact the percentage of saved storage. The size of the concatenated superdimension is significantly less than the size of the fact table, so the saved storage depends mainly on the number of concatenated

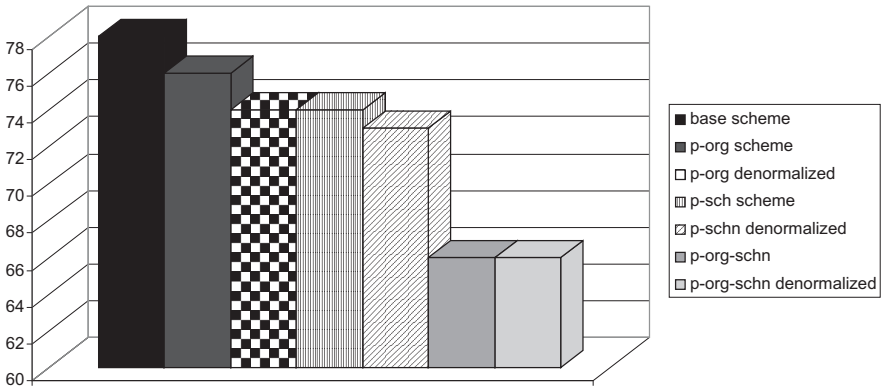

Fig. 4. Average time for the queries and B-tree index scheme

Table 2. The average time of the query series with a Bitmap index scheme

	SQL 1 (slice)	SQL 2 (roll up)	SQL 3 (dice)	SQL 4 (dice)	SQL 5 (dice)	Total time	Average time
base scheme	11.34	221.31	36.54	2.21	2.32	273.71	55
p-org scheme	12.63	224.89	36.32	3.16	3.18	280.18	56
p-org denormalized	11.11	220.13	36.30	1.16	3.18	271.87	54
p-sch scheme	10.19	225.55	32.54	1.91	2.12	272.30	54
p-schn denormalized	9.16	222.30	32.49	1.77	1.95	267.66	54
p-org-schn	10.57	218.58	29.81	2.86	3.07	264.89	53
p-org-schn denormalized	9.32	217.53	31.01	1.95	1.15	260.96	52

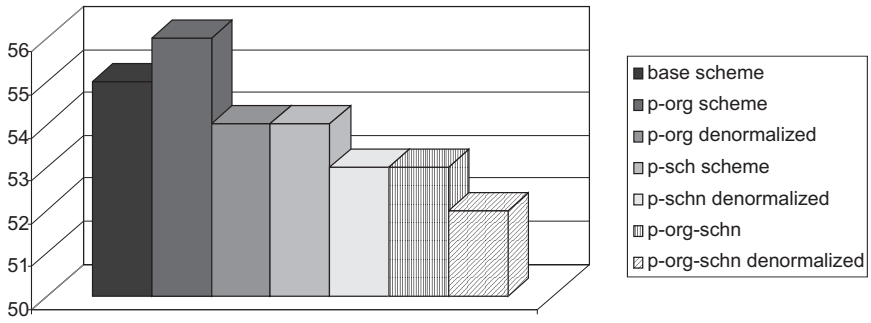

Fig. 5. Average time for the queries and Bitmap index scheme

Table 3. Allocated and saved storage for different design and indexing schemes

	Btree Sheme storage size (MB)	% saved storage towards base scheme	Btimap Sheme storage size (MB)	% saved storage towards base scheme
base scheme	1665		1801	
p-org scheme	1474	13%	1597	13%
p-org denormalized	1481	12%	1604	12%
p-sch scheme	1474	13%	1600	13%
p-schn denormalized	1474	13%	1600	13%
p-org-schn	1335	25%	1447	24%
p-org-schn denormalized	1347	24%	1459	23%

dimensions;

- The denormalized design schemes have an insignificantly better performance that the corresponding normalized design scheme;
- If the sparsity is greater, the performance is better, but this dependence is very weak;
- If the sparsity between concatenated dimensions is less than 99% the performance benefits become more perceivable (mostly in the case of a B-tree index scheme).

5. Conclusions and future directions. The dependence between dimensions can be a reason for a set of dimensions being stored together in a joint relational scheme, instead of separate dimension schemes. In some sources this technique is also referred to as "flattening dimensions" (see [3]).

Our conviction is that the design technique is not widely used in ROLAP systems, because the model becomes more complicated and the dependence be-

tween dimensions could not be accurately computed (it changes over time). The OLAP designers have to decide which part of the dimensions to form a composite of on the basis of their knowledge about the business area and inner object dependences. But this technique certainly brings storage benefits. The results of our experiments suggest that if there is less than 20% sparsity between two dimensions, their concatenation will decrease the allocated storage and increase the performance in specific queries, while the others won't be affected.

In fact, the productivity gains are guaranteed when we want to show the relationship between the dimensions on the OLAP users before the execution of the main report query, for example, when a user selects a product to see what the relevant sales channels are. We have not included such a test in our experiment because there are indisputable benefits (a query on the superdimension is significantly faster than a query with distinct operators over a fact table).

It is difficult to define a precise sparsity threshold above which speed-up of the queries will compensate for the overhead of the additional join operations and deliver a significantly better performance. Our experiment is based on the real enterprise data, but there an additional test over simulated data is necessary to find a more accurate answer to the question what dimension is it reasonable to concatenate into a composite to achieve better query performance in addition to better storage size.

REFERENCES

[1] Lemire D., O. Kaser. Lectures in Data Warehousing. `http://pizza.unbsj.ca/~owen/backup/courses/OLAP-2004/olap2.pdf`, 2004

[2] Lemire D. O. Kaser. Attribute Value Reordering for Efficient Hybrid OLAP. In: Proceedings of the 6th ACM international workshop on Data warehousing and OLAP, 2003, ISBN:1-58113-727-3, 1–8.

[3] Cognos ULC. Improved Performance by Flattening Dimensions. `http://www.ibm.com/developerworks/data/library/cognos/page174.html`, 2008

[4] Kimbal R. Showing the Correlation between Dimensions. Kimbal Univeristy. `http://www.rkimball.com/html/designtipsPDF/DesignTips2000%20/KimballDT6ShowingThe.pdf`, 2010

[5] Chaudhuri S., U. Dayal. An overview of data warehousing and OLAP technology. *ACM SIGMOD Record*, **26** (1997), 65–4.

[6] Singh A. Handling inter-dimensional members dependency and reducing cube sparsity using reference dimensions in Analysis Services 2005 SP2. `http://asmdx.blogspot.com/2008/05/handling-inter-dimensional-members.html`, 2008

[7] Thomsen E. OLAP Solutions Building Multidimensional Information Systems, ISBN:0-471-40030-0, John Wiley and Sons, USA, 1997.

[8] Wu K., A. Shoshani, K. Stockinger. Analyses of Multi-Level and Multi-Component Compressed Bitmap Indices. *ACM Transactions on Database Systems*, **35** (2010), No 1, Article No. 2.

[9] Pedersen T., Ch. Jensen. Multidimensional Database Technology. *IEEE Computer Society Press*, **34** (2001), No 12, ISSN:0018-9162, 40–46.

[10] O'Neil P., D. Quass. Improved Query Performance with Variant Indices. In: Proceedings of the ACM International Conference on Management of Data (SIGMOD 1997), May 13-15, 1997, Tucson, Arizona, USA, 38–49.

[11] O'Neil P. Model 204 architecture and performance. In Proceedings of the 2nd International Workshop in High Performance Transaction Systems, Lecture Notes in Computer Science, **359** (1987), Springer, 40–59.

[12] O'neil E., P. O'neil, K. Wu. Bitmap index design choices and their performance implications. In: Proceedings of IDEAS'07, Banff, Alberta, Canada, 72–84.

[13] Niemi T., L. Hirvonen, K. Järvelin. Multidimensional data model and query language for informetrics. *Journal of the American Society for Information Science and Technology*, **54** (2003), No 10, 939–951.

[14] Levene M., G. Loizou. Why is the snowflake scheme a good data warehouse design?, *Information Systems,* **28** (2003), No 3, 225–240.

Ina Naydenova
Faculty of Mathematics and Informatics
St. Kliment Ohridski University of Sofia
5, J. Bourchier Blvd
1164 Sofia, Bulgaria
e-mail: `ina@fmi.uni-sofia.bg`

Received January 3, 2011
Final Accepted March 2, 2012