Meta Galaxy: A Flexible and Efficient Cube Model for Data Retrieval in OLAP [*]

Jie Zuo[1], Changjie Tang[1], Lei Duan[1], Yue Wang[1], Liang Tang[1], Tianqing Zhang[1,**], and Jun Zhu[2]

[1] School of Computer Science, Sichuan University, Chengdu 610065, China
{zuojie,tangchangjie,zhangtianqing}@cs.scu.edu.cn
[2] National Center for Birth Defects Monitoring Chengdu 610041, China

Abstract. OLAP is widely used in data analysis. The existing design models, such as star schema and snowflake schema, are not flexible when the data model is changed. For example, the task for inserting a dimension may involve complex operations over model and application implementation. To deal with this problem, a new cube model, called Meta Galaxy, is proposed. The main contributions of this work include: (1) analyzing the shortcoming of traditional design method, (2) proposing a new cube model which is flexible for dimension changes, and (3) designing an index structure and an algorithm to accelerate the cube query. The time complexity of query algorithm is linear. The extensive experiments on the real application and synthetic dataset show that Meta Galaxy is effective and efficient for cube query. Specifically, our method decreases the storage size by 95.12%, decreases the query time by 89.89% in average compared with SQL Server 2005, and has good scalability on data size.

1 Introduction

OLAP provides users quick answers to multi-dimensional analytical queries [1]. The process of OLAP combines technologies of database, data warehouse and data mining. In OLAP, data cubes provide data summaries in different points of view or dimensions. Generally, there are two kinds of OLAP systems: multi-dimensional OLAP (MOLAP) and relational OLAP (ROLAP) [2]. MOLAP and ROLAP are widely used in business related domains. However, both of them have disadvantages respectively. For example, MOLAP approach introduces data redundancy, while ROLAP is not suitable when the model is heavy on calculations which don't translate well into SQL statements.

In the real world applications, the number of dimensions in a data cube may be huge. The traditional design of a data cube creates a dimension table for each dimension.

[*] This work was supported by the National Natural Science Foundation of China under grant No. 60773169 and the 11th Five Years Key Programs for Sci. &Tech. Development of China under grant No. 2006BAI05A01.

[**] Correspondence author.

However, the dimensions may change due to the new requirements. As a result, a new attribute is added in the facts table, and a new dimension table is created. It follows that the traditional design is not flexible and efficient enough to meet the complex real application requirements. Example 1 illustrates the model design in birth defects monitoring database.

Example 1. In the real application of birth defects monitoring database, each record contains the information of a baby with birth defects as shown in Table 1. All birth defects are list as attributes for a baby record. For each defect attribute, the value describes the status of corresponding defect.

Table 1. Sample records stored in birth defects monitoring database

Baby_id	Weight	Gender	Defects_1	Defects_2	…	Defects_n
001	2800	Male	No	Serious	…	No
002	3500	Female	Weak	Weak	…	No
…	…	…	…	…	…	…

Table 1 shows that a new attribute must be inserted if a new defect is discovered, and needs updating operations over the data model and application design.

To solve the problem, we design a new cube model named Meta Galaxy. Basically, Meta Galaxy consists of three tables:

- Meta-dimension Table. It contains the definitions of all dimensions and uses unique ID to indicate each dimension.
- Meta-item Table. It defines all available values for each dimension.
- Meta-value Facts Table. It records the values on all dimensions of all objects.

The Figure 2, Figure 3 and Figure 4 in the rest sections describe the rough sketch of this model, the skeleton of the model likes a star model (or a galaxy when problem is complex), while the attribute value is meta value, such as *object_id*, *dim_id* and *itm_id*. Hence we call it as Meta Galaxy. It is with following advantages:

- Flexibility. It is unnecessary to modify the model design no matter inserting or deleting a dimension. Additionally, the storage space is saved since many default values are not kept in Meta Galaxy.
- Efficiency. An index structure (called Meta Index) is created and an algorithm is designed to accelerate the data query. Our experiments show that our method decreases the storage size by 95.12%, decreases the query time by 89.89% in average compared with SQL Server 2005, and has good scalability on data size.

2 Related Work

Svetlana et al. applied OLAP to business analysis [3]. The OLAP they implement OLAP is called ROLAP, since it is based on relational DBMS. Based on the concept of emerging pattern, Sébastien et al. introduced the concept of emerging cube, which is materialized in MOLAP, to reveal the changing of trends between two pre-computed cubes [4]. Perez et al. used XML files to store the raw data and established a related cube (R-cube) for MOLAP [5]. Doug et al. extended the OLAP data model to deal with ambiguous, imprecision data, and provided an allocation-based approach to tackle semantic problems in aggregation [6]. In [7], Thomas et al. allocated relational data warehouses based on a star schema and bitmap index, and used a facts table with multi-dimensional hierarchical data fragmentation to support queries referencing subsets of schema dimensions. Zhongzhi Shi et al. investigated the way to map multi-dimensional operations into SQL statements [8]. Zohreh et al. established views and index for OLAP and solved the index-selection problem to improve the processing efficiency [9]. Elaheh et al. designed a novel method, called Partial Pre-aggregation (PP), to estimate approximation results of joint queries in MOLAP [10]. Ruoming Jin et al. optimized the cost of main memory to deal with the performance problems in multi-dimensional hierarchical data [11]. Yon Dohn et al. used the RD-Tree to process top-k queries in OLAP aggregation [12]. T.S. Jung et al. proposed an index structure to accelerate the MOLAP operations [13].

The main differences between our method and existing results are: (a) we pay more attention to the situation of dimension evolutions. (b) It is unnecessary to modify the data model and application implementation due to the proposed Meta Galaxy to handle dimension evolution.

3 The Structure of Meta Galaxy

We propose a new cube model named Meta Galaxy. Intuitively, the design of Meta Galaxy is similar to the Triples. However, Meta Galaxy is a cube model with index and query algorithms (discussed in Section 4). Basically, the Meta Galaxy consists of three tables: Meta-dimension Table, Meta-item Table and Meta-value Facts Table. Figure 1 illustrates the main structures of these three tables.

- The Meta-dimension Table defines all dimensions in the cube model. The attributes include: *dim_id*, *name* and *description*. In this table, attribute *dim_id* is a unique key to indicate each dimension, attribute *name* is the name of a dimension, and attribute *description* is the note for a certain dimension.

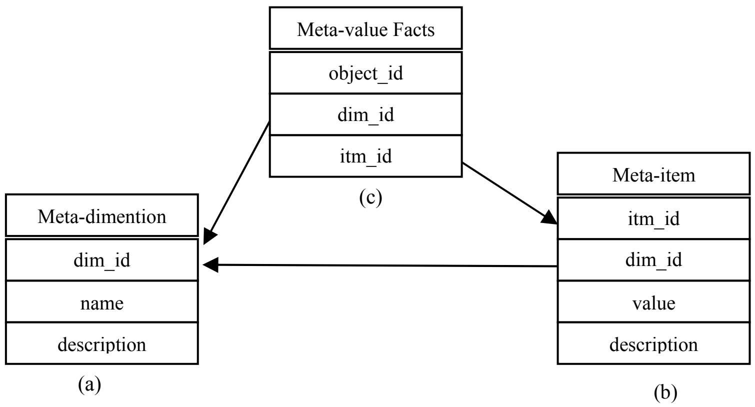


Fig. 1. The relationships among three tables in Meta Galaxy: (a) Meta-dimension Table, (b) Meta-item Table, and (c) Meta-value Facts Table

- The Meta-item Table defines all available values for each dimension. There are four attributes: *itm_id*, *dim_id*, *value*, and *description*. In Meta-item Table, attribute *itm_id* is a unique id for an available value (*value*) of dimension *dim_id*, and attribute *description* is the corresponding note.
- The Meta-value Facts Table (mvFacts) defines values on all dimensions of all objects by recording *object_id*, *dim_id* and *itm_id*.

Example 2. In Example 1, the baby with id "001" is male. Suppose the dimension id (*dim_id*) of "Gender" is *dim003*, item id of (*itm_id*) "Male" is *itm001*. Then, in Meta Galaxy, Meta-dimension Table contains record (*dim003*, "*Gender*", "*the gender information*"), Meta-item Table contains record (*itm001*, *dim003*, "*Male*", "*male information*"), and Meta-value Facts Table contains record (*001*, *dim003*, *itm001*).

Since dimension information is recorded in the Meta-dimension Table, it is flexible to handle dimension evolution. For example, to insert a dimension d_1, the record of d_1 will be added in the Meta-dimension table. Furthermore, all available values of d_1 will be added in the Meta-item table. To delete a dimension d_2, all corresponding records related to d_2 in Meta-dimension table and Meta-item table will be removed.

Example 3. Consider adding a new birth defect, *def*, in the birth defect monitoring database. Suppose there are three statuses of *def*: s_1, s_2, and s_3. Then a new tuple containing *def* and related information will be inserted into Meta-dimension table, and three tuples containing *def* and each of its three statuses will be inserted into Meta-item table.

4 The Meta Index and Fast Cube Query Algorithm

4.1 The Design of Meta Index

A typical cube query on a relational data table is a SQL statement containing "group by" clause. For example, Figure 2 describes a typical query statement on dimension d1, d2, d3 with aggregation function agg().

```
SELECT d1, d2, d3, agg()
FROM facts
WHERE dn = value1 and dm = value2
GROUP BY d1, d2, d3
```

Fig. 2. A typical SQL statement for cube query

Figure 2 shows that three key steps are in the query process.

Step 1: Filter objects based on the "where" clause.
Step 2: Group objects based on the "group by" clause.
Step 3: Calculate the results of each group based on the aggregation functions.

The efficiencies of first two steps are important. Since in real OLAP analysis, the number of data size is always very large. In Meta Galaxy, dimensions are not attributes

```
SELECT a, b, c, COUNT(*) FROM
(SELECT m.id AS id, a, b, c FROM
  (SELECT DISTINCT oid AS id FROM mvFacts) m
  LEFT JOIN
  (SELECT object_id AS id, itm_id AS a FROM mvFacts WHERE dim_id = d1) a ON m.id = a.id
  LEFT JOIN
  (SELECT object_id AS id, itm_id AS b FROM mvFacts WHERE dim_id = d2) b ON m.id = b.id
  LEFT JOIN
  (SELECT object_id AS id, itm_id AS c FROM mvFacts WHERE dim_id = d3) c ON m.id = c.id
) AS tt
GOURP BY a, b, c
```

Fig. 3. A modified SQL statement for query in Meta Galaxy

in the Facts Table. Thus to execute queries in Figure 2, we modify the SQL statement for Meta Galaxy. The statements are shown in Figure 3.

Figure 3 illustrates two disadvantages of this SQL statements template:

- It is difficult to write this kind of query statements.
- The efficiency of query is low. The Meta-Value Facts Table should do self-join several times.

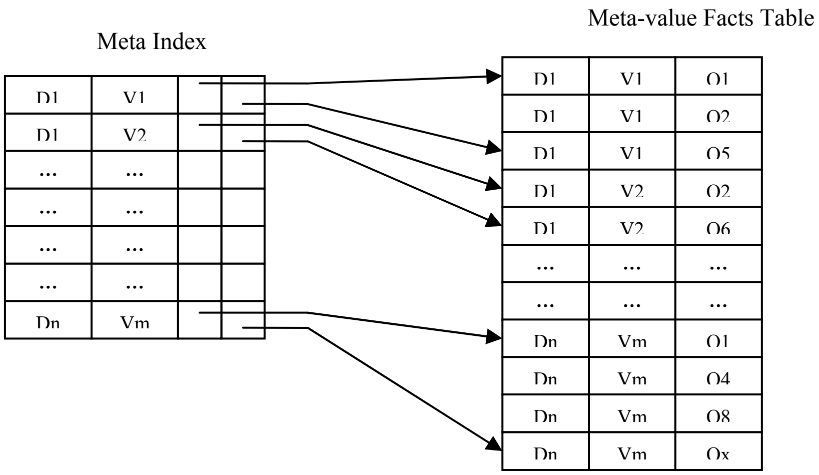


Fig. 4. The index for Meta-value Facts Table

Thus, it is necessary to design query algorithm for Meta Galaxy to improve the efficiency. The key points are: (a) in cube query, all query conditions are presented in the form of "slice". That is, "dimension 1 = value 1 and dimension 2 = value 2 and … and dimension n = value n". (b) Based on the selection conditions template, we design an efficient index structure which is shown in Figure 4.

The index is a map whose key is a pair (*dim_id*, *itm_id*), and value is a pair (*start*, *end*). The index is constructed in the following way. First, for all records in Meta-value Facts Table, sort them according to (*dim_id*, *itm_id*, *object_id*). As a result, all objects that have the same value on certain dimension will be arranged together. Next, for each key (*dim_id*, *itm_id*) in the index, (*start*, *end*) indicates the first and last positions of records containing (*dim_id*, *itm_id*) in Meta-value Facts Table.

One advantage of adopting this index is saving storage space which can save disk accessing time. The size of index depends on the number of items on all dimensions. Usually, it is small enough to resident in memory. Two key points for Meta-value Facts Table:

(a) All records in the table must be ordered once the index is created.
(b) The attributes *dim_id* and *itm_id* are redundancy. To minimize the storage space and query response time, these two attributes are removed.

As a result, there is only one column in Meta-value Facts Table. That is, the size of Meta-value Facts Table is decided by the number of non-default values of all objects on each dimension.

4.2 The Fast Cube Query Algorithm

We design the cube query algorithm based on the index. The basic idea is filtering objects which do not satisfy the selection conditions and grouping the satisfied objects. Index is used in both filtering and grouping to accelerate the query. We will give the algorithms for filtering and grouping, respectively.

In cube query, typical selection conditions like "dim1 = itm1 and dim2 = itm2…". Based on selection template and Meta Index, we can find the satisfied set of objects by Algorithm 1. The inputs of Algorithm 1 are dimension id and item id given by selection conditions.

Algorithm 1. IndexScan(dim, itm)
Input: dim: the dimension id, itm: the item id
Output: P: the id set of objects whose values on dim are itm

```
begin
1. construct the index key K = <dim, itm>
2. search the index by K, get the value (s, t)
3. scan Meta-value Facts from s to t, get object id set P
4. return P
end.
```

If there are several selection conditions, we use Algorithm 1 to find corresponding objects of each condition, and the final result is their intersection.

The filtering Filter (SC) is based on Algorithm 1.

Algorithm 2. Filter(SC)
Input: SC: the set of selection conditions
Output: P: the id set of objects that satisfy selection conditions

```
begin
1. set P is the id set of all objects
2. for each clause in SC, (dim, itm)
3.    Q ← IndexScan(dim, itm)
4.    P = P ∩ Q
5. return P
end.
```

The Algorithm 2 shows several advantages in the process of finding the objects satisfying selection conditions: (a) No unrelated object is scanned. (b) The efficiency of disk access is high due to the sequence scan. (c) All IDs got by scan are in sequence. It is efficient to execute intersection operation.

Algorithm 3 gives the details of grouping operation based on the results of filtering step (Algorithm 2).

Algorithm 3. Group(R, D)
Input: R: the id set of objects which satisfy filtering conditions, D: dimension set
Output: P: the id set of objects in each group

```
begin
1.  P ← {R}
2.  for each dimension dim in D
3.    for each available item in dim, itm
4.      T ← IndexScan(dim, itm)
5.      for each id set S in P
6.        Q ← Q ∪(S ∩ T)
7.    P ← Q
8.    remove all elements in Q
9.  return P
end.
```

Note that, In Step 4, the function IndexScan(dim, itm) returns the id set of objects based on (dim, itm)

The result of Algorithm 3 is the set of groups of object ids which are sorted in lexicographically order. The Algorithm 2 scans Meta-value Facts Table in sequence only once, so the efficiency is high. Since the main part of Algorithm 3, like the Filtering Algorithm, is the intersection operation, we apply bit operation to improve the efficiency. Our cube query algorithm as shown in Algorithm 4.

Algorithm 4. CubeQuery(SC, D)
Input: SC: the set of selection conditions, D: the set of dimensions
Output: P: the id set of objects in each group

```
begin
1. R ← Filter(SC)
2. P ← Group(R, D)
3. return P
end.
```

As a cube query may involve the access of data on disk, we should try to minimize the times of disk access. In filtering step (Algorithm 2), our method scans the records related to selection conditions in Meta-value Facts table in sequence only once. In grouping step (Algorithm 3), our method scans the related aggregation dimensions on Meta-value Facts Table in sequence only once. For all meaningful queries, the parts of Meta-value Facts Table scanned in filtering and grouping are different, so the whole Meta-value Facts Table is scanned once in the worst case. As a result, the time complexity of our method is linear. It is worth to note that in most queries of real application, the dimensions involved is a subset of all dimensions, thus only a small part of Meta-value Facts Table will be scanned.

5 Experimental and Performance Study

To evaluate the performance of Meta Galaxy, we test it on the National Birth Defects Monitoring Database of China with about 1,000,000 instances of defect babies. The number of baby information is more than 200, including birth date, gender, body length, weight, parent ages, parent occupations, family history, defect information, and so on. Meta Galaxy is implemented in Java. The experiments are performed on an Intel Core2 1.86 GHz (2 Cores) PC with 4G memory running Windows Server 2008 64 bit Edition operating system.

5.1 Efficiency Comparison

Since the original data are kept in MS SQL Server 2005, we compare our Meta Galaxy with it. We implement two versions of Meta Galaxy, one is implemented based on SQL Server without Meta index, and the other is implemented based on file system with Meta index, denoted as R-Meta Galaxy and F-Meta Galaxy respectively. Table 2 gives the storage size used by each method in our experiment.

Table 2. The storage sizes used by SQL Server, R-Meta Galaxy and F-Meta Galaxy

SQL Server	R-Meta Galaxy	F-Meta Galaxy
2GB	2 GB	100 MB

Table 2 shows that F-Meta Galaxy can save the storage space greatly. Moreover, F-Meta Galaxy is flexible to the change of dimensions. Next, we demonstrate the efficiency of F-Meta Galaxy is higher than other models by four different methods.

- Method 1: Execute SQL statements for cube queries on original data model.
- Method 2: Create a view on R-Meta Galaxy to construct the original data model, and execute queries on this view.
- Method 3: Materialize the view created by Method 2, and execute cube queries after building the index.
- Method 4: Execute cube query algorithm (Algorithm 4) on F-Meta Galaxy to do cube queries.

In Method 2, the view is created by the statements like those in Figure 2.

For each method list above, we execute 4 typical queries. Query 1 is a simple one-dimension grouping query. Query 2 is a group query on three dimensions. And Query 3 and Query 4 are more complex by adding some selection conditions.

- Query 1: SELECT a, count(*) FROM facts GROUP BY a
- Query 2: SELECT a, b, c, count(*) FROM facts GROUP BY a, b, c

- Query 3: SELECT a, b, count(*) FROM facts WHERE c = sc1[1] GROUP BY a, b
- Query 4: SELECT a, b, count(*) FROM facts WHERE c = sc1 AND d = sc2 GROUP BY a, b

Note that, in Method 2, 3 and 4, we slightly revise the statements to adapt to Meta Galaxy, and keep the same semantic meaning. Figure 5 illustrates the query time of each method executing these four queries. The query time of Method 2 is much longer than those of other methods. The query time of Method 2 executing each query is: 6891 ms, 7750 ms, 13947 ms, and 5344 ms, respectively.

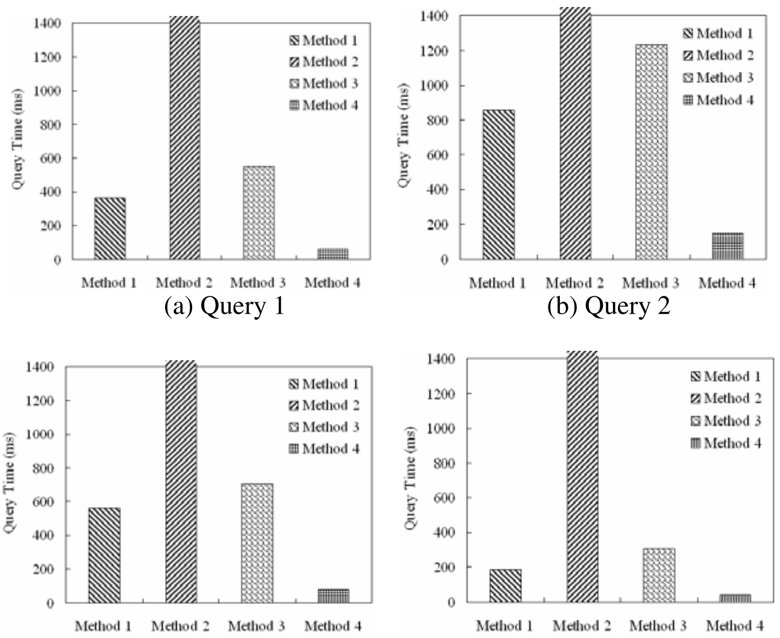


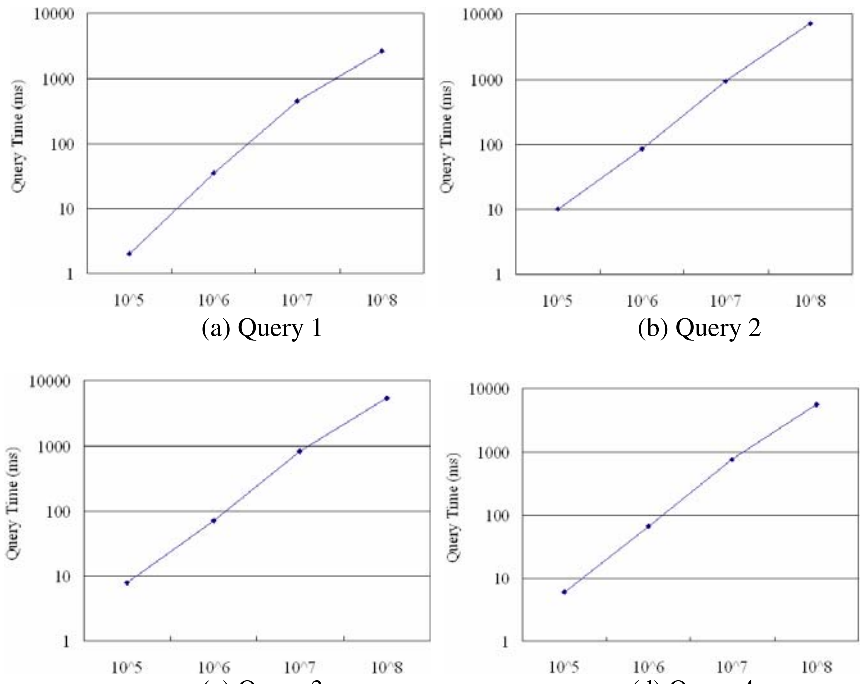
Fig. 5. The query time of each method executing four queries

Figure 5 shows that the proposed method (Method 4) greatly decreases query time. Specifically, compared with Method 1, 2, 3, the query time is decreased by 82.61%, 98.97%, 88.11%, respectively. Moreover, we can see that Method 1 and Method 3 have the similar query performance. The reason is that the query performances on a materialized view and on a table are nearly the same. As our method is implemented in Java, the performance can be improved further if we adopt C++.

5.2 Scalability Validation

To demonstrate the linear time complexity of our method, we record the query time of four queries on different sizes of data. Since the number of records in the National Birth

[1] We use sc1 to indicate a selection condition. It is an item id in our experiment.



(a) Query 1 (b) Query 2

(c) Query 3 (d) Query 4

Fig. 6. The query time of four queries in Method 4 under different data sizes

Defects Monitoring Database of China is limited, we built a data generator to randomly generate synthetic datasets. The generated datasets have following characteristics: (a) each dataset contains 200 dimensions, (b) each dimension has 20 items in average, (c) there are 20 dimensions have non-default values in average for each object.

We generate four datasets of which the numbers of objects are: 10^5, 10^6, 10^7, and 10^8. In Meta Galaxy, the storage sizes of these four datasets are: 8M, 80M, 800M and 8G bytes. For each of these datasets, we execute these four queries in Method 4 on it and record the query time. Figure 6 illustrates the query time of each query under different data size. It is clear to see that the query time is increased linearly with the dataset size becomes larger. So our proposed Meta Galaxy has a good scalability.

6 Conclusions

OLAP provide users quick answers to multi-dimensional analytical queries. The traditional design is not flexible when the data model is changed. To deal with this problem, a new cube model, called Meta Galaxy, is proposed. Moreover, an index structure and an algorithm are designed to accelerate the cube query. The time complexity of query algorithm is linear. The extensive experiments demonstrate that the newly proposed method decreases the storage size by 95.12%, decreases the query time by 89.89% in average compared with SQL Server 2005, and has good scalability on data size. The future work includes applying Meta Galaxy to more real world applications.

References

1. Han, J., Kambr, M.: Data Mining Concepts and Techniques. Higher Education Press, Beijing (2001)
2. Pedersen, T.B., Jensen, C.S.: Multidimensional database technology. Computer 34(12), 40–46 (2001)
3. Mansmann, S., Neumuth, T., Scholl, M.H.: OLAP technology for business process intelligence: Challenges and solutions. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2007. LNCS, vol. 4654, pp. 111–122. Springer, Heidelberg (2007)
4. Nedjar, S., Casali, A., Cicchetti, R., Lakhal, L.: Emerging cubes for trends analysis in OLAP databases. In: Song, I.-Y., Eder, J., Nguyen, T.M. (eds.) DaWaK 2007. LNCS, vol. 4654, pp. 135–144. Springer, Heidelberg (2007)
5. Perez, J.M., Berlanga, R., Aramburu, M.J., Pedersen, T.B.: R-Cubes: OLAP Cubes Contextualized with Documents. In: ICDE (2007)
6. Burdick, D., Deshpande, P.M., Jayram, T.S., Ramakrishnan, R., Vaithyanathan, S.: OLAP over uncertain and imprecise data. The VLDB Journal (2007)
7. Stöhr, T., Märtens, H., Rahm, E.: Multi-Dimensional Database Allocation for Parallel Data Warehouses. In: Proceedings of the 26th VLDB, Cairo, Egypt (2000)
8. Shi, Z., Huang, Y., He, Q., Xu, L., Liu, S., Qin, L., Jia, Z., Li, J., Huang, H., Zhao, L.: MSMiner—a developing platform for OLAP. Decision Support Systems, 42 (2007)
9. Asgharzadeh, Z., Chirkova, R., Fathi, Y.: Exact and Inexact Methods for Selecting Views and Indexes for OLAP Performance Improvement. In: EDBT (2008)
10. Pourabbas, E., Shoshani, A.: Efficient Estimation of Joint Queries from Multiple OLAP Databases. ACM Trans. on Database Systems V(N) (September 2006)
11. Jin, R., Yang, G., Agrawal, G.: Communication and Memory Optimal Parallel Data Cube Construction. IEEE Trans. on Parallel and Distributed Systems 12 (2005)
12. Chung, Y.D., Yang, W.S., Kim, M.H.: An efficient, robust method for processing of partial top-k/bottom-k queries using the RD-Tree in OLAP. Decision Support Systems 43 (2007)
13. Jung, T.S., Ahn, M.S., Cho, W.S.: An Efficient OLAP Query Processing Technique Using Measure Attribute Indexes. In: WISE 2004, pp. 218–228 (2004)