http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 5 (3): 565-572, 2006
ISSN 1812-5638

Partitioned Cube: Condensed Storage and Fast Access for Data Cube

Mohammad Kasem Ajaja
Jordan University for Graduate Studies (JUGS), Damascus University, Syria, Jordan

Abstract: In this study a new approach is proposed for partitioning and storing data cube efficiently. The new method partition the storage of the cube into multiple tables and uses a pointer table to link attribute combinations to attribute values. Experiments on synthetic and real data sets show that the required storage for the new method is about 60% of that required for flat table cube. Also the proposed approach has the advantage of fast access to the attribute values. Also it supports fast roll up and drill down without the need of any indexes for the attribute values.

Key words: Data cube, data warehouse, multidimensional database, OLAP, data analysis, aggregation, summary tables

INTRODUCTION

In recent years, data in enterprise operational information systems has grown to a very large size. The analysis of this data became a major objective for higher productivity in decision-making process. Data warehouse represents an essential part of data analysis. A data warehouse is subject-oriented, integrated, time-variant and nonvolatile collection of data in support of management decision making processes (Inmon, 1996). Data cube represents the heart of a data warehouse. The data cube was proposed (Gray *et al*., 1996) to pre-compute the aggregation-such as sum, average, maximum or minimum-for all possible combinations of dimensions to answer analytical queries efficiently.

Unfortunately, the sizes of the extracted data cubes for any real life databases are huge (Sismanis *et al*., 2002; Barbara and Sullivan, 1997; Lakshmanan *et al*., 2002). Huge data cubes require large storage and the features finding process is very slow. Several methods have been proposed for compressing and condensing data cubes (Sismanis *et al*., 2002; Wang *et al*., 2002; Barbara and Sullivan, 1997; Lakshmanan *et al*., 2002; Fengand Shan, 2002). Some of these methods are used to get approximate answers (Feng and Shan, 2002; Acharaya, 2000; Barbara and Sullivan, 1997; Vitter *et al*., 1998) while others try to compress the content of the cube and store it (Wang *et al*., 2002; Sismanis *et al*., 2002). In the later methods a decompression processing is required to interpret the data in the data cube.

Data cubes are used to show different views of the cube data according to a subset of dimensions with their values. Because of the large size of the extracted data cube, it is not efficient to access this specific information through exhaustive search, even if the cube is stored in a compressed format. To speed up search, several indexes are required on the attribute combinations. But these indexes require extra storage and may be they require more storage space than the cube itself. Supporting fast access to data cube data without the need of these indexes is considered a big challenge.

In this study a new method is proposed for the reduction of the storage required for the data cube. The proposed method supports the fast access for the cube data without any extra indexes. The new method depends on eliminating the storage of the value ALL from the cube by partitioning it into a number of tables equal to the number of cube dimensions. To denote the attribute combination for a specific row in the partitioned table with m dimensions, a pointer table is used to store the different combinations with m attributes where each row in the pointer table links each attribute combination to its first row in the partition with m attributes.

The proposed method does not compress any data, so it does not need any de-compression. Also it stores the cube without any loss of information, so it is used for exact answers. The proposed method has the advantage of fast access to any attribute combinations through the usage of small pointer tables. The pointer tables have the role of fast index for each attribute combination first row in the partition.

PROBLEM DESCRIPTION

Consider for example the sales database shown in Table 1. This database stores information about sales for products according to specific months and selling employees. The table that stores the original data used for data cube extraction is called fact table. The data cube that consists of the dimensions, product, month and

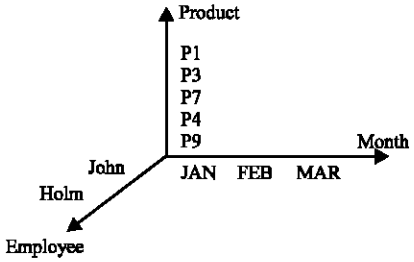


Fig. 1: Data cube for the example sales database

Table 1: Example sales database

ID	Product	Month	Employee	Amount
1	P1	Jan	John	100
2	P3	Jan	John	50
3	P7	Feb	John	60
4	P4	Mar	John	40
5	P9	Mar	Holm	90

Table 2: The data cube for the sales database

Cube

ID	Product	Month	Employee	Sum
1	All	All	All	340
2	All	All	Holm	90
3	All	All	John	250
4	All	FEB	All	60
5	All	FEB	John	60
6	All	JAN	All	150
7	All	JAN	John	150
8	All	MAR	All	130
9	All	MAR	Holm	90
10	All	MAR	John	40
11	P1	All	All	100
12	P1	All	John	100
13	P1	JAN	All	100
14	P1	JAN	John	100
15	P3	All	All	50
16	P3	All	John	50
17	P3	JAN	All	50
18	P3	JAN	John	50
19	P4	All	All	40
20	P4	All	John	40
21	P4	MAR	All	40
22	P4	MAR	John	40
23	P7	All	All	60
24	P7	All	John	60
25	P7	FEB	All	60
26	P7	FEB	John	60
27	P9	All	All	90
28	P9	All	Holm	90
29	P9	MAR	All	90
30	P9	MAR	Holm	90

Table 3: The attribute cardinalities of the example sales database

Attribute	No. of values (cardinality)
Product	5
Month	3
Employee	2

employee for the sales database is shown in Fig. 1. It is usually stored as flat table as seen in Table 2 where the value ALL in any attribute represents the computed aggregate for all values in this attribute.

In the example sales database, the data cube has three dimensions: product, month and employee that has the values and attribute cardinalities as shown Table 3.

The data cube for this dummy database consists of 30 rows as shown in Table 2. For real data sets, the extracted data cube has more than ten dimensions and the cardinalities of these dimensions ranges from 2 to several hundreds so its size is huge (Wang *et al*., 2002; Sismanis *et al*., 2002; Han *et al*., 2001; Lakshmanan *et al*., 2002) and in some cases cannot fit in main storage during its extraction. In some cases, the cubes are extracted using lower granularities for the attribute values using concept hierarchies for these attributes (Ester *et al*., 2000). Usually, the extracted cube is represented as a flat table for the attribute values by adding the computed aggregate. In this case, the value ALL in any column represents the aggregate computation for all values in this column.. Some literature uses the symbol * to denote the value ALL and some commercial databases uses null for the same purpose. All the previous representations require storage space for the value ALL. The dimensions of the data cube are either categorical with limited number of values or continuous for numeric and date attributes. The continuous attributes are divided into a reasonable set of intervals. Now handling the data cube is done by mapping continuous attributes to the chosen intervals. In the case of continuous attributes, the number of intervals cannot be selected to be very large because this tends to a larger data cube. Also very small number of intervals will hide a lot of trends represented in the cube. To reduce required storage for the cube, both categorical attributes and mapped intervals for continuous attributes are indexed and represented by small number of values. Attribute indexes have very small length and the value ALL still exists in the cube represented as flat table and requires storage space to represent it. Consider the example cube shown in Table 2 there are $30 \times 3 = 90$ attribute values represented and stored in this data cube. From these attribute values there are 37 'ALL' values that represent a ratio of about 41% from all stored values. This ratio will increase as the number of dimensions increases or the attribute cardinality increases. Eliminating the storage of 'ALL' values will decrease the size of the stored cube. In this paper a new storage approach is proposed for data cubes that eliminate storing 'ALL' values while providing fast access for the cube contents. Data cubes are used mainly for analyzing the data and extracting unknown and unusual features through visualization and other methods (Sathe and Sarawagi, 2001; Dong *et al*., 2001; Keim and Kriegel, 1996). The proposed method has the advantage of fast roll up and drill down throw the structure of the pointer tables as will be described later.

RELATED WORK

The problem of huge data cube has attracted researchers to develop new methods for extracting, representing and exploiting them efficiently (Barbara and Sullivan, 1997; Beyer and Ramakrishnan, 1999; Fang *et al*., 1998; Han *et al*., 2001; Lakshmanan *et al*., 2002; Lakshmanan *et al*., 2003; Ross and Srivastana, 1997). In (Wang *et al*., 2002) a new structure called condensed cube was proposed to reduce the represented rows in the cube. This method depends on the combinations of values for cube dimensions. It depends on eliminating cells that can be computed from other cells in the cube. However, this method adds an un-normalized column to the cube where this column stores more than one value to show the attribute combinations for the values in this row. This column requires large storage space compared with the compression ratio achieved. Furthermore, this method requires exhaustive search for attribute values to get a specific value or new indexes are required to speed up the search. These indexes require extra storage space. A dwarf cube is proposed by Sismanis *et al*. (2002) that eliminates the representation of redundant values in the cube combinations. In (Lakshmanan *et al*., 2002) the cube is partitioned according similar values of the computed aggregates which is rare in large real life databases.

THE PROPOSED APPROACH

To the best of my knowledge, the problem of eliminating 'ALL' values and de-normalizing the storage structure of the data cube has not been addressed before. The proposed approach depends on eliminating 'ALL' values represented in the extracted cube by using a storage structure suitable for this purpose.

Let us consider the database shown in Table 1 and the extracted cube for this fact table shown in Table 2. The storage of ALL values can be eliminated if the structure of the representation has changed. Looking again at the data cube shown in Table 2, we notice that there are several attribute combinations that store values other than 'ALL'. For example row 2 has value for one attribute (Employee = Holm) and row 10 has values for two attributes (Month = MAR and Employee = John). To de-normalize the representation of the data cube we can divide this flat table into many tables each of them has different attribute combinations. Figure 2 shows all the partitions that have different number of attributes. For Fig. 2a-c, they have one dimension from the cube in addition to the aggregate function computed for the values of this dimension. Table d-f in Fig. 2 have two dimensions and the computed aggregate. Similarly, the Table g in Fig. 2 has three dimensions and the computed aggregate function for the different combinations of the attribute values. If one needs to divide the computed cube to this form, too many tables are required. So we need a distinct table for each attribute combinations where the number of attributes in the combinations ranges from 1 to the number of dimensions in the cube. For a cube with N dimensions, the number of tables with x (where $1 \le x \le N$) attributes is:

$$M(x) = \frac{N(N-1) \ldots..(N-x-1)}{x!} \quad (1)$$

The total number of tables for all attribute combinations is

$$T = \sum_{x=1}^{N} \frac{N(N-1) \ldots..(N-x-1)}{x!}$$

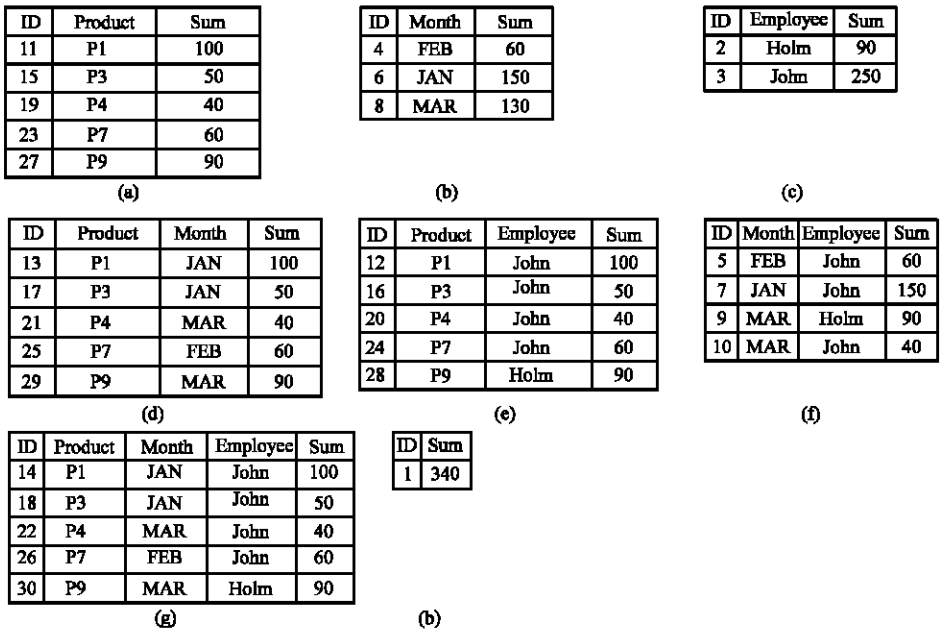
Multiplying with (N-x) factorial we can find

$$T = \sum_{x=1}^{N} \frac{N(N-1) \ldots..(N-x-1)(N-x).....1}{x!\,(N-x)!} = \sum_{x=1}^{N} \frac{N!}{x!\,(N-x)!} \quad (2)$$

So for example, if number of dimensions N is 5 the total number of tables is 31 and the total number of tables is 1023 for N = 10.

From the previous analysis, if one needs to partition the cube into many tables each of which has different attribute combination, it is concluded that the number of tables increases dramatically by the increase of the number of dimensions. Also the number of attribute values that contain 'ALL' in the original flat table cube represents a high percentage of the total attribute values in the cube and these values if eliminated from the representation will result in a high reduction of the storage required for the computed cube. Looking again to the example partitioned Tables from Fig. 2, it is seen that there are many tables with the same number of dimensions but with different attributes. If one merges all tables with the same number of dimensions in one table, the total number of tables required for the storage will be reduced very much. In this method, only N tables are required to store all cube data without any loss of information and 'ALL' attribute values from the original flat table cube are eliminated.

To merge the tables with different attributes but have the same number of dimensions, a pointer is required to link the part of the merged table to the attribute



ID	Product	Sum
11	P1	100
15	P3	50
19	P4	40
23	P7	60
27	P9	90

(a)

ID	Month	Sum
4	FEB	60
6	JAN	150
8	MAR	130

(b)

ID	Employee	Sum
2	Holm	90
3	John	250

(c)

ID	Product	Month	Sum
13	P1	JAN	100
17	P3	JAN	50
21	P4	MAR	40
25	P7	FEB	60
29	P9	MAR	90

(d)

ID	Product	Employee	Sum
12	P1	John	100
16	P3	John	50
20	P4	John	40
24	P7	John	60
28	P9	Holm	90

(e)

ID	Month	Employee	Sum
5	FEB	John	60
7	JAN	John	150
9	MAR	Holm	90
10	MAR	John	40

(f)

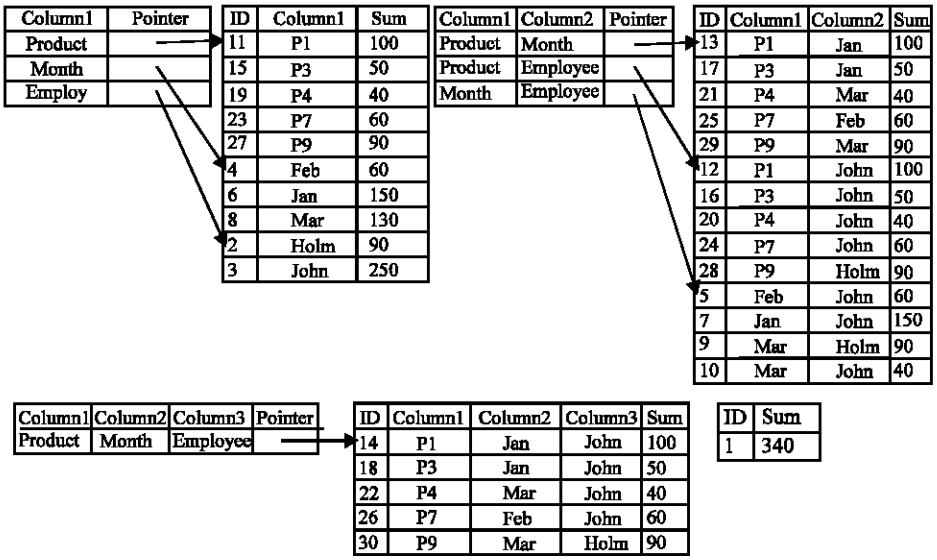
ID	Product	Month	Employee	Sum
14	P1	JAN	John	100
18	P3	JAN	John	50
22	P4	MAR	John	40
26	P7	FEB	John	60
30	P9	MAR	Holm	90

(g)

ID	Sum
1	340

(b)

Fig. 2: The cube partitioned into a table for each attribute combination



Column1	Pointer
Product	
Month	
Employ	

ID	Column1	Sum
11	P1	100
15	P3	50
19	P4	40
23	P7	60
27	P9	90
4	Feb	60
6	Jan	150
8	Mar	130
2	Holm	90
3	John	250

Column1	Column2	Pointer
Product	Month	
Product	Employee	
Month	Employee	

ID	Column1	Column2	Sum
13	P1	Jan	100
17	P3	Jan	50
21	P4	Mar	40
25	P7	Feb	60
29	P9	Mar	90
12	P1	John	100
16	P3	John	50
20	P4	John	40
24	P7	John	60
28	P9	Holm	90
5	Feb	John	60
7	Jan	John	150
9	Mar	Holm	90
10	Mar	John	40

Column1	Column2	Column3	Pointer
Product	Month	Employee	

ID	Column1	Column2	Column3	Sum
14	P1	Jan	John	100
18	P3	Jan	John	50
22	P4	Mar	John	40
26	P7	Feb	John	60
30	P9	Mar	Holm	90

ID	Sum
1	340

Fig. 3: Partitioned cube storage, a distinct table for all attribute combinations with the same number

combination. Consider this pointer has the structure as seen in Fig. 3. The pointer has a structure of a table with many rows for all attribute combinations with the same number of attributes. The storage structure of the merged tables with the pointer tables is called the partitioned cube.

It is clear from the partitioned cube representation that they do not contain the value 'ALL', so no lost storage for these values. Using this method for data cube storage requires number of tables equals to the number of dimensions in the computed data cube. Each table stores the attribute values for all attribute combinations with the same number of columns (Fig. 3). The pointer tables have a very important advantage where they point directly to the attributes combination of the computed aggregate. To access some attribute values from the partitioned cube, one first searches the pointer table with the same number of attributes. The size of this pointer table is too small

where each pointer table used for x number of attributes in a cube with N dimensions contains only number of entries as computed in Eq. 1. For example, a cube with 10 dimensions has 252 entries in the pointer table with 5 attributes and 120 entries in the pointer table with 7 attributes. The data in the data cube tends to be highly static, so each set of rows related to the same attribute combination can be sorted according to the attribute values. This storage enables fast access for the attribute values.

It should be noted that, the total number of rows in the partitioned cube equals to the number of rows in the flat table data cube and only the value ALL has been eliminated from the partitioned storage. According to the type of data stored in the partitioned cube columns, it should be able to represent any simple type that can be stored in the database such as number, string, date ….etc. The type varchar from the SQL standard is able to represent all the values from the other types. For example, numeric values are represented by digits from 0 to 9 and date is represented as a string with the form dd/mm/yyyy. As mentioned before, the actual values of the categorical attributes are mapped to an index values and so for intervals of categorical attributes. In this case, the indexes are always numbers and require small storage compared to the original values.

Concerning the extraction of the data cube using the proposed storage structure, all proposed algorithms can be used (Beyer and Ramakrishnan, 1999; Han *et al*., 2001; Rossand Srivastana, 1997) with a very slight modification. Instead of using a single storage structure for the extracted cube, multiple structures are used according to the attributes combinations suitable for the proposed approach.

EVALUATION STUDY

Here an evaluation study will be presented for the proposed storage method. The storage required for the proposed method is compared with that required for the cube represented in a flat table. The unit of the comparison is the number of attribute values required for each of the two methods.

The evaluation study uses synthetic data generated for this purpose. The fact table has the following structure:

FACT(C1,C2,C3,C4,C5,C6,C7,C8,C9,C9,C10,C11,C12,C13,M)

It consists of 13 attributes each has a cardinality of 25 and a quantitative attribute for the computed aggregate.

The content of the fact table ranges from 10 to 100 k records. The first experiment consists of computing the cube for all dimensions. Figure 4 shows the required storage in number of attribute values for flat table cube and partitioned cube according to the number of records in the fact table.

It is clear from this figure that the partitioned cube requires about 50% storage space compared with the flat table cube. Also the flat table cube requires several indexes to access the computed values for some attribute values. Consider that the visualization requires a roll up for the columns C1, C2, C7 and C9, it requires an index that contains these attributes as major key in the case of flat table representation. To visualize the same attributes in the partitioned cube, it is done directly from the pointer table that points to the partition with 4 attributes (which is very small compared to the flat table cube and it contains at most 1365 entries according to equation 1) and searches for the columns C1, C2, C7 and C9 where this entry points directly to the required data in the partition that contains four attributes. It should be noted that the contents of the pointer table can be sorted because it is never changed and binary search or another fast search method can be used.

The second experiment has a fact table of 20k records and attributes cardinality is 25 and the cube is computed for different number of dimensions ranges from 2 to 10. Figure 5 shows the required storage (again as number of values) for flat table cube and for the total storage required for the partitioned cube according to the number of dimensions in the cube. It is clear that the required storage for the partitioned cube is less than that required for the flat table representation. Figure 6 shows the same information but as percentage storage ratio.

The third experiment has a fact table of 20k records and the number of attributes is 10 and attributes cardinality ranges from 2 to 100. Figure 7 shows the required storage for flat table cube and for partitioned cube. It is clear that the required storage for the partitioned cube is about 50% of that required for the flat table cube. Figure 8 shows the ratio between required storage in partitioned cube and flat table cube according to the attributes cardinality.

In the practical experiments, a comparison study is presented for the storage required in the proposed method and the original cube measured in number of values for data cubes computed for real life data sets. Also, the storage required in Kbytes in one of the commercial databases is recorded for both methods. The first real life data set is used for tracking purchase orders for a large network of retailers. The fact table has the attributes shown in Table 4 together with their number of values and the intervals for continuous attributes.

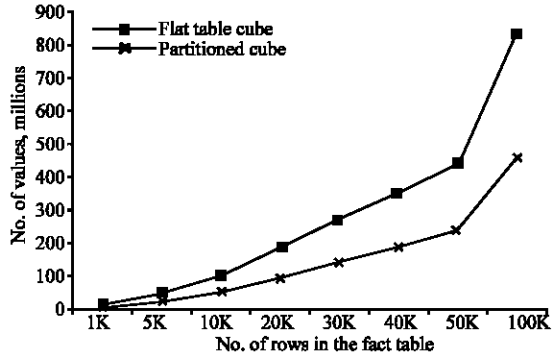


Fig. 4: Number of stored attribute values for flat table cube and partitioned cube according to the number of rows in the fact table

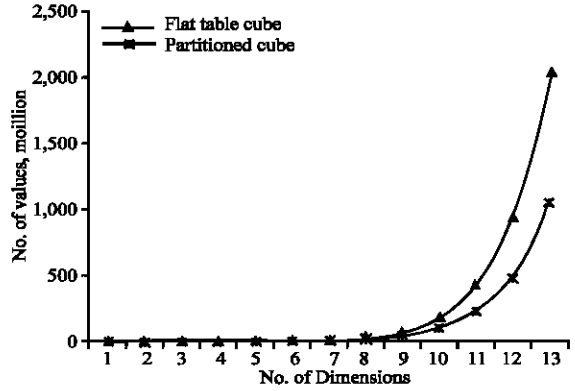


Fig. 5: Required storage for flat table cube and for the partitioned cube according to the number of dimensions in the cube

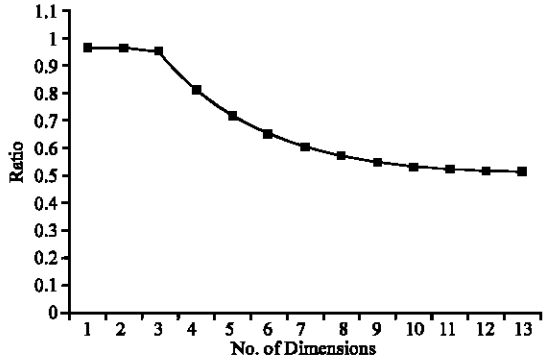


Fig. 6: The ratio between storage required for partitioned cube and storage for flat table cube according to the number of dimensions

The purchasing database contains 8245 records. The extracted data cube is used for all dimensions and the measure is the average period between due date and arrival date.

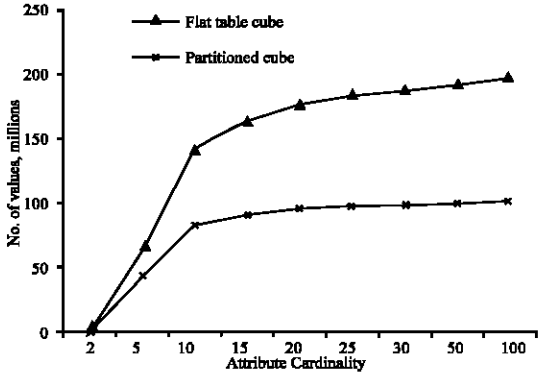


Fig. 7: The required storage for flat table cube and for partitioned cube according to the attributes cardinality

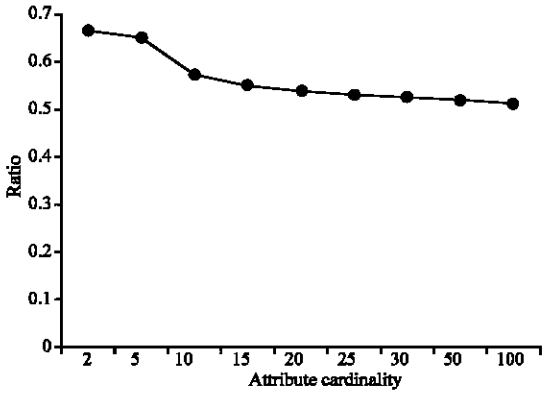


Fig. 8: The ratio between required storage in partitioned cube and flat table cube according to the attributes cardinality

Table 4: The structure of the purchasing database for a retail network

Attribute	Type	No. of values	Interval
Purchase order date	Continuous	1/1/1991- 30/12/2000	Week
Purchase order type	Categorical	2	
Supplier ID	Categorical	1377	
Employee ID	Categorical	132	
Due date	Continuous	20/1/1991-30/12/2000	Week
Shipping date	Continuous	23/2/1991-14/4/2001	Week
Arrival date	Continuous	26/2/1991-22/6/2001	Week
Shipping type	Categorical	4	
Shipping cost	Continuous	3k-31k	0.1k
Total cost	Continuous	17k-852k	1k

Table 5: The structure of the employee annual tax database

Attribute	Type	Values/Range	Interval
Employee_number	Identifier	237452	
Year	Continuous	1982-1992	1 year
Age	Continuous	18-72	1 year
Employment type	Categorical	5	
Education level	Categorical	5	
Education type	Categorical	8	
Gender	Categorical	2	
Marital_status	Categorical	4	
Own_house	Categorical	2	
Nationality	Categorical	17	
Annual_tax	Continuous	3-61k	

Table 6: Required storage for flat table cube and for the partitioned cube for two real life data sets

			Storage for flat table cube		Storage for partitioned cube		
Data set	Rows in fact table	Rows in data cube	No. of values	Kbytes	No. of values	Kbytes	Percentage (Values)
Purchasing	8245	8009620	80096200	267880	41245941	151968	51%
Emp. annual tax	1977975	69666096	626994864	2258392	382466341	1422799	61%

The second real life data set is used to record the annual tax for employees in the years from 1982 until 1992. The fact table has the attributes shown in Table 5 together with their number of values and the intervals for continuous attributes.

The data cube extracted for the annual employee tax consists of all dimensions except the identifier of the employee. The annual employee tax contains 1977975 rows since not all the employees has a distinct row for each year in the interval from 1982 until 1992 where new employees entered the work after 1982 and old employees left the work before 1992.

Table 6 shows a comparison for the required storage for both flat table cube and partitioned cube for the previous data sets. It is clear from this table that the proposed method requires less than 60% storage space of that required for the flat table cube. The table also shows the required storage in Kbytes for both methods using one of the most common commercial databases.

CONCLUSION AND FUTURE RESEARCH

In this study a new approach is proposed for partitioning and storing data cube efficiently. The proposed method eliminates the storage of the value ALL by partitioning the flat table cube into number of tables equals to the number of dimensions in the cube. Each table stores the attributes values and the computed measure for combinations with the same number of attributes. A set of pointer tables are used to link different attribute combinations to the attribute values in the partitioned tables. The pointer tables have the advantage of fast access for the attribute values since they are very small compared to the original cube size. The fast access for the attribute values is very important in rollup and drill down without the need of any indexes for the attribute values.

Experiments on synthetic and real data sets show that the required storage for the new method is about 60% of that required for flat table cube.

All previously proposed methods for data cube extraction can be used and adopted to use the proposed storage approach.

Developing new visualization method for the cube content stored in partitioned cube is currently under development.

ACKNOWLEDGEMENTS

I would like to thank Mr. Rudi Bayer for his practical suggestions and notes on the implementation of the proposed method. Also many thanks to Advanced Data View for providing the real life data sets used in analyzing the storage space for the proposed method.

REFERENCES

Acharaya, S., P.B. Gibbons and V. Poosla, 2000. Congressional samples for approximate answering of group-by queries. In SIGMOD 200, Vol. 29.

Barbara, D. and M. Sullivan, 1997. Quasi-cubes: Exploiting approximation in multidimensional databases. In Proceedings of International Conference on Management of Data (SIGMOD).

Beyer, K. and R. Ramakrishnan, 1999. Bottom-up computation of sparse and iceberg cubes. In Proceedings of International Conference on Management of Data (SIGMOD).

Dong, G., J. Han, J. Lam, J. Pei and K. Wang, 2001. Mining Multi-Dimensional Constrained Gradients in Data Cubes. In Proceedings of the VLDB 2001 Conference, Italy.

Ester, M., J. Kohlhammer and H.P. Kriegel, 2000. The DC-Tree: A Fully Dynamic Index Structure for Data Warehouses. In Proceedings of 16th International Conference on Data Engineering (ICDE'2000).

Fang, M.N., H. Shivakumar, R. Garcia-Molina, Motwani and J.D. Ullman, 1998. Computing iceberg queries efficiently. In: Proceedings of International Conference on Very Large Databases (VLDB).

Yu, F. and W. Shan, 2002. Compressed data cube for approximate OLAP query processing. J. Compute. Sci. Technol., 17: 625-635.

Gray, J.A., Bosworth, A. Layman and H. Pirahesh, 1996. Data cube: A relational aggregation operator generalizing group-by, cross-tab and sub-totals. In Proceedings of International Conference on Data Engineering (ICDE).

Han, J., J. Pei, G. dong and K. Wang, 2001. Efficient computation of iceberg cubes with complex measures. In: Proceedings of International Conference on Management of Data (SIGMOD).

Inmon. W.H., 1996. Building the Data Warehouse. John Wiley and Sons.

Keim, D.A. and H.P. Kriegel, 1996. Visualization Techniques for Mining Large Databases: A Comparison. In IEEE Transactions on Knowledge and Data Engineering, 8: 6, Dec. 1996.

Lakshmanan, L.V.S., J. Pei and J. Han, 2002. Quotient cube: How to summarize the semantics of a data cube. In Proceedings of International Conference on Very Large Database (VLDB).

Lakshmanan, L.V.S., P. Jian and Y. Zhao, 2003. Snakes and sandwiches: Optimal clustering strategies for a data warehouse. In: Proceedings of International Conference on Management of Data (SIGMOD).

Ross, K.A. and D. Srivastana, 1997. Fast computation of sparse data cubes. In: Proceedings of International Conference on Very Large Database (VLDB).

Sathe, G. and S. Sarawagi, 2001. Intelligent Rollups in Multidimensional OLAP Data. In Proceedings of the 27th VLDB Conference, Roma, Italy.

Sismanis, Y., A. Deligiannakis, N. Roussopoulos and Y. Kotidis, 2002. Dwarf: Shrinking the peta-cube. In Proceedings of International Conference on Management of Data (SIGMOD).

Vitter, J.S., M. Wang and B.R. Iyer, 1998. Data cube Approximation and histograms via wavelets. In CIKM98.

Wang, W., J. Feng, H. Lu and J. Yu, 2002. Condensed cube: An effective approach to reducing data cube size. In: Proceedings of International Conference on Data Engineering (ICDE).