An Efficient Approach for Providing multidimensional Data on Relational DBMSs°

Jinho Kim, Sung-Hyun Shin, Hun-Young Choi, and Yang-Sae Moon

Department of Computer Science, Kangwon National University

192-1 Hyoja-2 Dong, Chuncheon, Kangwon 200-701, Korea and

Advanced Information Technology Research Center (AITrc)

Korea Advanced Institute of Science and Technology (KAIST)

373-1 Guseong-dong, Yuseong-gu, Daejeon 305-701, Korea

E-mail: {jhkim, shshin, hychoi, ysmoon}@kangwon.ac.kr, Fax:+82-33-250-8440

Abstract Multidimensional data are used in popular to support OLAP(On-Line Analytical Processing) operations efficiently. Many OLAP tools have been developed to manipulate the multidimensional data to provide users with useful information for decision making. While they are equipped with useful features, they seem be sometimes complicated to relational database users. In this paper, we developed an easy and effective interface manipulating multidimensional data on the top of relational DBMSs. The proposed interface provides a mechanism handling data in relational tables like spreadsheet (e.g., Microsoft Excel) does. Spreadsheet is the most popular tool for end users to analysis their data conveniently. The interface provides the features browsing relational tables virtually with sheet by sheet and creating pivot tables. Pivot tables are a useful presentation form of multidimensional data, which have attribute values (not attribute names) as their columns thus have horizontal format. We store pivot tables of horizontal format as vertical tables in relational DBMSs and we provide the transformation of queries on horizontal table into ones on vertical tables. With this user-friendly interface, users will be able to perform analysis activities on data in relational DBMSs easily and effectively.

Keywords: Data warehouse, OLAP, Multidimensional data, PIVOT, Spreadsheet.

1. Introduction

OLAP(On-Line Analytical Processing) is technology supporting multidimensional data analysis used to effectively extract and analyze information from huge data[1]. Various tools have been developed to provide OLAP activities effectively, which maintain data in multidimensional form and manipulate the data over various dimensions. These OLAP tools have many useful features to process OLAP operations analyzing a huge size of business data and extracting valuable information from them for decision making, but their elegant features sometimes seem to make end users complicate to use. In real world, end users use spreadsheet tools like MS Excel to deal with business data and to analyze them simply the most.

Because spreadsheets represent data with simple 2-dimensional grid form, support calculation over the data easily, and provide various charts, they became the most popular tool to manipulate and to analyze data easily. Because of simple but useful property of spreadsheet tools, several researches have been done to take advantage of spreadsheets in OLAP systems. A straightforward way of utilizing spreadsheets is to check out data in databases (or data warehouses) into a file then to load it into spreadsheets. In general, spreadsheets can have the limited number of rows and columns thus they may not be able to handle directly the whole data stored in relational tables which can have a huge number of tuples. An alternative way is to extend database systems to incorporate spreadsheet-like features. As one of such kinds of effort, Witkowski et al. [2] extended SQL

° This work was supported by Korea Science and Engineering Foundation(KOSEF) through the Advanced Information Technology Research Center(AITrc).

statements to add spreadsheet clause which permits array-based calculation being used in spreadsheet tools. Through this new clause, they tried to provide multidimensional array-based calculation functions into SQL language but they just provide users with such new features in a syntactical form and they don't provide them with a GUI(Graphic User Interface) facility just like spreadsheet tools do .

In this paper, we developed a graphical interface which can browse multidimensional data stored into databases like spreadsheets, which is a virtual spreadsheet interface over databases. The interface has features partitioning database tables into sheets and browsing them with sheet by sheet. This sheet-partitioning function partition tables over either a specific number of rows(i.e., tuples) or the values of a specific column (i.e., attribute). It also provides the feature building pivot tables over database tables. Each pivot table has a 2-dimensional array (or table) form which has attribute values (not attribute names) of a relational table as its columns. While it is a popular form presenting multidimensional data, the pivot table has a horizontal schema with many columns (sometimes, it can have more than thousands of columns) [2, 3, 4, 5]. Because relational DBMSs allow relations to have limited number of attributes (i.e., columns), it is difficult to store horizontal tables into relational databases and to provide users with horizontal tables directly [3, 4].

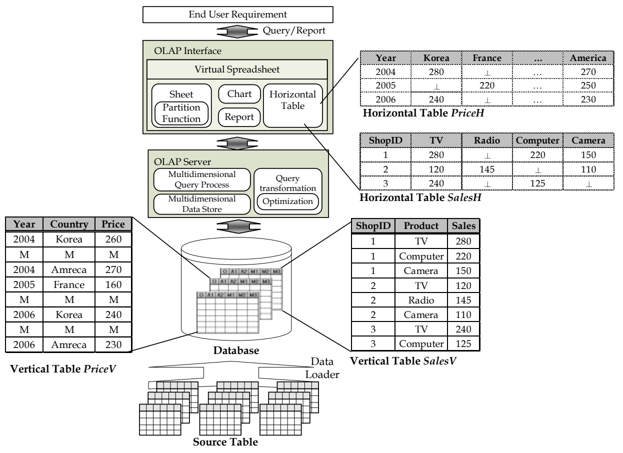
To solve the problem of storing horizontal tables, there have been some researches which convert horizontal tables into vertical tables and store the vertical tables physically. Agrawal et al. [3] proposed a method which stores horizontal tables by using vertical tables and convert queries for horizontal tables into ones for vertical tables. Cunningham et al. [6] extended the work of [3] by proposing PIVOT/UNPIVOT operations as new basic algebraic operations which relational DBMS engines have to provide. PIVOT (or UNPIOVT) is the operation converting vertical tables (or horizontal tables) into horizontal tables (or vertical tables). Chen and Rundensteiner [7] introduced GPIVOT operation which generalized PIVOT operation to handle a pivot table with a hierarch of attribute values in its columns. Existing research [3] converting vertical tables into horizontal tables use basic relational algebraic operations only and they didn't utilize PIVOT operators[8] which recent commercial DBMSs provide. In this paper, therefore, we propose transformation rules employing the PIVOT operation of commercial DBMSs when converting vertical tables into horizontal tables. With these rules, we can provide users with user-friendly view of horizontal table and we can efficiently transform queries of horizontal table into ones of vertical table. Also we investigate an optimization strategy of transformed queries including PIVOT operators.

The rest of the paper is organized as follows. Section 2 introduces the overall architecture of the proposed OLAP system. Section 3 introduces a spreadsheet-based OLAP interface which employs sheet-partitioning and pivot functions. Section 4 presents query transformation rules to convert horizontal table queries into vertical table queries including PIVOT operations. We also present an optimization technique for processing transformed vertical queries in Section 5 and we finally conclude this paper in Section 6.

2. Overall Architecture of the OLAP system

Figure 1 shows the overall architecture of the OLAP system aimed in this paper. As shown in the figure, when users send their requests or queries for data analysis into OLAP server by using OLAP interfaces, the system processes them over source tables stored in vertical form and provides users with their results in the form of spreadsheet. The OLAP system consists of three modules such as OLAP interface module, OLAP server module and relational DBMS. First, the OLAP interface module provides users with an interface for multidimensional analysis. Users can manipulate multidimensional data easily by using a GUI interface which provides the results of queries. The results will be presented to users with horizontal tabular views which is widely used in the applications of data analysis, like cross tabulation in statistics. These horizontal results may have a huge size of data which cannot be loaded in a spreadsheet file. Thus the OLAP interface in the proposed system provides a function to browse huge results of queries virtually with sheet by sheet.

Next, the OLAP server module provides the functions: a) processing users' ordinary queries in OLAP tool; b) transforming horizontal tabular views into vertical tables which will be physically stored in relational DBMSs; and c) converting queries for horizontal tables into ones for vertical tables. Especially, PIVOT operators are utilized for the query transformation between horizontal and vertical table queries in order to process efficiently.



Year	Korea	France	…	America
2004	280	⊥	…	270
2005	⊥	220	…	250
2006	240	⊥	…	230

Horizontal Table *PriceH*

ShopID	TV	Radio	Computer	Camera
1	280	⊥	220	150
2	120	145	⊥	110
3	240	⊥	125	⊥

Horizontal Table *SalesH*

Year	Country	Price
2004	Korea	260
M	M	M
2004	Amreca	270
2005	France	160
M	M	M
2006	Korea	240
M	M	M
2006	Amreca	230

Vertical Table *PriceV*

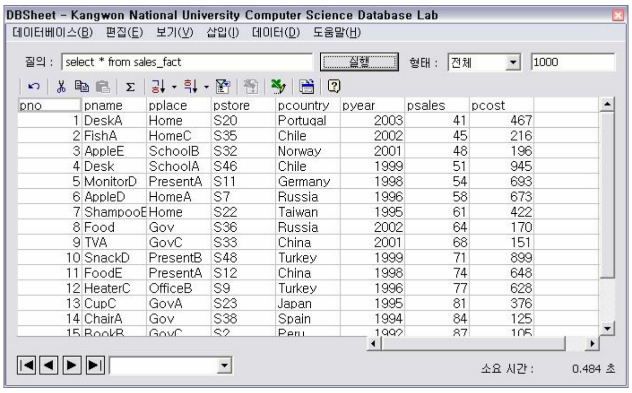
ShopID	Product	Sales
1	TV	280
1	Computer	220
1	Camera	150
2	TV	120
2	Radio	145
2	Camera	110
3	TV	240
3	Computer	125

Vertical Table *SalesV*

Figure 1. Overall architecture of the OLAP system.

3. The OLAP Interface

This section describes the spreadsheet-based OLAP interface which is a user interface of the proposed OLAP system to handle data analysis and queries. This interface has functions which help users to compose queries over multidimensional data in accordance with their OLAP requests and present results of the queries as user-friendly formats. The figure 2 shows a typical example of a query result. The result was represented as a form similar to spreadsheet, which can be used for further manipulation by users.


DBSheet – Kangwon National University Computer Science Database Lab

데이터베이스(B) 편집(E) 보기(V) 삽입(I) 데이터(D) 도움말(H)

질의 : select * from sales_fact 실행 형태 : 전체 1000

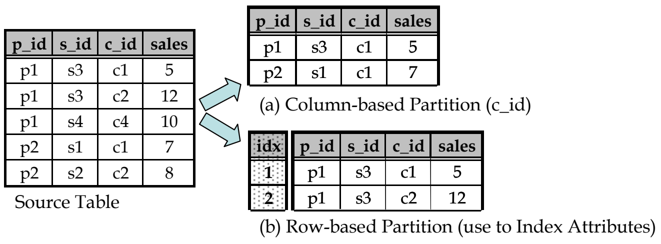
pno	pname	pplace	pstore	pcountry	pyear	psales	pcost
1	DeskA	Home	S20	Portugal	2003	41	467
2	FishA	HomeC	S35	Chile	2002	45	216
3	AppleE	SchoolB	S32	Norway	2001	48	196
4	Desk	SchoolA	S46	Chile	1999	51	945
5	MonitorD	PresentA	S11	Germany	1998	54	693
6	AppleD	HomeA	S7	Russia	1996	58	673
7	ShampooE	Home	S22	Taiwan	1995	61	422
8	Food	Gov	S36	Russia	2002	64	170
9	TVA	GovC	S33	China	2001	68	151
10	SnackD	PresentB	S48	Turkey	1999	71	899
11	FoodE	PresentA	S12	China	1998	74	648
12	HeaterC	OfficeB	S9	Turkey	1996	77	628
13	CupC	GovA	S23	Japan	1995	81	376
14	ChairA	Gov	S38	Spain	1994	84	125
15	BookB	GovC	S2	Peru	1992	87	105

소요 시간 : 0.484 초

Figure 2. OLAP interface.

Spreadsheet is a very convenient tool for simple analysis over data. But it is very inefficient to load the whole content of data stored in relational DBMS into a spreadsheet file. Because it takes much time to access the entire data and users don't need all of them at every time. Users may often want to handle some portion of query results at a specific time. Thus it is recommendable to access some portion of the data in a large table (or query result) which users want to focus on at a specific time.

In order to compromise the problem of handling tables with large size, this research develops a spreadsheet-like interface which partitions tables into a set of sheets and makes users to browse them sheet by sheet like spreadsheet. When a user wants to use a specific sheet, the corresponding small portion of a table only is accessed. The interface developed in this research permits the mechanism partitioning tables into sheets over both the number of rows (i.e., tuples) and attribute values: row-based and column-based partitioning. Row-based partitioning partitions a table by the number of rows specified by users. Column-based partitioning does it over the values of a specified attribute. Figure 3(a) and (b) show the basic concept of these two sheet partitioning methods.



p_id	s_id	c_id	sales
p1	s3	c1	5
p1	s3	c2	12
p1	s4	c4	10
p2	s1	c1	7
p2	s2	c2	8

Source Table

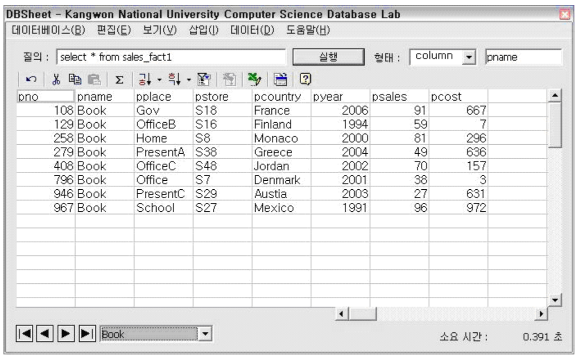
p_id	s_id	c_id	sales
p1	s3	c1	5
p2	s1	c1	7

(a) Column-based Partition (c_id)

idx	p_id	s_id	c_id	sales
1	p1	s3	c1	5
2	p1	s3	c2	12

(b) Row-based Partition (use to Index Attributes)

Figure 3. Row-based and column-based sheet partition.


DBSheet – Kangwon National University Computer Science Database Lab

데이터베이스(B) 편집(E) 보기(V) 삽입(I) 데이터(D) 도움말(H)

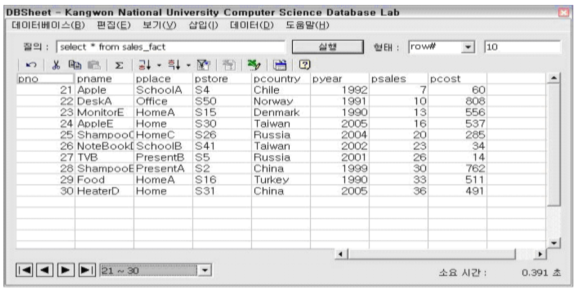
질의 : select * from sales_fact1 실행 형태 : column pname

pno	pname	pplace	pstore	pcountry	pyear	psales	pcost
108	Book	Gov	S18	France	2006	91	667
129	Book	OfficeB	S16	Finland	1994	59	7
258	Book	Home	S8	Monaco	2000	81	296
279	Book	PresentA	S38	Greece	2004	49	636
408	Book	OfficeC	S48	Jordan	2002	70	157
796	Book	Office	S7	Denmark	2001	38	3
946	Book	PresentC	S29	Austia	2003	27	631
967	Book	School	S27	Mexico	1991	96	972

Book

소요 시간 : 0.391 초

(a) Column-based sheet partition


DBSheet – Kangwon National University Computer Science Database Lab

데이터베이스(B) 편집(E) 보기(V) 삽입(I) 데이터(D) 도움말(H)

질의 : select * from sales_fact 실행 형태 : row# 10

pno	pname	pplace	pstore	pcountry	pyear	psales	pcost
21	Apple	SchoolA	S4	Chile	1992	7	60
22	DeskA	Office	S50	Norway	1991	10	808
23	MonitorE	HomeA	S15	Denmark	1990	13	556
24	AppleE	Home	S30	Taiwan	2005	16	537
25	ShampooC	HomeC	S26	Russia	2004	20	285
26	NoteBookD	SchoolB	S41	Taiwan	2002	23	34
27	TVB	PresentB	S5	Russia	2001	26	14
28	ShampooE	PresentA	S2	China	1999	30	762
29	Food	HomeA	S16	Turkey	1990	33	511
30	HeaterD	Home	S31	China	2005	36	491

21 ~ 30

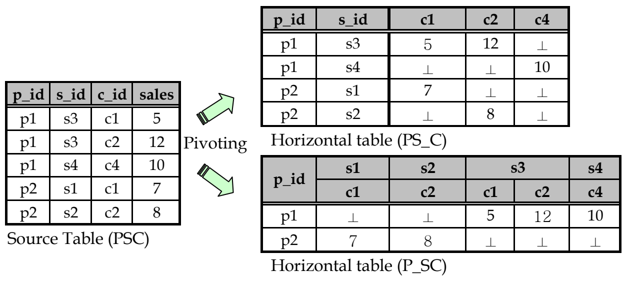
소요 시간 : 0.391 초

(b) Row-based sheet partition

Figure 4. Sheet partition execution and result in OLAP interface.

Figure 4 shows the presentation results of the OLAP interface using column-based and row-based sheet partitioning for an example query on the *sales_fact1* table which is supposed to have a large volume of data. Figure 4(a) shows a sheet partitioned by the values of specified attribute (i.e., *pname*). Figure 4(b) shows one sheet partitioned by a number of rows (i.e., *10 rows*). All sheets for the example query result are listed in the list box shown in the bottom of the result window.

Next, we provide the facility making pivot tables like MS Excel, which change the role of rows and columns to have the values of attributes as columns. Pivot tables are usually horizontal tables which have a lot of columns and have lots of null values. These horizontal tables are being used very popularly in e-business applications and statistical analysis. (They are often called cross tabulation in statistics). This system incorporates pivoting feature over relational tables to provide users with pivot tables of horizontal format as users' views. Figure 5 shows example pivot tables derived from a source table stored in relational databases. As shown in the figure, the pivoting feature creates pivot tables over several rows and columns specified by users and it provides horizontal representation which is a user-friendly form in OLAP.

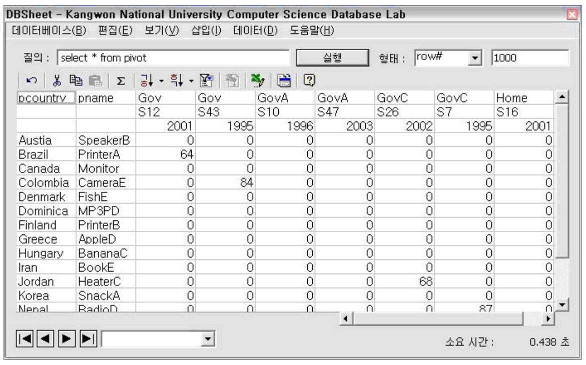

Pivoting
Source Table (PSC)
Horizontal table (PS_C)
Horizontal table (P_SC)

p_id	s_id	c_id	sales
p1	s3	c1	5
p1	s3	c2	12
p1	s4	c4	10
p2	s1	c1	7
p2	s2	c2	8

p_id	s_id	c1	c2	c4
p1	s3	5	12	⊥
p1	s4	⊥	⊥	10
p2	s1	7	⊥	⊥
p2	s2	⊥	8	⊥

p_id	s1	s2	s3		s4
	c1	c2	c1	c2	c4
p1	⊥	⊥	5	12	10
p2	7	8	⊥	⊥	⊥

Figure 5. Pivot table derived from relational table.


DBSheet – Kangwon National University Computer Science Database Lab
데이터베이스(B) 편집(E) 보기(V) 삽입(I) 데이터(D) 도움말(H)
실행
소요 시간 : 0.438 초

질의 : select * from pivot

형태 : row# 1000

pcountry	pname	Gov	Gov	GovA	GovA	GovC	GovC	Home
		S12	S43	S10	S47	S26	S7	S16
		2001	1995	1996	2003	2002	1995	2001
Austia	SpeakerB	0	0	0	0	0	0	0
Brazil	PrinterA	64	0	0	0	0	0	0
Canada	Monitor	0	0	0	0	0	0	0
Colombia	CameraE	0	84	0	0	0	0	0
Denmark	FishE	0	0	0	0	0	0	0
Dominica	MP3PD	0	0	0	0	0	0	0
Finland	PrinterB	0	0	0	0	0	0	0
Greece	AppleD	0	0	0	0	0	0	0
Hungary	BananaC	0	0	0	0	0	0	0
Iran	BookE	0	0	0	0	0	0	0
Jordan	HeaterC	0	0	0	0	68	0	0
Korea	SnackA	0	0	0	0	0	0	0
Nepal	RadioD	0	0	0	0	0	87	0

Figure 6. Result of pivoting feature in the OLAP interface.

Figure 6 shows an example result of pivot table created by the pivoting feature. As shown in the figure, the rows of the example pivot table came from two attributes of source table (i.e., *pcountry* and *pname*) and its columns from three others (i.e., *pplace, pstore,* and *pyear*). Users can select dynamically rows and columns as they want. Therefore the pivoting feature can create pivot tables over multiple dimensions and it can provide users with a user-friendly representation for multidimensional data.

4. Storing Pivot Tables and Query Transformation Rules

We can not store directly pivot tables of horizontal structure in relational databases which have a large number of columns. In this paper, therefore, we store horizontal tables by vertical table form and we convert queries of horizontal tables into ones of vertical tables through query transformation rules. In this section, we introduce a method of storing horizontal tables by vertical tables and query transformation rules between them. We assume that a horizontal table, H, has the schema (O, $A_1, A_2, \ldots, A_n$) and each tuple of H is represented as ($O, M_1, M_2, \ldots, M_n$). The vertical table, V, storing horizontal table H has the schema(O,A,M) and each tuple in V is represented as (O_i, A_i, M_j).

Now, we introduce a method storing horizontal table H into vertical table V. That method converts the horizontal table H into the vertical table V by changing the role of row and column. This converting can be defined as a relational algebraic expression as shown in [7]. The following Eq. (1) shows the relational algebraic expression for storing horizontal table H into vertical table V from [7].

$$V = [\cup_{j=1}^{n} \pi_{Oid,' A_j', A_j} (\sigma_{A_j \neq \perp}(H))] \quad (1)$$

According to Eq. (1), each tuple $(O_i, M_1, M_2, \ldots, M_n)$ of H is represented as a set of (O_i, A_j, M_j) tuples of V if M_j, the value of attribute A_j, is not null.

Then, we introduce PIVOT operation which converts vertical table V into horizontal table H. It is the contrary operation of the storing process described in Eq. (1). The PIVOT operation was defined as the following algebraic expression in the following Eq. (2) by [7].

$$H = \mathrm{PIVOT}\,{}^{[A_1,\ldots,A_n]}_{A \ on \ M}(V)$$

$$= [\bowtie_{j=1}^{n} \pi_{Oid,M}(\sigma_{A='A_j'}(V))] \quad (2)$$

When PIVOT operation over $[A_1, \ldots, A_n]$, the value of attribute A, applies to vertical table V, as shown in Eq. (2), we get horizontal table H as the result. In this paper, therefore, we propose a method for query transformation of horizontal table to vertical one which utilizes PIVOT operation(= $\mathrm{PIVOT}_{A\ on\ M}^{[A_1,\ldots,A_n]}(V)$) provided in recent commercial DBMSs rather than complicate relational algebra operations (= $[\bowtie_{j=1}^{n}\pi_{Oid,M}(\sigma_{A='A_j'}(V))]$).

The following Figure 7 shows an overall framework that processes queries on horizontal tables by using vertical tables. From the figure, we can see that each horizontal table query(*Query*) is processed by a vertical table query(*Query'*) followed by PIVOT operation. By using the proposed transformation method, we can get the result derived from vertical table in the right side of the Figure 7 which is equivalent to the result from horizontal table in the left side.

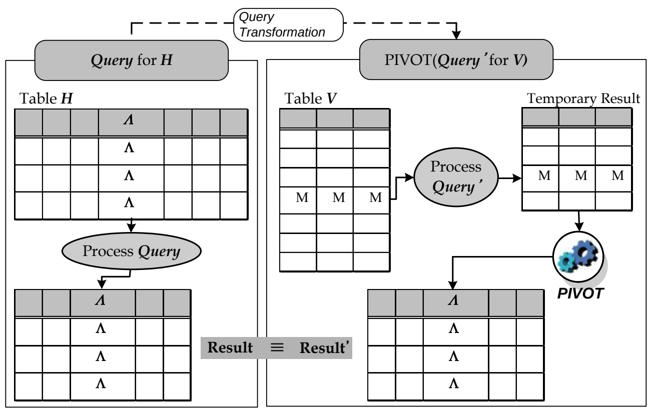


Figure 7. Query transformation framework between horizontal table and vertical table.

In the followings, we present transformation rules for two algebra operations such as selection and projection by using the PIVOT operation. We omit the transformation rules for the other operations due to space limitation.

Projection. As shown in Eq. (3), the projection on Horizontal table *H* can be converted to the PIVOT operation on vertical table *V* over the attributes to be projected from *H*.

$$\pi_{Oid,B_1,\ldots,B_k}(H) = [\bowtie_{j=1}^{k}\pi_{Oid,M}(\sigma_{A='B_j'}(V))]$$

$$= \mathrm{PIVOT}_{A\ on\ M}^{[B_1,\ldots,B_k]}(V) \qquad (3)$$

where, $B_j = A_i (1 \le j \le i \le n)$.

Example 1: The Figure 8 is an example converting a projection operation on the horizontal table *SalesH* into a PIVOT operation on the vertical table *SalesV* in the Figure 1. As shown in the figure, we can see that Eq. (3) converts the projection query of *SalesH*, $\prec_{ShipID,TV,Camer}(SalesH)$, into the PIVOT operation of *SalesV*, $\mathrm{PIVOT}_{Product\ on\ Sales}^{[TV,Camera]}(SalesV)$. The final results of these two queries are equivalent each other. ☐

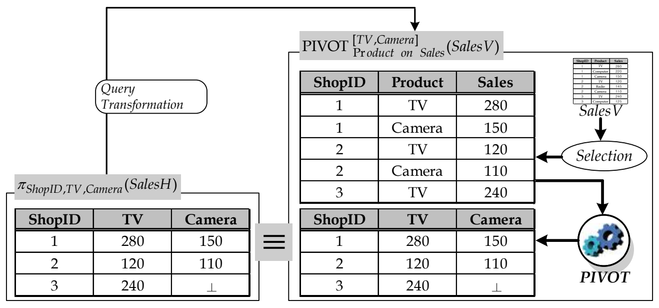


Figure 8. Example of projection query transformation.

Selection. The selection operation on horizontal table *H* can be converted into selection and projection on vertical table *V* followed by a PIVOT operation as defined in Eq. (4). Here, the predicate of selection operation is expressed in local ANDs(∧) of A_i□`M_i□ terms.

$$\sigma_{\wedge_{i=1}^{k}(A_i\theta' M_i')}(H) =$$

$$[\bowtie_{j=1}^{n}\pi_{Oid,M}(\sigma_{A='A_j'}((\bigcap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta' M_i'}(V)))\bowtie V))]$$

$$= \mathrm{PIVOT}_{A\ on\ M}^{[A_1,\ldots,A_n]}((\bigcap_{i=1}^{k}\pi_{Oid}(\sigma_{A='A_i'\wedge M\theta' M_i'}(V)))\bowtie V) \quad (4)$$

Example 2: The Figure 9 is an example converting a selection operation on the horizontal table *SalesH* in the Figure 1. By the Eq. (4), as shown in the figure, the selection query on *SalesH*, $\sigma_{TV\le 200}(SalesH)$, was converted into a corresponding expression on the vertical table *SalesV* in the Figure 1. These two queries produce the same result in horizontal table form. ☐

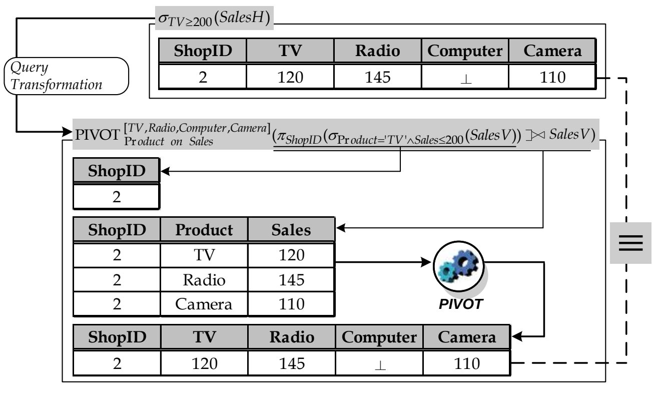

$\sigma_{TV\ge 200}(SalesH)$
ShopID TV Radio Computer Camera
2 120 145 ⊥ 110
Query Transformation
$\mathrm{PIVOT}_{Product\ on\ Sales}^{[TV,Radio,Computer,Camera]}(\pi_{ShopID}(\sigma_{Product='TV'\wedge Sales\le 200}(SalesV))\bowtie SalesV)$
ShopID
2
ShopID Product Sales
2 TV 120
2 Radio 145
2 Camera 110
PIVOT
≡
ShopID TV Radio Computer Camera
2 120 145 ⊥ 110

figure 9. Example of selection query transformation.

5. Query Optimization Technique

In this section, we introduce a query optimization technique for query transformation between horizontal and vertical tables. We suppose that queries on a horizontal table *H* are transformed into ones on a vertical table *V* through the Eq. (5) using PIVOT operation. We describes possible ways to process transformed queries and we propose an efficient way processing the transformed queries.

$$H \xrightarrow[Transformation]{Query}$$

$$PIVOT\left[\begin{matrix} A \text{ in} \\ (A_1, A_2, \ldots, A_n) \end{matrix} \mathrm{F}_{Function\ \exists M}(V)\right] \qquad (5)$$

As we can see at the above Eq. (5), PIVOT operation is executed by first extracting $A_1, A_2, \ldots, A_n$, the values of attribute *A* , from vertical table *V*. The *in* operator is used to extract those over attribute *A*. After that, aggregate function(s), $\mathrm{F}_{Function}$, is/are applied to attribute *M* which includes measurements. We can use aggregation functions such as *sum, avg, max, min, count,* etc.

5.1. Processing Projection Operation

Figure 10 shows four possible ways transforming the projection operation on horizontal table *H* into the one on vertical table *V*. The left side of the figure shows a general form of projection operation on horizontal table. It can be converted into several ways of projection on vertical table which are shown in the right side of the figure. These ways produce same results but they may have different execution time. Through experiments, we compared their execution time and we found that Figure 10(a) is a better execution plan that the others.

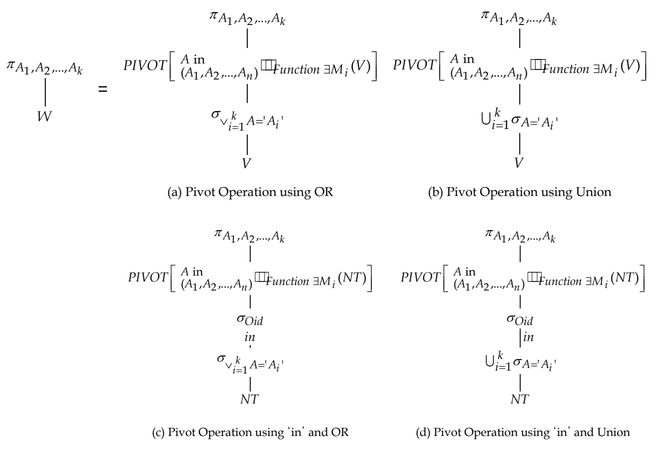

(a) Pivot Operation using OR
(b) Pivot Operation using Union
(c) Pivot Operation using 'in' and OR
(d) Pivot Operation using 'in' and Union

Figure 10. Transformation plans for projection operation.

5.2. Processing Selection Operation

In the case of selection operation which retrieves tuples satisfying a specific condition at horizontal table *H*, it can be also converted into several ways over vertical table *V*. The right side of Figure 11 shows two of them which are equivalent to the selection on *H* in the right side of the figure. These have different execution time thus we have done experiments to find a better way. From the experiments, we found that Figure 11(b) has better performance in overall cases.

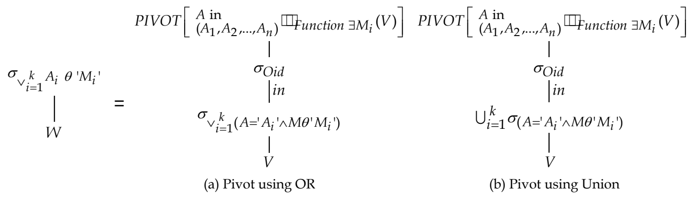


Figure 11. Transformation plans for selection operation.

5.3. Optimization Strategy on PIVOT Operation

Based on the processing plan for basic operations described above, Figure 12 shows an optimized execution process for transformed queries including PIVOT operation. The optimization strategy used in that figure is as follows: 1) PIVOT operation is applied after executing projection and selection on vertical table *V*; and 2) Join operation is applied after processing every operation to vertical table *V* and other operations (i.e., selection and projection) to another relational table *R* to be joined. This strategy was designed to find a better execution plan by comparing various ways of executing transformed queries.

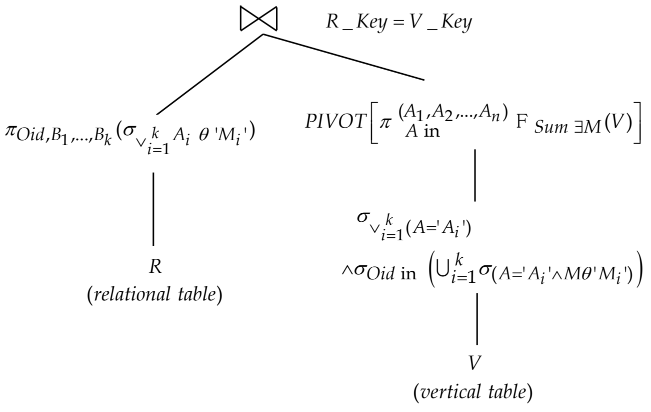


Figure 12. Optimization strategy on transformed query.

6. Conclusions

OLAP system supports various analyses from a huge volume of data which can be used for business decision making in an enterprise. To do it, we need useful tools that can show data in various criteria (i.e., dimensions). Spreadsheet is the most popular tool for end users to handle and to analyze simply data for their own purpose. In this research, we develop a graphical interface which users can handle data stored in databases just like spreadsheet. To browse data in relational tables with a large number of rows, we provide a mechanism partitioning virtually relational tables into multiple sheets based on attribute values or the number of rows. Also we provide a function making pivot tables from relational tables like MS Excel. Pivot tables have horizontal format which has attribute values (not attribute names) on their columns. Because they can have so many columns (e.g, several thousands), we convert the horizontal format into vertical table in order to store pivot tables into relational databases. We provide transformation rules between queries on horizontal tables and ones on vertical tables. Because there are several ways executing transformed queries, we also investigated an optimization strategy to find better execution plan for them. With these facilities, users can easily handle data in relational DBMSs like in spreadsheet. The proposed interface will be able to be used to manipulate huge data needed in OLAP analysis or e-business environment easily and effectively.

References

[1] Chaudhuri, S. and Dayal, U., "An Overview of Data Warehousing and OLAP Technology," In *Proc. the ACM SIGMOD Int'l Conf.*, Vol. 26, No. 1, pp. 65-74, March 1997.

[2] Witkowski, A., et al., "Spreadsheets in RDBMS for OLAP," In *Proc. Int'l Conf. on Management of Data*, ACM SIGMOD, San Diego, California, pp. 52-63, June. 2003.

[3] Agrawal, R., Somani, A., and Xu, Y., "Storage and Querying of E-Commerce Data," In *Proc, the 27th Int'l Conf. on Very Large Data Base*, Roma, Italy, pp. 149-158, Sept. 2001.

[4] Gray, J., et al., "Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals," *Technical Report MSR-TR-95-22*, Microsoft Research, Advance Technology Division, Microsoft Corporation, Redmond, 1995.

[5] Witkowski, A., et al., "Business Modeling Using SQL Spreadsheets," In *Proc. the 29th Int'l Conf. on Very Large Data Bases*, Berlin, Germany, pp. 1117-1120, Sept. 2003.

[6] Cunningham, C., Graefe, G., and Galindo-legaria, C. A., "PIVOT and UNPIVOT: Optimization and Execution Strategies in an RDBMS," In *Proc. the 30th Int'l Conf. on Very Large Data Bases*, Toronto, Canada, pp. 998-1009, Aug. 2004.

[7] Chen, S. and Rundensteiner, E. A., "GPIVOT: Efficient Incremental Maintenance of Complex ROLAP Views," In *Proc. the 21st Int'l Conf. on Data Engineering (ICDE)*, IEEE, Tokyo, Japan, pp. 552-563, Apr. 2005.

[8] Microsoft SQL Server 2005. http://www. microsoft. com/sql.